



# Observability and Containers

Monitoring Best Practices for  
Containerized Applications

# Table of Contents

---

03	Introduction to Containers and Observability	15	Antipatterns: How Not to Solve the Container Monitoring Challenge	26	Conclusion
07	The Visibility Challenge in Containerized Environments	18	Capabilities Required For Effectively Monitoring Containerized Microservice Applications	27	About Instana
12	Why Container Observability Matters	21	Enterprise Observability – the Best Solution for Monitoring Containers and Containerized Applications		

# Introduction to Containers and Observability



## Achieving Observability for Containerized and Orchestrated Environments

- Explaining Container Adoption
- Challenges of Container Observability

To say that containers have revolutionized software delivery and deployment is an understatement. In a few short years, container platforms like Docker have evolved from a novel, experimental technology to a core part of the infrastructure of organizations large and small, across a wide range of industries.

Before Docker's release in 2013, few software engineers or systems administrators used containers in a serious way, apart from small numbers of followers of platforms like OpenVZ and LXC. That quickly changed; by early 2016, studies of container adoption concluded that "Docker spreads like wildfire."

Containers are now an essential component of technology stacks at companies large and small. Perhaps no previous technology watershed — not the advent of virtual machines, not the open source revolution, not even the rise of Java — has matched the pace and passion with which containers have revamped the way organizations produce and deploy applications.

## Explaining Container Adoption

Ultimately, agility and speed are the two primary reasons that container adoption has been so meteoric. Containers enable organizations to achieve a level of agility that was simply not possible using traditional virtualization and software deployment technology. This agility leads to increased development speed, giving organizations the ability to rapidly build and deploy new business services.

There are several benefits for development and operations teams:

- IT Ops / DevOps and scale quicker and easier with containers
- Software delivery teams can switch seamlessly between different development frameworks
- Containers provide ideal building blocks for constructing continuous delivery pipelines
- Containers are perfect for developing and deploying modular, flexible microservice applications
- QA teams can more easily duplicate production environments for more salient testing

Ultimately, containers make all application stakeholders lives better, from architects and developers to QA, DevOps and SREs. They compartmentalize at multiple layers which makes the entire application delivery process more flexible and agile.

## The Container Observability Challenge

But with the revolutionary advantages of containers comes a new type of challenge – performance management. Application Observability, Monitoring, Visibility, Tracing, Profiling and performance tuning are much more challenging within containerized environments. There are several key reasons for this.

Containers are more than just glorified virtual machines. They can contain any number of platforms and technologies within a single stack. Even if individual containers are simple to use, the overall containerized environment can get complicated in a hurry, with registries, variable run-time configuration, loosely coupled connections and the ability to spin up and down in milliseconds – and that's before you get into orchestration.

The differences between containerized environment and traditional (including virtual) infrastructure are the primary reasons that organizations risk shooting themselves in the foot if they don't adopt new monitoring technologies to go along with their new architectures.

If you attempt to monitor your containerized applications and environments in the same way that you monitor traditional applications, you ultimately undercut the core advantages of adopting containers in the first place. There are several reasons traditional monitoring tools struggle with observability in container-based applications and infrastructure.

- Monitoring an application component inside a container involves observability of the container, the operating system, the middleware and the application code, all correlated as a single unit (and that's BEFORE orchestration is considered)
- Keeping up with the provisioning and destruction of container instances also requires more than simple observability at the code or infrastructure level)
- The rate of change of platforms, code and services (up to hundreds or thousands of changes per week), make manual monitoring configuration impossible to maintain

Fortunately, delivering effective observability and monitoring in containerized applications can be solved, but meeting the challenge requires a fundamentally new approach to observability and monitoring. Let's examine why container observability is difficult, the typical mistakes organizations make when trying to address this challenge, and how to develop an effective observability approach to maintain complete visibility and agility after making the transition to containers.



Visualizing how highly distributed applications are structured and operating is challenging.  
[uber.com/blog](https://uber.com/blog)

# The Visibility Challenge in Containerized Environments



— .

To understand how to develop an effective approach to container monitoring, you must first understand why maintaining visibility and optimizing application performance within containerized environments are uniquely challenging. Let's look at the primary reasons container observability and monitoring are so difficult that pre-container monitoring tools struggle with basic monitoring:

- Lack of built-in observability or monitoring
- Container orchestrators are not performance monitoring tools
- Traditional monitoring tools struggle with distributed microservice environments
- The wide variety of workloads that can be in a container
- Containers are dynamic and unpredictable
- Polyglot – the diversity of technology platforms and languages in containerized applications

## Lack of built-in observability and monitoring

Container platforms include only basic monitoring functionality. For example, the Docker stats command provides a limited amount of performance information about running containers, but on its own, the stats tool is insufficient for monitoring large-scale production environments. The lack of built-in monitoring beyond basic data sets containers apart from other application infrastructure technologies. Ironically, this observability and monitoring challenge occurs in an environment that has elevated monitoring needs, due to the speed and frequency of application updates. Similarly, operating systems provide significant data about system performance, but they don't include application-level data. Even advanced virtual machine platforms like VMware, which include relatively sophisticated monitoring tools, fail to provide application layer data or distributed tracing. The nature of containers make it much more difficult to collect sufficient observability, monitoring and performance information from the container, itself.

## Container orchestrators are not performance monitoring tools

Container orchestrators like Kubernetes (and all the derivative K8S distributions) are great for organizing, provisioning and managing the deployment of their production container environments and applications. There's hardly a containerized application left that isn't orchestrated, whether by Kubernetes, D2IQ or some other orchestration engine.

But Orchestrators are not meant nor designed for sophisticated monitoring, and while they have tremendous focus on container and host resources, there is no aspect of user experience or application performance included in orchestration management.

