INSTANA
an IBM Company

Observability Needs for Application Modernization

Table of Contents

INSTANA
an IBM Company

Executive Summary

Perform a Google search for “microservices,” and the results are endless. Clearly, everyone is using microservices in their environments. Except maybe they’re not.

A [2020 O’Reilly survey]() of more than 1,500 technology workers found that 61% of organizations have been using microservices for a year or more. However, 23% do not use microservices at all. Of those that do, 29% are migrating or implementing a majority of their systems using microservices.

In other words, application modernization paints a more complicated picture than the developer advocates on social media would have you believe. Many container-based microservice applications were built from legacy environments, and there’s a lot of migration to come over the next few years. On-premise bare-metal servers and monolithic applications are the starting point, and Cloud-Native Applications are the endpoint, but each organization plans its own route based on where it is now, its needs, and what is achievable.

Elements of Application Modernization

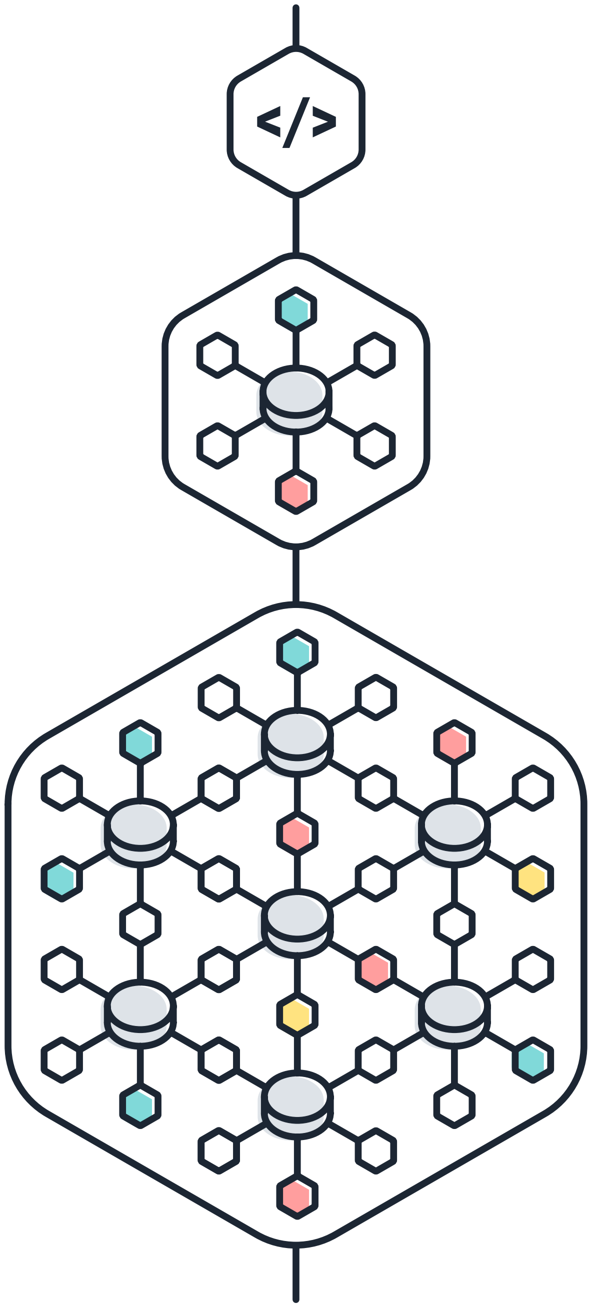
Application modernization has many layers, which organizations can address in parallel or individually. The order is subject to change because some elements are dependent on others. They create a cross-section of code, application components, and infrastructure.

Code

Writing and updating code for an application is an arduous task. Just getting it out the door is challenging. Pride of ownership causes many developers to wrap their arms around it and consider it “theirs.” This sense of ownership is fantastic, but there is often a dark side.

