INSTANA
an IBM Company

APM for Microservice Applications on Kubernetes

The Ultimate Guide to Managing Performance of Business Applications on Kubernetes

Table of Contents

There's a reason everyone is talking about Kubernetes these days. It has become the go-to container orchestration solution for organizations of all sizes as they migrate to microservice application stacks running in managed container environments.

Kubernetes is certainly worthy of the recent excitement it has garnered, but it doesn't solve every management problem, especially around performance. It's important to understand what Kubernetes does, and what it doesn't do; and what specific capabilities DevOps teams require from their tooling to fully manage orchestrated microservice applications and achieve operational excellence.

This eBook examines Kuberenetes, the operational issues it addresses, and those that it does not. Additional examination of modern DevOps process is included, with a discussion on management tooling needed to achieve continuous delivery of business services leading to excellent operational performance. The eBook concludes with a detailed analysis of the capabilities needed from your tooling to successfully operate and manage the performance of microservice applications running on Kubernetes.

Kubernetes Basics (or the A-B-C's of K-8-S)

Kubernetes (sometimes abbreviated K8s) is a container orchestration tool for microservice application deployment. It originated as an infrastructure orchestration tool built by Google to help manage container deployment in their hyper-scale environment. Google ultimately released K8s as an open source solution through CNCF (the Cloud-Native Computing Foundation).

Notice there is a critical aspect of operational management missing - application performance management. The whole discipline of application performance visibility and management is not part of the Kubernetes platform.

Orchestration is just a fancy word that summarizes the basic Kubernetes features:

- Container deployment automation, relieving admins of the need to manually start them
- Instance management - balancing the number of instances of a given container running concurrently to meet application demand
- DNS management regarding microservice / container load balancing and clustering to help manage scaling due to increased request load
- Container distribution management across host servers to spread application load evenly across the host infrastructure (which can help maximize application availability)

Why Kubernetes is Important

Remember, the goal of DevOps is speed!! Orchestration is all about enabling fast and easy changes to production environments so that business applications can rapidly evolve. The message is clear: speeding up your application delivery cycles adds huge value to your business. Automating container orchestration is a great complement to agile development methods and the microservice architecture. Modern CI/CD is automating the testing and delivery stages of development - containers and Kubernetes make it much easier to get your code into production and manage resources.

Kubernetes distributions

Many cloud providers have their own versions of Kubernetes (called a “Distribution”) that have unique enterprise capabilities added to the open source Kubernetes version, which provide a few distinct advantages:

- Organizations concerned about enterprise readiness get a fully tested and supported version of k8s.
- Additional enterprise functionality is included - for example, Red Hat’s OpenShift K8s distribution adds security features and build automation to the mix.

For most enterprise use cases, it‘s much faster and easier to use a cloud provider’s Kubernetes distribution than to set up the open source version. A wide variety of Kubernetes distributions are available, designed to run either on local infrastructure or as a hosted service in the cloud. You can get an updated list of distribution providers in Kubernetes online docs.

Kubernetes Application Monitoring Challenges

- Container Management is NOT Application Performance Management
- More Moving Parts - and Complexity
- Decoupling of Microservices from Physical Infrastructure
- Service Mapping - A New Layer of Abstraction
- Root-Cause Ambiguity

Container Management is NOT Application Performance Management

Now that we've discussed what Kubernetes does, let's explain what it does not do. Remember, Kubernetes orchestrates containers that are part of an application. It does not manage application performance or the availability of highly distributed applications. Similar to applications, Kubernetes doesn't consider performance when managing infrastructure.

Kubernetes effectively adds a layer of abstraction between the running application (containers) and the actual compute infrastructure. On its own, Kubernetes makes decisions about where containers run, and can move them around abruptly. Visibility of exactly how your technical stack is deployed, and how service requests are flowing across the microservices is not easily available via Kubernetes, nor is performance data (request rate, errors and duration or latency) of services a native part of Kubernetes. Operational production monitoring of application performance and health is absolutely not available via Kubernetes.

Let's look at other aspects of orchestrated containerized application environments that further complicate monitoring.

More Moving Parts – and Complexity

Any microservice application creates a trio of issues:

- Exponentially more individual components
- Constant change in the infrastructure and applications (the application stack)
- Dynamic application components, In a Kubernetes environment, there are many more moving parts than there would be in a traditional application stack.

With the addition of containers - and then orchestration with Kubernetes - each of these management challenges becomes even more difficult. Every time there is a decoupling of physical deployment from the application functionality, it becomes more difficult to monitor application performance and solve problems. Instead of host servers connected with a physical network, Kubernetes utilizes a cluster of nodes and virtualizes the network, which can be distributed across a mixture of on-premise and cloud-based infrastructure,
or even multiple clouds.

