StackState

Monitoring in a Cloud-Native Era

By Bernd Harzog, 2021

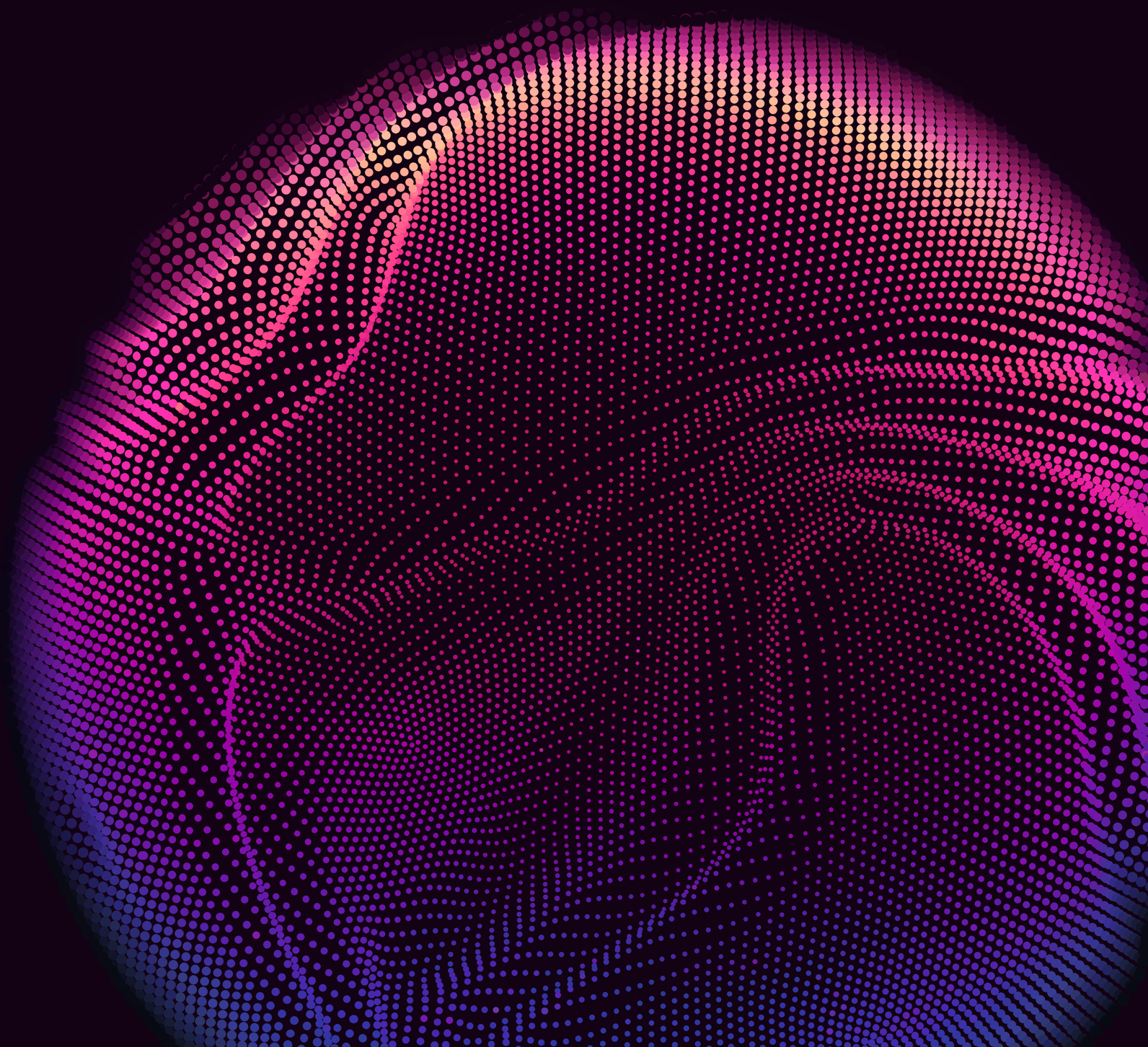
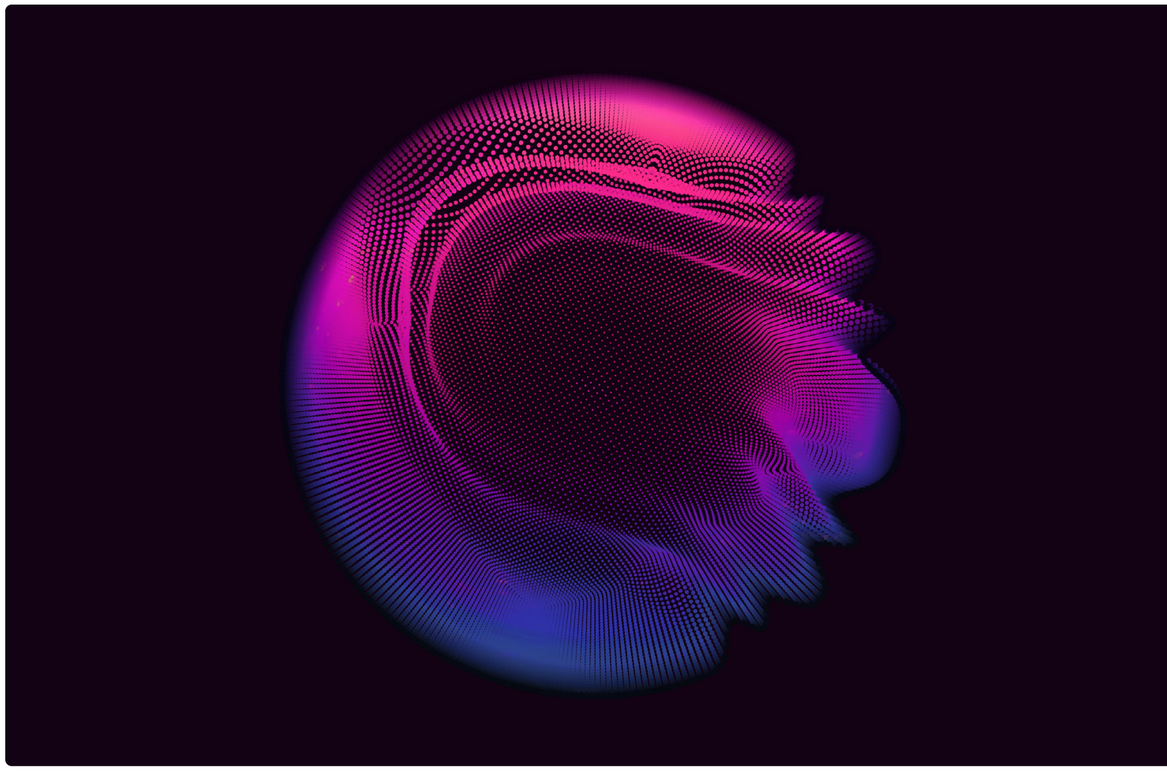