Murphy’s Laws tells us that eventually, a legacy application will have to be updated while the code owner is on vacation or otherwise unavailable. This leads to a classic Theory of Constraints scenario, where the team will have to choose between waiting for the owner or having someone else work on it. The time will also come when the owner of the application leaves the company and is no longer available to even answer questions.

The new person who gets into the application could be in for a rude surprise. Custom code, personal comments, odd ordering, no comments, not using accepted syntax, and poor formatting can all turn simple tasks into detective work and make introducing automation almost impossible. Outdated Java could disable critical integrations on your website. If suddenly your website can’t provide services or collect payments, you’re damaging the business.



INSTANA
an IBM Company

Application components

A monolithic application on bare-metal servers can no more move to Cloud-Native with a few tweaks than a car can fly by bolting on wings. Break a monolithic application into microservices. Place the components into containers. Orchestrate those containers with Kubernetes or another tool.

Just that is a lot of work, and that's not even counting the needs of shifting the infrastructure to cloud-based workloads, but it provides a start for application components. Of course, the more complex the components, the more time is required of developers and engineers, which often creates more urgency to move to a cloud hosting service: let them manage the infrastructure so your team can manage the applications, especially if you plan to re-platform or refactor an application (more on that in a bit).

"Application modernization paints a more complicated picture than the developer advocates on social media would have you believe."

Infrastructure

Remember when we said that code, component, and infrastructure changes can happen in parallel? If you're migrating an application to a cloud environment, you have several options.

- Re-host – lift and shift the application and re-host it in the cloud
- Re-platform – host the application in the cloud and make minor infrastructure changes
- Repurchase – purchase SaaS versions of your applications
- Refactor – rewrite some or all of an application and possibly deploy new application architecture such as converting a SOA application to containers

Beginning the Modernization Journey

Where do you start? It depends on where you are. Generally, you want to move to the next step on the sliding scale toward Cloud-Native, but each step has its challenges.

You Are Here	Your Next Step
Bare metal, on-premise	Virtualized, on-premise
Virtualized, on-premise	Data centers
Data centers	Private cloud
Private cloud	Public cloud
Public cloud	Hybrid cloud
Hybrid cloud	Cloud-Native

Determining the success of each step in the process requires both a benchmark measurement of application performance before and a new measurement for comparison after. Maintaining visibility to collect the same set of metrics at each stage can be challenging for legacy APM tools.

Defining *Legacy* for THIS eBook

Depending on your organization, and your specific title, the phrase legacy application could have one of a half-dozen different definitions. To make our discussion and analysis easier, we're going to set a definition of Legacy Applications here. This doesn't mean that other definitions are wrong, or not even the most important definition for their particular situation. It just makes it easier for us to write "legacy" and "legacy application" than the longer fully qualified name.

So…. for the purposes of this paper, legacy application shall come to mean some form of a Java EE or .NET application, either deployed as a monolith or in a Services-Oriented Architecture environment. For that purpose, this paper might also refer to generation 1 and 2 APM tools as legacy monitoring or legacy APM. There are other tools that might be considered "legacy," but any discussions of those tools will be called out more specifically.

Benefits of Modernizing Applications

Migrating code, application components, and infrastructure makes for a complicated set of decisions that requires a great deal of planning. Considering how difficult this task is, are you sure this is something you want to do?

Modernizing applications is a key part of the digital transformation journey, which helps to automate previously manual processes. Automation has a great impact on both service performance and on human resources. Processes that require frequent manual intervention can be slower, subject to more errors, and more expensive to operate.
With that in mind, here are some reasons for modernizing applications:

- Optimize developer time
- Accelerate innovation and the CI/CD pipeline
- Reduce costs and redundancy
- Become more responsive to end user needs
- Build more uniformity into
- organization and processes
- Deprecate older environments
- Issue resolution

Optimize developer time

Many legacy applications are based in Java or .Net, but some lack the sophistication of newer applications. They can make updates and even routine maintenance difficult for development teams. As noted above, the lack of tribal knowledge can make one missing developer a bottleneck for all work on an application. It's classic Theory of Constraints.