With so many different pieces of infrastructure and middleware, as well as the polyglot of application languages used to create the microservices, it's difficult for monitoring tools to distinguish the different needs and behaviors of all these critical components in the application stack. For example, collecting and interpreting monitoring data from any one platform is different from all other platforms. What do you do when you have Python, Java, PHP, .NET, Application Proxies, 4 different databases and a multitude of middleware?

Decoupling of Microservices from Physical Infrastructure

Kubernetes takes control of running the containers that make up the microservices of your application, completely automating their lifecycle management and abstracting the hardware.

Kubernetes will run the requested workloads on any available host/node and using software-defined networks to ensure that those workloads are reachable and load balanced. Compute resources (memory and CPU) are also abstracted with each workload having a configured limit for those resources. Because containers are ephemeral, any long term storage is provisioned by Persistent Volume Claims provided by various storage drivers.

The already deep level of abstraction may be further compounded by the Kubernetes nodes running on external cloud computing services such as EC2, GCE or Azure.

The high level of disconnect from the application code to the hardware it's running on makes traditional infrastructure monitoring less critical. It is considerably more important to understand how the microservices and overarching applications are performing and if they are meeting their desired SLAs. An understanding of the overall health of the Kubernetes backplane is also essential to ensure the highest levels of service for your application.

Service Mapping – A New Layer of Abstraction

As noted earlier in this eBook, one of the main reasons for using an orchestrator like Kubernetes is that it automates most of the work required to deploy containers and establish communications between them. However, Kubernetes on its own can't guarantee that microservices can communicate and integrate with each other effectively. To do that, you need to directly monitor the services and their interactions.

That is challenging because Kubernetes doesn't offer a way to automatically map or visualize relationships between microservices.

Admins must manually determine which microservices are actually running, where within the cluster they exist, which services depend on other ones and how requests are flowing between services.

They must also be able to quickly determine quickly how a service failure or performance regression could impact other services, while also looking for opportunities to optimize the performance of individual services and communications between services.

Root-Cause Ambiguity

APM tools exist because middleware-based business applications - first using Java and .NET, then using SOA principles, and even microservices and containers - make it difficult to monitor performance, trace user requests, then identify and solve problems. The more complex the application environment, the harder it becomes for DevOps teams to get the performance visibility and component dependencies needed to effectively manage application performance.

In a Kubernetes environment, determining the root cause of a problem based on surface-level symptoms is even more difficult, because the relationships between different components of the environment are much harder to map and continuously change. For example, a problem in a Kubernetes application might be caused by an issue with physical infrastructure, but it could also result from a configuration mistake or coding problem. Or perhaps the problem lies within the virtual network that allows microservices to communicate with each other.

Of course, when the problem lies within the application code, it's important to have the deep visibility required to debug actual code issues, even understanding when bad parameters or other inputs are causing application problems. Ultimately, there could be a myriad number of root causes for the issue, ranging from configuration problems in Kubernetes, to an issue with data flows between containers, to a physical hardware failure.

To put it simply, tracing problems in a Kubernetes environment back to their root cause is not feasible in many cases without the help of tools that can automatically parse through the complex web of data and dependencies that compose your cluster and your microservice application's structure.

Seeing Through the Complexity

By now, it should be clear that managing the performance and availability of Kubernetes applications is challenging and scary! It's not hopeless, though. With the right APM (Application Performance Management) tool, you can manage your Kubernetes environment in a way that maximizes uptime and optimizes performance, combining the benefits that K8s offers with the goal of achieving DevOps excellence.

Let's look at key types of visibility that your monitoring should support for applications running in a Kubernetes environment.

Application Service Identification and Mapping

As discussed earlier, Kubernetes injects a new level of application abstraction, making it difficult to know how well individual services are running, or the interdependencies between all the deployed services. Your APM tool must be able to see past Kuberenetes and the container system to identify the application services - and how they are related to each other.

> ***$ kubectl get svc***
> Lists out Kubernetes service definitions but not their relationships.

Kubernetes services are NOT the same as application services.
The K8s documentation states:

"A Kubernetes Service is an abstractionwhich defines a logical set of Pods and a policy by which to access them"

There can be multiple application services within a Pod.

