The OpsCruise Monitoring Manifesto:

Actionable Observability for Your Modern Applications

OpsCruise

Prologue

This is an update to the original manifesto that OpsCruise published in April 2020. We have since seen a progressive trend towards microservice applications and Kubernetes adoption, as well as a growing trend towards using Open Source monitoring and OpenTelemetry. Having founded OpsCruise to build actionable observability for Kubernetes applications, it is fitting we provide the next revision of our manifesto. While much of the content and motivation has not changed, we have now had the opportunity to learn from lessons from the field given the broader adoption of OpsCruise and added capabilities that enhance observability built on open source monitoring.

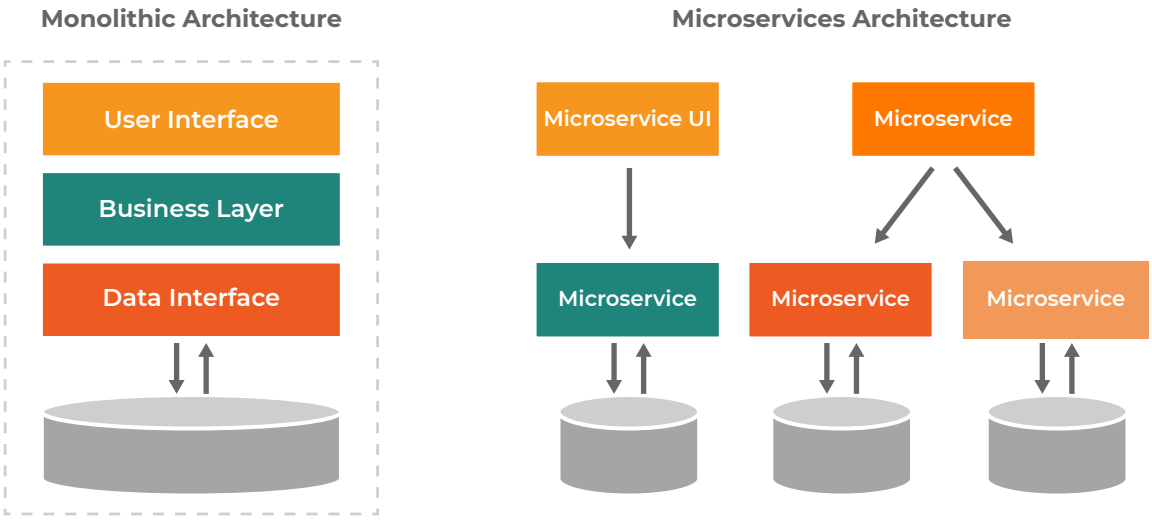
Modern Cloud Applications are Fundamentally Different

It is well-established that the adoption of cloud computing is continuing at a significant pace. Gartner[1] predicts by 2026 public cloud spending will exceed 45% of all enterprise IT spending, from less than 17% in 2021. With the move to cloud, more applications driven by the need for scalability, agile and rapid deployment, are cloud native and built as microservices. Per Red Hat's 2022 State of Enterprise Open Source survey over two-thirds IT organizations are using containers[2].

Interestingly, while cloud-native architectures fundamentally changed how applications are written and deployed, the challenges in observability of cloud applications are still similar to those of legacy monolithic applications in terms of detecting performance related concerns:

- How to detect a condition that may impact a customer-facing service?
- Is the alert from an incident benign or a false positive?
- Does the alert indicate whether end users are being negatively affected or not?
- How to ensure that Ops is fixing the root cause and not just the alert symptoms so that they do not face the same problems again?

It is therefore worth exploring why the observability challenges became bigger.



Microservices versus Monolithic application architecture (ref: BMC)

[1] Gartner Says Four Trends Are Shaping the Future of Public Cloud

[2] Red Hat's 2022 State of Enterprise Open Source

Why Are Modern Applications More Challenging to Troubleshoot?

What is different in the case of microservice applications is that it is harder to answer these questions given:

- Complexity caused by sheer scale and the multi-tiered nature of orchestrated applications: obfuscation within microservice applications since they are more than two-dimensional service maps. They are built on containers, provisioned on an orchestration layer, i.e., Kubernetes, which in turn is built on a cloud infrastructure layer. There are multiple points of performance loss: an incident in the application, in the container, in the orchestration layer, or in the infrastructure services can have an impact on multiple services that are masked by these layers of obfuscation.
- Dependencies across services and down to infrastructure: shared infrastructure resources: as with many distributed systems that share infrastructure services, especially a network of services, there are many points of failure in the microservices application. Even before a failure is manifestly obvious, the application could be operating in a degraded mode given there are many more states of the overall application. Third- party SaaS and 3rd party APIs add to the heterogeneity of service types.
- Dynamism: applications are ephemeral and hourly/daily code pushes and releases creates changing topologies and services. Couple that with transient workloads and services to create more challenges in not just the spatial but also temporal aspect of the monitored application.

Existing Approaches Don't Address Today's Challenges

Organizations get lost on one or both of the illustrated roads above.

Naturally, many organizations head down the path of applying traditional monitoring tools to these new modern application environments. Unfortunately, the old ways of monitoring are no longer enough! Disjointed isolated monitoring of discrete aspects of the application down to the infrastructure means that while there is a lot of data, there is relatively poor insight into the application state resulting in mostly manual processes that reduces time to resolve problems. Further, these vendors rely on proprietary 'agents' to be deployed on every host and insist that all the raw telemetry is stored in their cloud. That's an expensive and unnecessary model for these newer environments.

Alternatively, many organizations have embraced the ecosystem of CNCF observability projects such as Prometheus for metrics, Loki and FluentD for logs, Jaeger for tracing, etc. These increasingly popular projects/ tools are a great starting point.

Unfortunately, just getting different telemetry and monitoring does not really help without considering them in context! With more sources of data, there are more dashboards adding to the 'cognitive overload' forcing the Ops team to switch between different tools, and run through many visual queries looking for patterns that may not be applicable.

The conventional wisdom was to get the 'three pillars of observability', comprising logs, metrics, and traces. Unfortunately, metrics, logs and trace are no longer enough when trying to detect and resolve problem scenarios! Besides, switching Ops has the mental burden of maintaining context to link them, even as the application structure is changing given the ephemeral nature of microservices. We now need to check if new added services created more new dependencies or old ones removed when services are deleted. Also, configuration of the underlying Kubernetes infrastructure, which can also change, may be a cause of a problem. Ops needs to be aware of the changing topology and dependencies as well as configuration of the provisioned infrastructure!