As one developer takes over for another, they can Frankenstein together an application with fixes and patches that each makes with his own style. The new person comes in, and the inconsistencies are overwhelming: links to JavaScript function files in some places, and JavaScript directly in the code in others. Simple fixes turn into hours poring over code to find the needle in the haystack. These scenarios lead to more mistakes and developer burnout and turnover.

When developers apply rigor and apply UX principles to their code, they make it easy for anyone to come in and work with the code quickly or -- better yet -- automate away routine work. Comments tell the next person what each section pertains to in the application. References to scripts and stylesheets indicate what those elements do. And whichever way the originating developer decides -- tabs versus spaces -- is consistent and meaningful throughout.

The result is less time figuring out code and more time doing work. Less stress and more accomplishment. And better team performance.

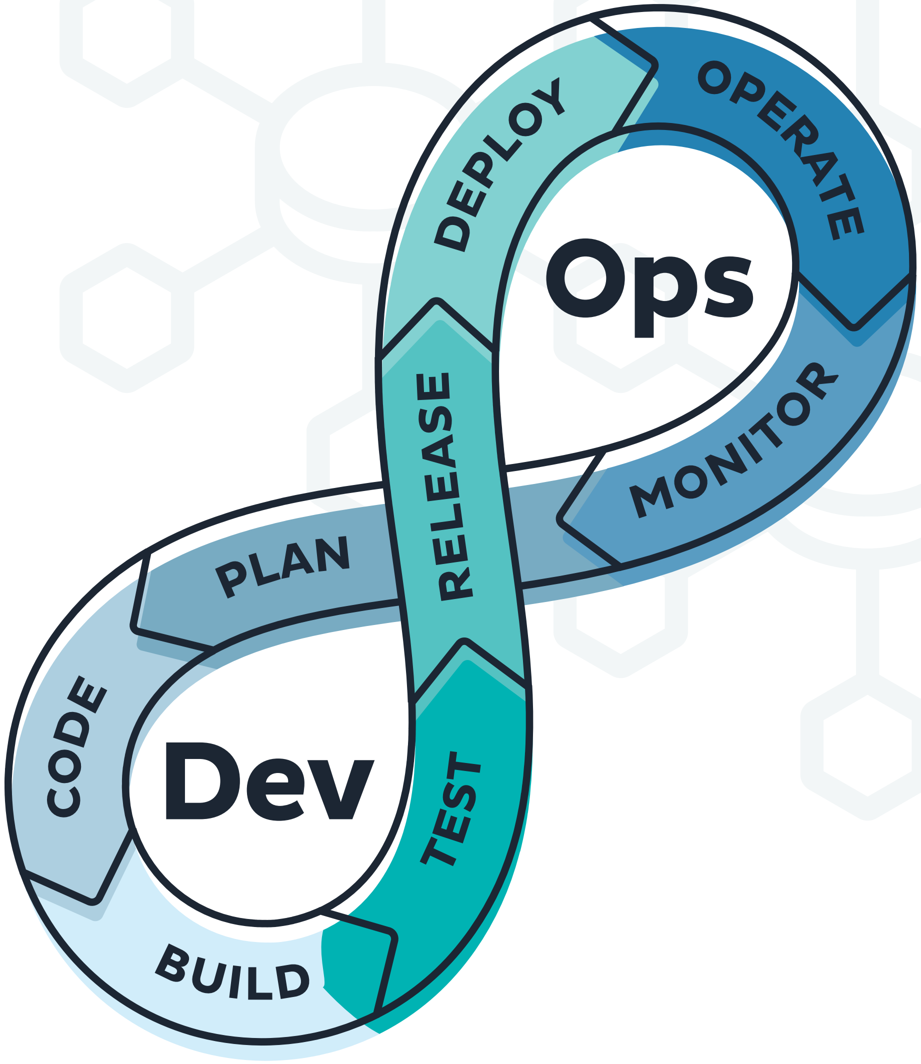
Accelerate innovation and the CI/CD pipeline

More commonly, older applications are less resilient. As a result, developers spend too much time firefighting issues to keep the lights on and not enough time innovating and building new features for customers. A 2019 study by Tidelift and The New Stack found that developers spend less than a third of their time actually writing or improving code.