Furthermore, If either the applications or the infrastructure is setup in a hybrid environment, orchestrators can actually get fooled into thinking everything is okay when, in fact, the overall system is crashing or hung. This is because container orchestrators can only manage things that they are aware of – which means the infrastructure and services running in containers. In a hybrid environment, container orchestrators cannot help to monitor resource usage of any other resources.

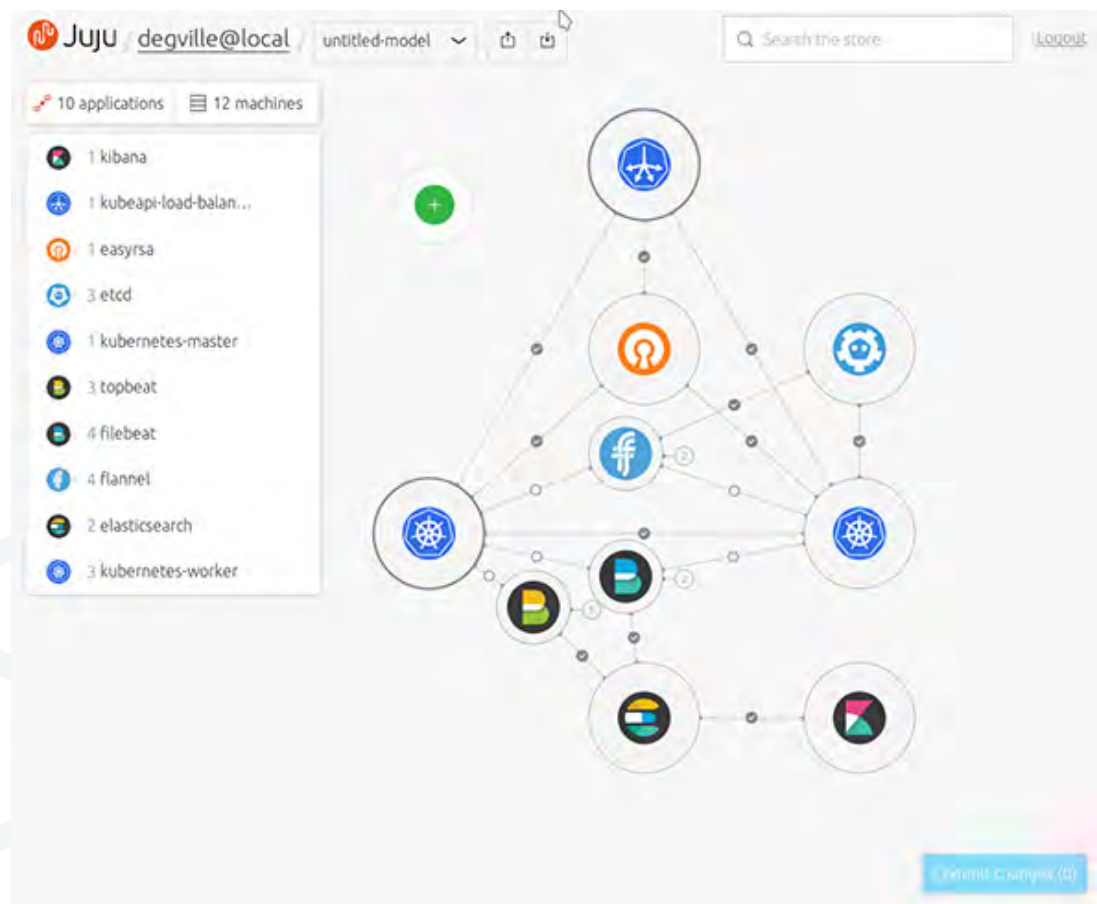
This creates a second major observability & monitoring gap for distributed applications. And that leaves a large part of your environment prone to performance problems. So remember, while orchestrators are excellent provisioning tools capable of finding and restarting failed containers, don't confuse these management tools with performance monitoring or observability.



## Traditional monitoring tools struggle with distributed microservices environments

Environments for container-based applications tend to be dynamic, distributed microservices, deploying those services across clusters of servers that are dynamic, themselves. A complete application could cross different kinds of infrastructure, platforms, languages and technologies to create just a single application. Requests between those services accomplish the business processing.

Traditional infrastructure and application monitoring tools struggle to visualize and understand distributed microservice environments. Their base technology was designed to monitor monolithic applications that are mapped to static individual servers, and they focus on providing visibility at the language level. These traditional monitoring approaches break down when the need arises to support microservices distributed across a large cluster of servers (composed of multiple languages, many middleware components, and multiple database systems).



Orchestrated applications tend to be highly distributed and “dis” integrated. This ‘Juju Charms’ illustration shows flow patterns of service requests between microservices deployed via Kubernetes.

## Containers host a wide variety of workloads

There is no single type of workload associated with containers. In some cases, they are used to deploy monolithic applications, like an Apache HTTP server. Some containers host Java virtual machines. Others host databases, which makes monitoring even more complicated. Why? Persistent data storage is typically outsourced on permanent servers, not hosted inside containers, creating a hybrid environment (remember the “Hybrid” problems?).

From a monitoring perspective, the multiplicity of different workloads that might run inside a containerized environment is challenging because your monitoring tools must support an unpredictable set of technologies, architectures and configurations.

This workload variability makes it difficult to configure traditional monitoring tools to handle containers because every container is different, and it is hard to determine which thresholds to set in advance for each one. In addition, monitoring tools need to be automated because configuring and enforcing monitoring policies manually for unpredictable types of workloads is not feasible.

Every container is different, making it difficult (if not impossible) to determine which thresholds to set in advance for each technology instance in each container. The variable workloads also drive a need for observability and monitoring automation. Configuring and manually enforcing monitoring policies for dynamic unpredictable workloads just isn't feasible.