Table of contents

Monitoring in the Cloud-Native Era

The move to the cloud creates massive opportunities to deliver great applications and experiences to customers and employees, but it also comes with a new set of complexities. These new environments, powered by containers and microservices, among others, are dynamic and ever-changing.



The old ways of monitoring don't apply anymore—but the need to ensure the reliability and performance of your applications is more important than ever.

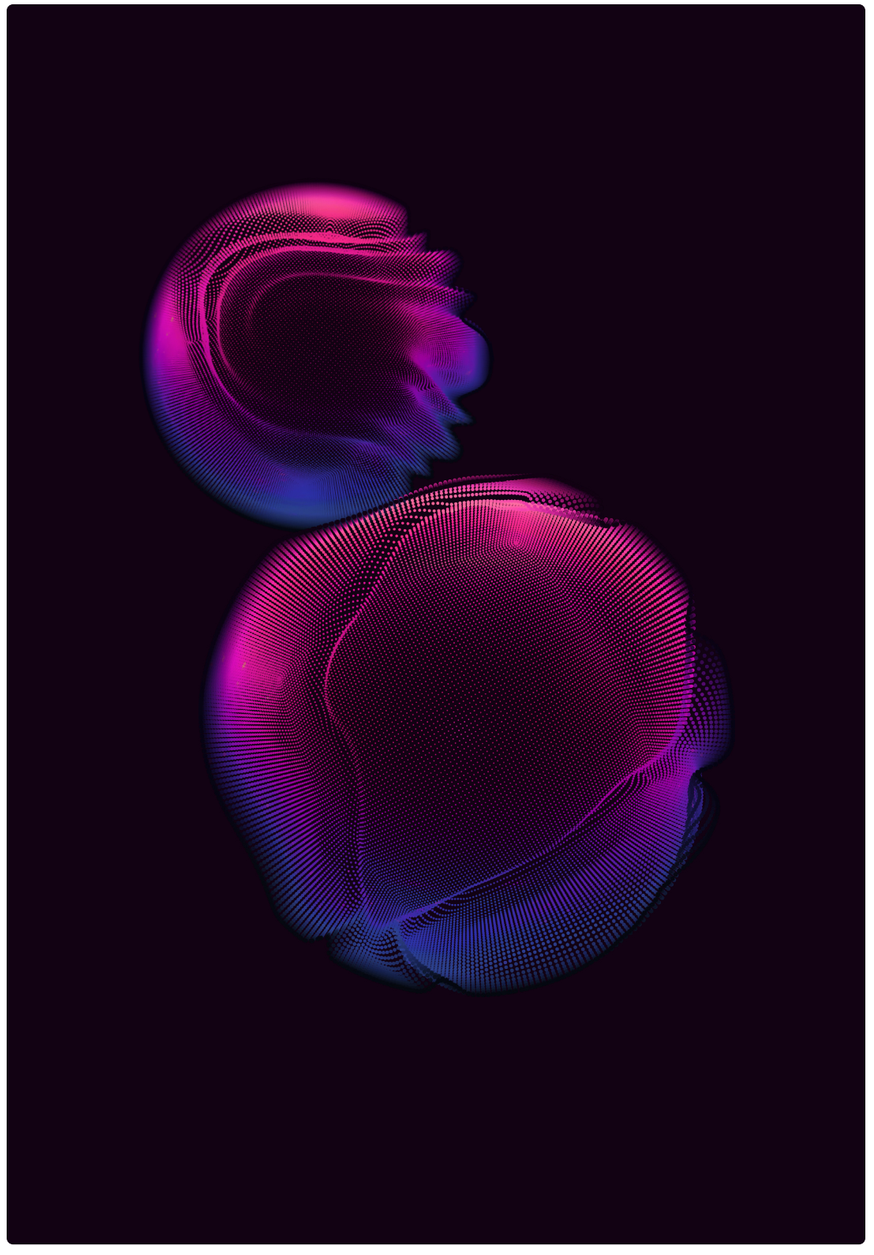
How do you rationalize this? With cloud **observability solutions.**