When developers complete more work early in a development project, they start a virtuous cycle of resource allocation that keeps projects on time and reduces errors. By contrast, when projects slip, they start a vicious cycle of firefighting.

The development process can be challenging for organizations that run it manually. They run tests at each stage manually instead of running continuous automated tests. They burn hours finding results of tests performed by other team members.

They manually instrument code instead of automating it. They painstakingly document their infrastructure instead of using a tool that can automatically discover it, potentially causing non-functional tests (performance tests, load tests, and stress tests) to give misleading results, which can cause code to hit production before it's ready. They run application components without isolating them from potential risk factors, causing the results to be inconclusive. Fortunately, automation is helping.



Organizations are applying architectures for running applications -- such as containers, microservices, serverless computing, and Kubernetes orchestration -- to application development and delivery. With more workloads moving to dynamic technologies, there are new realities for the pace and scale of change within application environments.

Automation and orchestration tools like Jenkins, Ansible, Chef, and Puppet can help keep teams on the same page as they navigate applications through the pipeline. Each of the major public cloud platforms (AWS, Azure and Google Cloud) offers Kubernetes managed container processing as a service which is easily integrated into CI/CD delivery pipelines.

Your customers depend on you to release products that work, and that requires your ability to monitor applications through the development lifecycle.

Reduce costs and redundancy

One of the advanced steps organizations take before moving to cloud hosting is to run data centers. Having global data centers is great for load balancing and redundancy, but it is expensive. Steam Data Centers estimates the annual cost of running a single data center at $10-25 million US.

Data centers sometimes come with long-term contracts. Migrating to the cloud is a time-consuming process, so leave yourself at least a year before your contract expires to start the move.

And if you thought data centers offered redundancy, try dozens of containers with microservices in the cloud. That's redundancy. Some large corporations still use on-premise architectures, and they make them work, but the pattern toward cloud adoption is unlikely to reverse.

Become more responsive to end user needs

The requirements and wish lists for customers and employees should be your guiding light. Customers have little patience anymore for delays or interruptions online or in apps:

- 47% of users expect a maximum of 2 seconds loading time for an average website
- 40% of consumers will wait less than 3 seconds for a web page to load before abandoning
- 88% of online consumers are less likely to return to a site after a bad experience
- Slow-loading websites cost retailers $2.6 billion in lost sales each year

“88% of online consumers are less likely to return to a site after a bad experience.”

Build more uniformity into organization and processes

Remember how developers create their own fixes and patches to cobble together Frankenstein code? There are real-world consequences.

Different applications are coded differently, making holding them to minimum standards difficult. How can you apply a standard to an application that doesn’t use any of the components you’re trying to standardize? Organizations modernize applications to bring them into alignment, so they can be brought into compliance.

Deprecate older environments

An application is built on Java 11. The current version is Java 16. As long as the team is going to upgrade the version and improve code accordingly, why not take this opportunity to modernize the application in other ways? Break it into microservices. Move it to the cloud. Make it a Cloud-Native Application.

Issue resolution

Distributed components enable incident responders to disable some containers or other pieces without turning off all of them. In other words, distributed, cloud-based microservice application components remove the single point of failure that plagues monolithic applications. Containers also enable developers to isolate and test pieces of an application without affecting the whole thing.

Orchestrators like Kubernetes also make it easier to apply fixes to applications without customer downtime, which is a huge win for provider and user alike.

Why Companies Make the Mistake of Not Modernizing

There are generally two reasons companies put off application modernization: short-term pain and inability to properly measure progress. Pitting short-term pain against long-term benefits is always a calculation organizations have to make, especially since short-term pain is more certain and long-term benefits are more of a gamble. The other major reason is the inability to measure progress along the modernization continuum.