## Containers are dynamic and unpredictable

By design, containerized environments are always in flux. Individual containers spin up and down rapidly. A container sometimes lives only a few seconds before it is shut down. The ability of containers to start and stop quickly is key to achieving the agility that a container-based environment offers.

Neither traditional monitoring tools nor manual observability solutions can provide the proper infrastructure and application visibility while supporting the rapid pace of change associated with containerized microservice environments.

Let's say, for example, that a container spins up an NGINX web server instance that handles 150 requests during a period of five minutes, but the container then shuts down due to a performance problem. Traditional monitoring tools won't be able to include the server in its list of running systems, while traditional APM tools won't be able to trace the problem because the container no longer exists. A modern APM solution or Observability platform, however, has the ability to look back in time, including traces through containers even after they cease to exist.



Typical spread of technologies deployed via container orchestration into a single server with varied workloads and priorities. Here we can see an Apache server, a JVM, an NGINX proxy, a Postgres SQL database and a random process each running as a container within a host.

## Containerized environments can be built using diverse technologies and languages

There is no single way to construct a containerized software environment. Organizations can choose from a range of different orchestrators (like Kubernetes, Swarm and Marathon), registries (like Docker Hub, Quay and VMware Harbor), and even runtimes (such as runC and Rkt).

It's also common in a containerized environment to run multiple microservices, each written using a different type of programming language. Traditional monitoring tools that support only specific languages will not work well in such an environment.

The large diversity of technology also creates expertise issues in Ops and DevOps teams. Rather than a single middleware system running Java, Ruby or .NET, containerized applications can run several different languages, multiple database servers, a combination of messaging and a host of other technology components such as security, storage or search. The idea of a technology-specific subject expert drilling down into a technology-specific monitoring tool is gone.

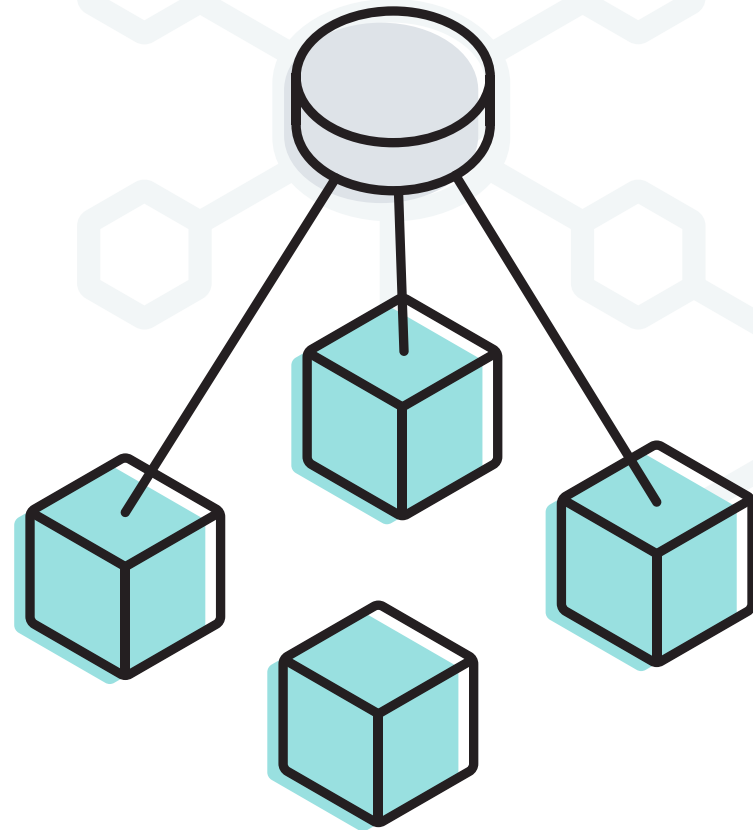
# Why Container Observability Matters



If there's anything IT has learned over the years of platform updates, it's that maintaining visibility is essential in any type of environment. In fact, it's the only way to keep applications running smoothly and to optimize resource usage and cost. In a containerized environment, visibility and monitoring are especially important. They are crucial not only for maintaining application health, but also for maximizing agility and efficiency – and thereby obtaining full return on the investment you make in containerized infrastructure. A lack of visibility in a containerized environment leaves you blind in ways that do not occur within traditional environments. This blindness creates real technical and business challenges, some of which are outlined below.

## The Visibility Challenge in Containerized Environments

- Inability to pinpoint the root of a problem
- Poor performance metrics
- Hindered Scalability





Lack of visibility inside containers hinders troubleshooting in Ops.

## Inability to pinpoint the root of the problem

Without proper observability and relevant visibility into containers, architecture and applications, you don't know whether the root cause of a problem is in the host server, the container runtime, the containerized middleware, the code running inside a container, or somewhere else.

Traditional monitoring tools that are unable to understand the context of a containerized environment will not help you solve this problem. Nor can you depend on your team to be able to troubleshoot issues in a containerized environment manually because production container environments are too complex and large to manage without automation. Even a small container-based application can end up with hundreds of containerized software parts.

## Poor performance metrics

Without a good observability platform, you'll struggle to get good data from inside containers; and that means you can't effectively determine whether or not your containerized apps are meeting your service quality goals. Since a major reason for adopting containers is to make application deployments more efficient and agile, operating without proper visibility is extremely risky. You will be blind to efficiency and agility, with no way of either capturing or maximizing those critical measurements.

For example, you won't be able to determine whether a new microservice or application deployment is meeting latency goals after an update. Traditional monitoring tools cannot track latency at the data granularity required to measure the performance of a containerized application effectively. Without this information, you have no way of knowing whether the time and money invested in a new deployment is being paid back to your team and business.