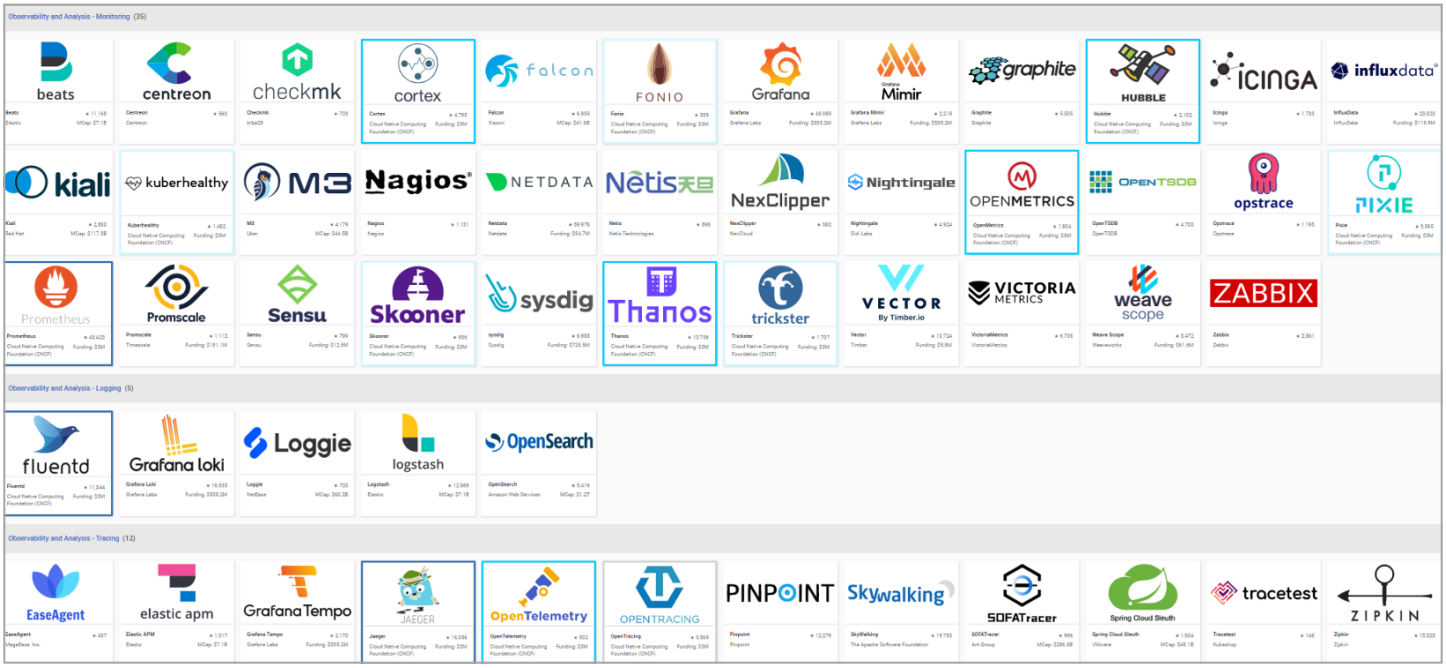
While distributed tracing tools can help in the diagnosis of performance problems, that approach forces the Ops team to operate primarily in a post-facto mode. What they lack is an integrated view in real-time into what is happening within and across the application without adding dashboards per metric.

Ops teams need intelligence from their observability tools that can automatically provide:

1. Real-time visibility into the ever-morphing application and its dependencies
2. Detect problems without guessing and tuning thresholds, and
3. Reduce the manual work to automate as much of the cause isolation

Let's start with how we get to the first step – the data needed for getting real-time visibility into the application.

The Monitoring Data You Need is Freely Available from Open Source!



Open source Observability projects (source: https://landscape.cncf.io/)

The Cloud Native Computing Foundation (CNCF) infographic above illustrates how these common classes of cloud application landscape have evolved. The takeaway from this infographic is that all types of telemetry used for monitoring are now freely available. With the support of the CNCF community, it also means using these monitoring tools provide a vendor-independent future-proof observability framework.

What we need is the contextual integration and analyses of this diverse telemetry that provides visibility into the state of the application.

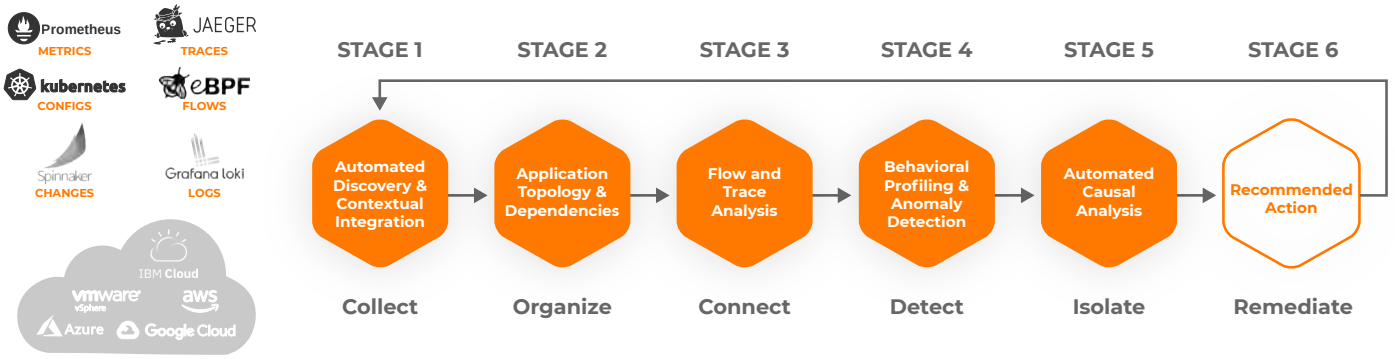
Building Real-Time Actionable Intelligence on the Open Source Monitoring

OpsCruise observability is built on the open source and cloud provider monitoring as shown in the figure below.

With OpsCruise, there is no more need to deploy proprietary agents or code instrumentation. Besides future proofing your observability platform, you gain several advantages, including:

- Your applications can be on public cloud, on-premise, or a hybrid of both and still be observable with a single platform
- Using OpenTelemetry (OTel) and open source tools means there is no need for proprietary host agents
- Data collection is lightweight as only a single collector of telemetry by type per cluster is all you need
- Monitored data stays in your control as the application owner and can be used for multiple business purposes
- Cost savings are significant as the stored telemetry can utilize low-cost cloud storage (e.g., AWS S3) as opposed to the vendors' monitoring services
- Accessing telemetry does not incur cloud egress charges
- Pick from the many deployment options including SaaS, on-prem software, or air gapped

OpsCruise's Observability: Smart Layer on Curated Open Source Monitoring



OpsCruise's pipelined knowledge-augmented ML processing for Observability

Now that you have access to all open source and OTel telemetry and monitoring data, OpsCruise can now go to work on embedding, supporting and unifying, i.e., contextually "stitching together" all telemetry to extract and provide deep insights. With OpsCruise you can:

- Auto-discover all full-stack dependencies - between services ('service map') and from services down to orchestrator and infrastructure ('three-layer view')
- Provide a single UI into different facets of the application from telemetry, performance, configuration and changes
- Expose dependencies in the application in real time through Flow and Trace Analytics
- Deliver predictive anomaly detection without requiring manual Ops intervention
- Automate causal analysis to isolate problem components

To improve the efficacy of analysis in each stage, OpsCruise embeds and uses curated knowledge on the technology stack, the known applications, and IT diagnostics processes.