Short-term pain

Here's a quick list of reasons organizations delay or cancel application modernization projects:

- Customer hesitation: The modernization process usually requires some disruption; and for industries like financial services and healthcare, privacy concerns and any disruption can be cause for alarm. Some larger customers are still hesitant to put applications and data in the cloud as well.
- Time: Remember our chart of next steps in application migration? Each of those steps is likely to take up to a year. Just breaking a monolith into microservices requires understanding which capabilities you want to deploy separately and breaking the process into steps so you can measure progress. It's not an exact science. As Martin Fowler says, "Microservices is a label and not the description."
- Money: In the short term, both the code upgrades and the architectural changes can be expensive in terms of both costs and resource use. Add up the cost of developer time spent on firefighting, adding new hires, turnover, training, and planned code freezes, and the hidden costs can really add up.

For companies looking to stop spending millions on data center operations, they often have to take up new costs of the hosting service.

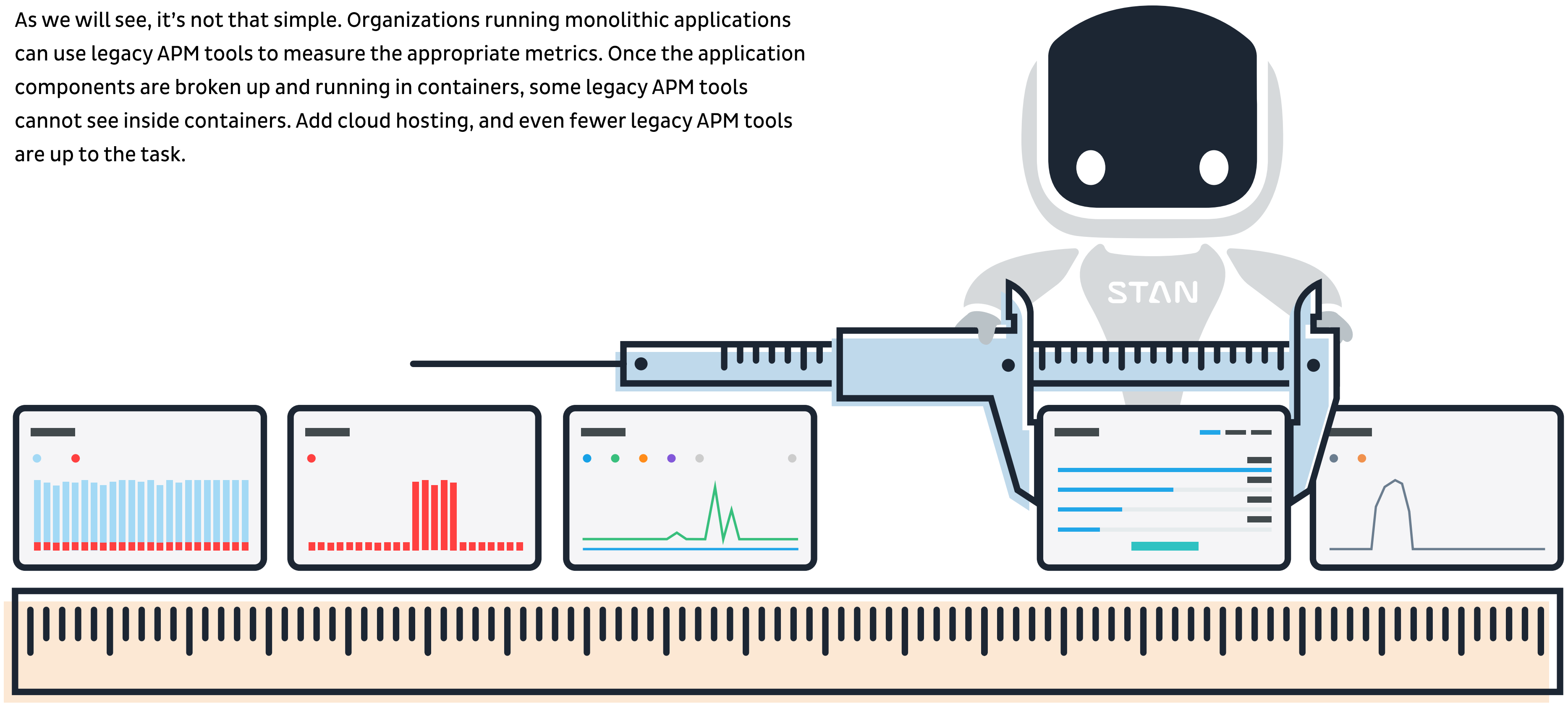
> "The annual cost of running a single data center is $10-25 million US."
>
> ***Steam Data Centers***

