Automating Root Cause Analysis for Kubernetes Applications

OpsCruise

TABLE OF CONTENTS

Automating Root Cause Analysis for Kubernetes Applications

The Problem

A 2022 survey [1] on the state of Kubernetes shows it is now a mainstream technology for software development and cloud adoption, with nearly half of the current users expecting to grow the number of clusters by more than 50%. At the same time, ensuring performance guarantees when there are continual changes in the application topology, service additions or deletions, or changes in the code ('image') behind a service, and a large number of containers creates significant challenges to the Ops or SRE teams. Because of the many dependencies between microservices and between application components and Kubernetes and underlying infrastructure, diagnosing problems are more complex in K8s applications. Even an innocuous small change in Kubernetes deployment can lead to an application slowdown or worse a crash of the business service. To paraphrase an old adage, 'for want of a nail the battle can be lost[2].'

Building an Automated Root Cause Analysis System

Trying to find an exact root cause in a large K8s application is not easy or deterministic given the high cardinality of interacting objects, the dynamism and scale of the environment. Ops or SRE teams have limited visibility into the state of the application in real-time while they look through metrics, logs, traces, or changes in deployments.

What we can do in an effective automated root cause analysis (RCA) system is to quickly narrow down the areas of the application and point the Ops team to focus on a few components or objects that are the likely cause—and surface the data and insights relevant to the fault domain. We take inspiration from how most expert SREs solve problems in war rooms use their extensive experience and different sources of information, including:

- Knowledge of IT stack, including understanding of Kubernetes
- Spatio-temporal dependencies and local interaction between connected services
- Dependencies on shared services (PaaS, external API calls, IaaS, etc)
- Problem indications in infrastructure (including changes) or in K8s deployment or runtime events
- Metrics and related alerts
- Logs or events
- Errors detected in traces, if available
- Emergent or learned behavior of individual services, using ML where needed

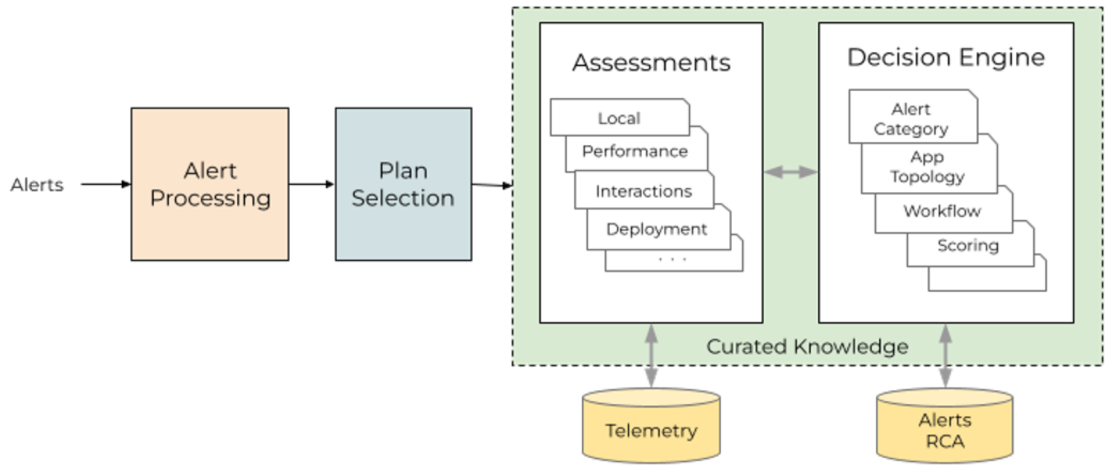


Figure 1: Dynamic decision tree approach to RCA

[1] 'The State of Kubernetes 2022,' VMware Tanzu, April 2022.

[2] 'For want of a nail' (the want of a nail leads to loss of the kingdom) https://en.wikipedia.org/wiki/For_Want_of_a_Nail

The dynamic decision engine makes assessments...to eliminate areas that are not relevant to the problem and isolate the fault domain.

