12

Immutable Rules for

Observability

splunk>

turn data into doing™

Introduction

Speed defines success in today's digital economy. With customers expecting flawless digital experiences and competition hovering just a click away, companies turn to cloud-native technologies like microservices, containers and Kubernetes to accelerate innovation, build applications faster and improve performance. However, moving to cloud-native technologies and distributed architectures introduces new challenges around speed, scale and complexity of data — challenges that traditional monitoring solutions simply weren't designed to handle.

This is where observability comes in.

Organizations must deliver high quality code and differentiated user experiences — fast. Observability lets DevOps and SRE teams understand and explain unexpected behavior, so they can effectively and proactively manage the performance of distributed microservices running on ephemeral infrastructure. The right observability strategy and solution translates into more reliability, better customer experience and higher team productivity.

Here, we provide 12 immutable rules for observability for ongoing success no matter the complexity of your environment. The more observable a system is, the quicker we can understand why it's acting up and fix it — which is critical when meeting service-level indicators (SLIs) and objectives (SLOs) and, ultimately, accelerating business results.

First things first, a quick observability definition:

Observability is a measure of how well we can infer or answer any question about the state of our systems (infrastructure, services, etc.) using their telemetry data (metrics, traces and logs).

Observable systems empower DevOps teams to troubleshoot any issue that may occur in their systems, including unknown failures and failures with root causes buried deep in a microservices maze.

Beyond troubleshooting, observability gives teams a window into their systems and the opportunity to proactively improve code releases and system architecture and adapt to change faster.

However, the mere availability of data doesn't deliver an observability solution. As observability becomes an integral part of the DevOps toolchain, it's important to keep in mind some immutable rules when considering, adopting and improving your observability.

Observability starts with data

How do you *really* know what your services are doing, even during development? It all starts with the data.

01 An observability solution uses all your data to avoid blind spots

The only way to troubleshoot a needle-in-a-haystack unknown failure condition and optimize an application's behavior is to instrument and collect all the data about your environment at full fidelity and without sampling. It's the only way to guarantee no visibility gaps. You have the data you need, when you need it.

Service-oriented distributed architectures create more complex cross-service boundary interactions, dependencies and error propagation that result in highly unpredictable systems with a long tail of infrequent but severe issues. Traditional observability solutions can fall short for monitoring microservices-based applications because they use head-based probabilistic sampling, which randomly samples traces and often misses the ones that you care about (unique transactions, anomalies, outliers, etc.).

When assessing observability solutions, look for those that do not sample and also retain all your traces — you can decide which traces you want to keep — as well as populate dashboards, service maps and trace navigations with meaningful information that will actually help you monitor and troubleshoot your application.

02 Operates at speed and resolution of your new software-defined (or cloud) infrastructure

Different use cases with varying degrees of criticality require different resolutions. The resolution at which you collect data from your monolithic application will most likely not be sufficient as you start to collect data from more dynamic microservices running on ephemeral containers and serverless functions. For instance, if you're measuring the performance of a mature monolithic application running on over-provisioned virtual machines (VMs) with a relatively consistent number of users, it may be sufficient to have relatively coarse visibility (minute-level resolution) into your infrastructure. On the other hand, if you have microservices that are running on short-lived, Kubernetes-orchestrated containers that spin up and down automatically in minutes, or serverless functions that instantiate for only seconds, you'll need much finer granularity (one-second resolution) to effectively monitor the performance of your application and infrastructure.

As you begin to adopt microservices, it's a good idea to err on having higher resolution rather than lower, since the process of re-architecting an application or building a net new application natively in the cloud can often involve trial and error.

