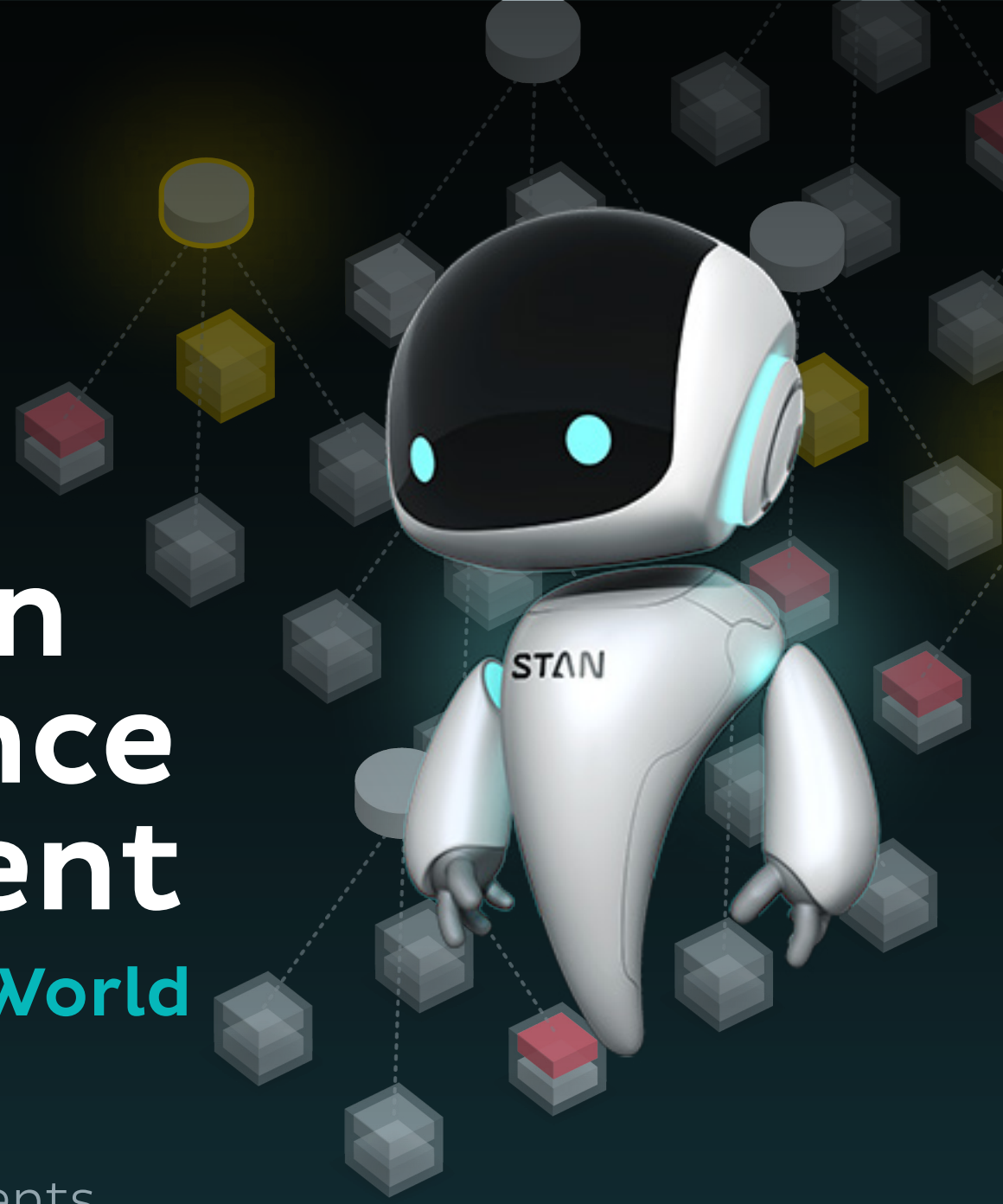
INSTANA

Application Performance Management

in a Containerized World

Achieving visibility in orchestrated environments



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." Today, containers are an essential component of the technology stacks at companies as diverse as General Electric and Oxford University Press.