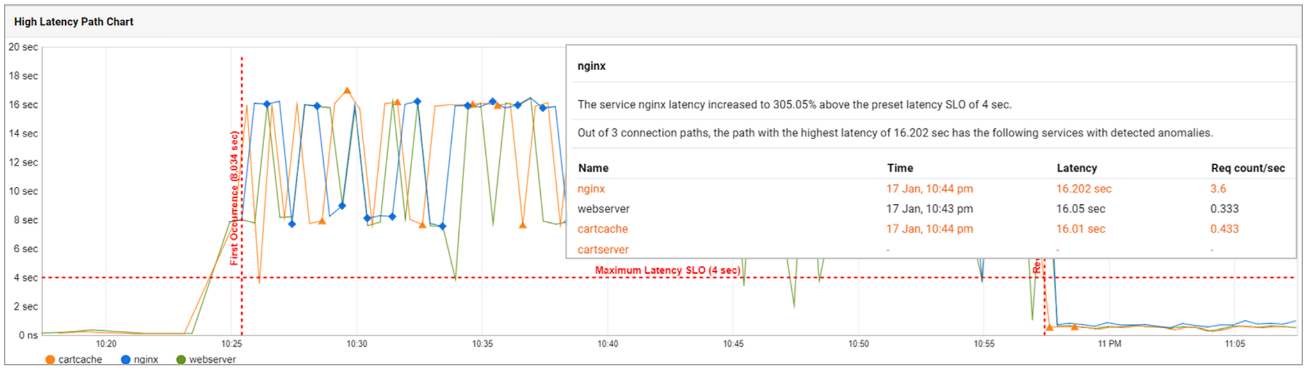
In effect, an automated RCA system is an "SME in a box" which determines the overall 'state' of the application across all telemetry and eliminates components that are not likely responsible for the performance problem. We have implemented automated RCA as a dynamic decision AI engine that:

- Analyzes the telemetry and alert details, both explicit (e.g., failures, saturation) and predictive from ML models
- Utilizes discovered and learned aspects of the application including topology and dependencies, and expected behaviors of service components
- Uses embedded curated knowledge of known applications (databases, storage systems, queuing systems, K8s)
- Selects a assessment plan depending on the alert category (known application, K8s or infrastructure) to follow a prescriptive diagnostic workflow

The dynamic decision engine makes assessments across all telemetry, dependency, alert details and its analyses, configuration, etc. to eliminate areas that are not relevant to the problem and isolate the fault domain. An example will illustrate how the automated RCA works.

Example: Application Performance Slowdown RCA

We show how automated RCA works in isolating the cause of performance slowdown in a sample application which is not instrumented for traces. The slowdown is detected using flow metrics from eBPF (note OpsCruise uses only open source and OTel monitoring sources) and as shown below detects when the ingress service 'nginx' exceeds the preset SLO set at 4 seconds. Automated RCA dynamically creates the high latency path chart that contains the anomalous services and containers.