Monitoring in a Cloud-Native World

Cloud-native applications are built and run in a way that takes advantage of cloud computing to deliver services. This enables companies to bring ideas to their users and target markets faster. The cloud also empowers companies to respond to user needs faster than in a traditional development and management model.

Typically, cloud-native applications depend on a container-based infrastructure. With containers, the ways in which the applications interface with their dependencies can be tweaked and enhanced to create novel efficiencies, more options, and new solutions.

Cloud-native applications are also structured around microservices. With microservices, the elemental functions of an application are executed by individual services, which can be managed, repurposed, and automated within containers.



Advantages of Cloud-Native Applications

Cloud-native applications come with several advantages, and each benefits a different aspect of your organization and its systems, including the speed and ease with which development happens, as well as the culture supporting the process.

Speed

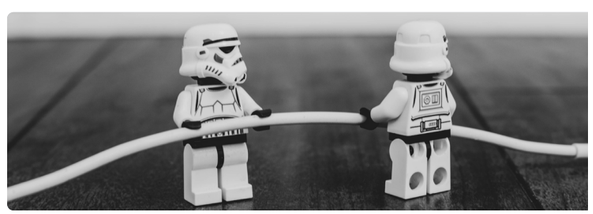
The speed at which a service is delivered used to be hindered by the hardware that delivers it. With a cloud-native approach, developers don't have to rely on outdated or weak hardware. You have access to the latest and greatest—and some of the fastest—components available.

The way speed benefits an application varies depending on the application, industry, and how it's being used. However, in the development phase, cloud-native application engineering has clear benefits, regardless of the business that's using it. For example, with a microservices-based architecture, you can strategically position your dependencies in a variety of ways without having to manually recode each one. In this way, you can simply tell the app which dependencies to use, as well as where in the process they should be utilized. If it doesn't work, or if performance doesn't meet your benchmarks, you can simply try a new configuration.

Back in the day, you had to make everything by stuffing pieces together, one by one.

When compared to the traditional development model, a container-based, microservices-dependent architecture can help development teams get an application just right faster and easier.

After an application has been built and deployed, the speed you get with a cloud-native ecosystem further benefits the end user and development teams. If, for whatever reason, the application isn't meeting users' needs, there's no need to recode everything from the ground up. In many cases, adjusting how dependencies are utilized can solve significant problems for the end user. This saves time when it comes to improving applications for future updates. It also makes it possible to deploy more often, enabling the organization to respond better to the needs of their end users.



Ready-to-Use Infrastructure

Remember the first time you saw a Lego set with those cool, premade pieces? Back in the day, you had to make everything by stuffing pieces together, one by one. Now, you have loading buckets for trucks, entire tractor trailers, boats—they even have an entire Millennium Falcon. While those of us who tirelessly pieced together bucket loaders and laser blasters using about 837 little bricks may not be happy, the premade stuff makes building, say, a scene in the Kessel run a whole lot faster.