## Hindered Scalability

Flexible scalability - the ability to increase or decrease the size or number of instances of an application or microservice in response to fluctuations in load ON DEMAND - is a key benefit of containers. Containers enable easy scalability because they can be started, stopped, or migrated in just seconds, compared to minutes for virtual machines or days for bare-metal servers.

But while containers make the question of HOW to scale easy, intelligent analysis of accurate monitoring data is the only way to know WHEN to scale. You need to know when an increase in demand for your application merits the scaling up of your deployments (meaning rapidly allocating new containers and hosts). Just as important is determining when demand has decreased, allowing you to scale down to stop paying for compute resources that are under-utilized.

As infrastructure has virtualized, scalability considerations have become more recognized as critical to operational success. Containers are the latest iteration of virtualization, with misconfigured container environments resulting in the same problems as virtual server misconfiguration. Insufficient allocation of resources to containers will result in poor application performance and scalability issues, while excessive allocation wastes money and resources. Without precise real-time visibility into the details of the containers, overcoming these challenges is not possible.

# Antipatterns: How Not to Solve the Container Monitoring Challenge



— .

Faced with the observability, visibility and monitoring challenges described above, organizations that have migrated to containers are sometimes inclined toward solutions that appear to resolve the problem, but actually constrain their agility and ultimately negate the advantages of using containers. Following is a collection of common examples that don't solve the problems - they mask them.

Antipatterns: How Not to Solve the Container Monitoring Challenge:

- Trusting orchestrators to handle service quality
- Limiting technology choices
- Avoiding monitoring agents
- Using traditional infrastructure and application monitoring tools
- Keeping applications monolithic

## Trusting orchestrators to handle service quality

As discussed previously, orchestrators are not observability or monitoring solutions. You can and should use an orchestrator to help provision your containerized environment, but you need a separate monitoring solution because an orchestrator cannot, and is not intended to, provide the deep visibility required to guarantee service quality and application performance.

## Limiting Technology Choices

In some cases, modifying the applications that run inside containers can make observability and monitoring easier. For instance, you might decide to limit your developers to only use a single language that your legacy APM tool supports. While strategies like this can simplify monitoring, they defeat the purpose of container adoption because they constrain your agility. Limiting architectural, development or operational options because of your monitoring tools is the ultimate tail wagging the dog scenario.

## Avoiding monitoring agents

Avoiding monitoring tools that require agents to run inside containers can also help to make monitoring easier, because these agents are often difficult to deploy inside a containerized application. Programmatic data sources that generate monitoring information from within an application are easier to deploy. Here again, however, avoiding a certain type of monitoring technology can negate many of the benefits that containers offer. You should enjoy the flexibility to use either agent-based monitoring or programmatic data sources depending on your needs or preference.

Similarly, eschewing other types of desirable monitoring, such as opensource protocols like OpenTracing, Prometheus or Jaeger, can lead to a gap in your observability, visibility and monitoring. The most important aspect of your observability strategy is that you shouldn't close off any single method, allowing yourself to leverage the best of both worlds.



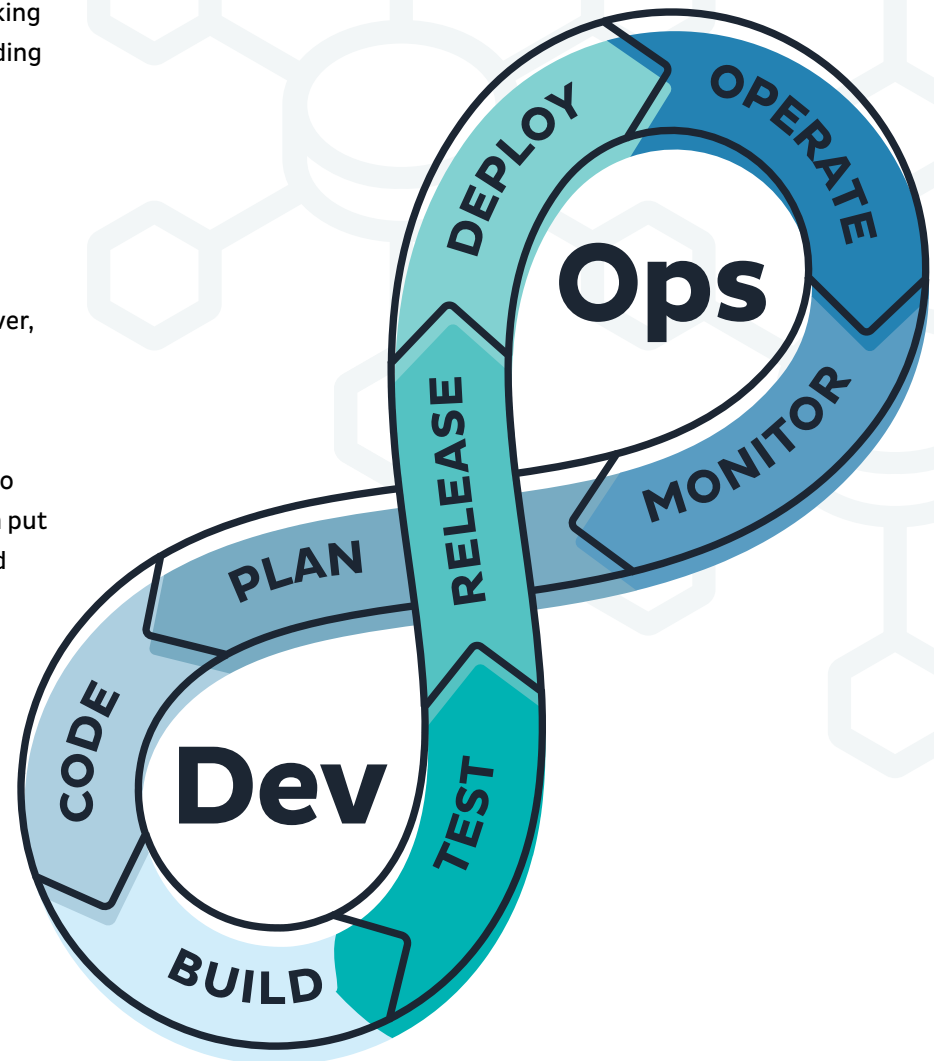
## Using traditional infrastructure and application monitoring tools