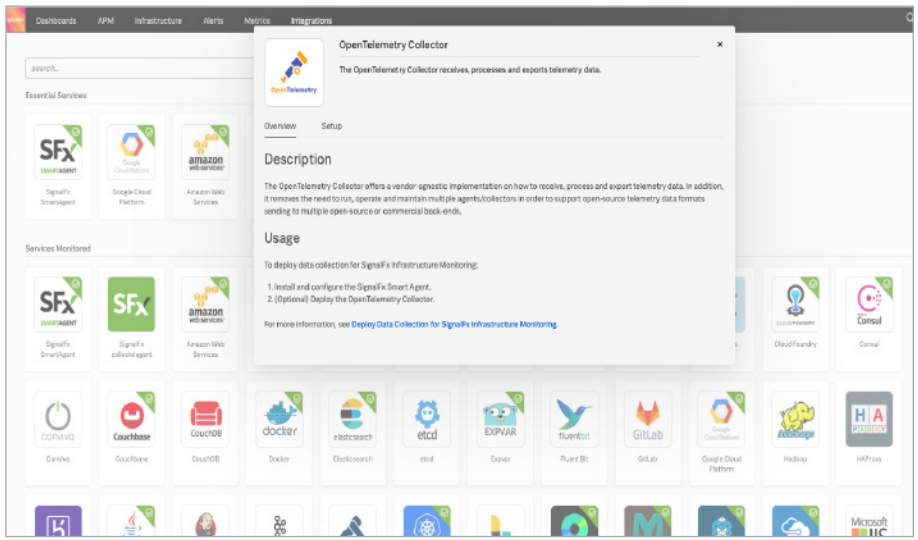
In other words, you need observability that operates in the same time scale as your software-defined, ephemeral infrastructure.

03 Leverages open, flexible instrumentation and makes it easy for developers to use

Plan on using open, standards-based data collection from day one. By selecting a standardized data format for trace, metric and log data and committing to open data ingest methods, it's easy to instrument your code and start capturing observability data, freeing up developer time.

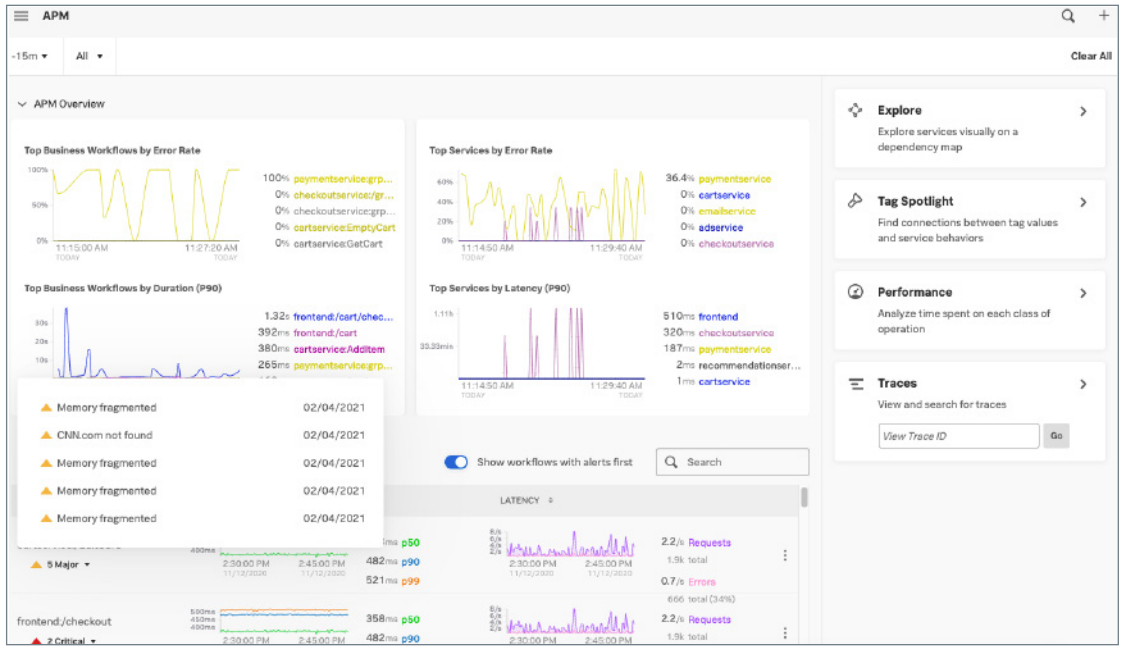
Heavy and proprietary agents are difficult to maintain, degrade service performance and are painful to replace, which all result in being locked into a specific observability solution that may not meet your expanding needs or may become increasingly expensive over time. Choosing to rely on common languages and frameworks — and leveraging OpenTelemetry, the second most active Cloud Native Computing Foundation project — will give you the most flexibility in not only how you collect data, but also what cloud solutions you use. Most importantly, using OpenTelemetry sets you up for success when the time comes to scale and expand monitoring and troubleshooting for ever-increasing distributed microservices.