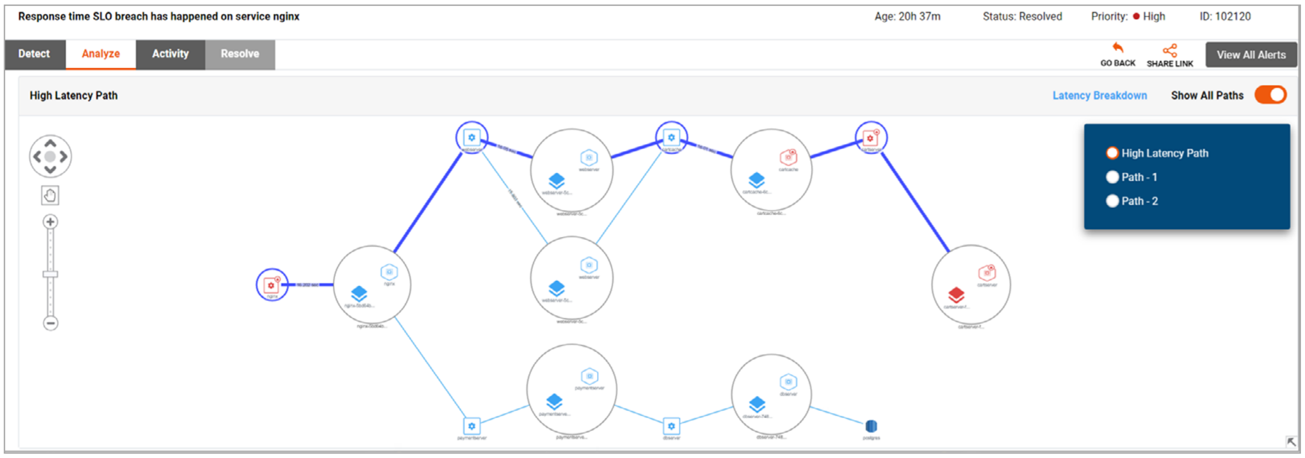
nginx

The service nginx latency increased to 305.05% above the preset latency SLO of 4 sec.

Out of 3 connection paths, the path with the highest latency of 16.202 sec has the following services with detected anomalies.

Name	Time	Latency	Req count/sec
nginx	17 Jan, 10:44 pm	16.202 sec	3.6
webserver	17 Jan, 10:43 pm	16.05 sec	0.333
cartcache	17 Jan, 10:44 pm	16.01 sec	0.433
cartserver	-	-	-

Figure 2



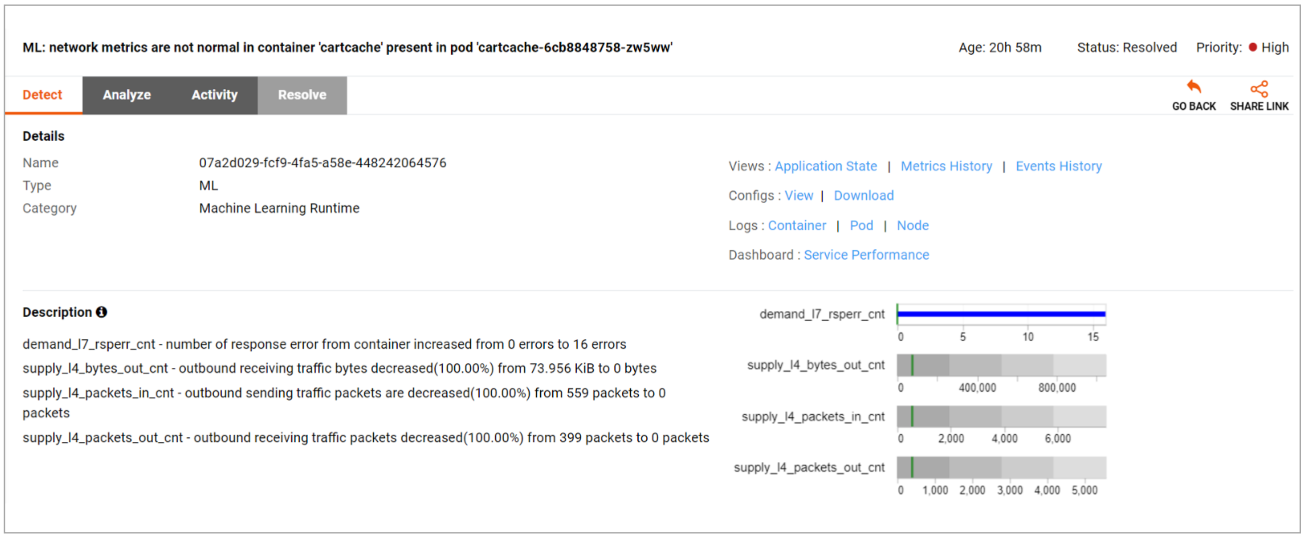
Response time SLO breach has happened on service nginx   Age: 20h 37m   Status: Resolved   Priority: ● High   ID: 102120

Figure 3

To eliminate manual searching for the slow path, the automated RCA checks all paths from the ingress and selects the highest latency path as shown above.