Container-based applications simply have too many differences from traditional infrastructure and monolithic applications; in the way they are architected, built, deployed, and executed. Traditional monitoring tools were designed before the high-speed agile development methodology and containers even existed. Thus, they were not designed to be able to keep up with the speed of containerized operations, taking too much continuous manual effort and additional expertise to derive understanding from the tools.

## Keeping applications monolithic

Maintaining a monolithic architecture instead of migrating or refactoring your applications to run as microservices is another way to simplify monitoring. However, this strategy also undercuts one of the main benefits of containers, which is their ability to support high speed agile development processes.

The common thread among all these antipatterns that represent the wrong way to think about how to solve the container visibility challenge – is because all of them put unnecessary limits that can negate the best value of converting to a containerized microservice environment.



# Capabilities Required For Effectively Monitoring Containerized Microservice Applications



As detailed above, the world of applications running in containerized environments is quite dynamic. DevOps needs precise realtime visibility, situational awareness and understanding into how applications are performing aligned with suggestions and advice on how to optimize the complex technical structures found in today's systems.

With containers (and the architectures they enable), there's simply too many differences and roadblocks to use conventional monitoring tools. To ensure proper service levels are continuously met, monitoring requires a new approach, built from the ground up to deal with the unique characteristics of these dynamic environments.

These are the key capabilities needed to effectively deal with the issues we've analyzed:

- Handle the Dynamism by Automating Everything
- Be Container Aware
- Automatically Discover, Map and Visualize the Full App Tech Stack
- Capture High Data Fidelity with Real-time Analysis
- Trace Every Request
- Assist with Root Cause Understanding of Microservice Issues
- Stop Relying on Humans to Configure Health Rules

# Handle the Dynamism By Automating Everything

When so much of your technical stack is changing so often, your team does not have the time or knowledge needed to continuously configure a monitoring tool manually. A modern observability solution must fundamentally be automatic. With zero configuration (intervention by a human), the platform must be able to visualize the situation, monitor application performance and trace requests.

With CI/CD (Continuous Integration and Delivery) processes streamlining and automating the development and deployment cycles for new services, you can't settle for a monitoring / observability tool that isn't also automatic!

## Be Container Aware

As mentioned before, every conventional APM solution was designed and built years before container infrastructure was even an idea, let alone dominating your environment. Over time, of course, these legacy tools were able to come up with workarounds to get their instrumentation into container-deployed code. That's NOT the same thing as understanding how containers are operating.

Your observability platform must be able to understand the full containerized situation – what is the container, is it orchestrated, what infrastructure is running inside it, instrument any code, and tie all those things together – along with all the information about every other container and tech stack.

The platform must also be able to handle the ephemeral nature of containers and the dynamic aspect of microservices by being able to automatically update all that information and monitoring in real time.

## Automatically Discover, Map and Visualize the Full App Tech Stack

In dynamic environments, understanding the structure and dependencies all your technical components over time is necessary for efficient analysis and performance monitoring. A modern observability platform must discover every application component and map the interactions & dependencies between them. The map visualization must automatically and continuously update on its own (complete with dependencies) and include real-time service and application performance changes as application updates / changes occur.

## High Data Fidelity, Real-time Analysis

Microservices and containers are short-lived and dynamic. A modern solution must be able to capture and analyze short spikes, which allows Dev+Ops to immediately understand whether a code rollout caused an issue. More than just real-time metrics, performance analysis and problems must be detected and identified in seconds. What used to pass for high granularity metrics gathering (15 seconds, 1 minute) now seems like an eternity. One-second granularity is the new requirement.

## Capture a Distributed Trace with Every Request

A core element of any Observability platform is the capture of distributed traces. But there are many ways distributed traces can be captured and/or presented to Dev+Ops users.

The only way to have complete analysis and understand exactly how, when and where problems exist is to capture a distributed trace of every request – no gaps, no sampling and no proxies. That means that tracing has to be supported on every piece of infrastructure, every platform and every line of code deployed into the containerized environment.

## Assist With Root Cause Understanding of Microservice Issues

Don't lose sight of the fact that containerized environments are delivering applications which must perform to the expectations of the business. No matter how complex your environment, the Devops team must identify and resolve performance issues as fast as possible. Look for a solution that can automatically point you to where and why distributed applications break down.


## Stop Relying on Humans to Configure Health Rules

Across application development and monitoring lifecycles, the more that humans are required to do (i.e., install, configure, customize, reverse engineer, etc.), the less complete the monitoring coverage will be, the weaker responses to changes will be, and the more likely that a problematic update will cause an application outage.

As element counts grow into the hundreds or thousands, humans cannot practically manage application performance on their own. Even adding in Machine Learning is not enough to eliminate the human drag. Tools for modern application environments must include a more complete level of AI-assisted functionality, including machine learning, advanced statistical analysis and curated knowledge-bases – all of which continue to learn from real-world experiences to automate even more of the solution.

Going beyond monitoring and root case analysis, modern monitoring solutions should include automated identification of Key Performance Indicators (KPIs), usually in the form of SLO/SLI management – all without human interaction.

