White paper

virtana

Intelligent Observability for Modern Applications

Real-Time, Context-Aware Insights Without the Bloat

Prologue

An explosive increase in the use of microservice applications and Kubernetes platforms continues year over year. A re-emerging growth in Cloud/Datacenter-hybrid architectures and the continued adoption of open-source monitoring tools like OpenTelemetry (OTel), are redefining the Application, Infrastructure, and Observability landscapes. Having built Virtana's Container Observability Platform for Kubernetes applications running in the cloud, data centers, and beyond, we've decided to share our container observability "manifesto." We've built our platform by leveraging experience and lessons from the field as the adoption of Virtana's Container Observability continues to grow. Virtana acts as a Smart Layer that enhances the detection, causal, and analytical capabilities of the open-source telemetry layer.

Modern Cloud Applications are Fundamentally Different

It is well-established that the adoption of cloud computing is continuing at a significant pace. In a recent press release, a Gartner analyst said, "Cloud use cases continue to expand with increasing focus on distributed, hybrid, cloud-native, and multicloud environments supported by a cross-cloud framework, making the public cloud services market achieve a 21.5% growth in 2025." This rapid growth underscores the need for deeper visibility into increasingly complex, containerized environments that span multiple cloud platforms.

Driven by the need for scalable, agile and rapid deployements, most new applications are cloud native and built as microservices. According to the Cloud Native Computing Foundation (CNCF), approximately 90% of all organizations report using containers in production environments.

Interestingly, while cloud-native architectures fundamentally changed how applications are written and deployed, the challenges in the observability of modern applications mirror 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 teams are 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 continue to increase.

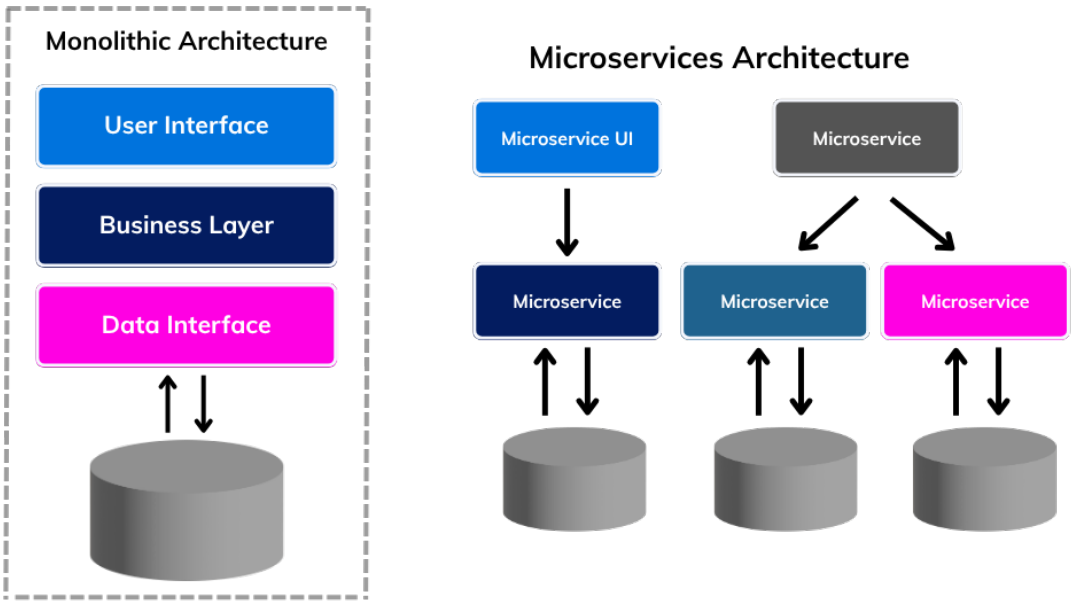

Figure #1: Microservices versus monolithic application architecture (ref: BMC)

Why Are Modern Applications More Challenging to Troubleshoot?

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

- Complexity Caused by the Sheer Scale and Multi-tiered Nature of Orchestrated Applications: obfuscation is present 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 is built on top of a cloud or on-premise infrastructure layer. There are multiple points of performance loss: an incident in the application, in the container, in the orchestration layer, or the infrastructure services can have an impact on multiple services that are masked by these layers of obfuscation.
- Dependencies Spread Across Services and Down to Infrastructure: 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 distributed nature of modern applications means that they could be operating in a partially degraded mode with only certain, hard-to-detect components failing. Third-party SaaS and 3rd party APIs add to the complexity, heterogeneity, and difficulty in pinpointing failures of services.
- Dynamism: Applications are ephemeral, with frequent code pushes and daily releases constantly changing topologies and service structures. These shifts, combined with transient workloads, introduce challenges in understanding where issues occur and when they occur.

Existing Approaches Don't Address Today's Challenges

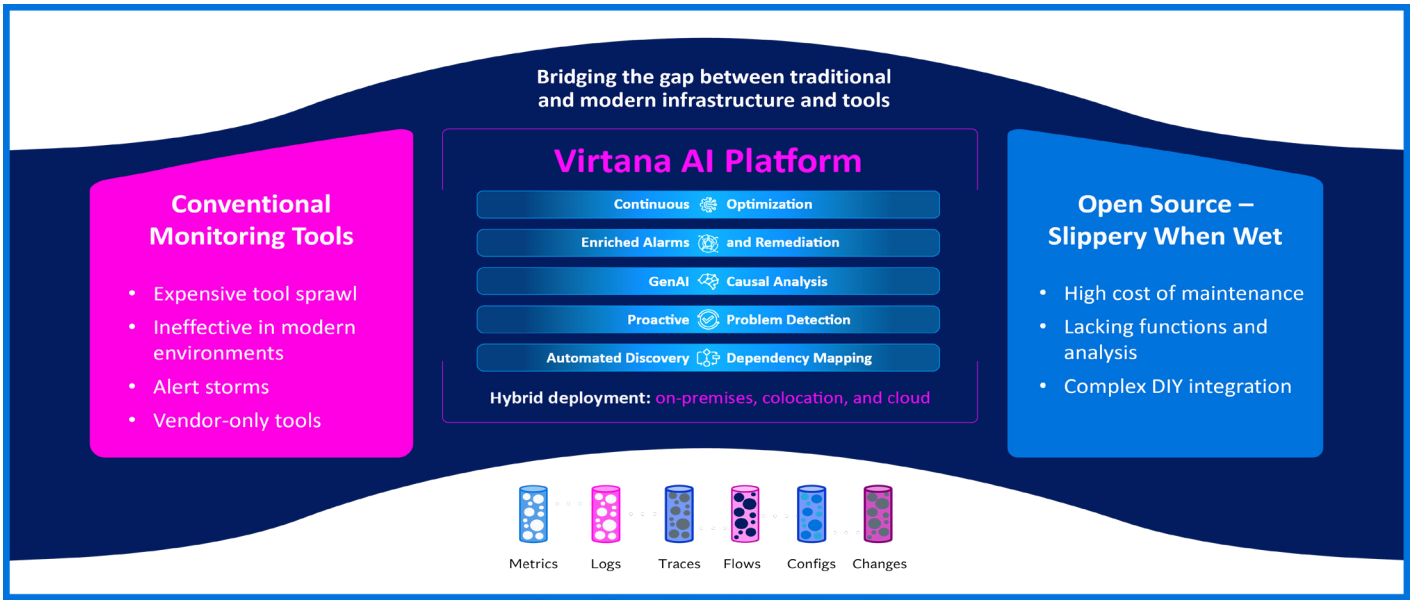

Figure #2: Organizations get lost on one or both of the illustrated roads above

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 poor insight into the application state, resulting in mostly manual processes that reduce the time to resolve problems. Further, the vendors providing this visibility rely on proprietary "agents" to be deployed on every host and often insist that all the raw telemetry is stored in their own backend. That's an expensive and unnecessary model for these newer environments.