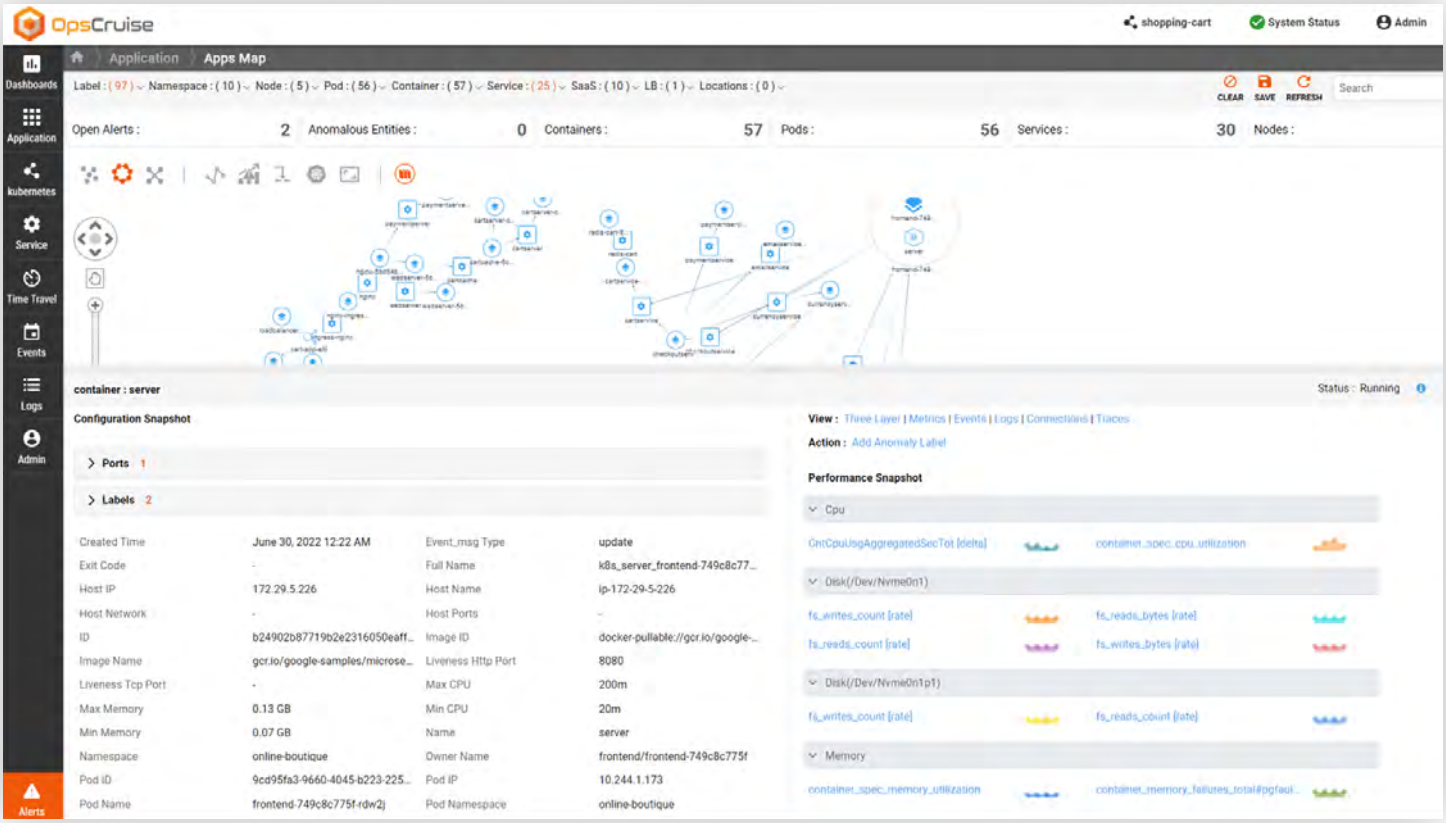
Here is a summary of the capabilities in each stage.

Stage 1 – Automated Discovery and Contextual Integration

OpsCruise's platform unifies the telemetry data (metrics, logs, traces, flows, etc.) in a distributed object model representing the microservice application as a dynamic distributed system. This enables representing application services of different types, whether microservice, SaaS, serverless or FaaS, understand services can be auto-scaled and work with application components that are ephemeral in nature, e.g., services that are added and removed and actual physical instances are not permanent.

This enables contextually associating all aspects of the topology of a container backing a service, allowing users to:

- Quickly navigate to an object (e.g., Kubernetes pod) and see all the associated metrics, logs, events, connections (into and out of), and traces of that pod without needing to maintain topology mappings or maintain mental context
- Automatically see changes in the topology even as services are modified, whether through auto-scaling, service additions or deletions, or even when the image in the container is changed
- Check on configuration and deployment details of all objects in the Kubernetes estate directly from the UI without requiring users to run kubectl commands, allowing SREs who are not Kubernetes experts to easily explore the correctness of the deployment



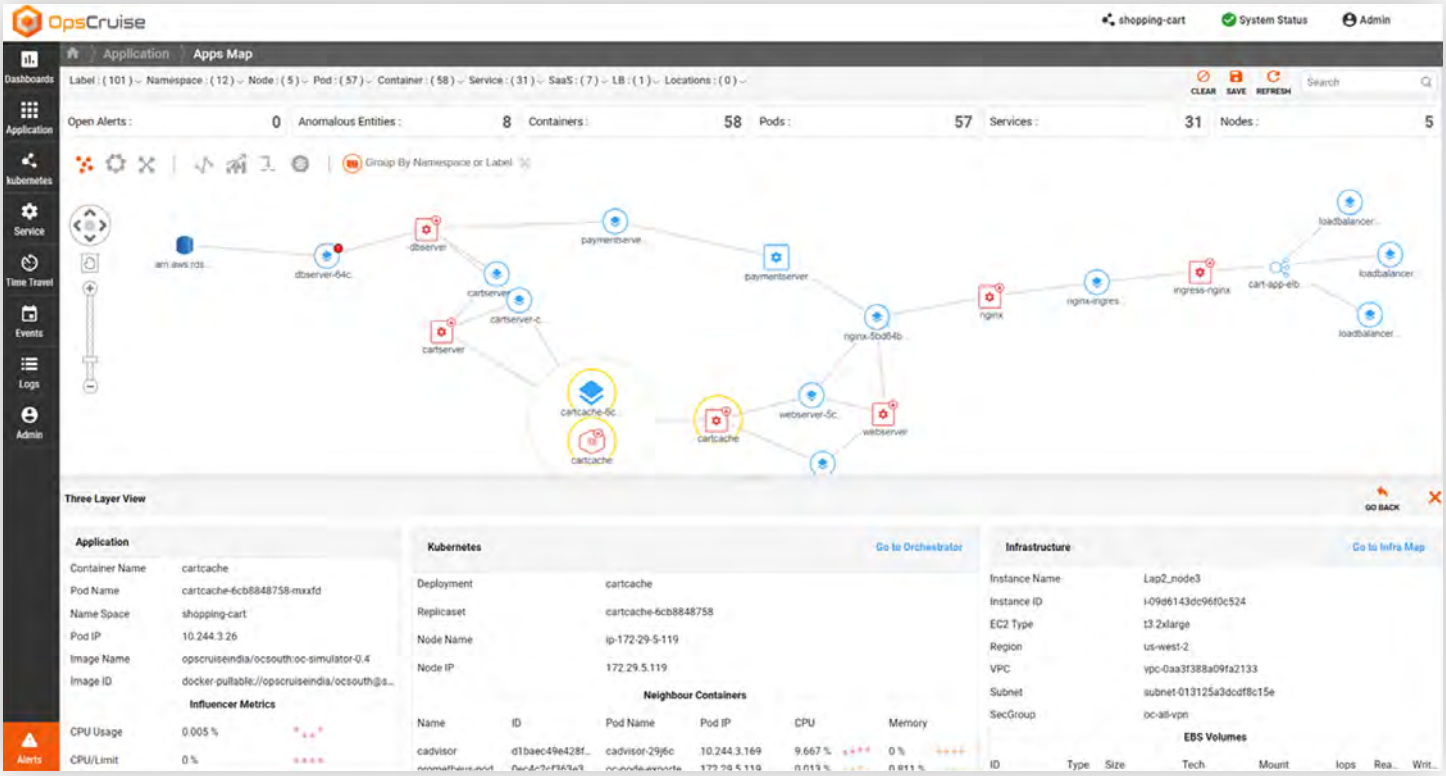
Automated contextual linking of metrics, logs, events, connections, and traces of an application container