Open instrumentation also makes it easier to integrate with everything in your existing toolchain for code-to-cloud visibility.



Observability provides visibility into your DevOps practices and toolchain visibility. That visibility is key to sustaining application velocity and supporting new tools and processes. While there is no one-size-fits-all cloud solution, language, incident response product or set of DevOps tools, your observability solution should be able to easily integrate with and provide insights into any tools that you're using or may use in the future.

04 Enables a seamless workflow across monitoring, troubleshooting and resolution with correlation, and data links between metrics, traces and logs



Organizations manage multiple point tools. It's not uncommon to find application owners flagging a performance degradation with a point APM tool — then contacting a different IT operations team that has a separate view of servers and hosts with a point infrastructure monitoring tool to try to understand whether infrastructure problems are impacting critical workloads and business performance.

This approach isn't sustainable when trying to resolve issues quickly. You should be able to easily understand interdependencies and any upstream or downstream effect of a particular issue in one system, and have a cohesive workflow no matter where exploration starts.

Your observability solution should have all capabilities fully integrated, providing you with relevant contextual information throughout your troubleshooting workflow, regardless of your

title (frontend engineer, SRE, DevOps engineer). Go anywhere the data takes you without running into dead ends. For example, it should be easy to jump from an alert to relevant trace details and carry that information to correlate application degradation with underlying infrastructure or frontend problems.

Many teams create and deliver modern applications, making issue resolution challenging and siloed. Observability helps reduce unnecessary discovery steps and troubleshooting dead ends by seamlessly passing information, allowing for a continuous investigative flow.

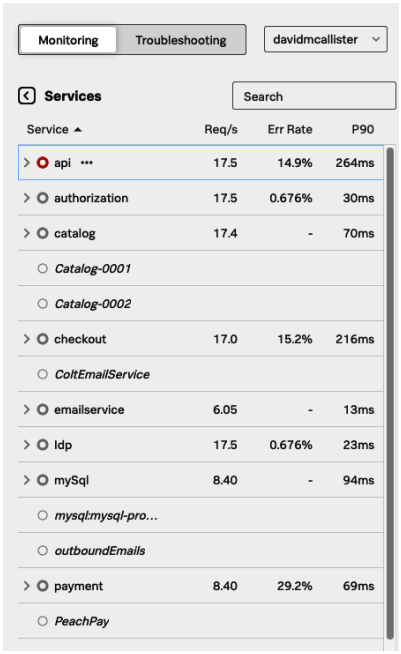


Fig: triggered RED metrics

Don't make teams repeat forensic steps. Why context and unified metrics, traces and logs are important in a troubleshooting workflow:

An alert on one service (metric) ····> SRE, operations

Leads to a timeout error (tracing) ····> DevOps engineer, software engineer

Leads to an infrastructure problem (metric) ····> DevOps engineer, SRE

Leads to a configuration issue (metric) ····> DevOps engineer, operations

Leads to a memory leak in an app (logs) ····> Developer, software engineer

Observability gets us to answers fast

With modern architectures comes a surge of data that impacts your understanding of your systems. But data alone is not meaningful; you need to be able to aggregate it, analyze it and respond to it as needed.

05 Makes it easy to use, visualize and explore data out of the box

Intuitive visualizations that require no configuration like dashboards, charts and heat maps make it easy to understand the enormous amounts of data your systems produce at a glance, and let you interact with key metrics in real time. Make sure your observability solution aggregates all data and automatically displays metrics dashboards, service maps and container architectures out of the box and allows for dynamic dimension-based filtering, grouping and aggregation. Your solution should also allow custom dashboards that can help keep an eye on particular services of interest.

