EBOOK

The Three Phases of Observability

What is observability?

Gartner defines observability as the evolution of monitoring into a process that offers insight into digital business applications, speeds innovation, and enhances customer experience.[1] Infact, the rise in popularity of the DevOps movement and Cloud-Native architecture is to enable digital businesses to become more competitive and great observability is a foundational requirement of this.

The need for observability was inherently born out of the DevOps movement — before DevOps, not many engineers needed to think about operating the systems that they built. Now that engineers both build and operate, it's critical to start thinking about building systems that are easier to observe.

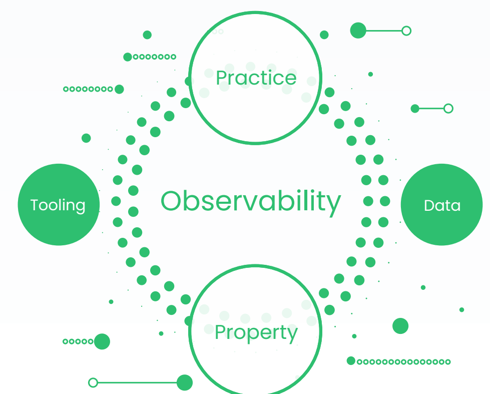
When we look at the outcome an engineer is trying to ultimately achieve with observability, it can be broken down into answering **three critical questions:**

1. How quickly do I get notified when something is wrong? Is it BEFORE a user/customer has a bad experience?
2. How easily and quickly can I triage the problem and understand its impact?
3. How do I find the underlying cause so I can fix the problem?

Regardless of what instrumentation exists and what tools or solutions are employed, the ability to answer the above three questions in order to remediate production issues as quickly as possible is fundamentally what we believe observability should be focused on.

OBSERVABILITY IS BOTH A PRACTICE AND A PROPERTY, AIDED BY TOOLING AND DATA

The term observability can be used to describe both a practice (or process), and to describe the property (or state) of a service. Observability, like DevOps, is a core competency of distributed systems engineering. It is the practice that cloud native developers do on a daily basis in increasingly complex systems as they answer the types of questions outlined above. Observability is also a property of a system — whether or not it produces data that can be used to answer any question that a developer asks of it. It is much easier to maintain and manage an observable system than a non-observable one.

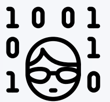


[1] Source: Innovation Insight for Observability
https://www.gartner.com/en/documents/3991053/innovation-insight-for-observability

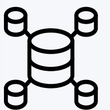
Why do companies need observability? Why now?

The need to introspect and understand systems and services is not new — many of the basic goals of observability have been in practice for decades. What's changed is the nature of the applications and infrastructure that teams are operating. Cloud-native applications running on containers and microservices have a completely different architecture and are designed to be more scalable, reliable, and flexible than legacy apps. Cloud-hosted monitoring and application performance monitoring (APM) were born in a pre-cloud-native world — one that had very different underlying assumptions. Cloud-native has forced organizations to revisit how they perform monitoring and observility because:

Data is growing in scale and cardinality.
Cloud-native environments emit a massive amount of observability data — somewhere between 10x and 100x more than traditional VM-based environments.



Systems are more flexible and ephemeral.
Both the usage patterns and retention requirements are vastly different to what they were pre-cloud-native.



Services and systems have greater interdependencies.
Breaking services down into microservices leads to more complex dependencies that engineers must understand in order to troubleshoot problems. This also results in a greater need to correlate and connect infrastructure to applications to business metrics.

All of this has led to an explosion in complexity that makes it nearly impossible to reliably and efficiently operate cloud-native services without dramatically increasing overhead or finding a new approach.

What observability is not

Today, there are many who define observability as a collection of data types — the three pillars: logs, metrics and distributed traces. While these are all critical inputs to observability, they are not observability solutions in and of themselves. Rather than focusing on outcome, this siloed approach to observability is overly focused on technical instrumentation and underlying data formats.

Simply having systems emit all three data types doesn't guarantee better outcomes — for example if a system emits metrics, logs and traces, there is no guarantee that you get notified in a timely manner, nor is there a guarantee you can triage issues quickly. What's more, many companies find little correlation between the amount of observability data produced and the value derived from this data — i.e., more logs or metrics doesn't equate to more value, even though it almost always equates to increased costs.