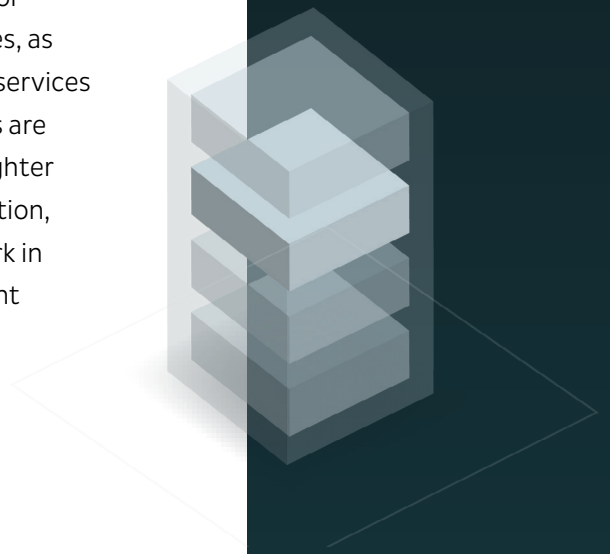
Perhaps no previous technological watershed — not the advent of virtual machines, not the open source revolution, not even the rise of Java — has matched the pace and passion with which containers have revamped the way organizations produce and deploy applications.

> **Achieving visibility in orchestrated environments.**

Explaining Container Adoption

What has driven the meteoric rise of containers? Two words - **Agility** and **Speed**. Containers have become so popular so quickly because they enable organizations to achieve a level of agility that was simply not possible using traditional virtualization and software deployment technology. And this agility leads to development speed, the ability to rapidly build and deploy new business services.

Containerized applications can scale incredibly quickly. Container environments make it easy for software delivery teams to switch seamlessly between different development frameworks. Containers provide ideal building blocks for constructing continuous delivery pipelines, as well as deploying modular, flexible microservices applications. Containerized environments are also much easier to duplicate, creating tighter parity between development and production, enabling developers and QA teams to work in an environment similar to the environment running the production applications.



The Container Monitoring Challenge

Yet with the revolutionary advantages of containers comes a new type of challenge. Application monitoring, visibility and performance tuning within containerized environments are much more challenging than they were when deploying applications using virtual machines or bare-metal servers. This is because containerized environments are composed of many components — an orchestrator, registry, runtime and more — whereas traditional environments were less complex.

For this reason, organizations that fail to overhaul their approach to monitoring when they adopt containers risk shooting themselves in the foot.

If you attempt to monitor your containerized applications and environments in the same way that you monitor traditional applications, you undercut the core advantages of adopting containers in the first place. Traditional monitoring tools can not keep up with the speed of containerized operations taking too much continuous manual effort to derive understanding from the tools.

Fortunately, achieving effective monitoring and visibility in a containerized world is a challenge that can be solved. As this eBook explains, however, meeting the challenge requires a fundamentally new approach to monitoring.

Below, we explain why monitoring within a containerized environment is so difficult, the mistakes organizations typically make when trying to address this challenge and how to develop an effective approach to maintaining complete visibility and agility after making the transition to containers.

Visualizing how highly distributed applications are structured and operating is challenging.
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. The reasons container monitoring is so difficult (and not possible to effectively achieve with monitoring tools designed for the pre-container era) include the following:

Lack of built-in 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 app technologies; and in an environment that has greater monitoring needs, due to the speed and frequency of application updates. And while operating systems provide significant data about system performance, they don't include application-level data.

Virtual machine platforms like VMware include relatively sophisticated monitoring tools, even if third-party add-ons are sometimes useful. With containers, however, it is much more difficult to collect sufficient monitoring and performance information from the container, itself.