In stark contrast to proprietary agents, the Observability world has been increasingly adopting the ecosystem of the Cloud Native Computing Foundation (CNCF) observability projects, including OpenTelemetry, Prometheus, Loki, and FluentD, alongside other modern, open-source projects.

While these increasingly popular projects and tools are quickly becoming the data collection standard, they are only a starting point. Just having telemetry data is not enough. We need to do something with that data. With a rich data set as a starting point, you can perform telemetry unification, topology and relationship understanding, anomaly detection, and problem remediation. Unfortunately, we've found that when using other platforms, more data usually means more dashboards, more dashboards, and more dashboards, adding to the "cognitive overload" of already overloaded engineers. This forces those same engineers to switch between different tools and manually run through visual queries, looking for patterns that may not be applicable, or looking for a problem they know is there but are not sure how to find.

The conventional wisdom is to implement the "three pillars of observability," comprised of logs, metrics, and traces. While this is a good start, metrics, logs, and traces are no longer enough to detect and resolve problem scenarios. When you have just this data, especially if it's disjointed, it is difficult to remediate issues while switching between telemetry context while and keeping a mental model of the problem and affected entities. This means that Mean Time to Resolution (MTTR) is negatively affected, translating to longer, unresolved outages.

Given the ephemeral nature of modern environments brought about by the use of advanced virtualization platforms and advanced container platforms like Kubernetes, a new challenge is the now ever-changing application structure. We now need to check if newly added services or previously existing ones removed have caused a negative performance or behavioral change in the application. Also, the configuration of the underlying Kubernetes infrastructure, which itself is constantly changing, can often cause problems. Engineers in every organization need to be aware of the changing topology, dependencies, and configuration of the provisioned infrastructure!

A note on distributed tracing: while distributed tracing data is incredibly valuable in diagnosing performance problems, heavy or exclusive reliance on this telemetry type forces the Dev and Ops teams to operate primarily in a reactionary, post-facto mode. What is needed is an integrated approach to telemetry and problem detection, where problems are detected in real time using not only traces, but metrics, logs, events, configuration changes, and relationship data. This data-complete approach is more capable of showing what is happening within and across the application without adding dashboards for every new metric an organization enables.

While, again, this is a good start, metrics, logs, and traces are no longer enough to detect and resolve problem scenarios.

Engineers in every organization need to be aware of the changing topology, dependencies, and configuration of the provisioned infrastructure!

Organizations need intelligence from their observability tools that can automatically provide the following:

1. Real-time visibility into the ever-morphing application and its dependencies
2. Detection of problems without guessing and manually tuning thresholds and
3. A reduction in the manual work to find a root cause

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

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

Figure #3: Open-source observability projects (source: https://landscape.cncf.io/)

The CNCF infographic above illustrates how the platforms and tools used in the observability landscape have evolved. It highlights how all types of telemetry used for monitoring are now freely available. With extensive, full-fledged support and ongoing development of these tools by the CNCF community, these telemetry tools also provide a vendor-independent, future-proof observability framework to start the journey.

With that data in hand, what we then need is contextual integration and analyses to provide visibility into the health, state, and performance of the application and infrastructure.

Building Real-time actionable Intelligence on Top of Open-Source Data

Virtana Container Observability is built on open source and cloud provider monitoring, as shown in the figure below.

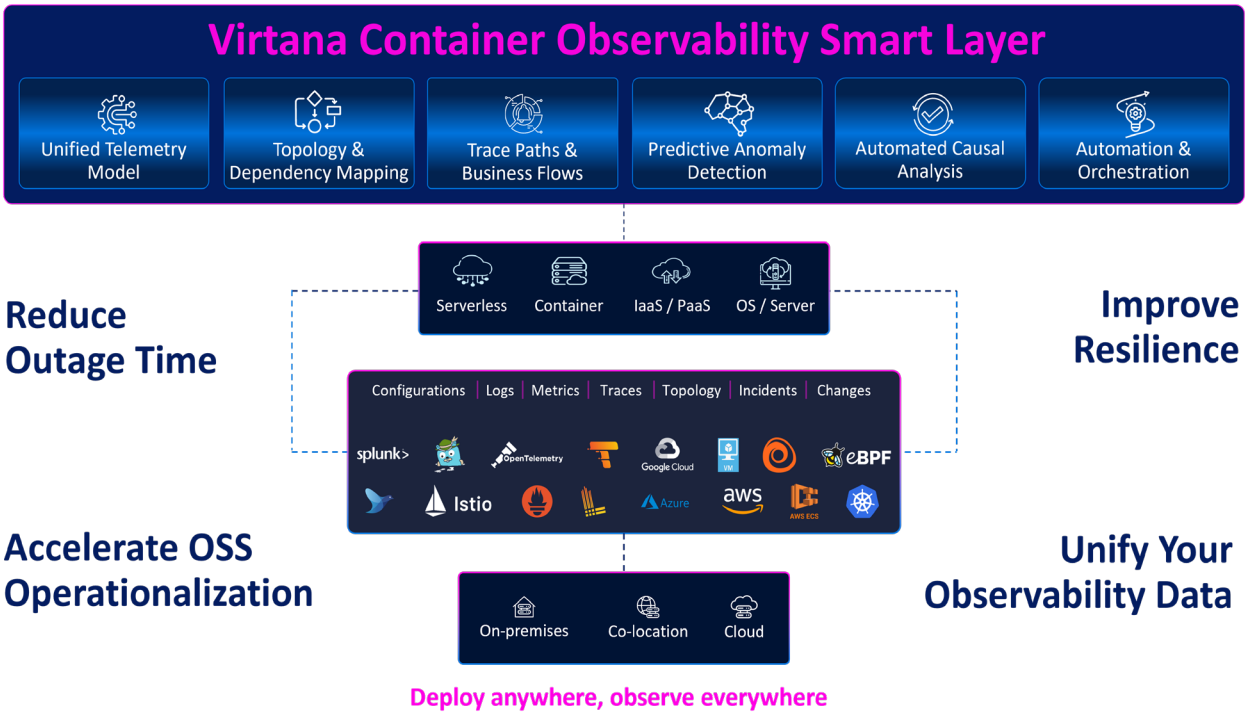

Figure #4: Virtana Container Observability's Smart Layer provides actionable insights into cloud and Kubernetes infrastructure and workloads

With Virtana Container Observability, there is no longer a need to deploy proprietary vendor agents or proprietary code instrumentation. Virtana not only provides a future-proof observability platform but also several other advantages, including:

- The ability to observe applications running anywhere, be it public cloud, on-premises, or a hybrid of both, and still be observable with a single platform
- No need for proprietary agents, as Virtana supports OTel and other open-source tools
- Data collection is lightweight and simple to deploy
- Monitored data stays in your control as the application owner and can be used for other purposes beyond observability, such as capacity planning and scaling decisions
- 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
- Immediate, out-of-the-box operationalization of telemetry data, meaning time to value is measured in minutes, not days or weeks
- Flexible deployment options, including SaaS, Self-Managed, and fully air-gapped

Virtana Container Observability's Smart Layer

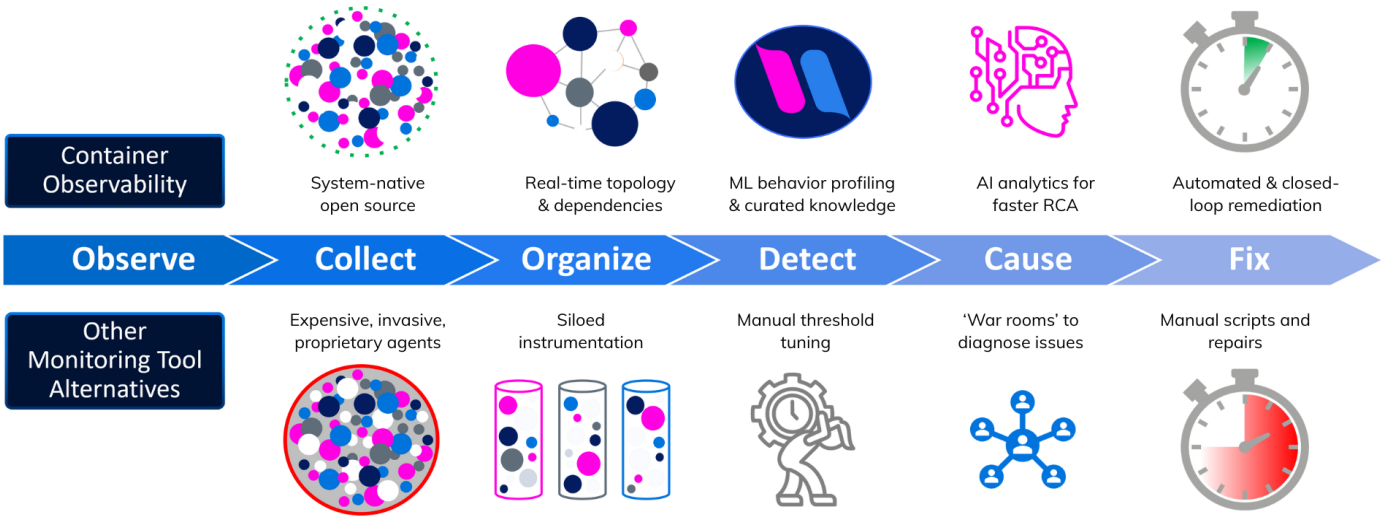

Figure #5: Virtana's Smart Layer processes data for correlation, enrichment, dependency mapping, and problem detection alongside root cause and remediation

With access to all the open-source and OTel telemetry data, Virtana Container Observability can now work on embedding, analyzing, and unifying, i.e., contextually "stitching together" all that telemetry to extract and provide deep insights. With Virtana, you can:

- Auto-discover all full-stack dependencies - between services and from services down to Kubernetes and the underlying infrastructure
- 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 eBPF Flow and Trace Analytics
- Deliver predictive anomaly detection without requiring manual Dev or Ops intervention
- Automate causal analysis to isolate problem impact domain and root cause

To improve the efficacy of analysis in each stage, Virtana embeds and uses AI, ML, and curated knowledge against the observed technology stack, known applications, and IT diagnostics processes.

Here is a summary of the capabilities in each stage.

Stage 1 – Automated Discovery and Contextual Integration

Virtana's Container Observability 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 the representation of application services of diverse types, whether microservice, SaaS, serverless, or FaaS. Because services can be auto-scaled and work with application components that are ephemeral in nature, Virtana continuously discovers entities, so services that are added or removed, along with infrastructure that is added or removed, are always accounted for and kept up to date in the relationship and topology data.

This enables contextually associating all aspects of the topology of a workload, allowing users to:

- Quickly navigate to an object (e.g., a VM or a Kubernetes pod) and see all the associated metrics, logs, events, connections, and traces of that entity without needing to maintain mental topology mappings or maintain mental context
- Automatically see changes in the topology even as services are modified, whether through auto-scaling, service additions, deletions, or something like a container image change
- See 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