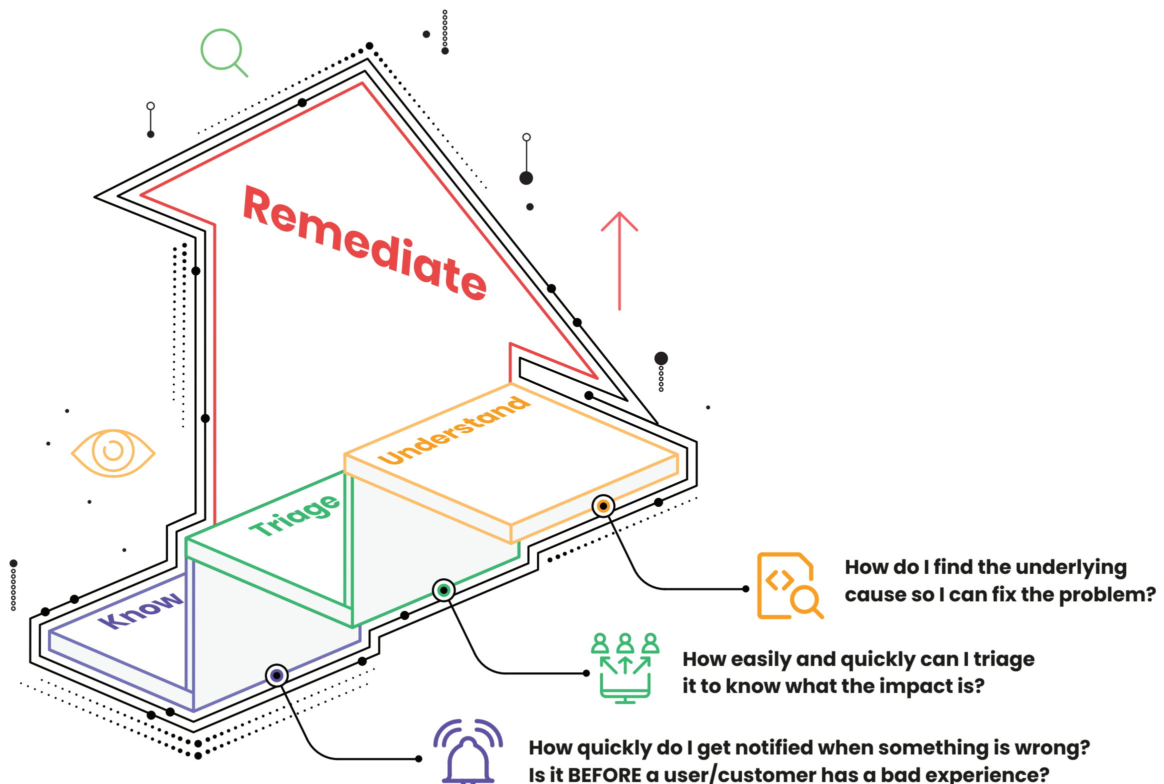
Break observability down into three phases

We're not the first to criticize the three pillars. We agree with much of the critique that others like Charity Majors and Ben Sigelman have put out there. Instead of the three pillars of observability, we've developed an approach to observability that is focused on the outcomes instead of the inputs and we call it **the three phases**. The phases are focused on positive observability outcomes and the steps teams can take to achieve these goals.

During each phase, the focus is on alleviating the customer impact — or remediating the problem — as fast as possible. Remediation is the act of alleviating the customer pain and restoring the service to acceptable levels of availability and performance. At each phase, the engineer is looking for enough information to remediate the issue, even if they don't yet understand the root cause. Each phase maps to answering one of the three critical questions we believe is required to achieve great observability.

"Too many companies look at their observability strategy and tick off the boxes: I have logs, I have metrics. I have traces. However, this doesn't necessarily mean you have observability, let alone great observability. Producing more of each of data type also doesn't lead to better observability. We think that an outcome based approach for the end user, which is the developer, is a better way to think about observability as a whole."

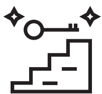
MARTIN MAO
Co-founder and CEO, Chronosphere



Phase 1: Know about the problem

The first step to resolving an issue is knowing the issue exists — ideally before your customer does. Often, knowing an issue is occurring is enough to trigger a remediation. For example, if you deploy a new version of a service and an alert triggers for that service, rolling back the deployment is the quickest path to remediating the issue without needing to understand the full impact or diagnose the root cause during the incident. Those can be examined after the issue is remediated, when there isn't active customer impact. Introducing changes to a system is the largest source of production issues, so knowing about problems as these changes are introduced is key

> "Fundamentally, what observability is trying to achieve is to create a model, or a map, of your system and your business in a way that humans can understand. The telemetry and data you produce should help people understand the system and mitigate problems faster."
>
> **ROB SKILLINGTON**
> Co-founder and CTO, Chronosphere



Keys to success:

Fast alerting: Shrink the time between a problem occurring and a notification firing.