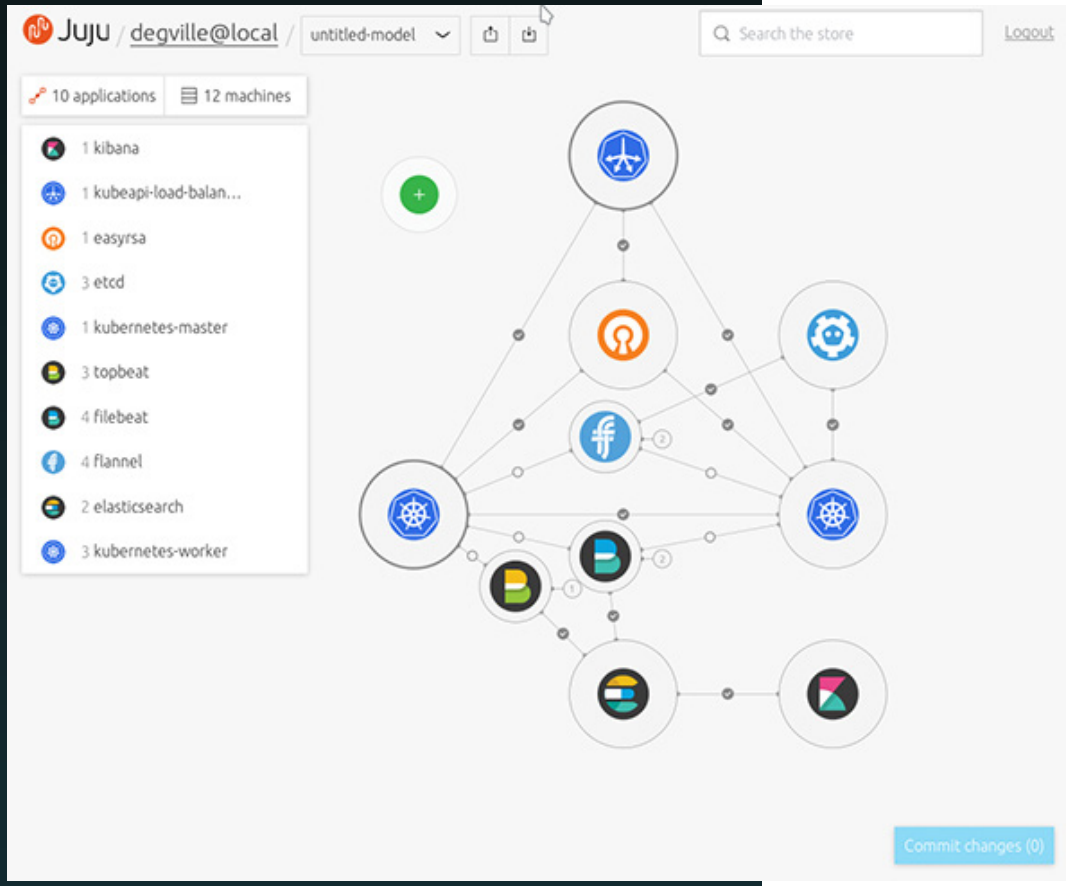
Container orchestrators are not performance monitoring tools

Container orchestrators have recently gained momentum to assist organizations in organizing, provisioning and managing the deployment of their production containerized applications. If you run containers in production, you almost certainly use an orchestrator, such as Kubernetes, DC/OS, Nomad or Docker Swarm, to provision your environment.

Orchestrators are not designed for sophisticated monitoring, and their focus on container and host resources prevents them from delivering a reasonable understanding of the performance and quality of APIs, microservices, middleware and applications.

Orchestrators also fail to provide sufficient monitoring functionality inside hybrid environments that amalgamate containers with other types of infrastructure. This is because container orchestrators can manage only the objects that they are aware of — which means containers. In a hybrid environment, container orchestrators cannot help to monitor resource usage of these other resources. This creates a second major monitoring gap for distributed applications. And that leaves a large part of your environment prone to performance problems.

> **While orchestrators are excellent provisioning tools, and are capable of finding and restarting failed containers in some situations, orchestrators should not be confused with performance monitoring tools.**



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.