Inability to benchmark and compare performance

Peter Drucker famously wrote, “If you can’t measure it, you can’t improve it.”

For something as complicated and variable as application modernization, measuring is difficult. Each level, from bare-metal monolith to cloud hosted to Cloud Native, is unique to each individual company. After every change, it’s crucial to measure progress to determine whether it’s safe to roll out changes to production and start preparing for the next step.

As we will see, it’s not that simple. Organizations running monolithic applications can use legacy APM tools to measure the appropriate metrics. Once the application components are broken up and running in containers, some legacy APM tools cannot see inside containers. Add cloud hosting, and even fewer legacy APM tools are up to the task.



The Role of Observability in Application Modernization

It’s critical to add observability into your portfolio as you plan and execute application modernization and digital transformation. Application quality control is a requirement to do modernization.

- Benchmark and compare with the same measurements
- Easier for a containerized, Docker solution to measure on-premise
- A legacy APM tool cannot measure Cloud-Native

Benchmarking before you plan to modernize an application is critical so you can determine where your application performance is weak, where it’s strong, where it’s on track, and set new targets. There are some basic best practices to keep in mind.

Benchmarking for application quality control

When you benchmark and compare performance at each stage, it’s critical to record the same measurements at each stage. Otherwise you’re not measuring apples to apples.

Benchmarking is a critical feature that allows you to track application quality control. There are few dependencies and few obstacles.

As a first step, you decouple capabilities within the monolith. As the monolith breaks into components and then into microservices, operating microservices effectively becomes critical -- and a risk factor -- so decoupling simple edge services can be a good place to start. It’s a small step, but already there are more complex metrics and request traces in play.

As a next step, microservices are placed in containers in a virtualized environment. Then the components are moved to data centers. Then to the cloud. You need new authentication services -- and new sensors for cloud architectures, integrations and APIs. Can your old APM solution do that?

Observability: More than just metrics, traces, and logs

While the de facto definition of Observability tools is Metrics, Traces and Logs, legacy applications won't provide much details within those three elements. That's one reason to look at an Observability Platform that can handle the unique needs of enterprises, especially those running legacy apps across legacy / hybrid environments.

Legacy APM tools can monitor monolithic applications, but they struggle with microservices, containers, and cloud architectures. When they move to containerized application infrastructure, they are running black-box monitoring, unable to see inside the application. Move to cloud-based or Cloud Native, and it's worse. So they cannot report on the same metrics for accurate comparisons.