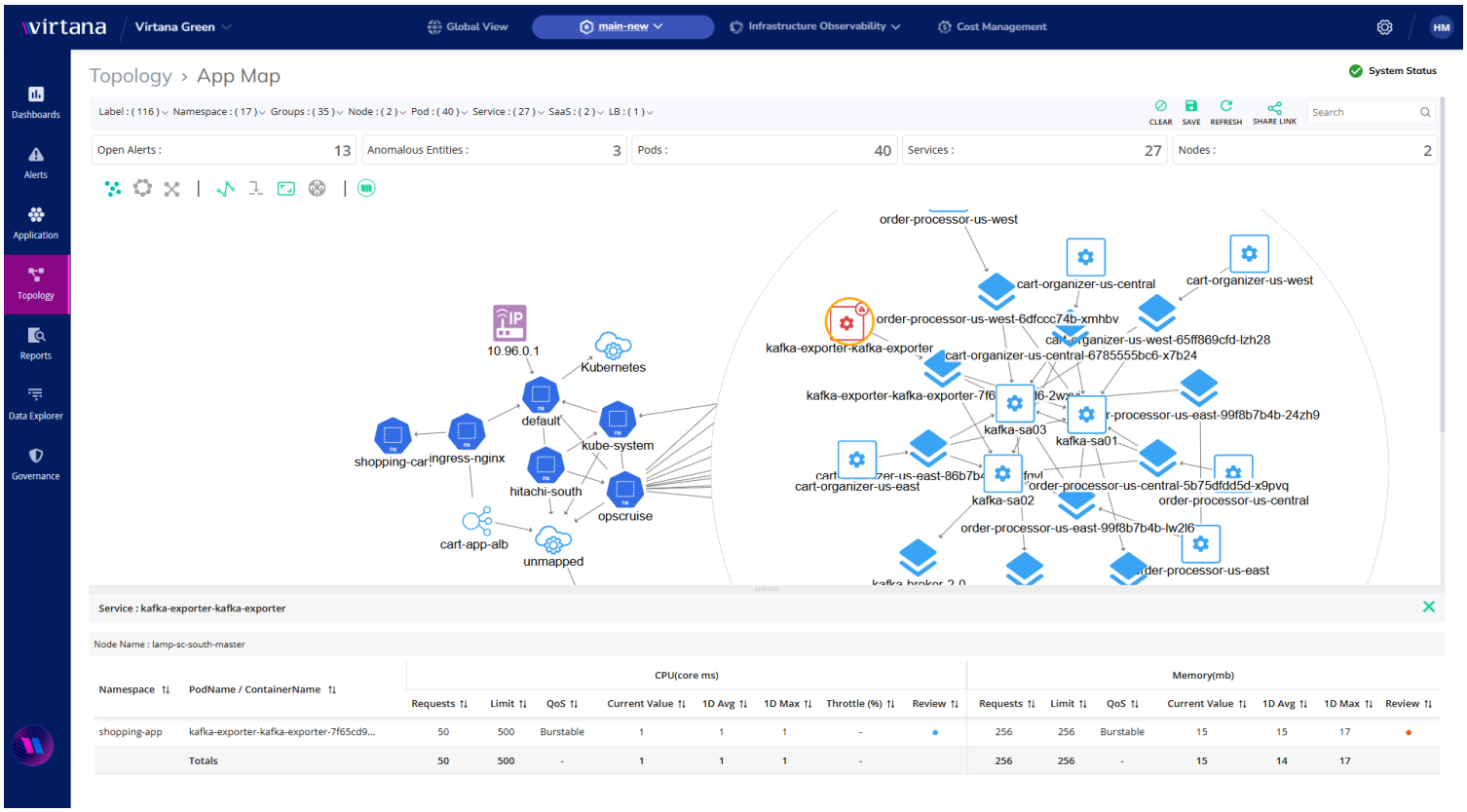

Figure #6: Automated contextual linking of metrics, logs, events, connections, and traces of an application container

Stage 2 – Application Topology and Dependencies

Once all telemetry is processed and contextually mapped, Virtana Container Observability is unique in creating:

- Application topology and all dependencies using eBPF network flow data (see Stage 3 Flow Analytics) without the need for enabling tracing or using proprietary agents (NOTE: we love and fully support distributed tracing in our platform, we just don't require it for toplogy!)
- Different views of the complete application structure, presented at the application level (Topology), Kubernetes node level (Node Map), VM and process level (Host Map), and trace level (Trace Map)
- Insights into 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, Virtana can be up and running in minutes.

> **There is no need to embed any proprietary agents into the application pods or containers, Virtana can be deployed and up and running in minutes.**

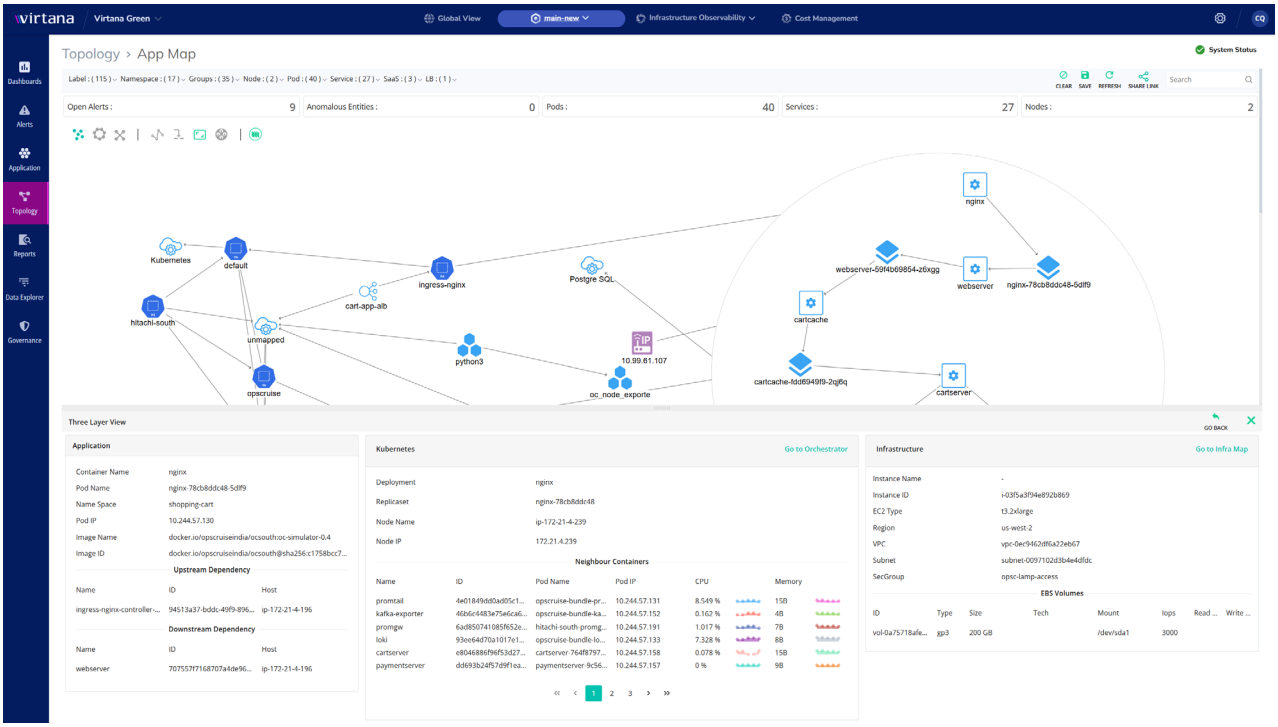

Figure #7: An application level view (App Map) showing dependency from container to infrastructure ('three-layer view')

Stage 3 – Flow and Trace Analytics

There are two approaches that Virtana Container Observability 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 and wired up to generate the topology and 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 a load balancer and a container, or between a container and a cloud service such as a cloud database.

Distributed Tracing: when application is instrumented for tracing

- When distributed tracing is enabled, it provides highly detailed information on every transaction, down to individual lines of code, with logs contextually linked to transactions.
- Engineers have access to a Virtana patented capability that discovers and differentiates transaction types based on common or divergent service pathways, called TracePaths.
- TracePaths enable the detection of performance issues in real time at an aggregate level. Teams can then further drill down to the problem transactions and the services/operations that are the source of the issue.