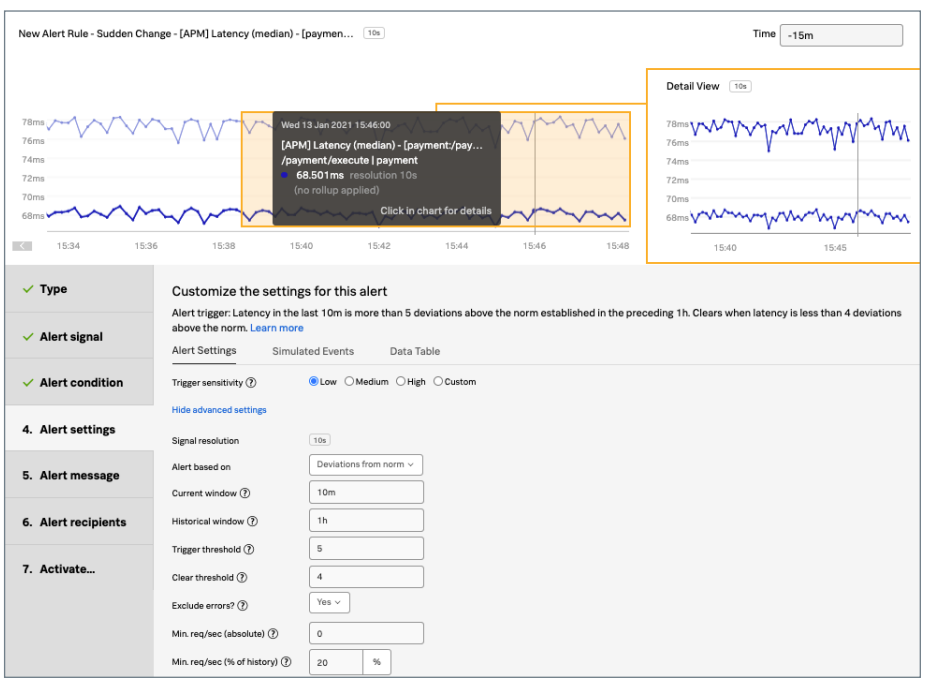
As mentioned, connected context is key. Imagine getting an alert from your observability solution that the 99th percentile latency of your service has gone up. You follow a link to the service dashboard right from the alert modal. The service dashboard shows all the components of the service, and the charts show that something's going wrong with the data store. Follow the link to the dashboard about the data store, and sure enough one of the instances started spiking latency about 15 minutes ago. You now know exactly where and when the problem started. Armed with the what, where

and when, now you can follow a link into the logs to discover the why, perhaps by looking at full stack traces written to logs.

Your observability tool should make it this easy — one alert, two dashboards and three clicks to narrow down the source of the problem.

06 Leverages in-stream AI for faster and more accurate alerting, directed troubleshooting and rapid insights

Cloud-native environments produce too much data to make sense of manually. To quickly process all this data, you need real-time analytics to surface relevant patterns and proactively deliver actionable insights. Basic alert triggers based on static thresholds and heartbeat checks are often inaccurate and result in a lot of noise. They typically cause floods of alerts that frustrate on-call engineers and add to the problem rather than helping to solve it. Instead of continuing to rely on these ineffective alert conditions, consider more dynamic thresholds based on advanced statistical models and AI, as well as more complex, multi-condition rules. Look for solutions that are effective at baselining historical performance, performing sophisticated comparisons and detecting outliers and anomalies in real time. They should also be able to tune and customize your alert rules to your specific application environment.



07

Gives fast feedback about (code) changes, even in production

Observability is not just for operations and should start during development.

Shifting left — which means starting DevOps processes like testing earlier in the pipeline — has become a popular strategy for teams working to find and fix issues sooner.

Shifting right — which involves extending pre-deployment processes into the production stage of the pipeline — helps achieve broader coverage of testing and monitoring.

Once code is deployed, teams need to understand what is happening within their applications as each release flows down the delivery pipeline. You can't understand your pipeline, or correlate pipeline events with application performance and end-user experience, if you don't understand what is happening inside your application.