Stage 2 – Application Topology and Dependencies

Once all telemetry is processed and contextually mapped, OpsCruise's platform is unique in creating the:

- Application topology and all dependencies using eBPF network flow data (see Stage 3 Flow Analytics) without need for enabling tracing or using proprietary agents
- Different views of the complete application structure is presented at the application level (App Map), Kubernetes node level (Node Map), VM and process (Host Map) as well trace level (Trace Map)
- Providing the golden signals, service interactions and service-level performance in real time using flow analytics

Furthermore, because there is no need to embed any proprietary agents into the application pods or containers, OpsCruise can be deployed and be up and running in minutes.



An application level view (App Map) showing dependency from container to infrastructure ('three-layer view')

Stage 3 – Flow and Trace Analytics

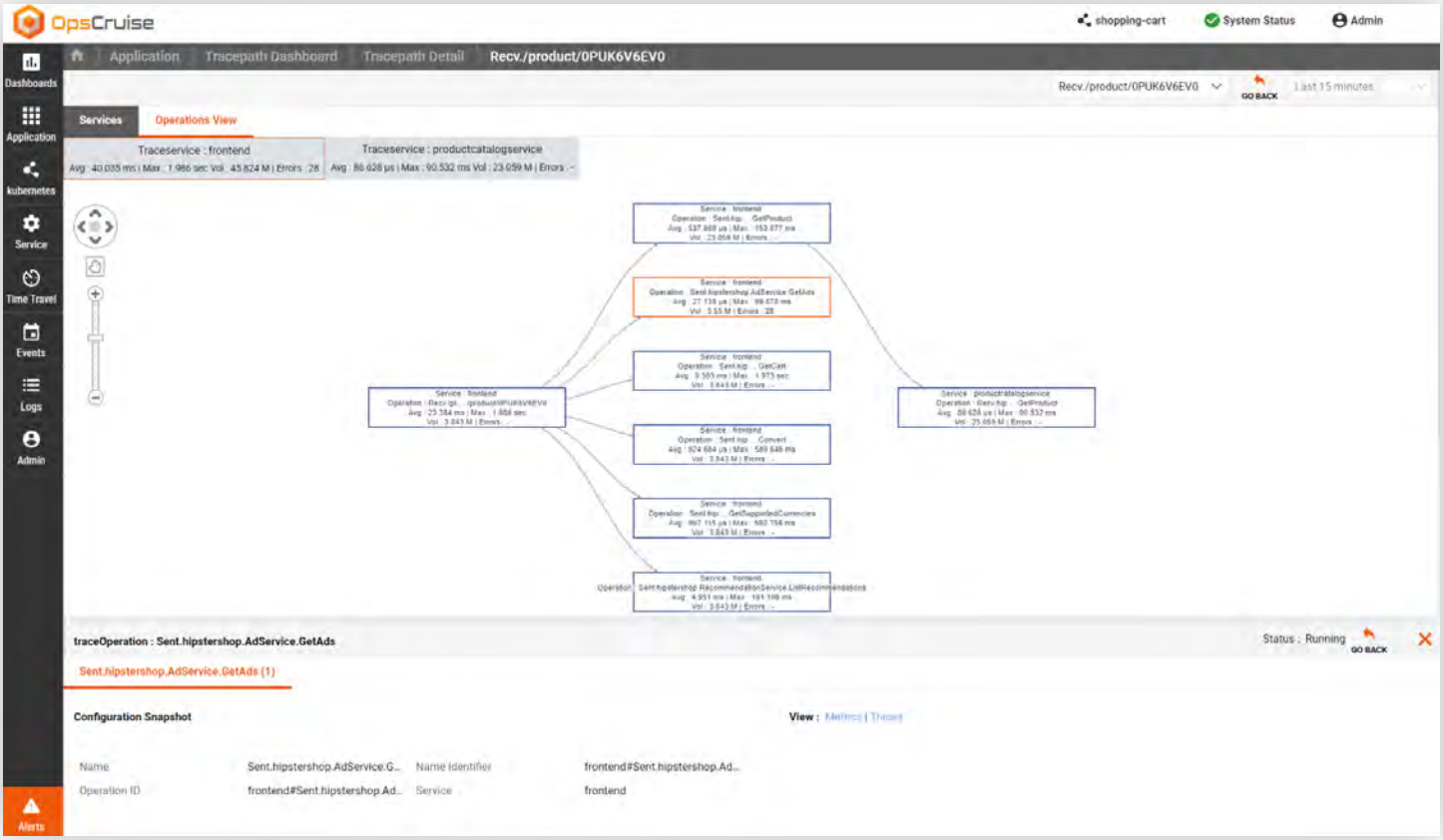
There are two approaches that OpsCruise uses for discovering the interactions and dependencies between components of the environment.