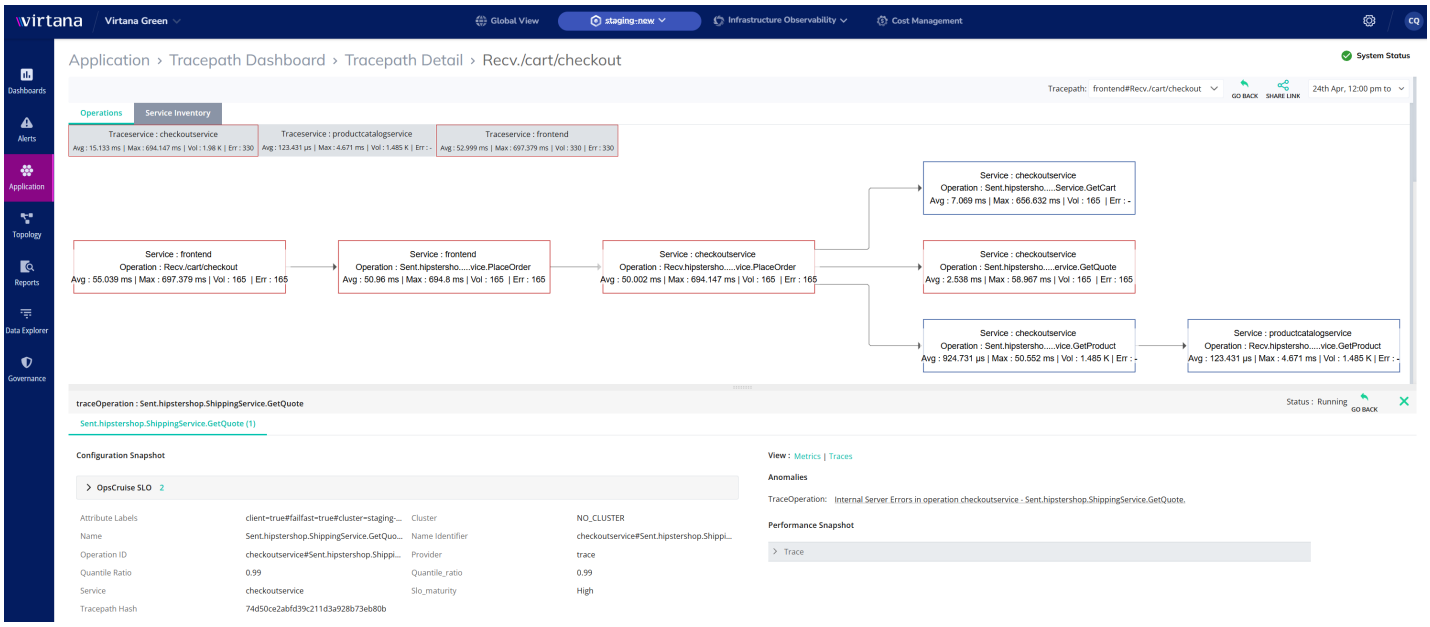

Figure #8: Transaction flows and contributing services discovered as part of a Virtana's Trace Paths

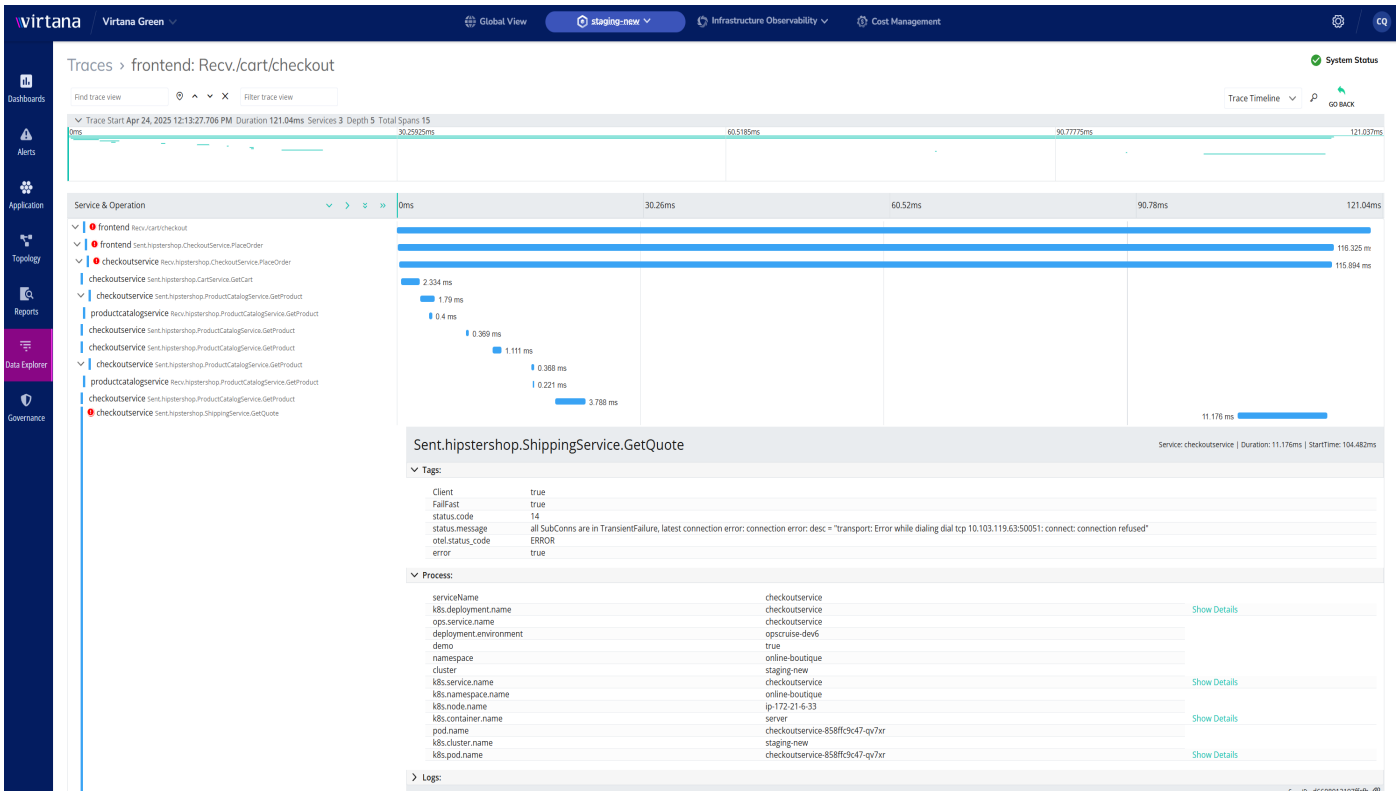

Figure #9: TCode-level breakdown of an individual request showing all services and operations with their contribution times and failures

Stage 4 – Behavioral Profiling and Anomaly Detection

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

Virtana Container Observability continuously learns the behavior of each component in the application estate. Each entity type (e.g., database, container, Kubernetes node, process, etc.) has a unique behavior that is learned using curated model templates and an algorithm ensemble.

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

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

Moreover, the problem detection engine provides explanations that are leveraged in automated Root Cause Analysis (RCA), which helps teams instantly understand the source of the problem.