Microservice Relationships

You also need to know how your Kubernetes services map to application services, the microservices they are built upon and their physical infrastructure in order to determine how the infrastructure impacts the services' availability and performance. Kubernetes doesn't easily reveal all of this information; you need to run multiple kubectl commands to manually build a mapping at a single point in time. Good luck doing that when there is a production issue that needs to be fixed immediately.

Application Request Mapping and Tracing

The microservices that comprise an application constantly send and receive requests from each other. Effective microservice application monitoring requires your APM tool to detect all the services, as well as the interdependencies between them - and visualize the dynamic relationships (i.e., map them) in real time.

Additionally, to solve problems, you will need exact traces from each individual application request across all the microservices it touches.

> There is no ***kubectl*** command to provide this information.

Deployment Failures

If Kubernetes fails to deploy a pod as expected, you want to know why and how it happened. However, it's more important to understand if your application functionality has been negatively impacted by this deployment failure. Is your application slower and handling less workload or is it throwing errors because a critical service is unavailable?

```
$ kubectl get events --field-selector involvedObject.name=my-deployment
```

The event stream will show where the deployment failed.
Since you cannot see the performance of your application using kubectl commands, the only way to answer the question above is with an APM tool that understands Kubernetes.

Performance Regressions

If your application is responding slowly, it's important to identify the issue and trace it to its root cause quickly. Since Kubernetes was not designed to help with this use case, there are no kubectl commands you can run to understand microservice or application performance.

Troubleshooting microservice applications running on Kubernetes requires your APM tool to have the ability to correlate metrics up and down the full application stack: infrastructure, application code, kubernetes system information, and the trace data between the services.

Infrastructure metrics like CPU, memory, disk I/O, network I/O, etc are good KPIs to reference while troubleshooting performance issues but they are only a part of the information required to fully ascertain root cause. There might also be issues with the application code or Kubernetes configuration issues that are causing resource contention. It's quite common to over-allocate CPU and memory resources on Kubernetes nodes with improper configuration.

Performance Optimization Opportunities

In Agile development environments, developers often push new code into production on a daily basis. How do they know that their code is delivering good response time and not consuming too many resources?

To help with this, the APM solution must work at the speed of DevOps, automatically and immediately recognizing when new code has been deployed - or any changes to the structure of the environment (including infrastructure). It must also make it easy for developers to analyze the efficiency of their code.

This use case calls for granular visibility into user requests, host resources (K8s nodes), and workload patterns. It's also critical that you have a robust analytics mechanism for all of this data. You cannot accomplish this use case with Kubernetes alone.

Should You Use Prometheus?

Prometheus has become the go-to monitoring tool for Kubernetes but it's missing some important functionality. Let's begin our Prometheus exploration by discussing what Prometheus does well.

Time series metrics | Flexible API | High cardinality | Monitoring and Alerting

Prometheus is a good stand-alone tool for collecting time series metrics but it is not capable of meeting the majority of use cases presented in this document. Here are the drawbacks of using Prometheus as your monitoring tool:

- No distributed tracing capability
- No correlation between service infrastructure and host
- No correlation between Kubernetes resources, request response times, and infrastructure metrics
- No analytics interface, roll-ups, or aggregates
- No automatic root cause analysis
- No automated alerting
- Management and Administrative costs