Traditional monitoring tools don't support distributed microservices environments

Containerized environments tend to be dynamic and distributed. They are composed of services that are hosted on clusters of servers. Also, containerized apps usually employ a microservice architecture, where multiple independent services combine to create a complete application. The requests between those services accomplish the business processing.

Traditional infrastructure and application monitoring tools struggle to visualize and understand distributed microservice environments. They are designed to handle 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 you need to support microservices distributed across a large cluster of servers (composed of multiple languages, many middleware components, and multiple database systems).

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 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. Containers' ability to start and stop quickly is key to achieving the agility that containers offer.

Traditional infrastructure and application monitoring tools cannot support the rapid pace of change within containerized environments. For example, if a container spins up an NGINX web server instance that handles 150 requests during a period of five minutes, and the container then shuts down due to a performance problem, a traditional monitoring tool won't be able to include the server in its list of running systems - and the traditional APM tool will not be able to trace the problem because the container no longer exists. A modern APM tool, however, can look back in time and trace containers even after they cease to exist.

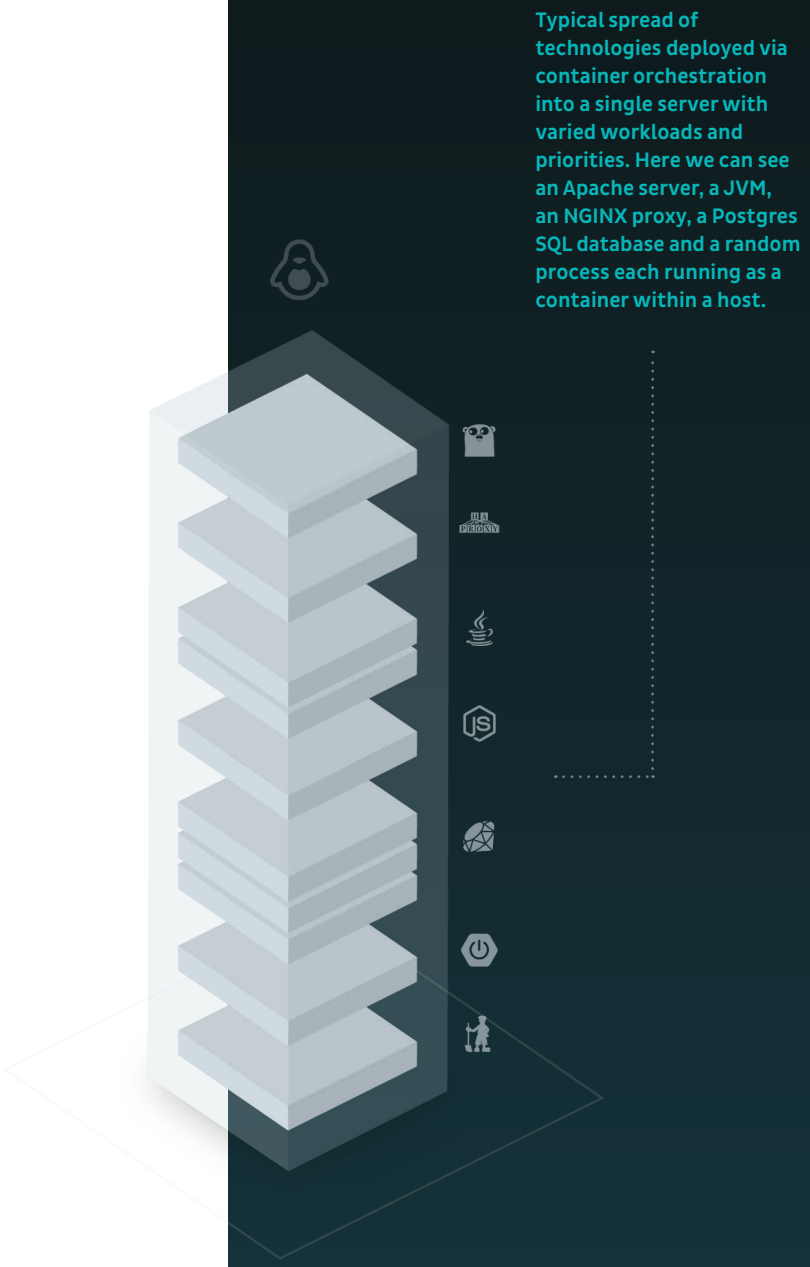
INSTANA

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/or DevOps. 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.