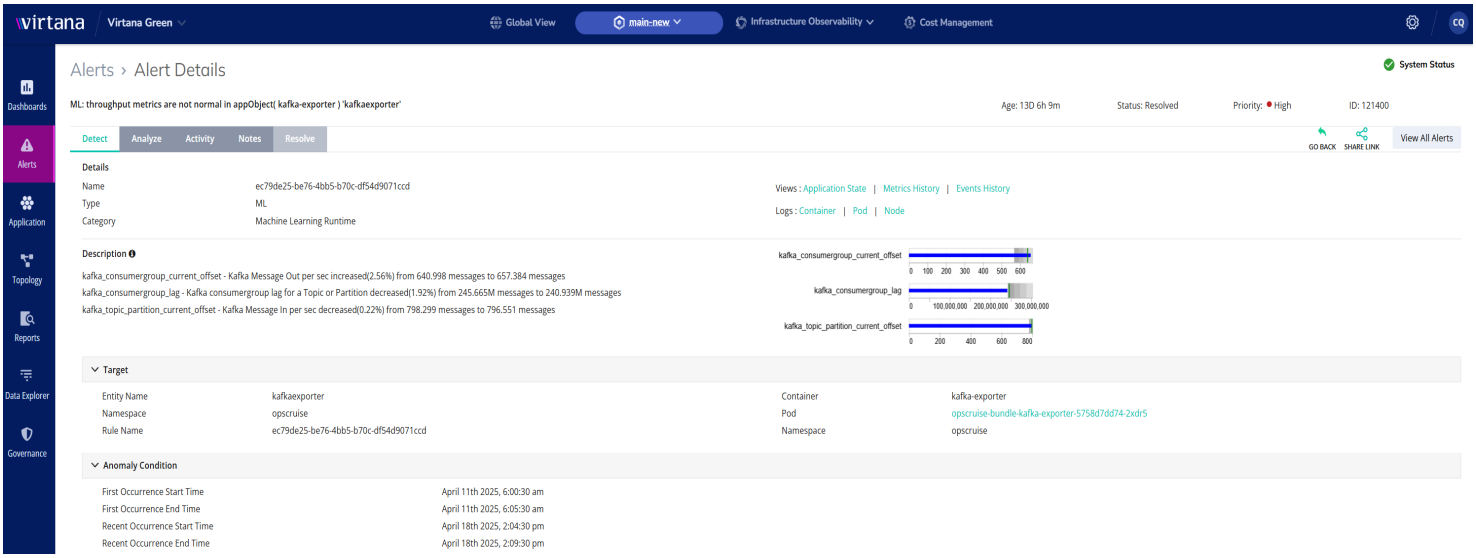

Figure #10: Automated, multi-variate problem detection showing multiple triggering causes for an alert

Stage 5 – Automated Causal Analysis

A key differentiator of Virtana Container Observability is reducing toil and MTTR via automated root cause analysis, which is triggered by problem detection.

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 issues in application and infrastructure, it leverages information on the current problem, configuration, and events, as well as learned dependencies and explanations from the behavior model.

By querying conditions relevant to the type of problem, the RCA engine can isolate the causal domain(s) and surface the relevant information to teams so that they focus only on the entities that are the likely cause of the problem.

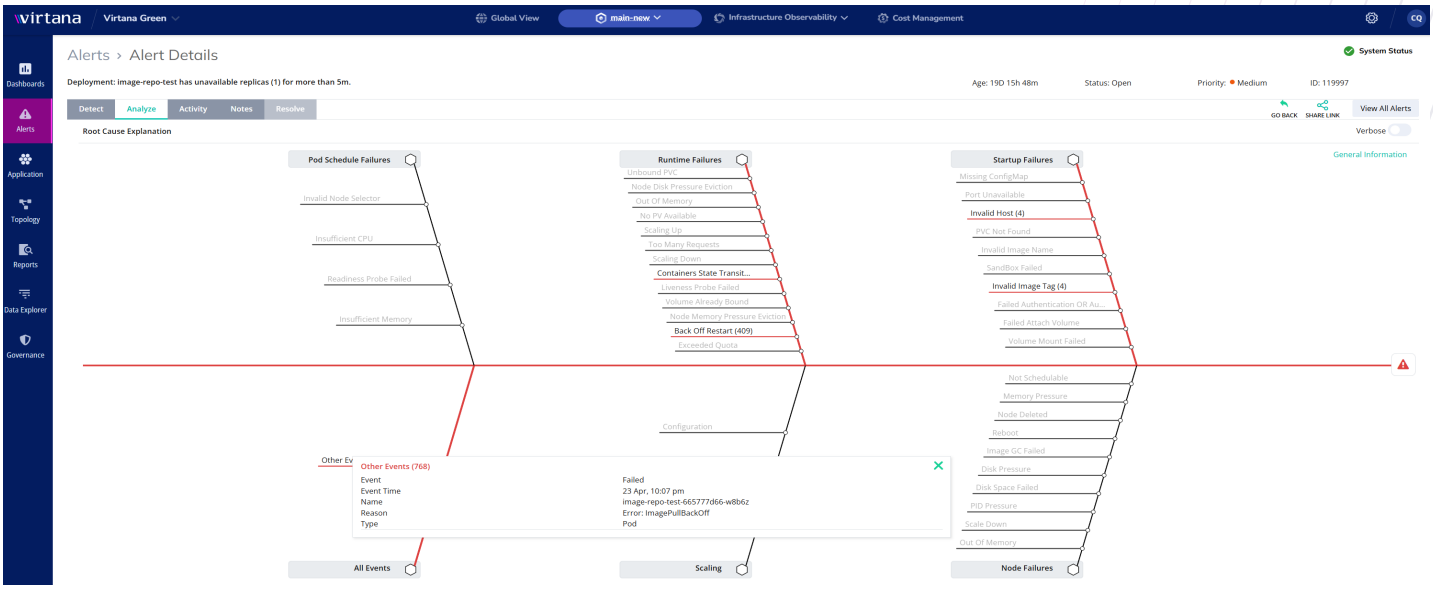

Figure #11: Virtana's Root Cause Analysis engine showing the root cause of a failure

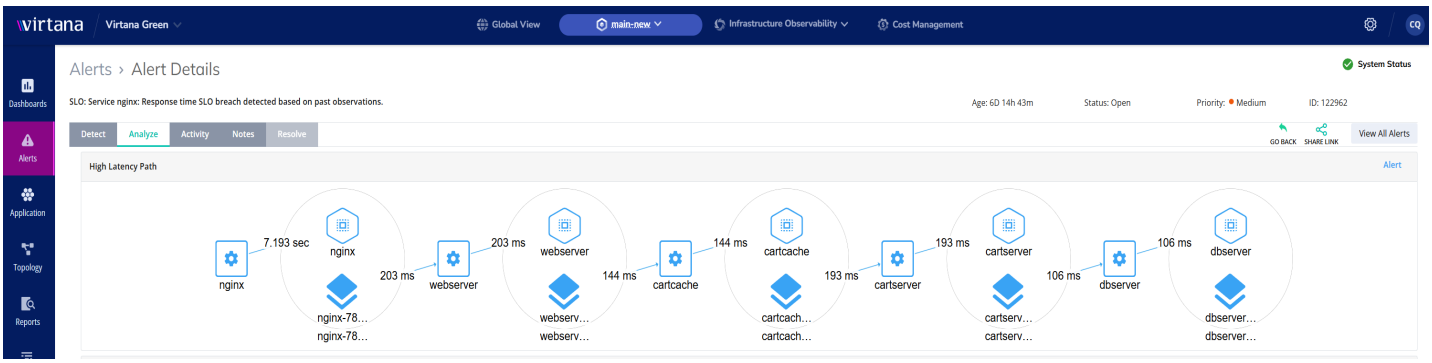

Figure #12: The dependency and relationship mapping helps identify problematic request paths

Stage 6 – Recommended Action

As environments grow larger and more heterogeneous than ever, taking the steps required to fix an issue, even after knowing how to fix it, still requires a large amount of mental load and manual effort. Organizations need to quickly get environments and applications back to a working state so as to not affect business outcomes, and accelerating this process using automated actions becomes a key part of reducing outage times and MTTR.

Inside Virtana, once an alert is triggered for a problem entity, a remediation action can be initiated against the involved entities. Remediation actions can be as simple as a notification to a particular endpoint, such as Slack or an email address, all the way to complex, multi-step remediation tasks involving customized scripts and conditional checks. The platform can execute in a fully autonomous fashion or require manual approval for each action to be executed so that all steps in the process can be carefully vetted.