This is where application testing and performance management come in to deliver code-to-cloud visibility. Observability delivers synthetic monitoring, analysis of real-user transactions, log analytics and metrics tracking, so teams can understand the state of their code from development through deployment. This understanding offers the depth that teams need to gain visibility into the state of each release across the development life cycle.

Splunk for DevOps

08 Automates and enables you to do as much “as code”

Dramatically improve the productivity, efficiency and predictability of your organization by leveraging programmability wherever possible. Leverage APIs to automatically manage infrastructure resources (e.g., via a Kubernetes orchestrator), change control and code deployments (e.g., via Jenkins integration). Build closed-loop automation throughout your production environment to perform sophisticated operations triggered by real-time alerts, such as automatic rollbacks and remediation, bringing mean-time-to-resolution (MTTR) down to machine time.

And in the “everything as code” movement, observability is no exception. The idea behind “observability as code” is that you develop, deploy, test and share observability assets such as detectors, alerts, dashboards, etc. as code.

Monitoring and alerting as code involve automated creation and maintenance of charts, dashboards and alerts as part of service life cycles. Doing so keeps visualizations and alerts current, prevents sprawl and allows you to maintain version control through a centralized repository, all without having to continuously manage each component manually. Leveraging available APIs and programmability also helps ensure that visualizations and alerts are consistent with best practices and enterprise policies throughout your organization.

```
provider "signalfx" {
   # It is strongly recommended to use secret management Terraform Provider such as Vault
   auth_token="<<your access token>>"
   api_url = "https://api.us1.signalfx.com" #use your custom SignalFx URL
}
resource "signalfx_detector" "application_latency" {
 name        = "application latency is high"
 description = "SLI metric for application latency is higher than expected."
 program_text = <<-EOF
       signal = data('demo.trans.latency').max()
       detect(when(signal > 250, '1m')).publish('application latency is greater than 250 ms')
   EOF
 rule {
   description   = "Application latency was high for last one minute"
   severity      = "Warning"
   detect_label  = "application latency is greater than 250 ms"
   notifications = ["Email,amitsharma@splunk.com"] #you can also configure slack, VictorOps and others
 }
}
```

Fig: An example of how to create an alert detector in Splunk using the Terraform provider

Observability is critical to your culture and business strategy

Observability is a critical business investment, especially when seconds of downtime can cost millions of dollars. It extends past DevOps teams to support resiliency and flawless customer experiences.

09 Is a core part of business performance measurement

Reliability is critical when it comes to delivering high performing applications and flawless customer experiences, but you can't do reliability without observability. Without observability, how do you know where to invest time and resources? If you're not measuring availability, how do you know your reliability? If you're not measuring performance, how do you know how well you're really doing? These measurements need to go from development all the way through to production. In the data age, you need to know what's going on at every stage of delivery.

Observability provides a window beyond CPU utilization and basic metrics into every layer of the stack, as well as outputs like user experience, SLX performance and other key metrics tailored to your business needs. In cloud-native environments, small upticks in a service can spiral into increased latency for even a specific customer.

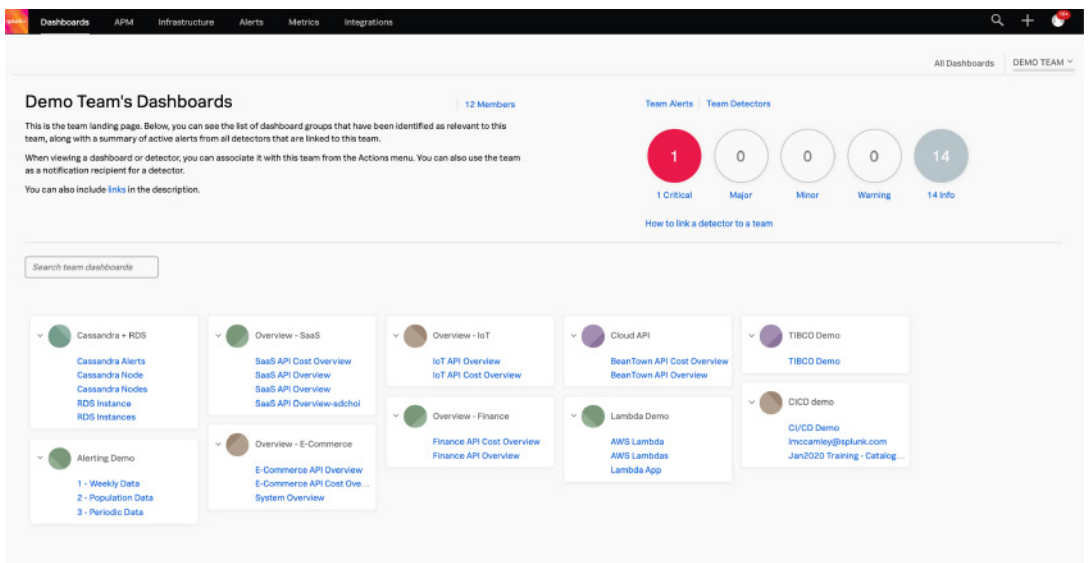
It's important to understand the KPIs by which your business is measured and how the teams within your organization will consume the data. You'll be able to:

- Anticipate what dimensions your monitoring data needs to have.
- Correlate data throughout your stack — from the underlying infrastructure to your applications and microservices.
- Correlate data across your entire digital business.

As a simple example, consider an application running on AWS that has users around the world. In order to get a complete picture of the end-user experience in each geographical region, it will be important to be able to slice-and-dice users, microservices and infrastructure based on AWS region or availability zone. Knowing ahead of time what kind of metadata is required will help you set up your visualizations right, the first time.

10

Provides observability as a service



The DevOps paradigm of "you build it, you own it" boosts agility in part by decentralizing operational responsibility to individual teams. More people across the organization now need access to observability, and this decentralization can easily lead to fragmented tools and data. Fragmentation can lead to higher costs and, even worse, highly inefficient operations. And with cloud resources being unlimited, costs can be even harder to manage.

Modern observability platforms should provide centralized management so teams and users have access controls and can gain transparency and control over consumption. Implementing clear best practices for observability across your business can not only cultivate a better developer experience, empowering them to work more efficiently and focus on building new features — it allows for improved cross-team collaboration, cost assessment and overall business performance.

11 Seamlessly embeds collaboration, knowledge management and incident response

While incidents may be inevitable, a strong observability solution can mitigate downtime and, perhaps, prevent it entirely — saving businesses money and improving the quality of life for on-call engineers. But most organizations haven't realized they control their preparation for recovery. To respond to and resolve issues quickly (especially in a high-velocity deployment environment), you'll need tools that facilitate efficient collaboration and speedy notification. Observability solutions should include automated incident response capabilities to engage the right expert to the right issue at the right time, all to significantly reduce downtime.

Other best practices to consider as you aim for observability:

- Evolve from tribal knowledge and hero-based incident resolution to standardized runbooks and knowledge bases. Providing all on-call engineers easy access to detailed context and suggestions about what has resolved similar issues in the past is a critical part of information sharing, collaboration and reducing MTTR.
- Allow seamless access to third-party tools (e.g., error tracking) via web links so that your on-call engineers can use them. This way, engineers can seamlessly carry the context of the incident to other systems and continue troubleshooting without missing a beat.

12

Scales to support future growth and elasticity

Invest based on your future observability needs rather than your current ones. How many containers do you have? What about the number of hosts in your environment, applications in production and code pushes per day? Per year? Answer these questions and you'll realize why you need a scalable monitoring system (or will need one in the future). To meet the needs of any environment — no matter how large or complex — observability solutions should be able to ingest petabytes of log data and millions of metrics and traces, all while maintaining high performance. This ensures that your investments are future-proof.