# Enterprise Observability – the Best Solution for Monitoring Containers and Containerized Applications

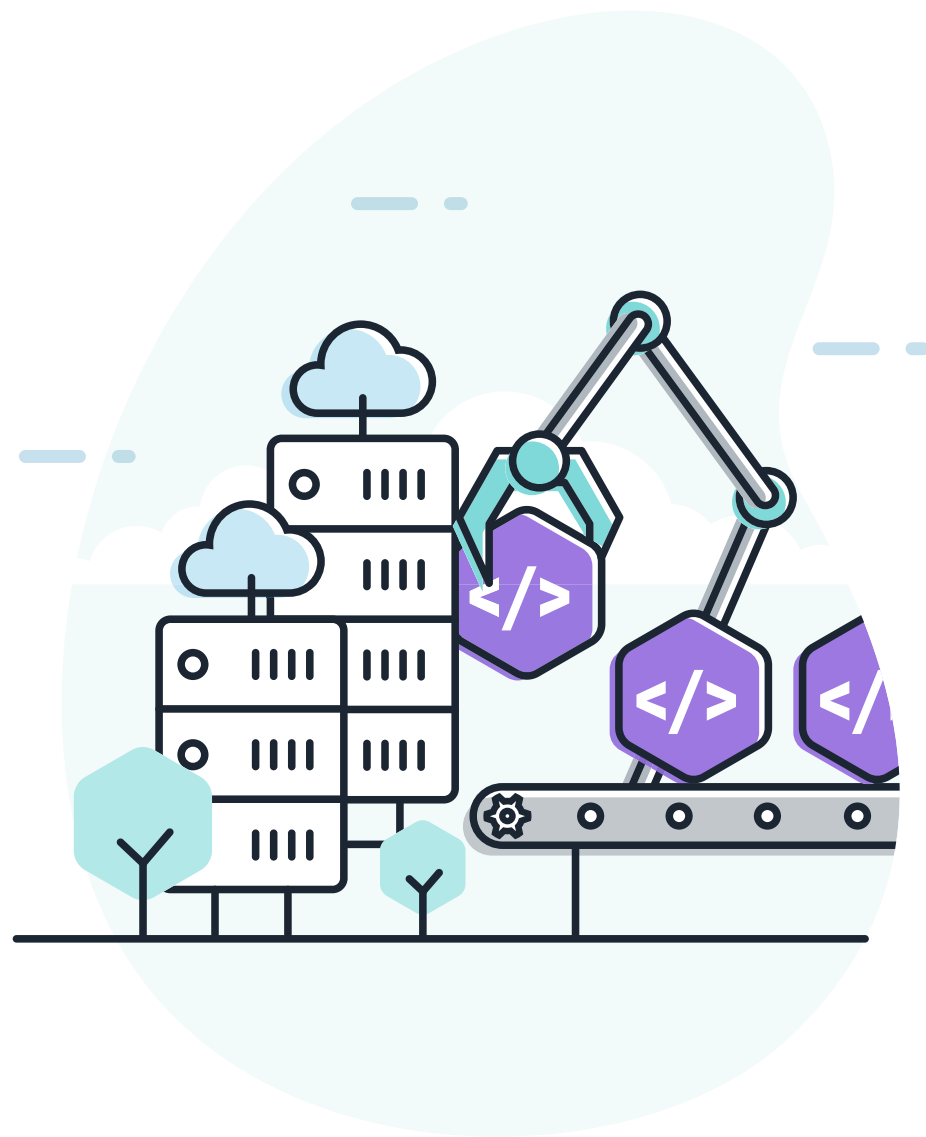


As detailed above, applications running in containerized environments are dynamic in multiple dimensions. To effectively monitor these environments – infrastructure, services and applications – DevOps needs a platform that can handle the unique challenges of containerized microservice application environments.

Traditional monitoring can't handle the ephemeral nature of containers, nor the complexity of microservices. Meanwhile, simple Observability tooling is can help individual stakeholders, but not the broad set of Dev, Ops and IT Decision Makers involved in modern application operations. That's where Enterprise Observability is required.

## Automation

Automation is a natural desire in any monitoring solution – but for modern application teams using agile development methodology and running an always-full CI/CD pipeline, automation is their life. But if Observability and Monitoring are manual, the entire process can get stalled or derailed at the monitoring step. Enterprise Observability automates the entire monitoring lifecycle – discovery, mapping, monitoring configuration, alerting, even root cause analysis.



## Context

In monolithic applications, each transaction runs through the same stack and code base as every other transaction, allowing certain functional shortcuts like sampling or reverse engineering when deploying monitoring and tracing; but in dynamic microservice applications, the only constant is change.

Understanding how every component (or entity) interacts with others is critical to understanding where to start solving problems. Further, sampling of application response metrics, infrastructure configuration and request tracing is no longer an effective proxy for reality.