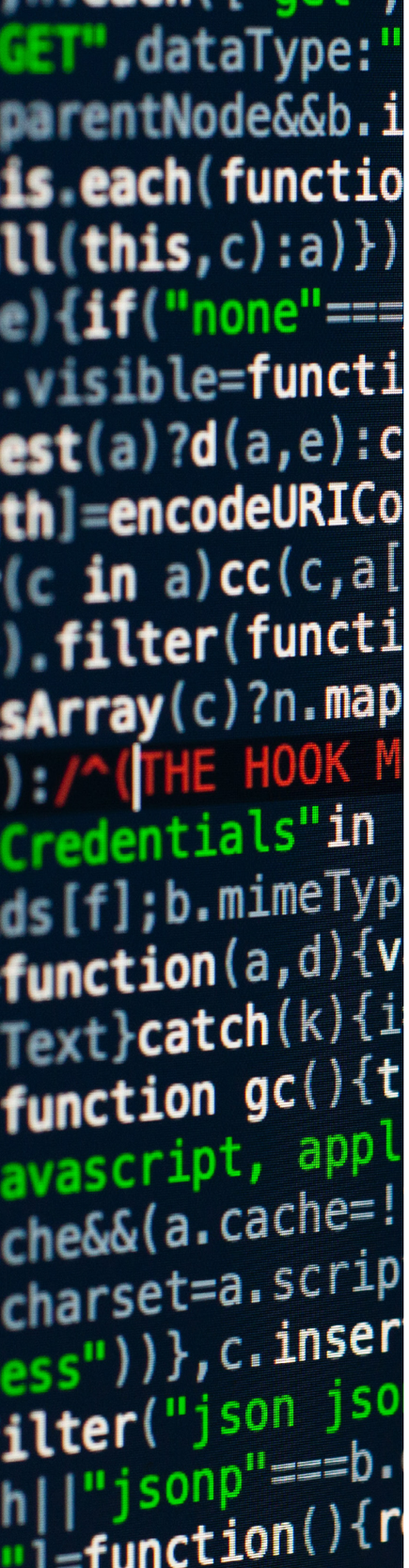
It's the same with cloud-native, ready-to-use tools. While hardcore development nerds may want to custom-code every aspect of their applications, the rest of the world—those with deadlines—would rather piece together the basic framework using prebuilt, proven components. These may include application programming interfaces (APIs), caching services, workflow engines, and operational rules. Because these core elements are preconstructed, you simply click them in place and then go about developing the things that make your application unique.

On-Demand Infrastructure

Instead of forcing your teams to build apps using limited resources that may take considerable time and investment to upgrade, with cloud-native architecture, you can get what you want when you want it. The cloud provider has all the tools and resources you need at the ready. Getting more processing power or storage, for example, is only a matter of requesting an upgrade. The resources you need can be ready to use in minutes.

In contrast, the traditional development model requires development teams to request just the right amount of resources prior to starting a project. If, partway through, they have an awesome idea that needs an additional resource, they may have to wait days or weeks for it to arrive. With a cloud-native architecture, it's like you're sitting in the middle of Home Depot with a gigantic shopping cart, ready to build a shed. Anything you need, you just grab it, and you're good to go. With the traditional model, you realize you don't have a heavy enough hammer, you have to order from the Sears catalog then wait for it to arrive.

With on-demand infrastructure, these kinds of unnecessary delays and expenditures can be avoided. Everything is at your fingertips.



Continuous Integration and Continuous Delivery (CI/CD)

In addition, cloud-native applications take advantage of continuous integration and continuous delivery (CI/CD) principles. This essentially means that DevOps teams can develop, integrate, and improve applications on a continuous basis through writing and deploying code reliably and frequently.