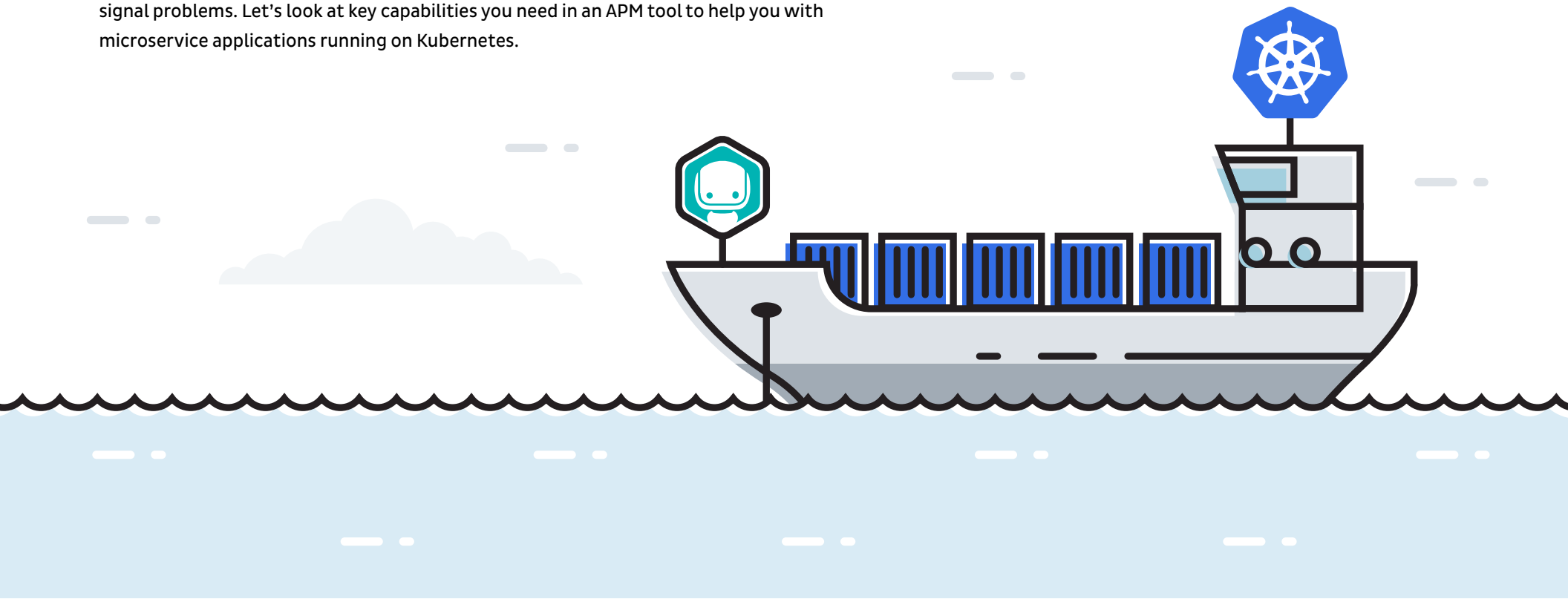
Prometheus is an open-source time series metrics monitoring and alerting tool. It is typically used to monitor KPIs, such as rates, counters, and gauges from infrastructure and application services. You can use Prometheus to monitor request response times but this often requires that you modify your source code to add the Prometheus API calls. This can be useful to understand overall response times and request rates but this approach lacks the detail required to troubleshoot or optimize application performance.

The Kubernetes distribution natively supports Prometheus, and when the Prometheus Helm package is installed, you'll find several dashboards pre-configured for the purpose of basic health checks. You'll also find a few predefined alerts configured on your cluster.

Ultimately, Prometheus is a good stand-alone metrics tool that cannot meet the challenges associated with running business critical microservice workloads on Kubernetes.

Kubernetes Monitoring Tools and Strategies

Achieving the elements described above in a Kubernetes application requires an APM tool that includes special features absent within traditional monitoring tools. For Kubernetes, you can not just collect monitoring data and detect anomalies that could signal problems. Let's look at key capabilities you need in an APM tool to help you with microservice applications running on Kubernetes.

