Measuring What Matters:

The 4 Golden Signals of Service Health and Performance in Cloud-Native Applications

NETSIL

Introduction

With advent of containerization and modern cloud platforms, applications are rapidly going through a paradigm shift from monolithic to microservices architectures. Modern cloud applications are increasingly services-driven and interact on the network using APIs, RPCs and database calls. These network interactions are either internal, between microservices, or external, with third party SaaS/PaaS services such as Twitter Auth, Twilio, AWS DynamoDB and so on. This rapid change comes with implications from an operational perspective for DevOps engineers and Site Reliability Engineers (SREs) as these distributed architectures expose a different kind of complexity, rendering most existing monitoring products, that were designed with code-level complexity in mind, ineffective.

In this white paper, we focus on best practices for measuring what matters when it comes to service health and performance of cloud-native applications -- what we refer to as “Golden Signals”. The golden signals of monitoring are the foundation of service-level observability for large-scale distributed applications. Proactive alerting on golden signals is critical to assuring service uptime and performance SLAs. Further, these golden signals ultimately help measure end-user experience, service abandonment and impact on business. After discussing these golden signals, we describe various approaches to measure them and discuss their implementation effort and costs.

The 4 Golden Signals for Service Health and Performance

The four golden signals are latency, traffic, errors, and saturation. These have been championed by the Google SRE team and the larger web-scale SRE community as the most fundamental metrics for tracking service health and performance.

Here is a brief description of these four golden signals:

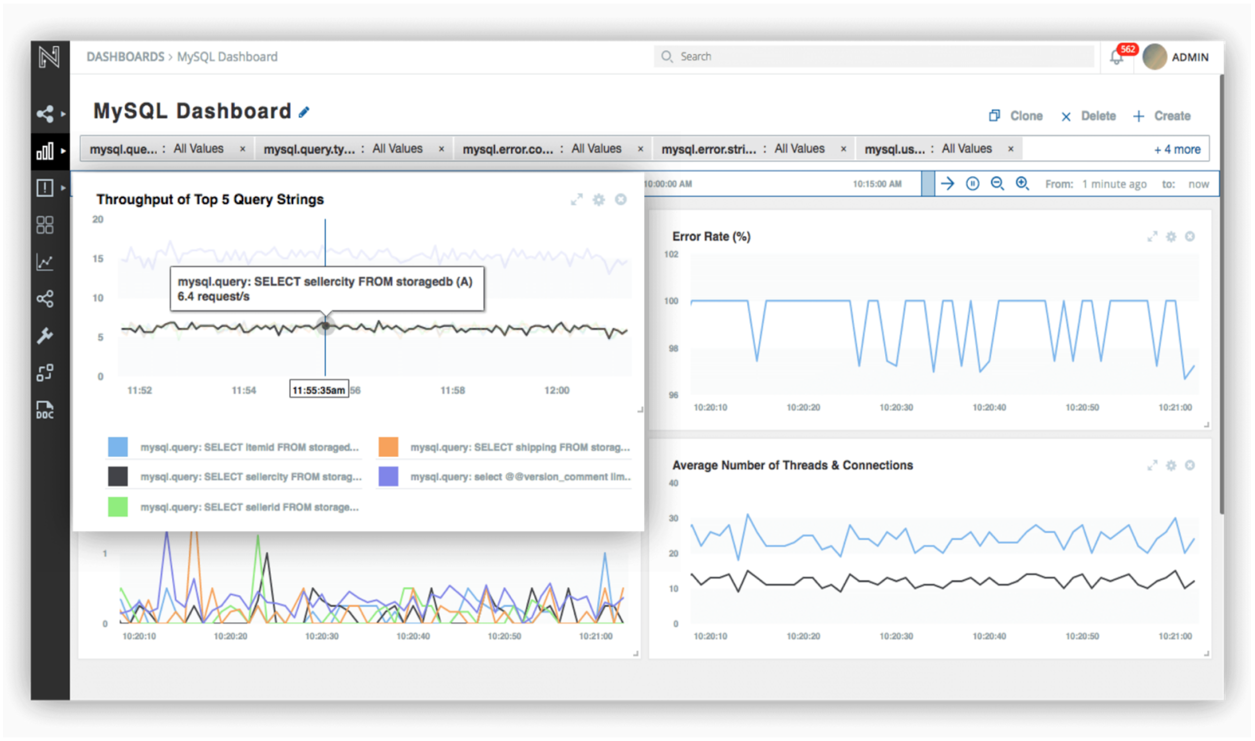
- **Latency:** The time it takes to service a request, with a focus on distinguishing between the latency of successful requests and the latency of failed requests.
- **Traffic:** A measure of how much demand is being placed on the service. This is measured using a high-level service-specific metric, like HTTP requests per second in the case of an HTTP REST API, queries per second in case of databases and so on.
- **Errors:** The rate of requests that fail. The failures can be explicit (e.g., HTTP 500 errors) or implicit (e.g., an HTTP 200 OK response with a response body having too few items).
- **Saturation:** How "full" is the service. This is a measure of the system utilization, emphasizing the resources that are most constrained (e.g., memory, I/O or CPU). Services degrade in performance as they approach high saturation.