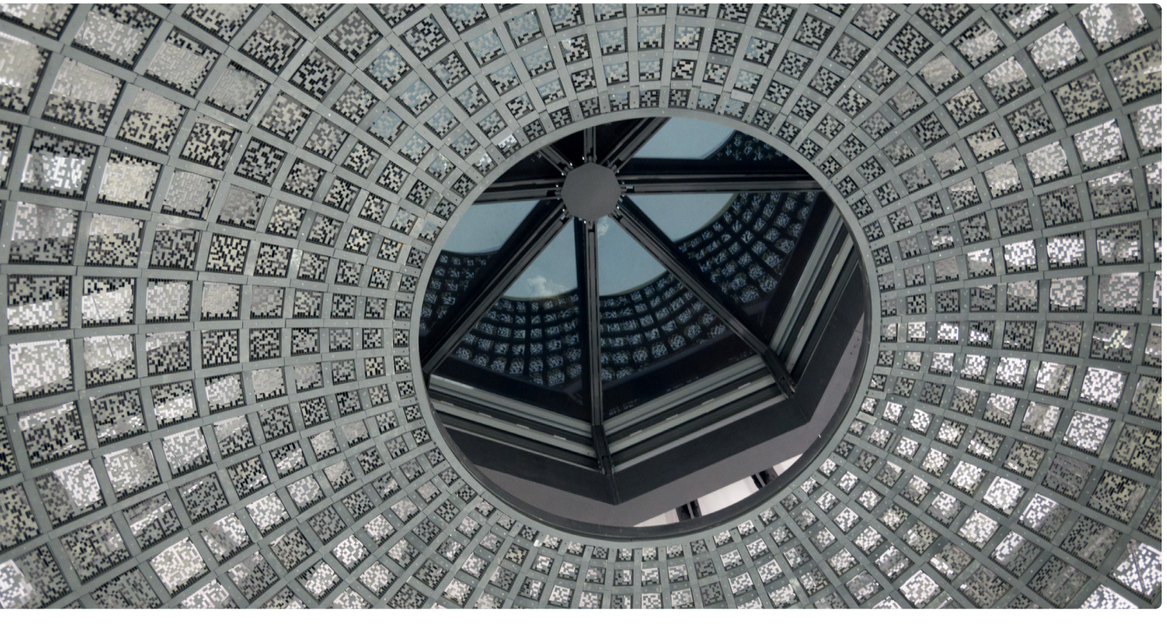
One of the earliest, and most common, types of cloud-native development is a perfect example of CI/CD: building a website on a site like GoDaddy. Before this was possible, you would have to write out your HTML painstakingly on your desktop, then upload it to the webserver and hope it looked how you want. If not, you had to dig back into your code file, find the problem, fix it, upload it again, make it live, and see how it looked. With GoDaddy, Wix, and others, you can make a little change and have it updated to your live site with a couple of clicks.

CI/CD in a cloud-native development platform enables this kind of agility but with complex applications. Instead of feeling like you're starting from scratch every time you have to make a change to the application, you can tweak it and redeploy in minutes. At the same time, you can obtain real-time feedback from the field, other developers, users, and other stakeholders. You can collect this input, apply another change, and see how they like it.

CI/CD also benefits the process of testing applications for vulnerabilities. If you have a white hat hacker on your team, for instance, you can deploy one version, see if the hacker can crack it, then try again—and again—until you have an airtight application.

DevOps Culture

DevOps culture is all about attaining agility through thoughtful teamwork across disciplines. With a cloud-native environment, it's easier for various team members to collaborate on projects regardless of where they are in the office—or the world. Everyone can hop on the platform and start collaborating. With a cloud-native infrastructure, you eliminate physical silos right off the bat, which paves the way for toppling other kinds of silos as well.



The Challenges of Monitoring a Cloud-Native Environment

These benefits are all well and good, but how do you monitor cloud-native apps? One of the strengths of cloud-native systems is the variety of different tools that you can combine to create exciting solutions. But how do you make sure these are all safe? Furthermore, if something goes wrong in the application or system you've built, how do you know from where the problem originated?

The first step in solving the monitoring puzzle involves asking three key questions:

1. Which technologies do the infrastructure and applications use?

The technology used will dictate how it needs to be monitored. Monitoring, after all, is only as good as the outputs you have to observe. Different technologies naturally necessitate different outputs. You wouldn't put a meat thermometer in a fried egg. (Well, you could, but…) So the first challenge in monitoring a cloud-native infrastructure is figuring out the tech you have and the possibilities for monitoring it.

2. How do you collect comprehensive data about the health and performance of each entity?

Every entity has pieces that make it work. In some cases, you need to monitor several of these in order to ensure it's working properly. In other situations, the data from one element may not be all that helpful.

Take a simple server, for example. You have a motherboard, a processor, memory, hard drives, a network connection, video cards, and a power supply. When monitoring this kind of server, one team may be focused on the speed at which it delivers services in comparison to its power consumption. Another team may be more interested in how much memory is used as it executes different processes. A totally different team—not even IT, necessarily—may be more concerned with how hot the motherboard gets during specific processes because this can impact cooling expenses for the data center that houses this server and similar units.

In an ideal world, you can present all the data you need on one screen, and different teams can choose what they want to observe. And with a server, this isn't too hard. However, with a complex IT environment, you may have to choose which facets of a component, network, application, or process are the most important to observe. This process, if not executed properly, can result in anything from time wasted tracking down where problems originated to undiscovered security vulnerabilities that expose your system to threats.