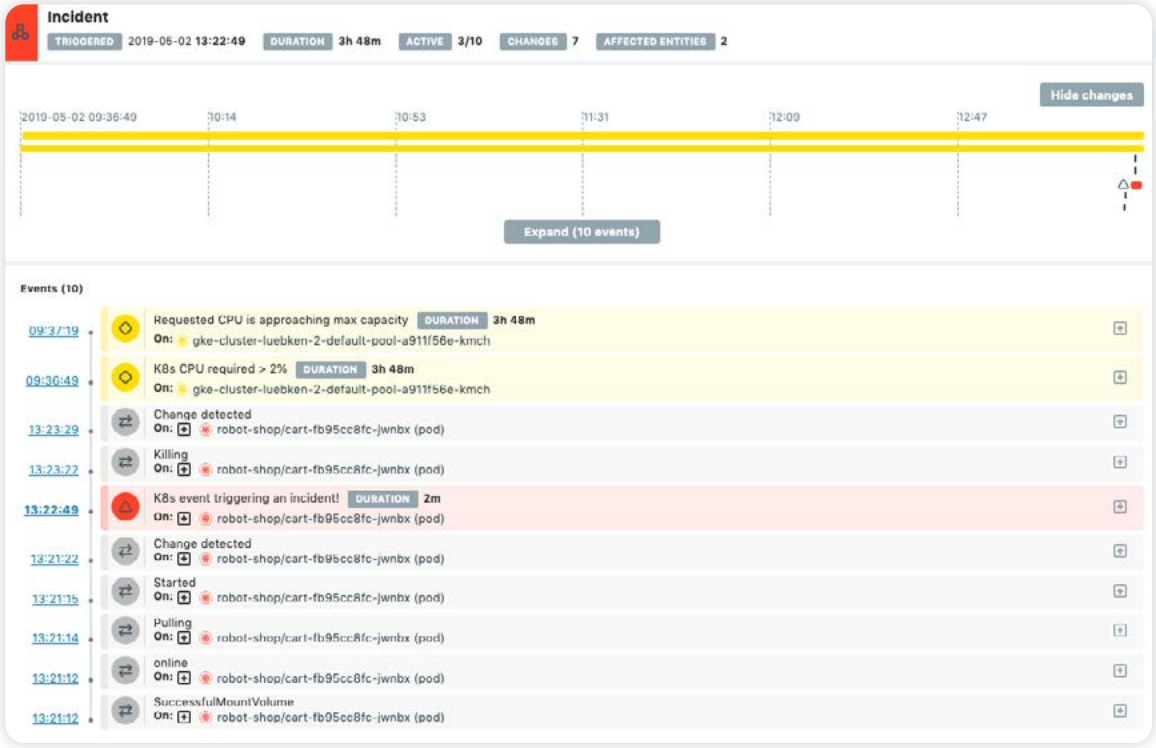


Root-Cause Analysis Within the Application, Containers and Orchestration

One is the ability to identify the root-cause of performance issues automatically. It's not good enough to just be aware of problems within your Kubernetes environment.

You must be able to trace those problems to their exact root cause and fix them in minutes.

Given the extreme complexity of a Kubernetes-based application and the lack of visibility into the environment, identifying the root causes of availability or performance problems is exceptionally challenging to do manually.

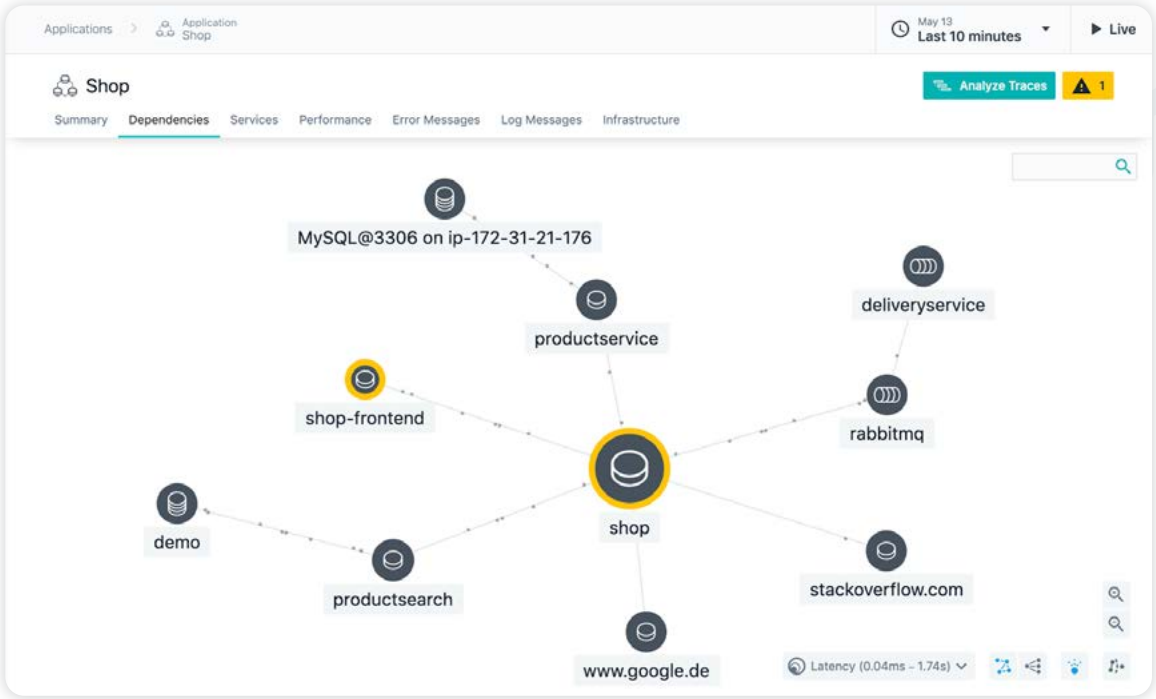


When your APM tool understands the relationships between Kubernetes, application services, and infrastructure, it can automatically identify the root cause of issues anywhere within the system.