For example, in the case of a container suddenly experiencing continuous memory spikes that in turn are causing OOM (Out of Memory) Kills, an automatic remediation action to increase the memory limits on the container's deployment can be triggered. The planned action can then be sent to the appropriate human operators to either approve or deny the change, followed by fully automated action if approved, all leveraging the Virtana Container Observability platform.

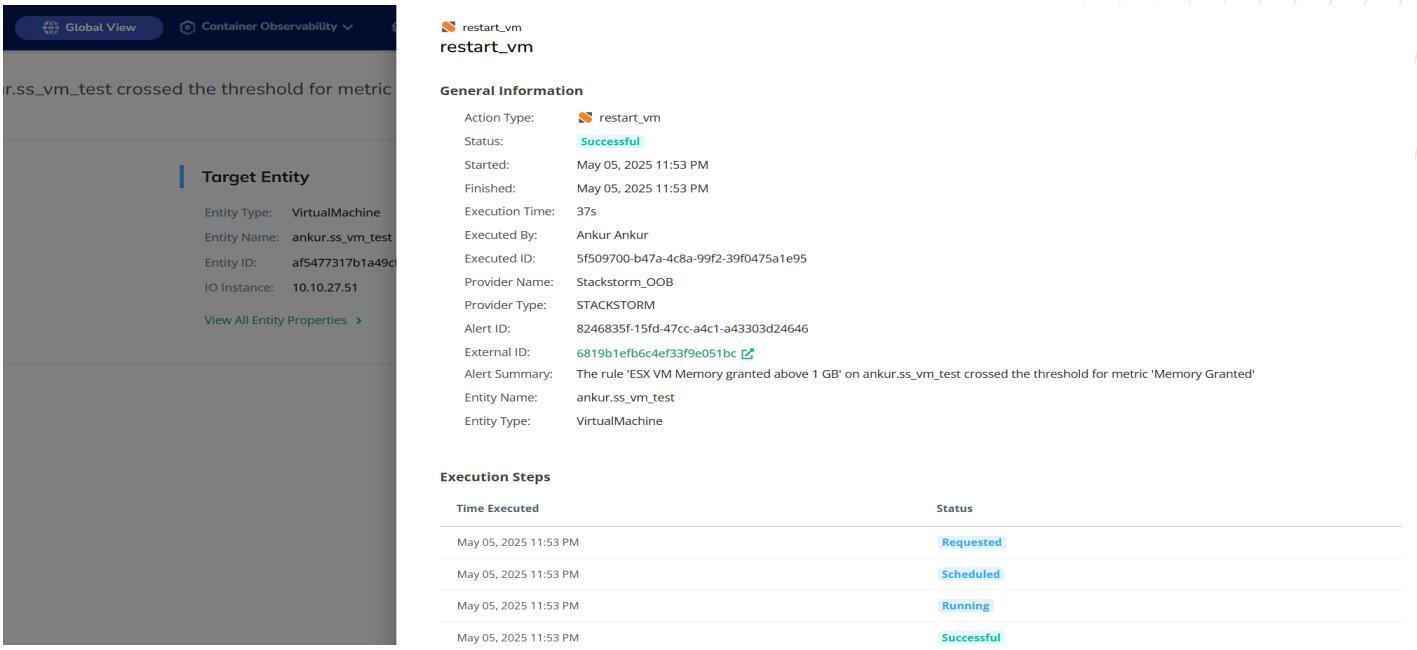

Figure #13: Restarting a VM with runaway memory utilization.

Virtana Container Observability in Action

The following pages show some example problems, how they are sub-optimally addressed in legacy solutions, and how they are solved in Virtana Container Observability.

Use-Case 1: Response Time Violation

SCENARIO

- Service Response Time (SLO) Violating by over 10x

VIRTANA CONTAINER OBSERVABILITY APPROACH

- Automatically find problematic service paths
- Analyze and correlate contributing anomalies

LEGACY APPROACH

- Isolated, manually created Response Time SLO violation alert
- Will not surface root cause
- Users must navigate screens and tools and mentally hold context

WHY THE VIRTANA APPROACH MATTERS

- Reduce dependence on experts, tribal knowledge
- Reduced MTTR and reliance on war-rooms
- Automatd problem and telemetry correlation

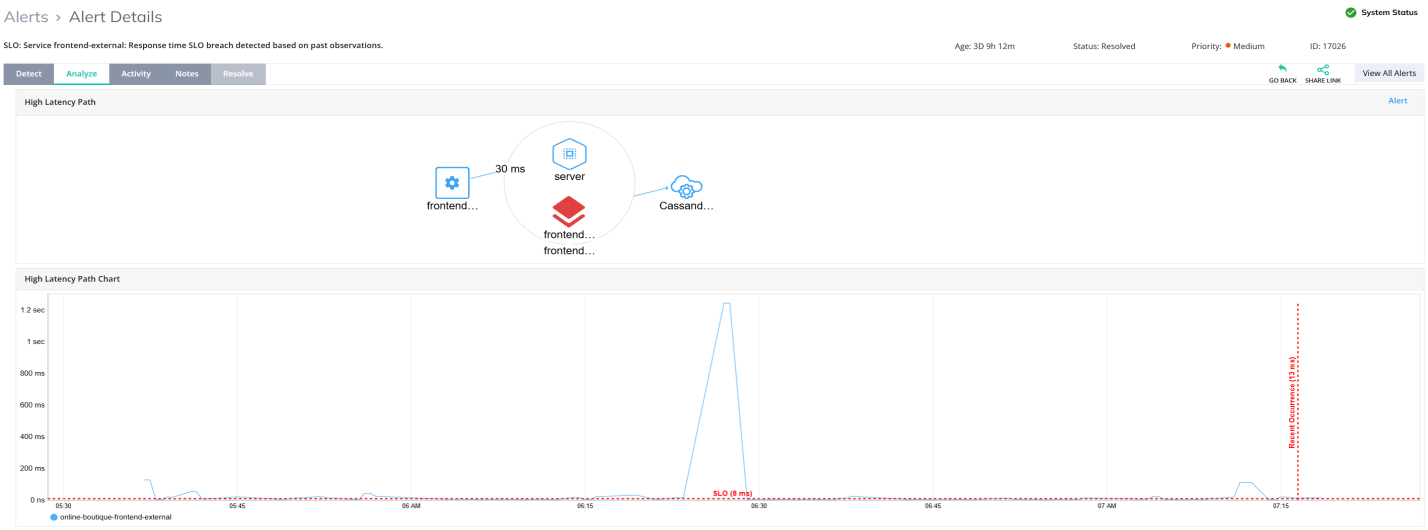

Figure #14: Automated detection of slow response times and identification of the offending service

Use-Case 2: Bad Image Name

SCENARIO

- Workload Not Starting and Unavailable

VIRTANA CONTAINER OBSERVABILITY APPROACH

- Correlate deployment state with K8s events
- Highlight Invalid Image Tag as failure point
- Immediate, automated Root Cause

LEGACY APPROACH

- Isolated event showing failed deployment
- Manual alert creation IF a subject matter expert thinks to create it
- No correlation

WHY THE VIRTANA APPROACH MATTERS

- Reduced outage time and MTTR
- Reduce reliance on senior subject matter experts being pulled in from multiple domains