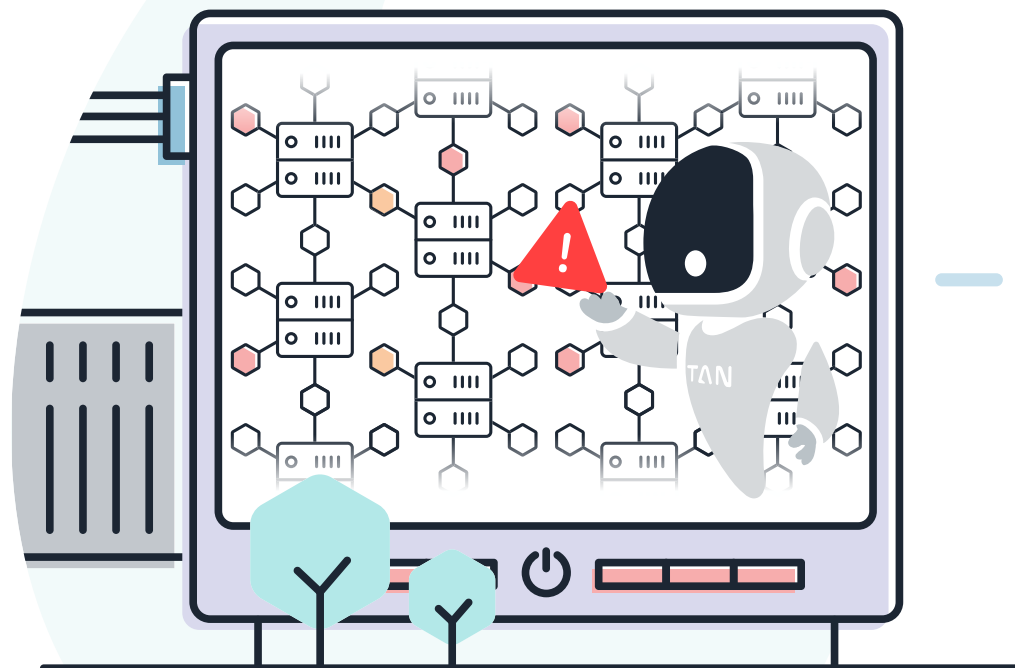
Enterprise Observability ties everything together – service maps, infrastructure configuration, request traces, open source protocols, even profiles – delivering understanding, both programmatically and to end users (Dev+Ops and decision makers).



## Intelligent Actions

Collecting data without the desire or ability to do anything with it is simply a waste of time and money. The purpose of Observability in container-based applications remains the same as application monitoring or APM solutions – to validate application performance; and to find and fix problems when they occur.

Since container-based applications are more complex, finding the root cause of problems requires assistance. Enterprise Observability uses its knowledge of application and infrastructure architecture, its immense collection of events and understanding of requests (traces) to identify triggering events and inter-component relationships to highlight where attention is required by DevOps to fix problems and optimize application performance.





## Ease of Use

As organizations roll out agile development processes, increase the frequency of their application updates and fill their CI/CD pipelines, more stakeholders require the actionable information we've been discussing. That's why ease of use, which was never a stalwart of APM solutions, is critical to Enterprise Observability. The more people that can leverage an Enterprise Observability platform, the better overall application performance will be, and the more efficient the entire DevOps organization will be.



# Conclusion

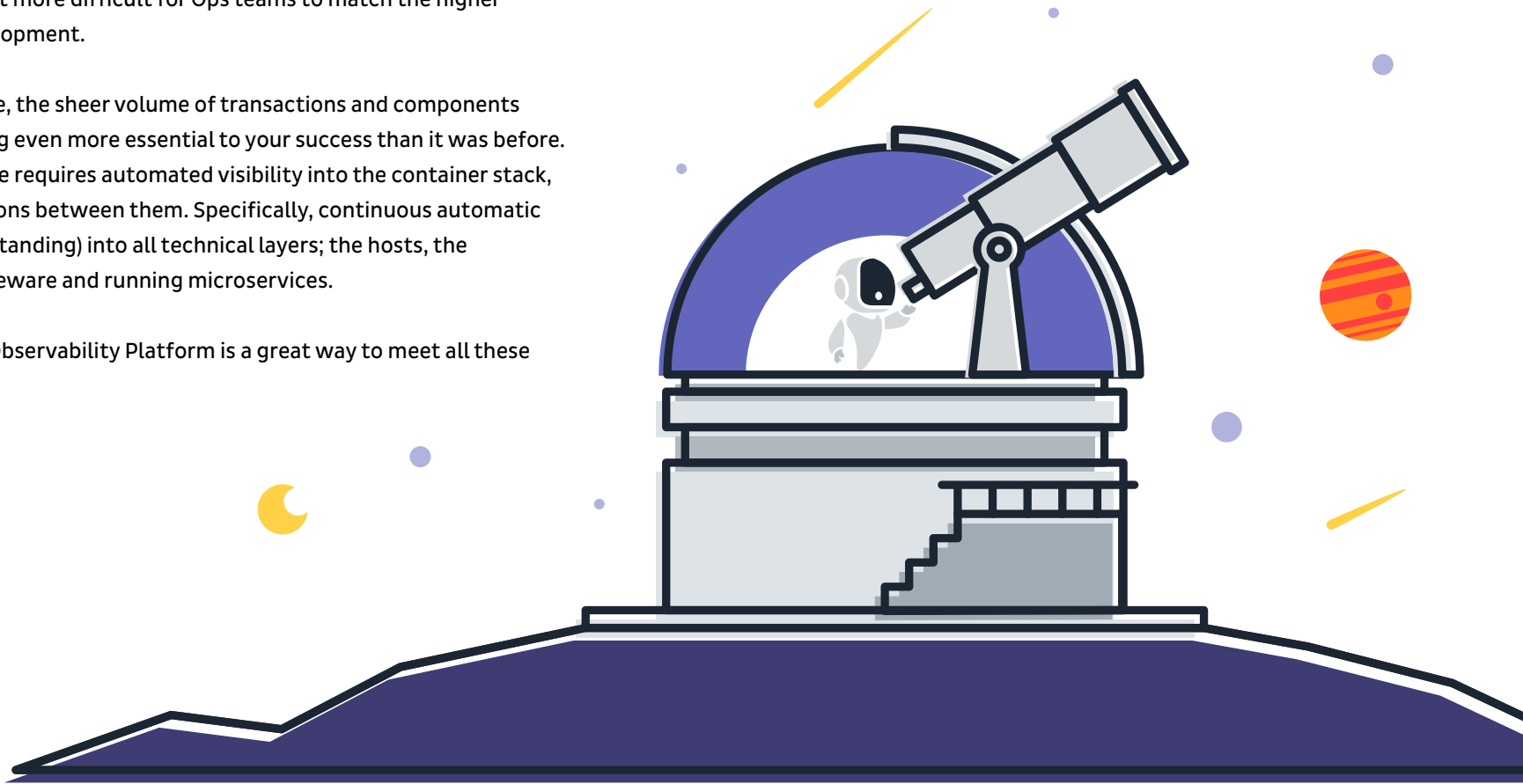
Containers have become indispensable for facilitating the acceleration of development velocity, but with this speed comes observability, monitoring and performance management challenges. Traditional monitoring tools, designed for older application technology, struggle with containerized applications, making it more difficult for Ops teams to match the higher velocity of agile development.