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.

INSTANA

Why Container Visibility Matters

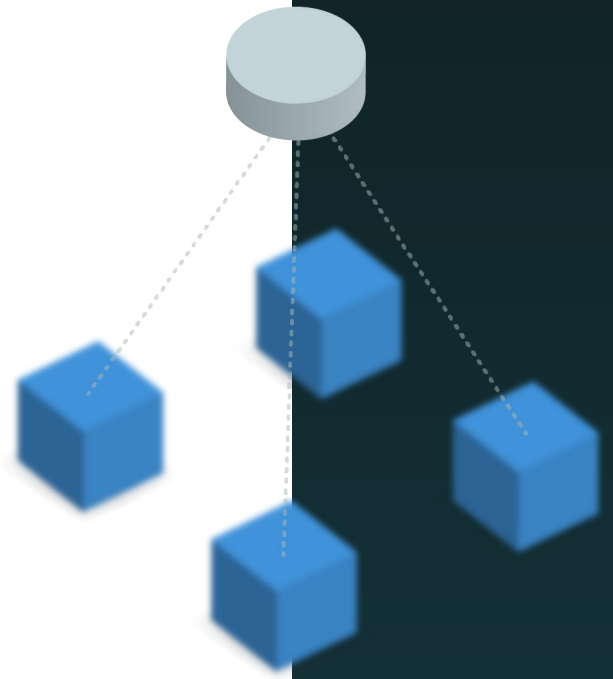
If there's anything IT has learned over the years of platform updates, it's that maintaining visibility is essential in any type of environment. In fact, It's the only way to keep applications running smoothly and to optimize resource usage and cost.

The Visibility Challenge in Containerized Environments

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

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.



Inability to pinpoint the root of a problem

Without visibility into your containerized application, you don't know whether the cause of an incident lies on the host server, with the container runtime, poorly operating middleware running in a container, within application code 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 small applications end up with hundreds of containerized software parts.

Poor performance metrics

Without the proper data from inside containers, you can't effectively determine that your containerized apps are meeting your service quality goals.

This means that you risk shooting yourself in the foot in the sense that the major reason for adopting containers is to make application deployments more efficient and agile. Without visibility, you have no way of measuring or maximizing efficiency and agility.

For example, you can't determine whether a new microservice or application deployment is meeting latency goals without the proper level of data. 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

Scalability, which means the ability to increase or decrease the size or number of instances of an application or microservice in response to fluctuations in 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.

Yet while containers make the ***HOW*** of scaling 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 and you can scale down in order to avoid paying for hosting resources that are under-utilized.

> **If you're familiar with virtual machines, you know how important scalability considerations are.**

Containers represent the modern form of virtualization, and misconfigured container environments will result in the same problems as virtual server misconfiguration. Insufficient allocation of resources to containers will result in application performance and scalability issues, while excess allocation leads to wasted money and resources. Without precise realtime visibility into the details of the containers, this will not be possible.

Antipatterns: How Not to Solve the Container Monitoring Challenge

Faced with the monitoring and visibility challenges described above, organizations that have migrated to containers are sometimes inclined toward solutions that appear to resolve the problem, but in reality constrain their agility and negate the advantages of containers.

Following is a collection of common examples that don't solve the problems - they mask them.

Trusting orchestrators to handle service quality

As discussed previously, orchestrators are not monitoring tools. 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 monitoring easier. For instance, you might decide to write all of your application code in a language that you know your legacy monitoring tool supports. While strategies like these will often help to simplify monitoring, they defeat the purpose of container adoption because they constrain your agility.

If you are forced to use only certain languages because your monitoring tools don't work effectively otherwise, you rob yourself of the flexibility that containers and microservices offer.