Scope notifications to just the teams that need to act: Scope the problem and route it to the right teams from the start.

Improve signal to noise ratio: Ensure that alerts are actionable.

Automate alert set-up: Most services or hosts produce the same metric data which means automated or templatized alerting can help engineers know about problems without a complicated set-up process.

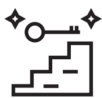
Tools and data:

- ✓ **Alerts**
- ✓ **Metrics** (native metrics as well as metrics generated from logs and traces)

Phase 2: Triage the problem

The goal of this phase is to quickly understand the context and impact of an issue. Once an alert goes off, if it is not obvious that a recent change to the system needs to be rolled back, the next step is to understand the business impact and the severity. Often, understanding the scope of the issue can lead to remediation. For example, if you determine that only customers in one experiment group are impacted, turning off that experiment would likely remediate the issue. Or, if requests to one availability zone or cluster are impacted, you can reroute requests to the other zones or clusters.

To help engineers triage issues, they need to be able to quickly put the alert into context of understanding how many customers or systems are impacted, and to what degree. Great observability allows engineers to pivot the data and shine a spotlight on the contextualized data to diagnose issues.



Keys to success:

Contextualized dashboards: Having alerts directly link to dashboards that show not only the source of the alert, but related and relevant contextual data.

High cardinality pivots: Allowing engineers to further slice and dice the data allows them to further isolate the problem.

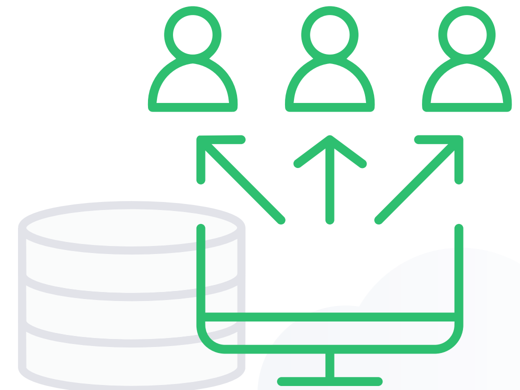
Leverage existing instrumentation: It's not practical to always assume that every use-case is instrumented perfectly, so it's important to be able to leverage existing instrumentation, but have them link as best possible for best contextualization.

Tools and data:

✓ Dashboards ✓ Metrics ✓ Logs

"If you emit 10 times the amount of logs or metrics as you did before, it doesn't mean you have a 10 times better mean time to resolution (MTTR). There's a mismatch in terms of the amount of data being produced and return on investment."

MARTIN MAO
Co-founder and CEO, Chronosphere



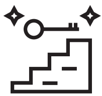
Phase 3: Understand the problem

This phase occurs ideally after remediation, when engineers can take the time to locate and understand underlying issues without the pressure of a ticking clock of customer expectations. With an ever increasing volume of microservices, doing a post mortem on an incident is often an exercise in navigating a twisted web of dependencies and trying to determine which service owner you need to work with.

Great observability gives engineers direct line of sight linking their metrics and alerts to the potential culprits. Additionally, it provides insights that can help fix underlying problems to prevent recurrence of incidents.

Tools and data:

- ✓ Traces
- ✓ Logs
- ✓ Metrics
- ✓ Dashboards



Keys to success:

Easy understanding of service dependencies: Identifying the direct upstream and downstream dependencies of the service experiencing the active issue.

Ability to jump between tools and data types: For complex issues, you need to repeatedly jump between details given by logs and traces to the trends and outliers given by metrics on dashboards and ideally in a single tool.

Time to root cause: Sometimes it's impossible to avoid having to perform root cause analysis during an incident and in those situations, having probable causes surface in alert notifications or during triage using dashboards reduces time to root cause.

Conclusion

Great observability can lead to competitive advantage, world-class customer experiences, faster innovation, and happier developers. But organizations can't achieve great observability by just focusing on the input and data (three pillars). By focusing on the three phases and the outcomes outlined here, teams can achieve the promise of great observability.

Ready to learn more? Book your demo today by visiting **chronosphere.io**.