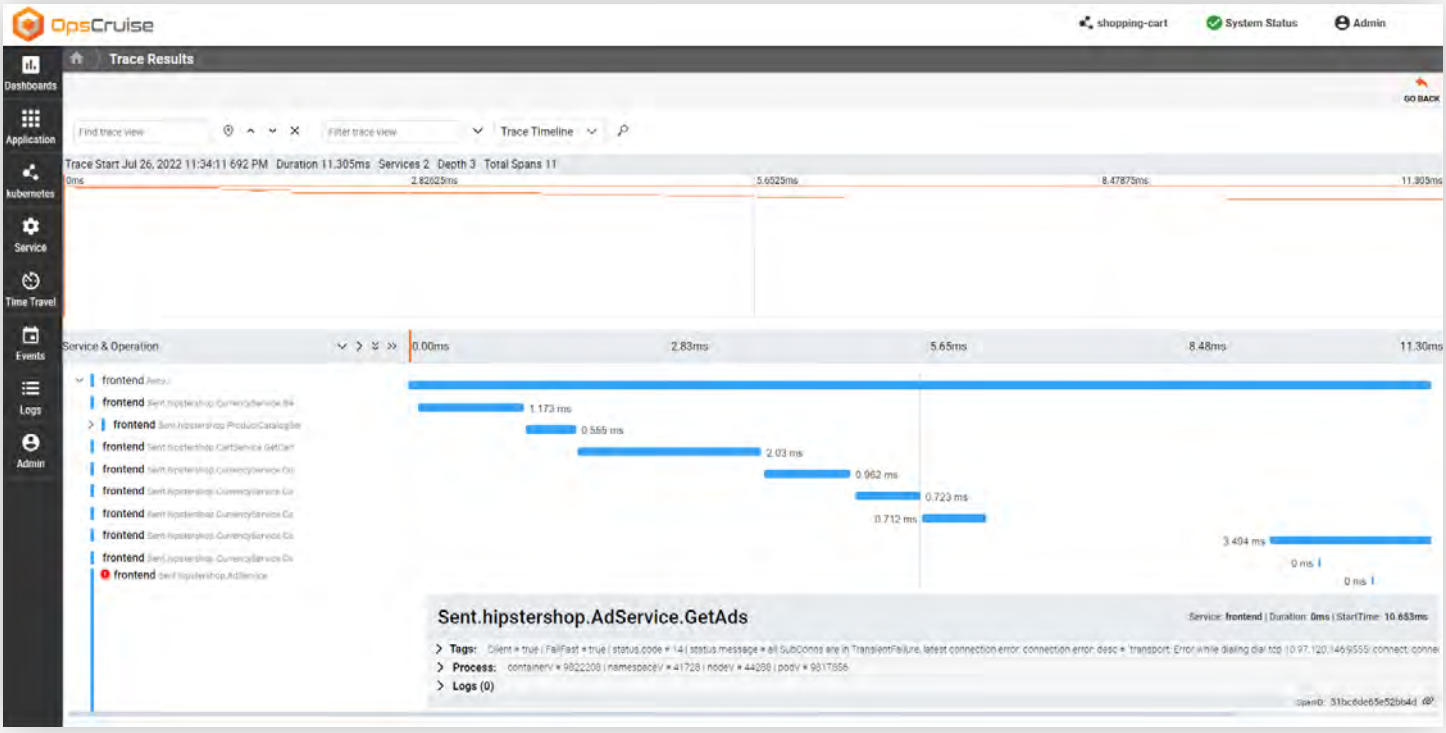
Flow Analytics: when code instrumentation is not available or possible

- eBPF, an Operating System technology, is used to discover the connections and the traffic between elements of the application.
- These network connections are mapped to and wired up to generate the App Map, as well as provide the golden signals at an aggregate level between services without any change to the application code.
- Within a few minutes of installation, the graph of all connected components is created and maintained in real-time. Examples of connections captured are those between containers, between load balancer and a container, or between a container and a cloud service such as AWS RDS.

Distributed Tracing: when application is instrumented for tracing

- When distributed tracing is enabled, it provides explicit information on every transaction. Capturing and persisting such detailed transactions requires computation, network and storage resources, so use of distributed tracing data is used for off-line debugging of application problems.
- OpsCruise's unique feature that discovers common shared routes, called TracePaths, are extracted from all collected traces.
- TracePaths enable detecting performance issues in real-time at an aggregate level, and then allow Ops to drill down to the problem traces and transactions and to the services or operations that are the source of the issue. This makes traces truly usable by Operations.





Using TracePath to detect and drilldown to specific problem trace and service

Stage 4 – Behavioral Profiling and Anomaly Detection

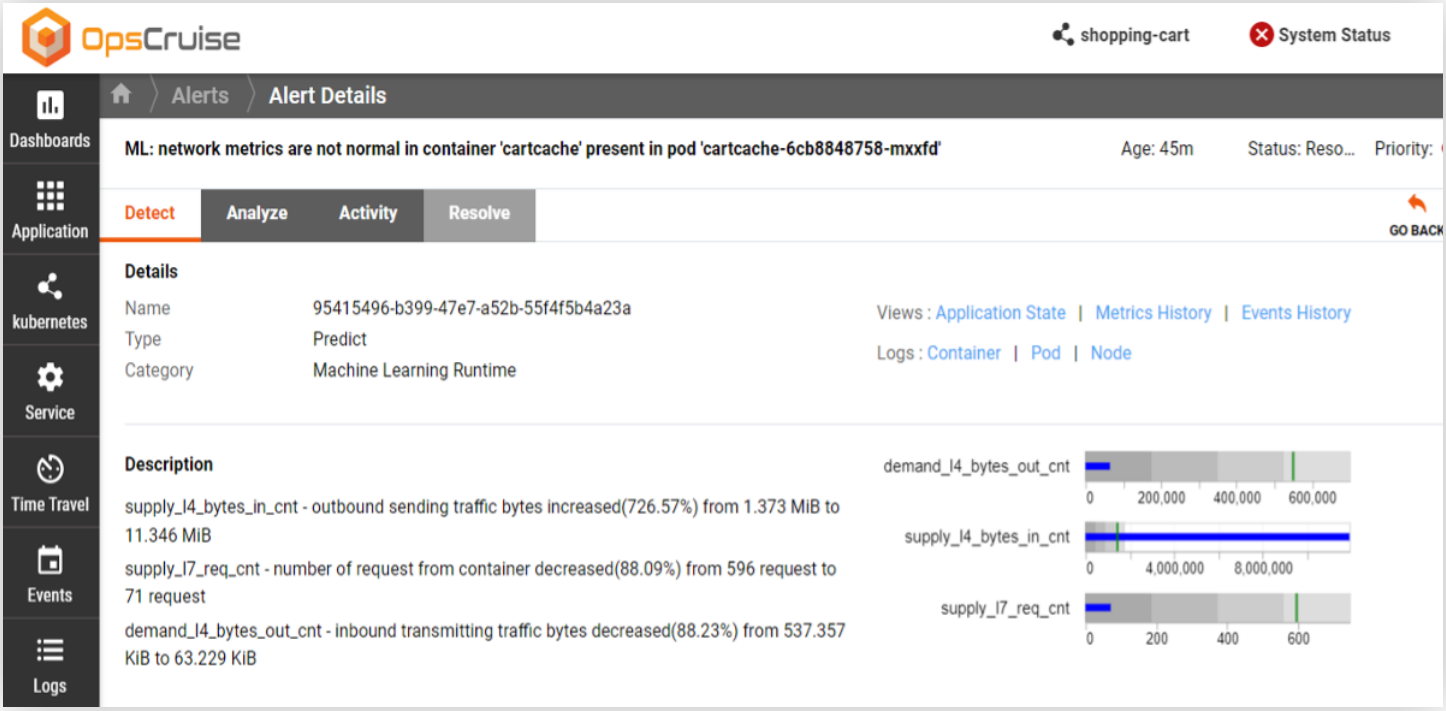
With the increase in the number of microservices and their instances, threshold settings have reached the end of the road. Rather than using single metric thresholds, a more holistic approach is needed.

OpsCruise ML continuously learns the behavior of each component in the application estate. Each deployment, DaemonSet, database and cloud service has a unique behavior that is learned using curated model templates and a machine learning algorithm ensemble.

The behavior of the component is compared to the predicted behavior and when there is a difference an alert is created.

This model based approach anomaly detection improves continuously without user involvement reducing false positives and negatives[3]. In field tests on production data, we have shown that this approach reduced false alert noise more than 50% compared to state of the art threshold based approaches over a 6-day period.

Moreover, the ML model provides explanations that are leveraged in automated RCA which helps Ops better understand the sources of the problem.