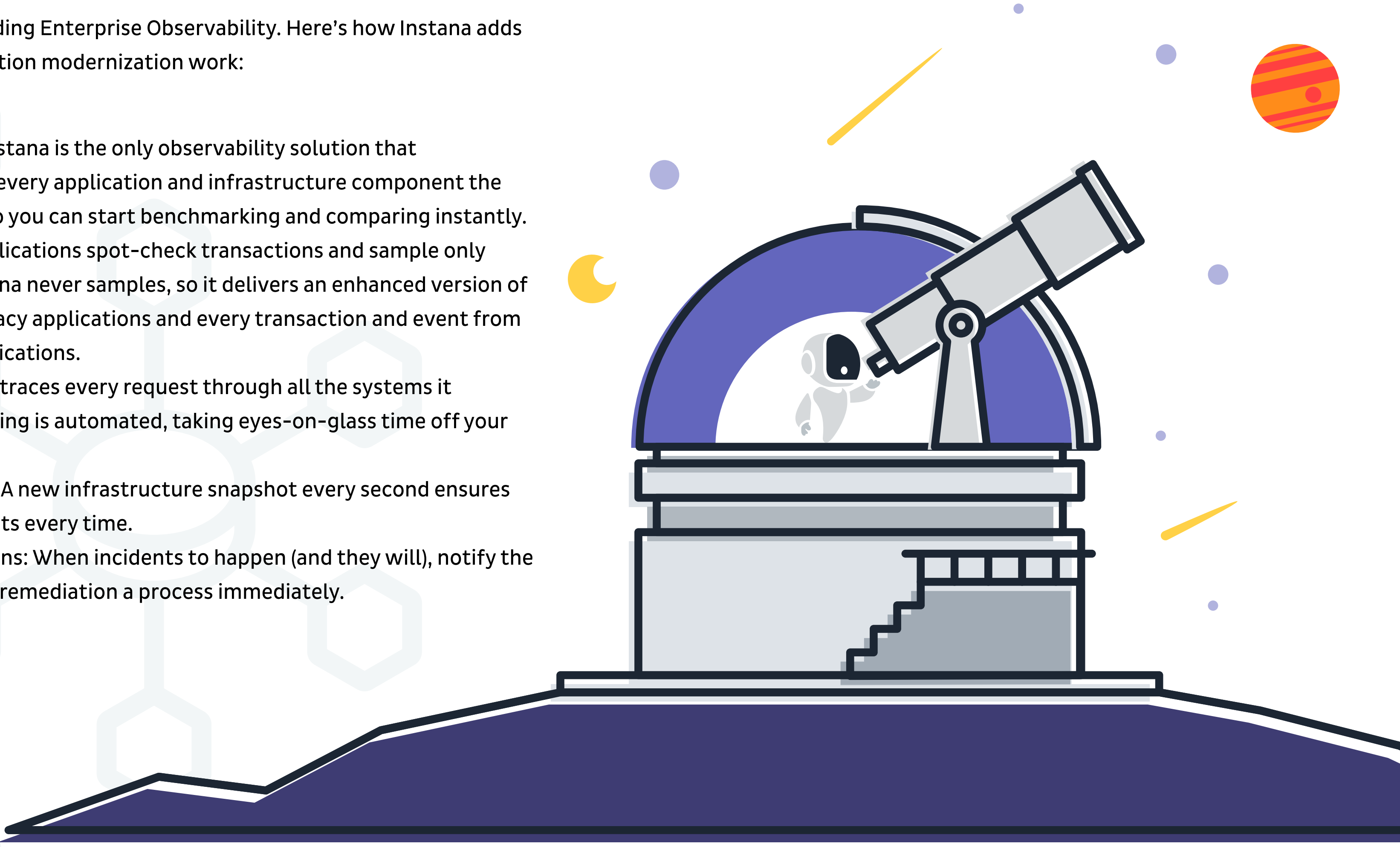
The solution is rather obvious: start with an Enterprise Observability tool. Here are some advantages of starting with an Enterprise Observability tool:

- Discovery: See every application and infrastructure component to miss nothing and bring context to everything
- Trace requests: Trace requests in legacy applications, and trace every request regardless of environment in even Cloud-Native Applications
- High granularity: Leave no more than a second between monitoring beats
- Instant notifications: Turn information into intelligent action

Instana Enterprise Observability

Instana delivers industry-leading Enterprise Observability. Here's how Instana adds functionality to make application modernization work:

- Automated discovery: Instana is the only observability solution that automatically discovers every application and infrastructure component the moment it is installed. So you can start benchmarking and comparing instantly.
- No sampling: Legacy applications spot-check transactions and sample only elements of traces. Instana never samples, so it delivers an enhanced version of the same metrics for legacy applications and every transaction and event from microservice-based applications.
- Request tracing: Instana traces every request through all the systems it moves through. This tracing is automated, taking eyes-on-glass time off your developers' plate.
- One-second granularity: A new infrastructure snapshot every second ensures up-to-date measurements every time.
- Three-second notifications: When incidents to happen (and they will), notify the right people and kick off remediation a process immediately.