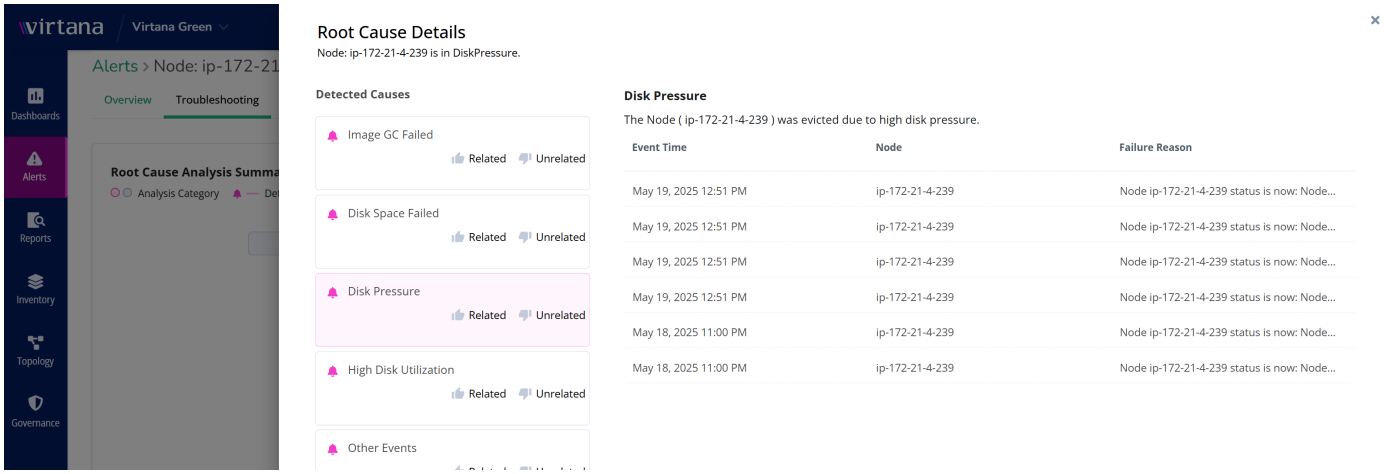

Figure #15: Virtana's Root Cause Analysis engine showing the root cause of failed container startup

Use-Case 3: Eviction Due to Node Disk Pressure and High Utilization

SCENARIO

- Service Unavailable

VIRTANA CONTAINER OBSERVABILITY APPROACH

- Correlate deployment state with K8s events, including evicted Pods
- Highlight Node has high disk utilization and disk pressure
- Immediate Root Cause

LEGACY APPROACH

- Isolated Event showing failed deployment
- No correlation

WHY THE VIRTANA APPROACH MATTERS

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

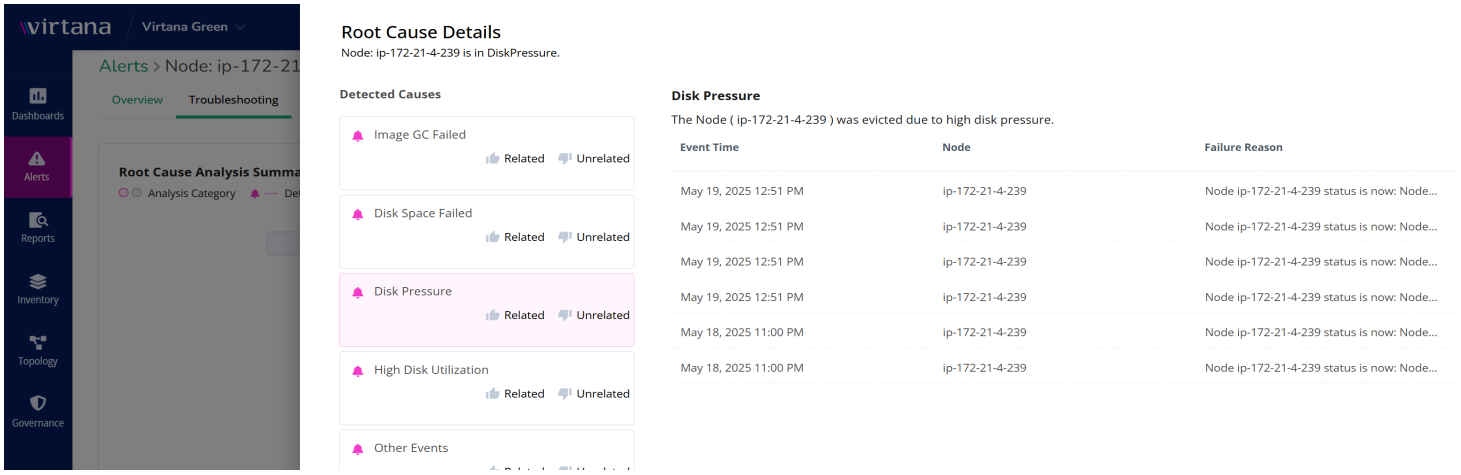

Figure #16: Virtana's Root Cause Analysis engine showing the root cause of pod evictions in a Kubernetes cluster

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

Higher Availability and Better Performing Digital Services

The relationship between application performance, customer satisfaction, and, ultimately, revenue has been proven in many industries. Virtana helps customers identify performance degradations before users notice and solve them more rapidly. This means fast recovery time, less downtime, and a direct, positive impact on the business's bottom line.

Up to 66% reduction in monitoring tool costs

Organizations are spending increasingly excessive amounts on legacy monitoring tools to perform basic functions. These costs are compounded when vendors store all the telemetry data in their cloud. Virtana Container Observability embeds the increasingly popular open-source monitoring tools as part of its foundation, allowing you to eliminate legacy, proprietary, and costly tools that may be in use in your modern environment.

50% improvement in SRE productivity

Virtana Container Observability can get organizations on the path to supporting more applications at higher release velocity with existing staff. Organizations have improved their SRE / Dev ratios by as much as 50%.

40% more alerts handled by L1.

Because Virtana Container Observability 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 oversized and underutilized sized instances and Pods inside Kubernetes. Virtana Container Observability enables organizations to make adjustments use as much as 20% fewer resources without impacting performance or risk while to reducing spend on oversized workloads

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; Virtana provides us an affordable, engaging, and approachable view that allows engineers to see how their microservices fit into the entire stack. Most importantly, because Virtana is building on our existing open source monitoring stack, deploying it was simple and required no changes to our running services."
– Director, DevOps Engineering, Global Car Rental Company

"Virtana is a modern and novel approach for integrating information from both CNCF telemetry/observability projects and Kubernetes to help visualize the environment and automate production troubleshooting."
– Founding infrastructure engineer, Prometheus.io

[Virtana] 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 Virtana

Virtana is the deepest and broadest observability platform for hybrid infrastructure. The AI-powered Virtana Platform delivers a unified view across applications, services, and underlying infrastructure, correlating user impact, service dependencies, performance bottlenecks, and cost drivers in real time. Trusted by Global 2000 enterprises, Virtana helps IT, operations, and platform teams improve efficiency, reduce risk, and make faster, AI-driven decisions across complex, dynamic environments.

Sources

Gartner Press Release

CNCF Annual Survey

CNCF Image

BMC Blog Image