ML-based problem detection and explanations without need for threshold settings

[3] Rethinking Anomaly Detection

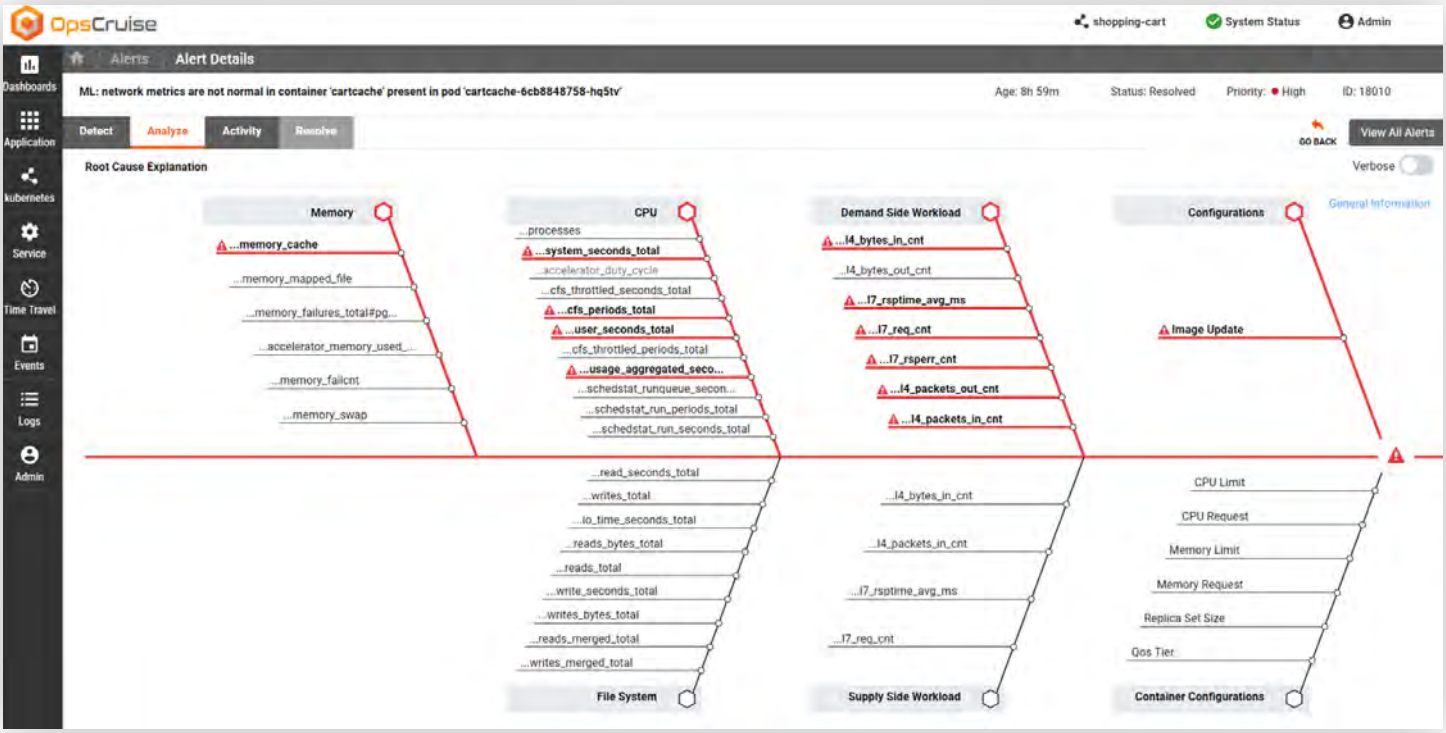
Stage 5 – Automated Causal Analysis

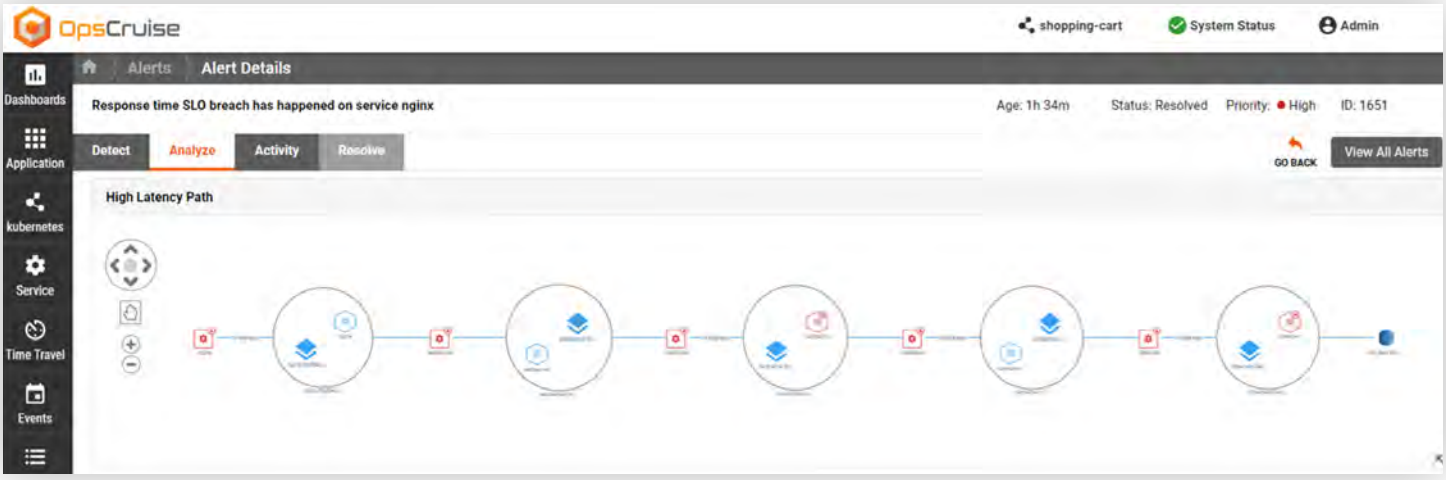
A key differentiator of OpsCruise in reducing toil as well as MTTR is automated root cause analysis (RCA)[4] which is triggered on any problem detection whether ML based anomalies or failure events.

The goal of the automated RCA is to conduct an AI-based diagnostics process which builds on all relevant information to the type of problem. Employing a dynamic decision tree process that uses curated knowledge ('SME in a box') for problems in application to infrastructure, it leverages both information of the current problem, configuration, events, as well as learned dependencies and explanations from the behavior model.

By querying conditions relevant to the type problem, it can isolate the causal domain(s), and surface the relevant information to Ops so that they focus on the application object(s) that are the likely cause of the problem.

Causal analysis for known problem classes related to Kubernetes, infrastructure or known application components are being continually added to the knowledge base increasing the effectiveness of RCA over time.