By focusing their monitoring and alerting on these golden signals, SREs and DevOps can proactively manage the Service Level Objectives (SLOs) for the services and APIs that make up their production applications.

Why Golden Signals Are Useful?

Let’s take the example of HTTP REST APIs to see how golden signals could be valuable to your application on latency, traffic and errors.

- **Latency:** Latency could be measured for each API call by timing the requests by clients on individual API endpoints and responses sent back by servers. Simply measuring latency at API endpoint-level is not as informational as it may lead to misleading conclusions about API health. For example, an HTTP 500 error that fails fast is better than a slow error. Similarly, a delayed HTTP 200 OK response may be considered an error in some cases. Drilling-down further into the latency of an HTTP 500 error versus the latency of an HTTP 200 OK response gives greater visibility into this signal.
- **Traffic:** Traffic (or throughput) could be measured as the number of requests per second (RPS) served (broken down by REST endpoints, in our example). Traffic signals provide key insights into how busy a service is at any given time and is valuable for capacity planning.
- **Errors:** Reliability Engineers need to monitor the error rates, or the rate of request failures, by explicitly tracking the server response codes. However, sometimes the server response codes alone may not be sufficient to identify failures. In such cases, errors can be identified using other metrics. For example, a malformed request for a search API might return no responses. Let’s say the average response size is 500KB but the size for the response to a malformed request is only 2KB (along with a response code of 200 OK). Then the error can be identified by monitoring the size of the responses for anomalies.
- **Saturation:** Many application health issues are rooted in problems related to the underlying network or infrastructure such as CPU utilization, Disk I/O, Network I/O and System Memory. Saturation helps understand how much spare capacity is available and roughly how much more a service can take. Also, saturation can help predict possible system failures, e.g. disk filling up in a few hours on a host.


DASHBOARDS › MySQL Dashboard
Search
562
ADMIN
MySQL Dashboard
Clone
Delete
Create
mysql.que... : All Values
mysql.query.ty... : All Values
mysql.error.co... : All Values
mysql.error.stri... : All Values
mysql.us... : All Values
+ 4 more
10:00:00 AM
10:15:00 AM
From: 1 minute ago to: now
Throughput of Top 5 Query Strings
mysql.query: SELECT sellercity FROM storagedb (A)
6.4 request/s
11:55:35am
mysql.query: SELECT itemid FROM storaged...
mysql.query: SELECT shipping FROM storag...
mysql.query: SELECT sellercity FROM storag...
mysql.query: select @@version_comment lim...
mysql.query: SELECT sellerid FROM storage...
Error Rate (%)
Average Number of Threads & Connections

MySQL Golden Signals in Netsil AOC

Approaches to Measure Golden Signals

Signals: Saturation

- **Collection Daemons:** This signal is easiest to collect and there are several open-source collection daemons available that can poll for saturation metrics for the infrastructure and various application stacks. A few of the popular open-source daemons are collectd, Telegraf and dd-agent.