Making matters worse, the sheer volume of transactions and components make good monitoring even more essential to your success than it was before. Tackling this challenge requires automated visibility into the container stack, including all interactions between them. Specifically, continuous automatic visibility (with understanding) into all technical layers; the hosts, the containers, the middleware and running microservices.

Instana's Enterprise Observability Platform is a great way to meet all these new requirements.

Learn more by reading the eBook:

**Foundations of Enterprise Observability**



# About Instana, an IBM Company

Instana is the leading provider of Observability and Application Performance Management solutions for Cloud-Native Microservice applications. The company's Enterprise Observability Platform, powered by automated APM, ingests all observability metrics, traces every request, and profiles all processes continuously and automatically, delivering the actionable information with full context to Dev+Ops to help them optimize their application performance and pipelines.

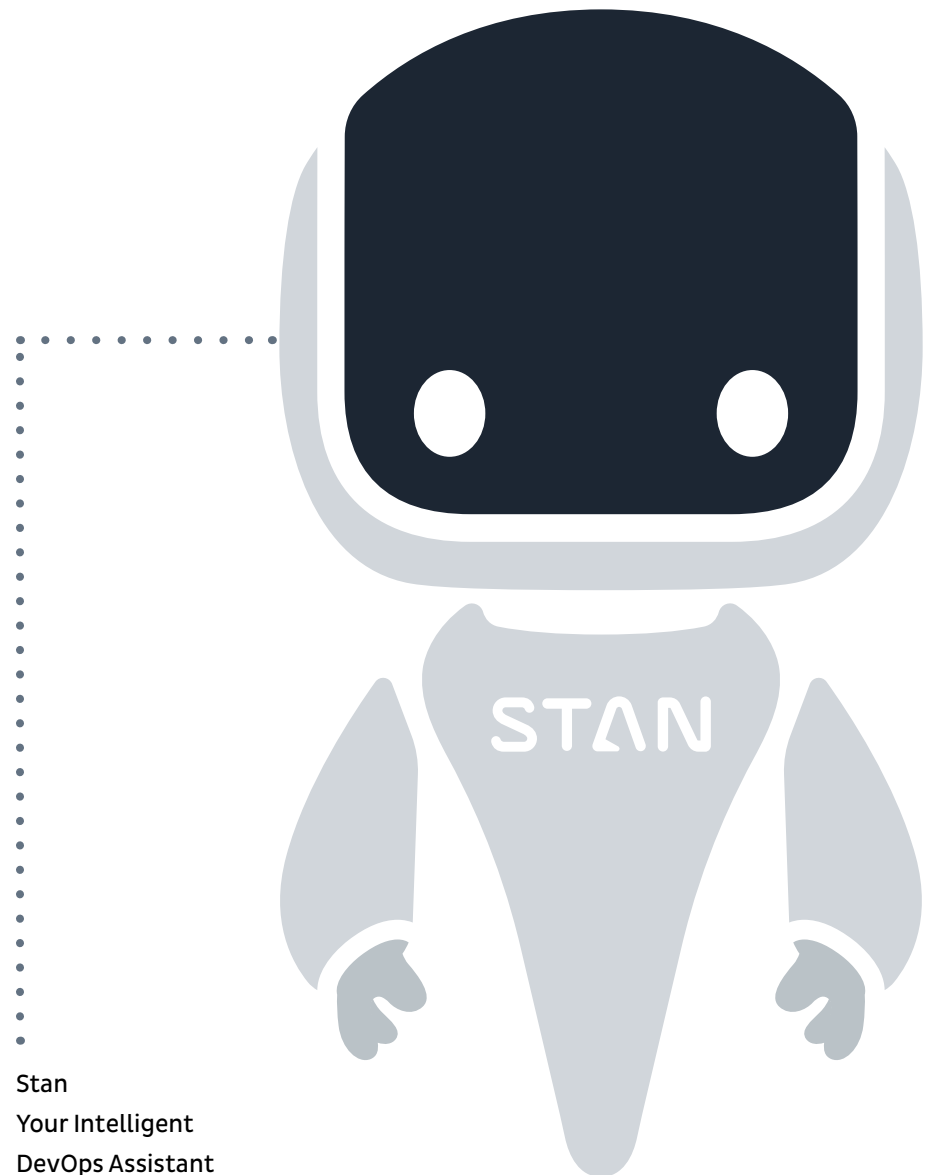
Learn more at <https://instana.com>

## Try it for Yourself

Don't just take our word for it.  
Try Instana's Automatic Monitoring and Enterprise Observability for yourself.

You can literally have full application observability  
in just a few minutes.

**Start Your Trial Today**



Stan  
Your Intelligent  
DevOps Assistant



**INSTANA**  
an IBM Company

IBM, the IBM logo and [ANY OTHER IBM MARKS USED] are trademarks of IBM Corporation in the United States, other countries or both.  
Instana® and its respective logo are trademarks of Instana, Inc. in the United States, other countries or both. All other company or product  
names are registered trademarks or trademarks of their respective companies.

©Copyright 2021 Instana®, an IBM Company