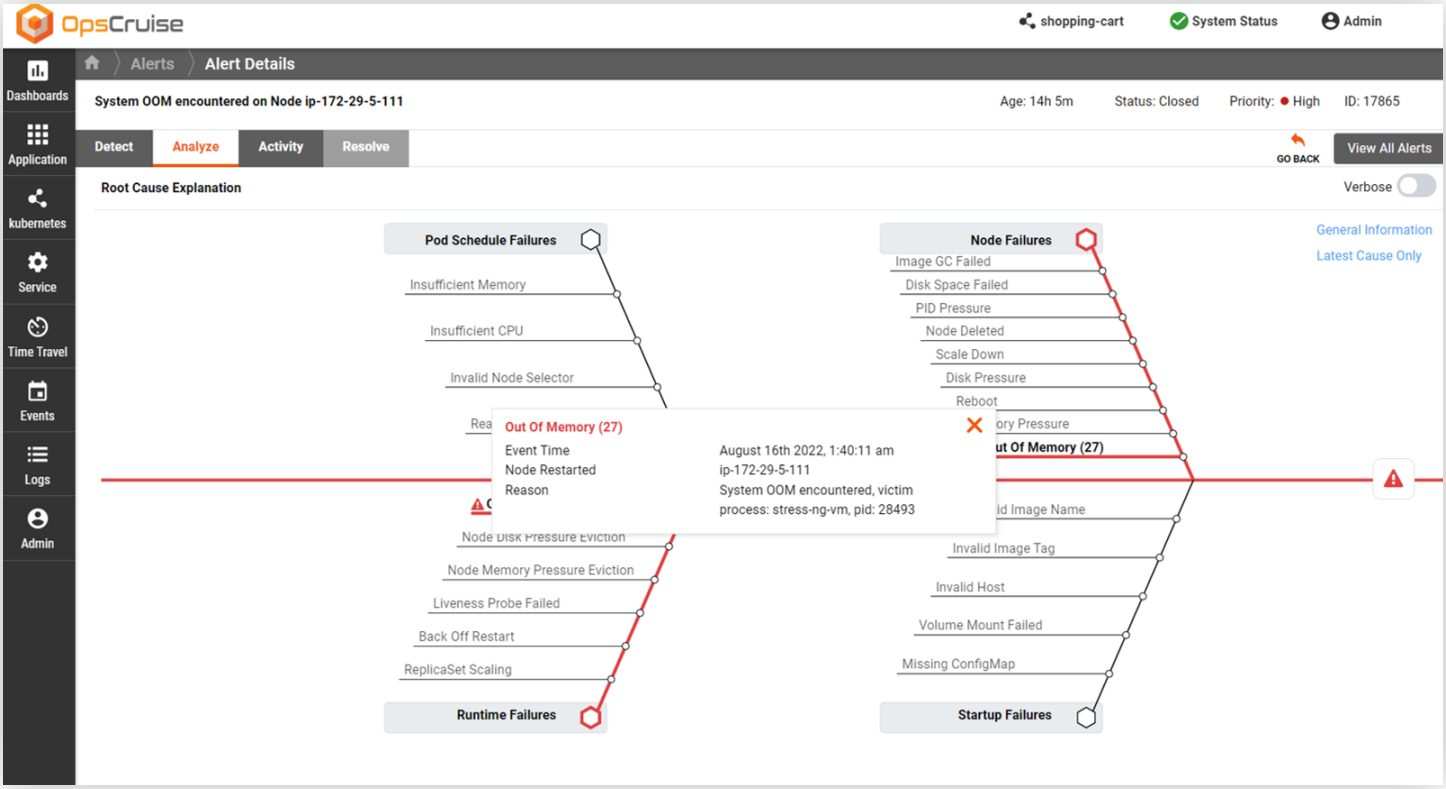
[4] Of Causality and Reasoning . . . OpsCruise's Automated Root Cause Analysis

Stage 6 – Recommended Action

Once OpsCruise isolates the problem source after it runs its automated causal analysis, a remediation action can be initiated. The level of automation that can be achieved depends on the nature of the problems, the operating environment, and the organization processes in place.

At the minimum for known repeatable problems, automated remediation actions can be initiated into runbooks already in place. While standards in automated remediation have not matured, OpsCruise believes in building on the 'Operator' concept[5].

OpsCruise envisions that as these remediation action frameworks mature, 'uber' operators will be implemented to make changes across the Kubernetes estate to automatically resolve problems. As an example, today OpsCruise can detect problems during failed deployment, and identify the cause being not having available replicas when there is limited memory in a node (see figure below). A Kube operator can be automatically triggered to increase the memory in the node to fix this problem.



Actionable causal analysis: increasing node resource to enable scale-out for a container that fails deployment

OpsCruise in Action

The following pages show some example problems, how they are suboptimally addressed in legacy solutions, and how they are solved in OpsCruise.

[5] Kubernetes operator

Use-Case 1: Response Time Violation

SCENARIO

Checkout Response Time (SLO) Violating by 250%

OPSCRUISE APPROACH

- Automatically find problematic service paths
- Analyze and correlate contributing anomalies
- Highlight image change as root cause

LEGACY APPROACH

- Isolated RT SLO violation alert
- Will not surface image change as root cause
- Users have to navigate screens and tools and mentally hold context

WHY IT MATTERS

- Reduce dependence on experts, tribal knowledge
- Reduced crisis time in war-rooms
- No code instrumentation required (w/ OpsCruise)
- Information correlation (OpsCruise's SmartWeave)

Use-Case 2: Bad Image Name

SCENARIO

Service Unavailable

OPSCRUISE APPROACH

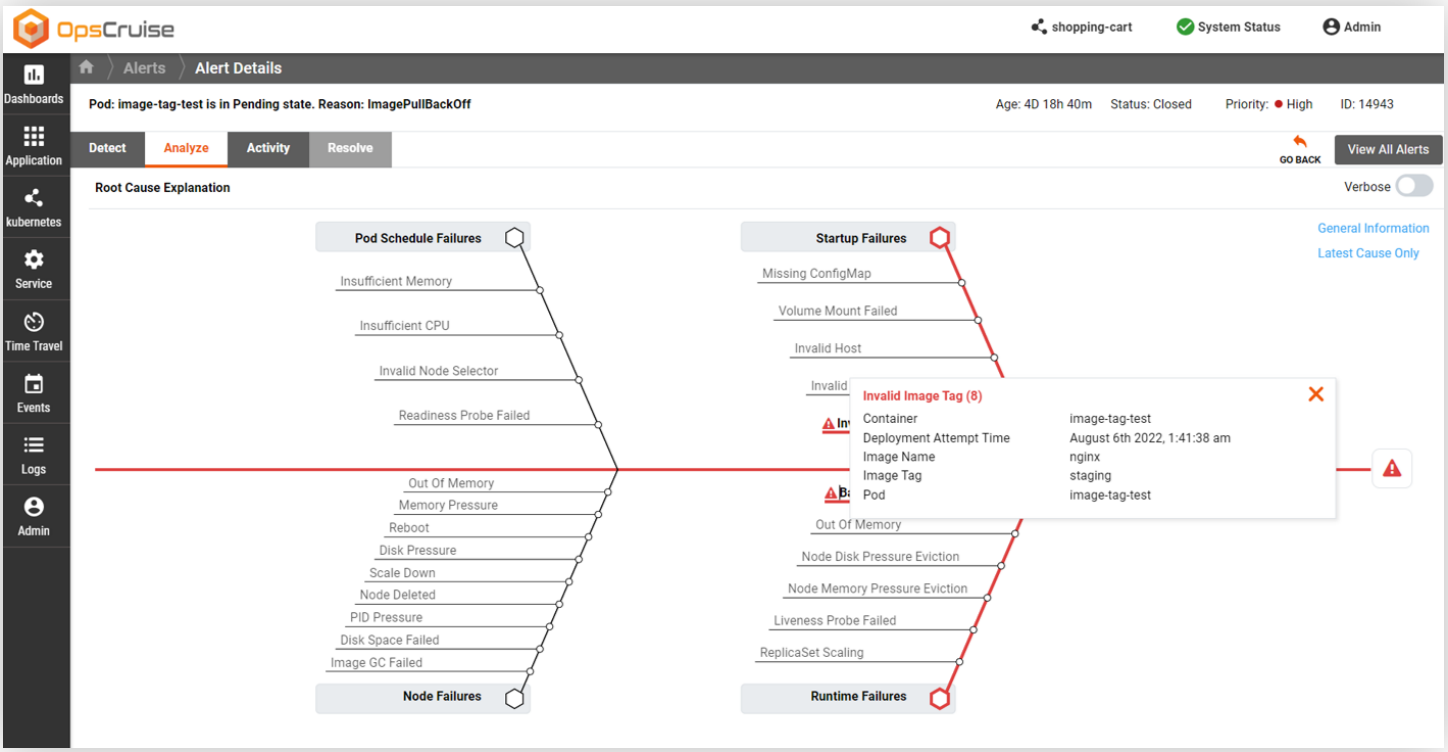
- Correlate deployment state with K8s events
- **Highlight ImagePullBackFailure due to Bad Image Name**
- Immediate Root Cause