It is not surprising that the high latency path shows 4 other anomalous components, services and components, that were discovered by OpsCruise's anomaly detection mechanism[3] which does not require setting thresholds or selecting metrics to monitor.

Examining these other alerts, we start examining the first one, 'cartcache', where the alert was detected by the ML based on its learned behavior model showing that there are increased errors detected in the container and that all traffic (L4 bytes or packets) interactions with the next container ("supply side") has dropped to 0.


ML: network metrics are not normal in container 'cartcache' present in pod 'cartcache-6cb8848758-zw5ww'   Age: 20h 58m   Status: Resolved   Priority: ● High

Detect | Analyze | Activity | Resolve   GO BACK   SHARE LINK

Details

Name	07a2d029-fcf9-4fa5-a58e-448242064576
Type	ML
Category	Machine Learning Runtime

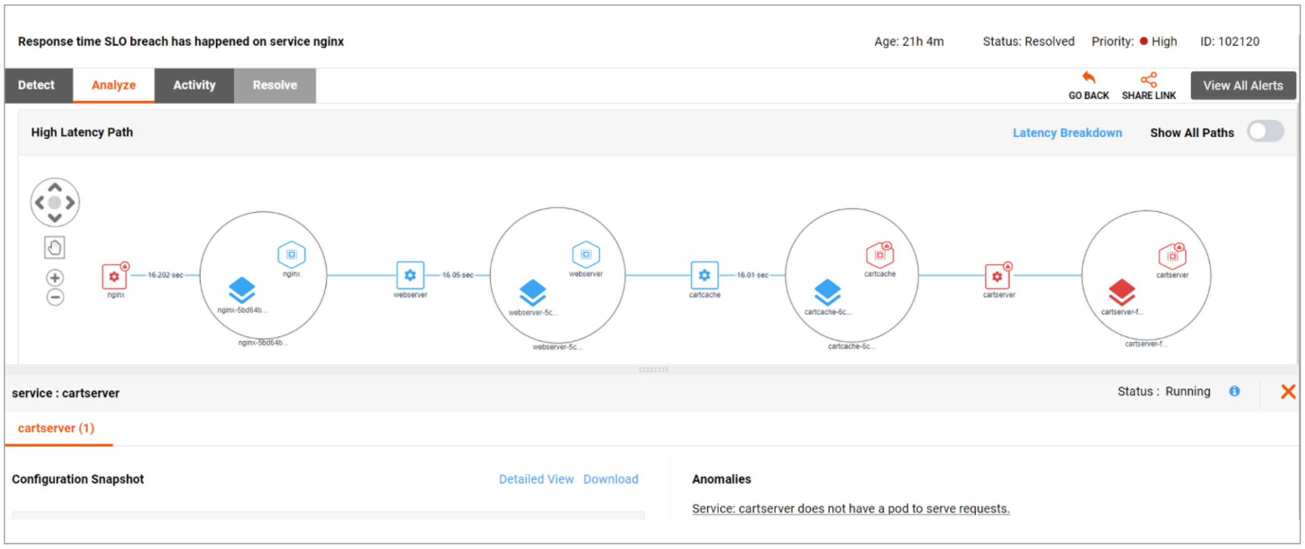
Views : Application State | Metrics History | Events History
Configs : View | Download
Logs : Container | Pod | Node
Dashboard : Service Performance

Description

demand_l7_rsperr_cnt - number of response error from container increased from 0 errors to 16 errors
supply_l4_bytes_out_cnt - outbound receiving traffic bytes decreased(100.00%) from 73.956 KiB to 0 bytes
supply_l4_packets_in_cnt - outbound sending traffic packets are decreased(100.00%) from 559 packets to 0 packets
supply_l4_packets_out_cnt - outbound receiving traffic packets decreased(100.00%) from 399 packets to 0 packets

Figure 4

The next alert is on the 'cartserver' service which indicates an immediate problem: there are no pods behind the service.