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 technology in order to enable monitoring negates 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.**

Using traditional infrastructure and application monitoring tools

> **Container-based applications simply have too many differences in the way they are architected, built, deployed and executed.**

Traditional monitoring tools were designed before the high speed agile development methodology and containers existed. They were not designed to be able to keep up with the speed of containerized operations taking too much continuous manual effort 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.

> These antipatterns represent the wrong way to think about how to solve the container visibility challenge.

The Right Solution:
Must-have Capabilities to Effectively Monitor Dynamic, Orchestrated Applications

The Right Solution: Implementing an Effective Approach to Container Monitoring

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

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.

Here are some attributes to look for:

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 manually configure a monitoring tool. A modern solution must fundamentally be automatic. In other words, with zero configuration (intervention by a human), the tool must be able to visualize the situation and monitor your application's performance.

Let's think of this another way. Continuous Integration and Delivery processes are about streamlining and automating the deployment of new services. Why would you settle for a monitoring tool that was not automatic?

> **Why would you settle for a monitoring tool that was not automatic?**

Be Container Aware

Your monitoring tools must be able to automatically look inside containers and understand the context of the environment in which they are running. They must also be designed to accommodate the rapidly changing state of containerized environments.

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 needed to analyze efficiency and performance. A modern solution must of discovers every application component and map the interactions & dependencies between them. The map must be automatically and continuously updated with dependency and visualize service quality information as changes occur.

High Data Fidelity, Real-time Analysis

Microservices and containers are short-lived and dynamic. A modern solution should be able to capture and analyze short spikes allowing operators to understand whether a code rollout caused an issue. Analysis must happen in real-time (under 5 seconds). Likewise, metric data collected must be very accurate and fine grained. 1 minute granularity is no longer sufficient. Look for 1 second granularity.

Capture a Distributed Trace with Every Request

To understand highly distributed, constantly changing applications, every request between every service and microservice must be monitored. A lightweight "Trace" which captures performance and flow detail, with dependency information about what middleware and hosts were involved for every segment of the request is preferred. The ability to drill down into code performance detail will also empower developers.

Assist With Root Cause Understanding of Microservice Issues

Let's not forget, 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 a new service will be missed.

As element counts grow into the hundreds or thousands, humans cannot practically manage application performance on their own. The latest buzzword in today's IT tools is ***Machine Learning***, but Machine Learning is actually not enough to eliminate the human drag.

It's important that your tool has been elevated to a more complete level of Artifical Intelligence assisted functionality, 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. Look for tools that automatically determine which Key Performance Indicators (KPIs) to collect, when measurements indicate service incidents, and determine the likely root cause of problems – all without human interaction.

Conclusion

Containers have become indispensable for facilitating the acceleration of development velocity, but with this speed comes visibility and management challenges. Traditional management tools struggle with these challenges, making it more difficult for IT Operations to match the higher velocity of agile development teams. Making matters worse, the sheer volume of transactions and components make good monitoring even more essential to your success.

> **Tackling this challenge requires visibility into the containers, as well as the interactions between them. Specifically, continuous automatic visibility (with understanding) into all technical layers; the hosts, the containers, the middleware and running microservices.**

Instana's monitoring solution embraces these new requirements.

Learn more by downloading our complimentary eBook:

The 6 Pillars of Managing Modern Dynamic Applications

About Instana

Instana provides the only APM solution that automatically discovers services, deploys agents and monitors component health for microservice and containerized applications.

Built to handle the demands of agile organizations, Instana continuously aligns application maps with any changes, detects behavioral anomalies, and automatically provides a health score for each technology component. Organizations benefit from real-time impact analysis, improved service quality, and optimized workflows that keep applications healthy.

The solution notifies DevOps teams when service quality is at risk, providing precise information identifying the triggering event and potential root cause.

We invite you

... to get a sense of just how much our solution can do. And if you're ready, give Instana a try today.

Try Instana today

Instana Container Solution

Stan
Your AI-Powered
DevOps Assistant

INSTANA

www.instana.com