3. How do you compile and analyze data to provide actionable insights into performance?

Harvesting data is sometimes the easy part. Organizing it into something you can use to improve and change how applications function can be another thing altogether. This is particularly challenging in a cloud-native environment where the smorgasbord of resources and processes may have several interdependent parts—and that the environment at this moment can look very different than a day ago, an hour ago, or even a minute ago.

Returning to the server example, how does the clock speed of the processor impact memory allocation? Does it? When? Why? If there's a bottle-

neck, what's causing it? The hard drive? The network adapter card? Something else? The list of questions can be nearly endless.

When dealing with a cloud-native system you have to monitor, you face a similar challenge. You have to figure out which components and services need to be monitored and which facets of their operation pertain most directly to the macro service being performed. Then you have to ascertain how to glean, organize, and process this information.

Monitoring Challenges Unique to Microservices

In a way, microservices work together like the components of a server. They all work together to provide a macro service. In some architectures, they work in unison after a single input has been submitted. In others, they may work in sequence.

With a system governed by container orchestration tools, services can be automatically engaged based on the demands of the system or a specific process within it. It can get complicated really quickly. But focusing on some of the core challenges can help simplify the process.

Monitoring the Orchestration Process

When running microservices in a system that has orchestration in place using a tool like Kubernetes, for example, resources can be scheduled, containers can be deployed, the system can scale automatically according to need, and more. Once you have a system in place to execute this level of orchestration, you need to make sure it can be monitored. With so many moving parts and parts that have a brief lifetime, if something goes wrong, it can take an inordinate amount of time to figure out what went wrong. However, if you have a way to monitor the orchestration process, you can **back-trace to the initial incident** that spawned the trouble.

Monitoring the Communication Between Microservices

Microservices have to be able to locate each other within the network, as well as how to adjust processes as issues arise, ensure workloads are balanced, and so on. This can be accomplished in a few different ways, such as with a remote procedure call (RPC) framework, service meshes, or network proxies.

Monitoring how these communications take place may involve drawing data from the RPC framework or other facilitation tools. You also have to know which data you want to collect, basing your decisions on the macro or microservices being performed by the system.

Monitoring Individual Microservices to Enhance Reliability

Microservice architecture opens a lot of possibilities, but it can also open a Pandora's box of unpredictable reliability issues. A monolithic infrastructure may seem cumbersome and limiting from a development and deployment standpoint, but at least it's easy to figure out what needs to be restarted in the event of a failure: the central server. On the other hand, if you take all the services running in a monolith and designate them as individual microservices running on separate hosts in a network, you suddenly have many more points of failure.

With appropriate monitoring, you can combat the resulting dip in reliability by monitoring each microservice's performance. If a problem occurs, you can pinpoint the cause in moments. You can also use the insights you gather to troubleshoot microservices or the ways in which they interact.

Monitoring Latency

Each microservice comes with latency. Latency is unavoidable, like traffic when you're trying to get somewhere… or taxes… or drama with your in-laws. Latency is something you learn to live with—at least until it behaves in a way you didn't expect. With microservices, the latency of each one may be relatively predictable, but every now and then, there can be a latency spike. Multiply the chances of this happening within one microservice by your total number of microservices, and the likelihood of troublesome latency skyrockets.

While this may be hard to prevent, it can be monitored, troubleshot, and mitigated with adjustments. In this way, you can get a step ahead of latency problems and the chain reactions of issues that can result. You can also use a monitoring system to reapportion microservices or redesign the infrastructure that manages them to create a more efficient system.

Downsides of Multiple, Disparate Monitoring Systems

Particularly when you're monitoring in a cloud-native environment, which may be populated by many systems running simultaneously, using multiple monitoring tools has significant drawbacks. Cloud-native monitoring is therefore best served with a unified system.

Here are some reasons why.

Siloed Data

Even several well-conceived monitoring systems may result in excess work and wasted time if they are separate from each other. The data each system produces will only be truly actionable if it is correlated and compared with the outputs from the other systems, which results in extra steps. If your cloud-native monitoring tools are unified, then you have a single pane of glass that results in reduced work and enhanced productivity.

Too Many Alerts

With disparate monitoring systems, you have to struggle with a plethora of alerts. Often, the number of alerts can be reduced with an end-to-end observability solution, particularly if all alerts are correlated according to severity or how imminently dangerous the corresponding fault is.

Chaotic War Room Meetings for Issue Diagnosis

With several disconnected monitoring systems, chaos is bound to happen in the troubleshooting process.
Best-case scenario: There's a fracas of opinions being levied in a din of confusion. *Worst-case scenario:* Finger-pointing and arguments erupt over which system or application is to blame. But with a system that unifies all your disconnected systems, you can avoid the conflict and confusion.