Signals: Latency, Traffic and Errors

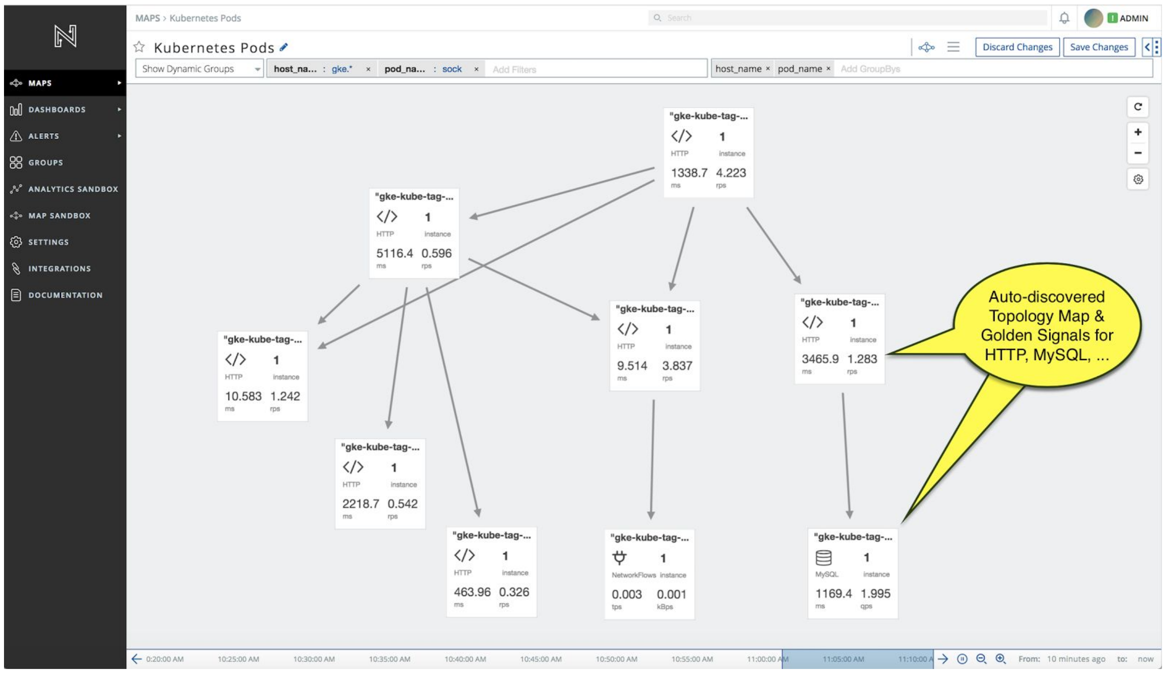
- **APM agents:** Application performance management (APM) approach can measure golden signals but they require code-embedded agents or language SDKs on all services in order to track code execution and request paths. AppDynamics, New Relic, and Dynatrace are some popular products in this category that leverage code profiling and transaction tracing to gather golden signals such as latency, traffic and errors. APM agents are hard to implement across heterogeneous languages and frameworks and configure in production. They are also known to make services instable due to compatibility issues and have inherent overheads. Hence, even though it is not ideal, many practitioners implement APM agents on a fewer hosts in their user facing services, forgoing monitoring and alerting on vast swaths of their applications.
- **Logging and metrics SDKs:** Some practitioners detect golden signals by using logs or publishing metrics from within the application code or proxies. Some technologies such as Apache web server and Nginx proxies can also provide detailed logs for each request. However, logs are very impractical in reality as they require emitting standardized or structured messages on each service and open-source components. In addition, logs have huge storage overhead and organizations can typically keep few days of logs for historical purposes. Metrics on the other hand are more practical than logs in terms of standardized formats and storage overheads, but require a lot of discipline on part of the development teams to publish them at entry and exit points across all services. Some popular metrics SDKs are statsd and Dropwizard. Statsd metrics collection is often bundled with, or available as a plugin in collection daemons such as collectd, Telegraf and dd-agent.
- **Distributed Tracing:** This approach requires developers to embed tracing SDKs in the application code and use them to track entry points and exit calls. These SDKs don't look at code execution but instead just inject headers in requests to help correlate across services. Some techniques apply sampling to help scale in production. SDKs emit trace spans, which contain the unique trace ID and other metadata/ tags. Some popular products in this category include OpenTracing, Datadog APM, AWS X-Ray, Finagle, inkerd and Envoy. Traces typically contain metadata such as errors and endpoints, in addition to timing data for calculating latency distributions. In order to get golden signals, trace samples are often aggregated over time. The downside of this approach is that