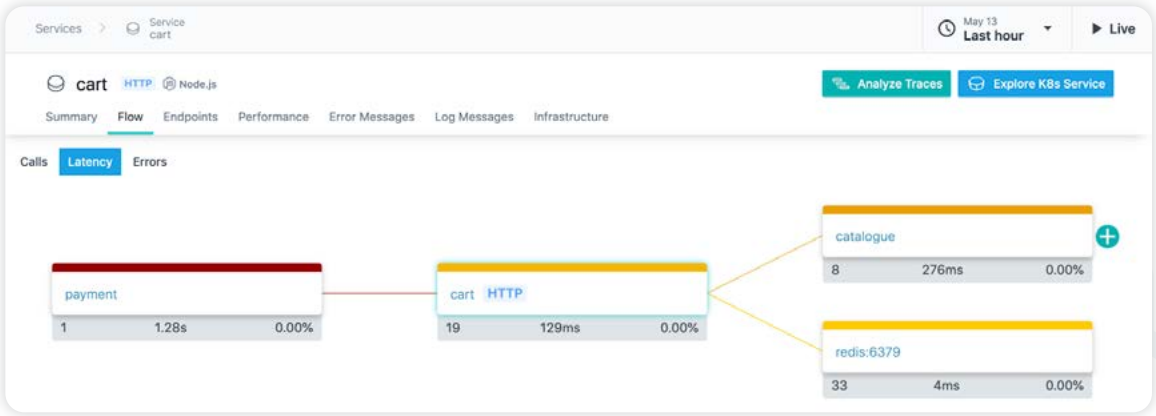
Integrated Service / Infrastructure Mapping

Given that Kubernetes doesn't offer full visibility into how services interact with each other, your monitoring tool must be able to map services automatically.

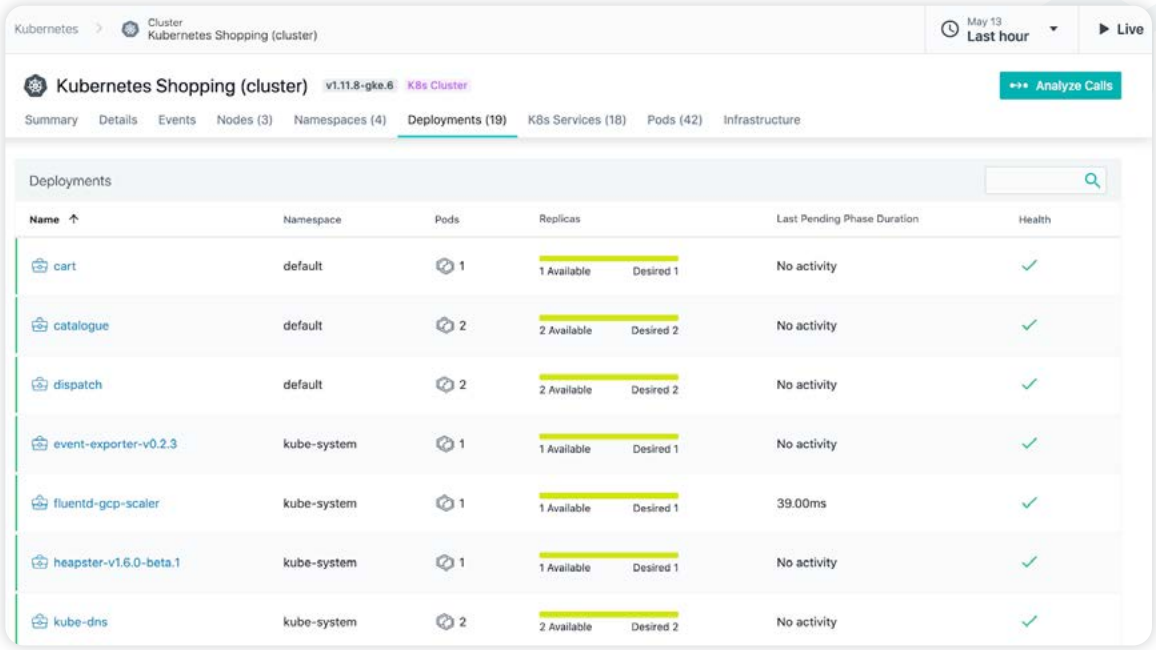
Equally important, it must have the ability to interpret the relationships and dependencies between those services in order to identify problems and understand how one service's performance will impact that of others.



Dependencies between all services are continuously mapped and monitored to understand the performance of the system as a whole.



Upstream and downstream dependencies of individual services are automatically identified. Every application service is correlated to its Kubernetes service so that you can seamlessly navigate between data sets.