Lack of Visibility in the Performance of Your IT Stack

An IT stack that's not truly observable may be the worst consequence of disparate monitoring systems. A system—if comprised of several standalone monitoring platforms—may not even answer some of your most important questions:

- *How are the different elements of the IT stack performing in relation to each other?*
- *At the moment of failure, did only one system generate a fault—or was it multiple?*
- *Is there a chain reaction happening where one system impacts another? If so, how long does it take for the issue to spread?*

Root-Cause Analysis Takes Too Much Time

While multiple monitoring systems can detect anomalies, each one adds an extra step in the process of root-cause analysis. With an observability platform that consolidates insights from different sources, you can ascertain the root cause faster because you have data from all systems in a single pane of glass.

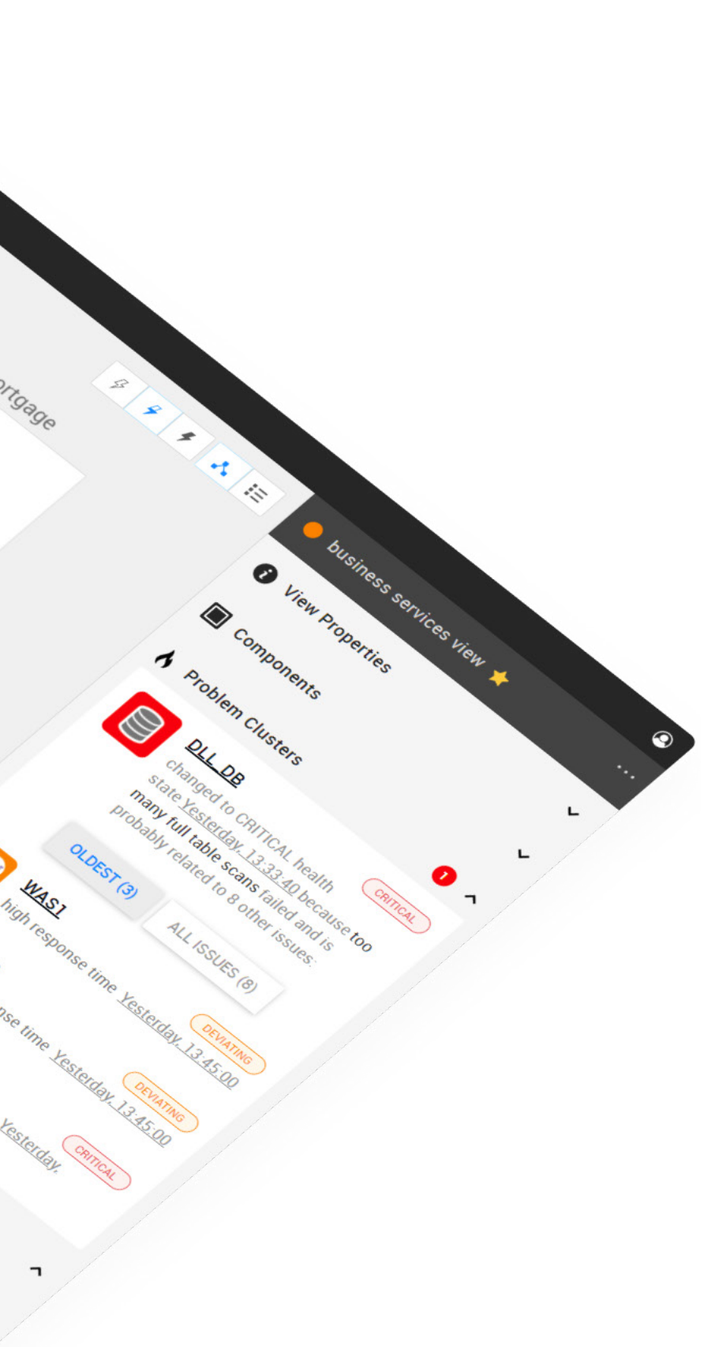
Taking Cloud-Native Monitoring to the Next Level with Relationship-Based Observability