Building your observability strategy with Splunk

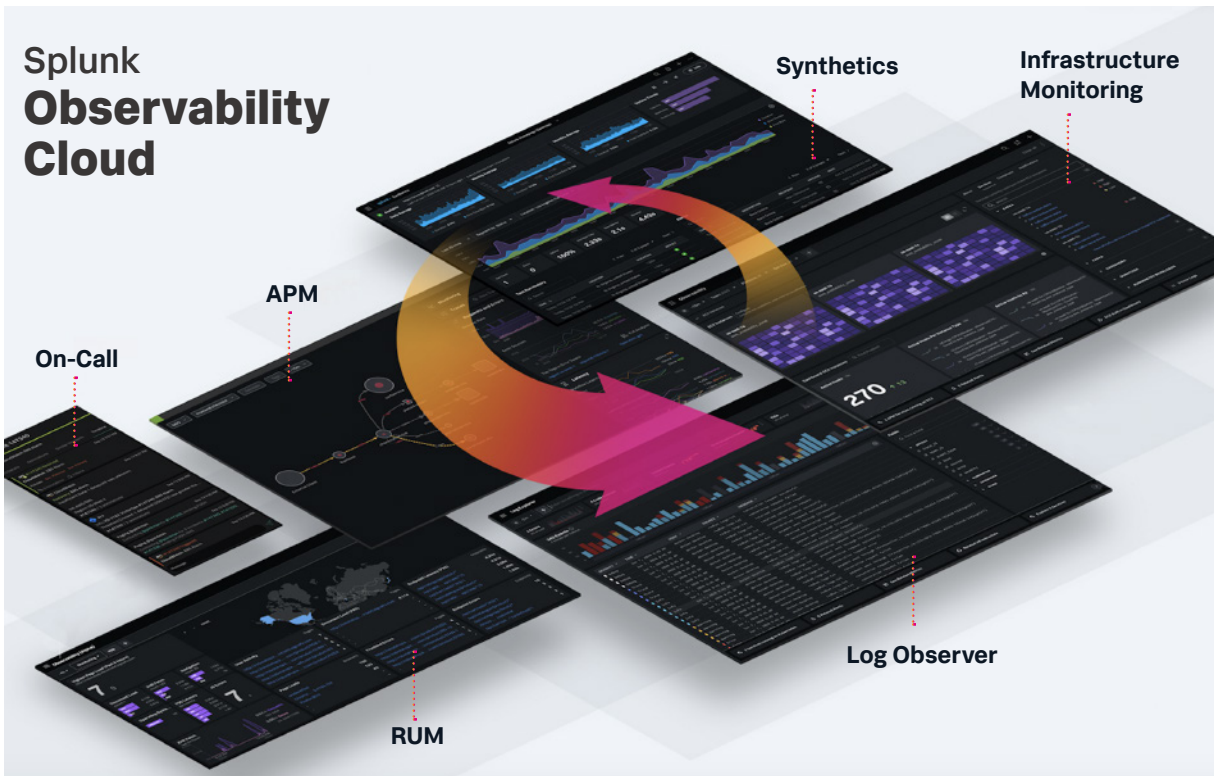
Observability is critical to success but it's not often an organization's core competency. That's why it's important to partner with an observability solutions provider like Splunk that can support you on your cloud journey.

Splunk Observability Cloud is the only full-stack, analytics-driven, enterprise-grade observability solution. It gives users a seamless and streamlined workflow for monitoring, troubleshooting and investigation — making it easy to go from problem detection to resolution in minutes. Whether you're a frontend developer who needs to know what customers are experiencing, a backend developer building APIs and services or an SRE who's frequently on call, Splunk Observability Cloud helps you get the insight you need to quickly resolve outages.

Splunk Observability Cloud includes:

Splunk Observability Cloud helps you conquer complexity, scale, and performance bottlenecks with:

- One tightly integrated user experience, enabling seamless and context-rich workflows for monitoring, troubleshooting and investigation — eliminating data silos and swivel-chair monitoring.
- End-to-end visibility based on open ingestion and correlation of ALL data — metrics, traces and logs.
- Unmatched speed and scale powered by a streaming analytics engine to detect issues and get feedback in real time (seconds, not minutes).
- AI-driven analytics that leverage the complete data set provide actionable insight, allowing you to proactively improve user experience.
- Centralized management, templatized best practices, cost control and observability-as-a-service for maximum ROI.



Learn More

To learn more about Splunk Observability Cloud, visit our website and sign up for our free trial.

21-17608-SPLK-12-Immutable-Rules-for-O11y-108

splunk> turn data into doing™