LEGACY APPROACH

- Isolated Event showing failed deployment
- No correlation

WHY IT MATTERS

- Reduced outage time and MTTR
- Reduce reliance on senior SMEs



Use-Case 3: Node Out of Memory (OOM)

SCENARIO

Service Unavailable

OPSCRUISE APPROACH

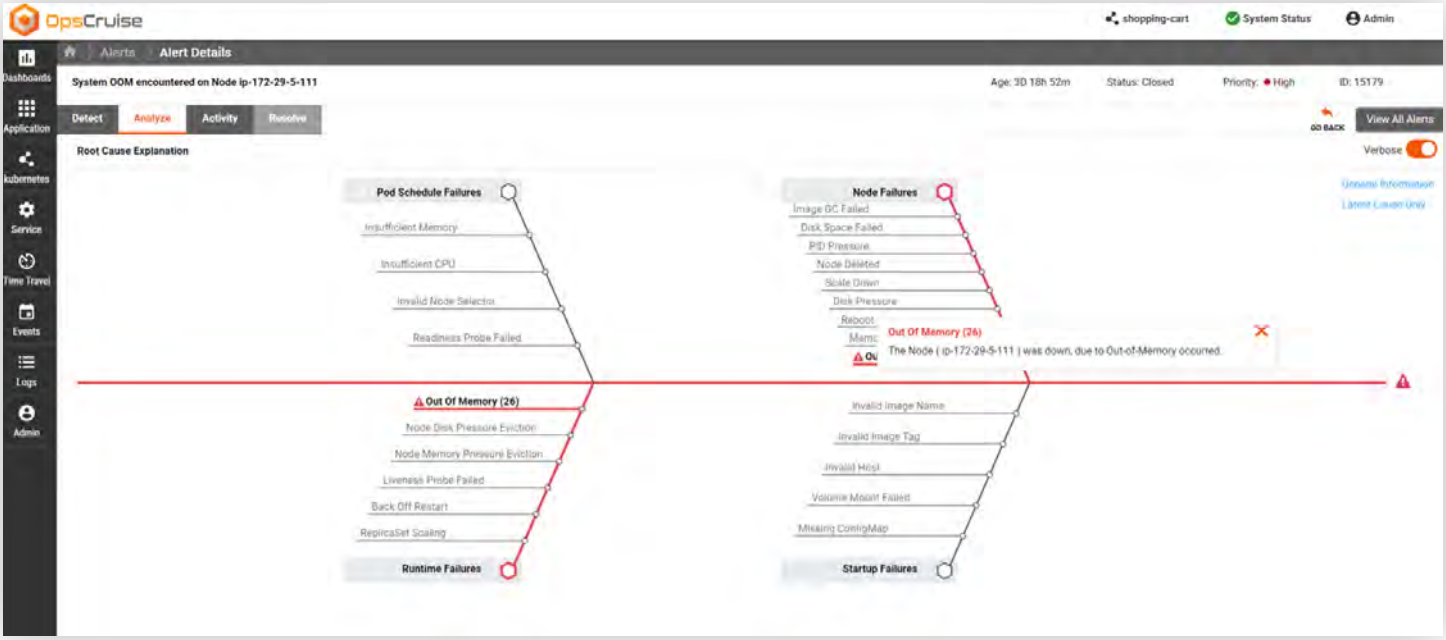
- Correlate deployment state with K8s events
- **Highlight Node is Out of Memory**
- Immediate Root Cause

LEGACY APPROACH

- Isolated Event showing failed deployment
- No correlation

WHY IT MATTERS

- Reduced outage time and MTTR
- Reduce reliance on senior SMEs



The Result: More Productive SRE/DevOps teams, Higher Availability, Lower Cost

Over the past couple of years with data and results from the field, OpsCruise has proven it can make significant improvements around the observability of cloud applications with the following business outcomes:

Higher availability and better performing digital services.

The relationship between application performance and customer satisfaction, and, ultimately, revenue has been proven in many industries. OpsCruise is helping customers identify performance degradations before users notice and solving them more rapidly. That has a measurable impact on the business.

Up to ⅔ reduction in monitoring tool costs.

Organizations are spending a small fortune on legacy monitoring tools to perform basic functions and costs are compounded when the vendors store your logs, metrics, traces in their cloud. OpsCruise embeds the increasingly popular open source monitoring tools as part of its foundation allowing you to eliminate the legacy, proprietary, and costly tools for your modern environment.

50% improvement in SRE productivity.

OpsCruise can get organizations on the path to supporting more applications at higher release velocity with existing staff. Organizations have been able to improve their SRE / Dev ratios by as much as 50%.

40% more alerts handled by L1.

Because OpsCruise alerts are highly enriched and prescriptive, it means fewer need to be escalated to expensive L2/L3 resources for resolution.

20% fewer cloud resources.

A lack of performance understanding leads to over-sized instances and pod allocations in K8s. OpsCruise has enabled organizations to make adjustments using as much as 20% fewer resources without impacting performance or risk.

Don't Take Just our Word for It:
What customers and thought leaders are saying

"In complex microservice architectures it can often be hard for engineering teams to see the big picture; OpsCruise provides us an affordable, engaging, and approachable view that allows engineers to see how their microservices fit into the entire stack. Most importantly, because OpsCruise is building on our existing open source monitoring stack, deploying it was simple and required no changes to our running services."

– Matt Surabian, Director, DevOps Engineering, Avis Budget Group (ZipCar)

"OpsCruise is a promising and novel approach for integrating information from both CNCF telemetry/monitoring projects and Kubernetes to help visualize the environment and automate production troubleshooting."

– Julius Volz, Creator of Prometheus.io, Infrastructure Engineer

"OpsCruise is cool because it natively uses ML profiling & open telemetry to rapidly identify performance issues in modern applications."

– Cool Vendors in Observability and Monitoring for Logging and Containers, Published 27 April 2022

About OpsCruise

Founded in 2019, OpsCruise is the leading open observability platform for modern Kubernetes-centric applications and infrastructure. Founded by AI/ML and infrastructure software veterans and backed by leading VCs and industry luminaries, OpsCruise is used by some of the most demanding SREs and DevOps Engineers at leading Global 2000 organizations and SaaS start-ups.