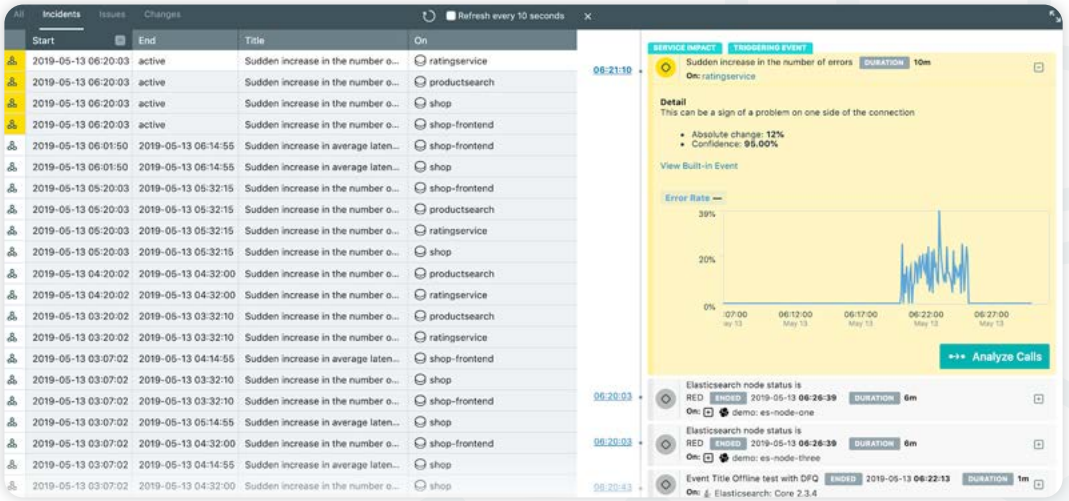


Kubernetes cluster data is collected and analyzed with the correlated application performance data to create a holistic understanding of the system.

Dynamic Baselining

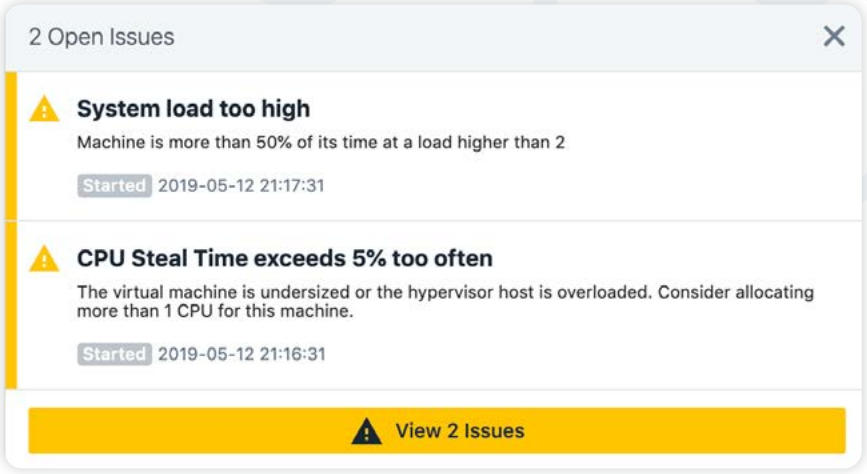
With environment architectures and configurations changing constantly, your APM tool must make sense of highly dynamic monitoring data and distinguish true anomalies from normal changes.

Identification of application performance issues requires anomaly detection based upon machine learning. Alerts are raised when performance indicators deviate too far away from normal behavior.



Remediation guidance

When something goes wrong in your enormously complex Kubernetes environment, you want to be able to resolve the problem quickly. That is difficult for human admins to do without the help or guidance from their APM tool. There's just too much data, and too many fast-changing variables, for humans to wade through to formulate an incident response plan on their own.