developers need to implement tracing SDKs across all their services and open-source components, making it an expensive solution to implement on all services and legacy systems in practice.

- **Operating System Tracing:** Modern Operating Systems provide various tracers that allow tracing not just the syscalls or packets, but also any kernel or application software. For example, tcpdump is a network tracer in Linux. Other popular tracers are eBPF and DTrace. Some popular products that rely on OS Tracing for gathering golden signals include Netsil and Sysdig. This approach started getting a lot of popularity with the recent push towards microservices architectures. This approach typically requires installing an agent at the host level that can capture network interactions from the OS and reconstruct their protocol state-machines to calculate latency of requests and gather metadata such as API endpoints, database queries, errors and so on. This approach requires the least effort and friction to implement as it is language and framework agnostic and provides the most accurate measurements for golden signals such as latency, traffic and errors. The downside of the OS tracing approach is that it does not work when the network traffic is encrypted, though some solutions make use of transparent MITM proxies to intercept and analyze SSL interactions to measure the golden signals.

Netsil’s Approach to Golden Signals

The Netsil Application Operations Center (AOC) is an observability and analytics product used by SREs and DevOps who run API and services driven cloud applications.


MAPS > Kubernetes Pods
Kubernetes Pods
Discard Changes
Save Changes
MAPS
DASHBOARDS
ALERTS
GROUPS
ANALYTICS SANDBOX
MAP SANDBOX
SETTINGS
INTEGRATIONS
DOCUMENTATION
"gke-kube-tag-... HTTP 1 instance 1338.7 ms 4.223 rps
"gke-kube-tag-... HTTP 1 instance 5116.4 ms 0.596 rps
"gke-kube-tag-... HTTP 1 instance 10.583 ms 1.242 rps
"gke-kube-tag-... HTTP 1 instance 9.514 ms 3.837 rps
"gke-kube-tag-... HTTP 1 instance 3465.9 ms 1.283 rps
"gke-kube-tag-... HTTP 1 instance 2218.7 ms 0.542 rps
"gke-kube-tag-... HTTP 1 instance 463.96 ms 0.326 rps
"gke-kube-tag-... NetworkFlows 1 instance 0.003 tps 0.001 kBps
"gke-kube-tag-... MySQL 1 instance 1169.4 ms 1.995 qps
Auto-discovered Topology Map & Golden Signals for HTTP, MySQL, ...

Automatically discovered service topology map with operational metrics (golden signals)

While the programming languages and web frameworks used to build the services change frequently with time, service communication protocols such as HTTP remain relatively constant, acting as the glue between them. Leveraging this insight, Netsil captures service interactions in real-time as its source-of-truth, without instrumenting the application code. Netsil’s specialized stream-processing technology analyzes these interactions and automatically discovers the service topology map of the application, overlaying it with operational metrics (including the golden signals).

Code-based application monitoring tools require upfront investment in the form of instrumentation of code before they can provide value. Further, they do not give visibility into calls made to external APIs as they cannot instrument them. The Netsil approach works seamlessly for both internal and external APIs, and the lack of upfront investment leads to fastest time-to-value.