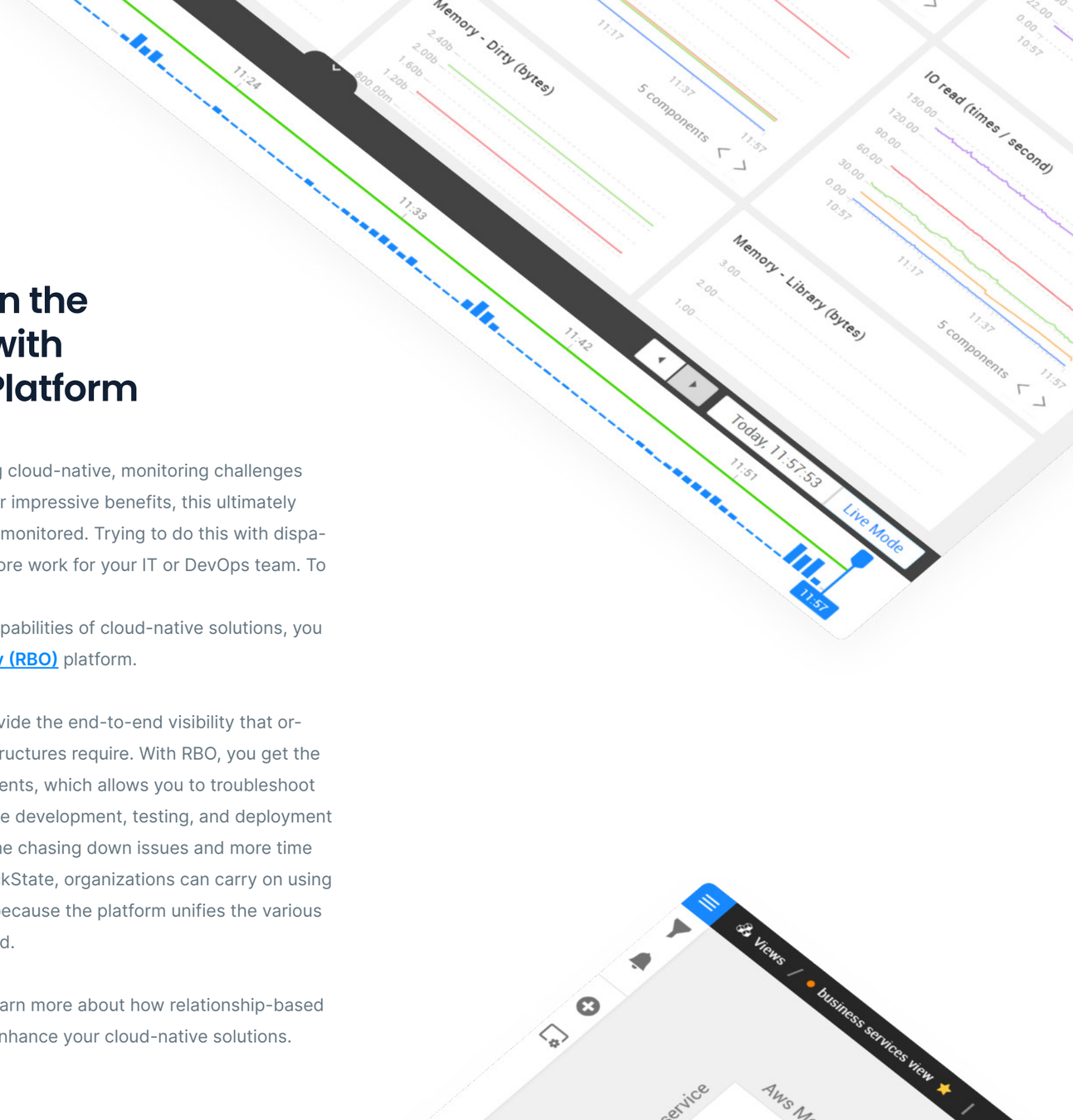
There are many downsides to using multiple monitoring tools, as discussed in the previous section, but there's a solution that can make them all work together—a relationship-based observability (RBO) platform. With it, you can:

- Leverage information from your existing monitoring tools
- Break down the silos between your various tools
- Create a unified view of the end-to-end IT environment

The results:

- Faster root cause analysis
- Unified insights
- Noise reduction
- Business impact analysis
- IT process automation





Better Monitoring in the Cloud-Native Era with StackState's RBO Platform

With more and more organizations going cloud-native, monitoring challenges abound. While cloud infrastructures offer impressive benefits, this ultimately results in more systems that need to be monitored. Trying to do this with disparate monitoring tools may only create more work for your IT or DevOps team. To benefit from the speed, ready-to-use,
on-demand infrastructure, and CI/CD capabilities of cloud-native solutions, you need a [relationship-based observability (RBO)] platform.

StackState is uniquely positioned to provide the end-to-end visibility that organizations with complex, dynamic IT structures require. With RBO, you get the ability to relate changes directly to incidents, which allows you to troubleshoot issues quickly. You can also speed up the development, testing, and deployment process because you can spend less time chasing down issues and more time testing out your solution. Also, with StackState, organizations can carry on using the monitoring tools they already have because the platform unifies the various data from those tools into one dashboard.

[Get in touch] with StackState today to learn more about how relationship-based observability can help you control and enhance your cloud-native solutions.