About Instana

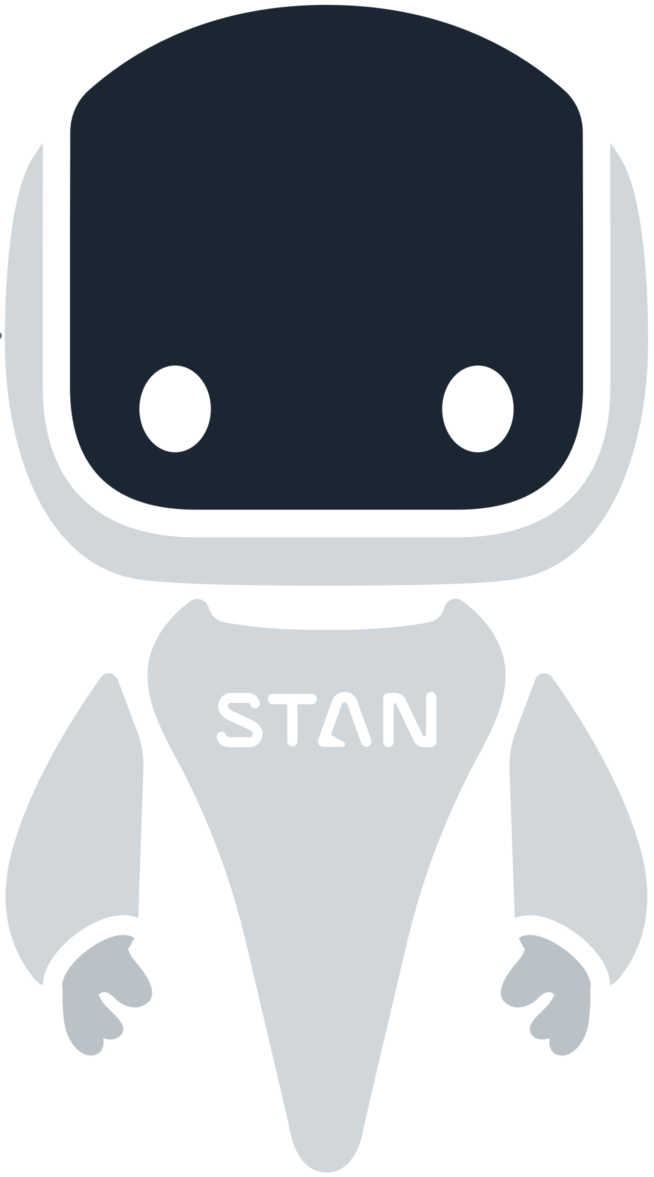
Instana, an IBM Company, provides an Enterprise Observability Platform with automated application performance monitoring capabilities to businesses operating complex, modern, cloud-native applications no matter where they reside—on premises or in public and private clouds, including mobile devices or IBM Z® mainframe computers.

Control modern hybrid applications with Instana's AI-powered discovery of deep contextual dependencies inside hybrid applications. Instana also provides visibility into development pipelines to help enable closed-loop DevOps automation.

These capabilities provide actionable feedback needed for clients as they optimize application performance, enable innovation and mitigate risk, helping DevOps increase efficiency and add value to software delivery pipelines while meeting their service and business level objectives.

For more information, visit https://instana.com.

Start Your Trial Today



Stan
Your Intelligent
DevOps Assistant

INSTANA
an IBM Company

INSTANA
an IBM Company