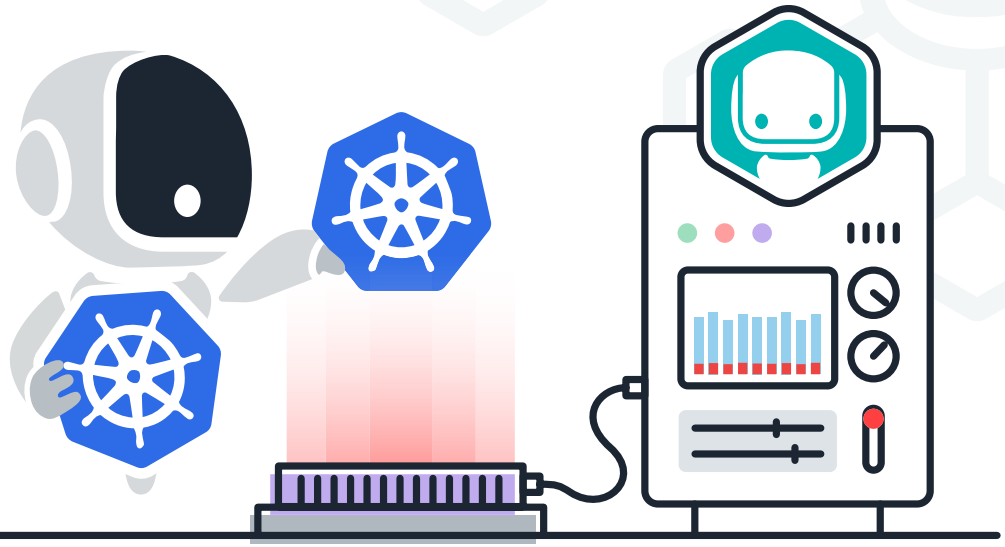


Conclusion

Kubernetes is rapidly becoming the standard orchestration platform in enterprises to augment and even complete the transition to DevOps, but does not include application performance visibility or management. Furthermore, Kubernetes introduces a new layer of abstraction into the datacenter creating observability challenges making it more difficult to manage application availability and deliver the needed performance SLAs demanded by your business.

To properly manage business critical applications on Kubernetes, Instana recommends an APM tool with these key capabilities:

- Full-stack visibility (including infrastructure, code, microservices, request traces, middleware, containers and Kubernetes) of all technology layers
- Continuous discovery of the full application stack to automatically adjust to changes in the environment
- Dependency mapping and correlation between the layers of technology
- Automatic root cause determination and assistance for the DevOps teams to troubleshoot application issues.

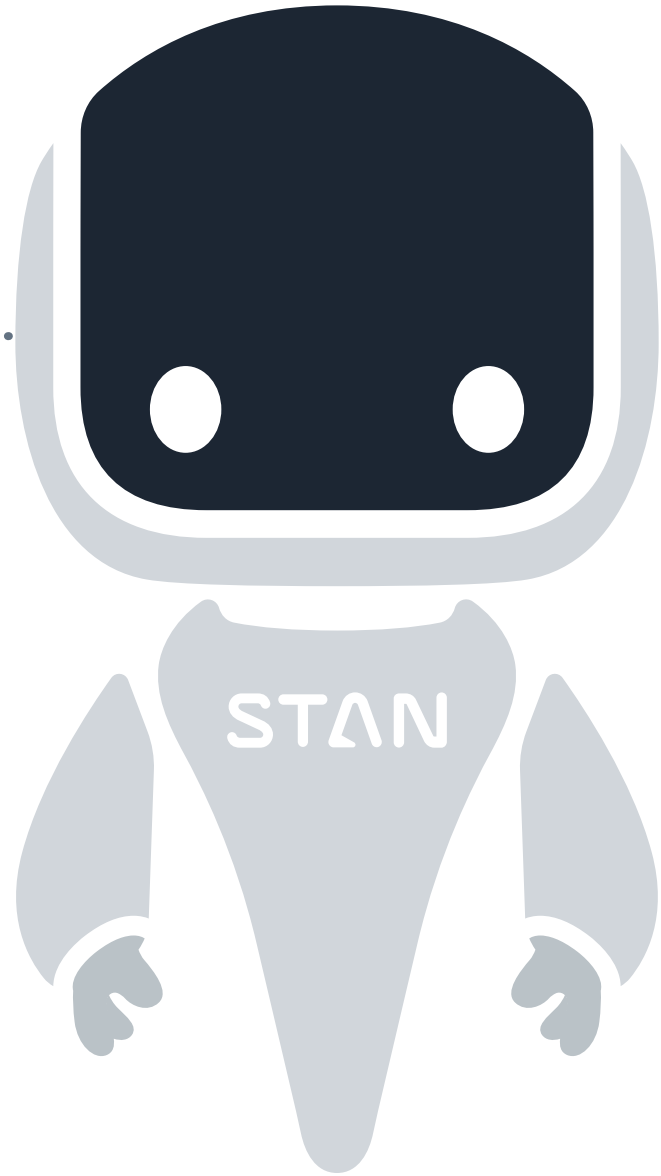


About Instana, an IBM Company

Instana is the only APM tool specifically built for microservice applications running on Kubernetes. The solution automatically discovers the full containerized application stack, automatically understands the performance of your microservices, and includes automatic determination of the root cause of performance issues.

The solution is designed to empower the full DevOps team.

Start Your Trial Today



Stan
Your Intelligent
DevOps Assistant

INSTANA

an IBM Company