Response time SLO breach has happened on service nginx   Age: 21h 4m   Status: Resolved   Priority: ● High   ID: 102120

Detect | Analyze | Activity | Resolve   GO BACK   SHARE LINK   View All Alerts

High Latency Path   Latency Breakdown   Show All Paths

service : cartserver   Status : Running

cartserver (1)

Configuration Snapshot   Detailed View   Download

Anomalies

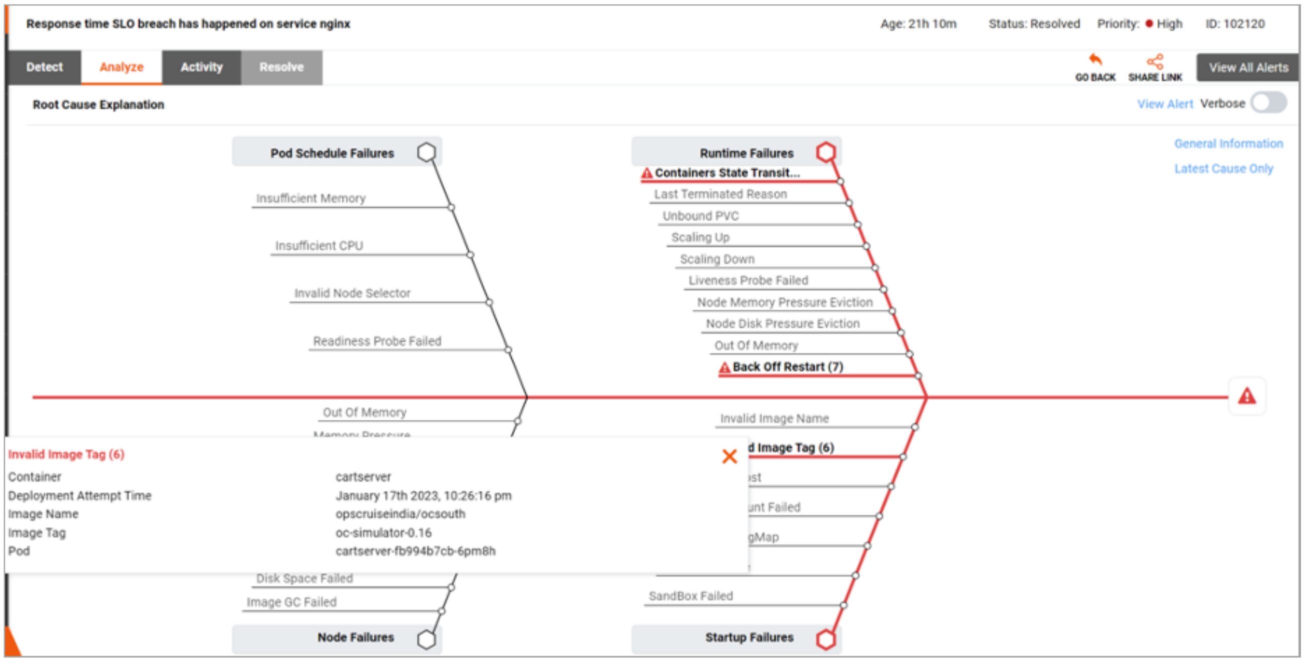
Service: cartserver does not have a pod to serve requests.

Figure 5

Following the causal path, we check the container and pods behind 'cartserver' reveals more specifics on the reason for the anomaly: the container is in pending state due to a ImagePullBackOff alert detected from Kubernetes.

[3] "Rethinking Anomaly Detection" https://www.opscruise.com/newsroom-post/ebook-on-rethinking-anomaly-detection

Finally, checking the RCA tab indicates the real cause for the image backoff error: a bad image name that was deployed creating a startup failure.



Response time SLO breach has happened on service nginx

Age: 21h 10m Status: Resolved Priority: ● High ID: 102120

Invalid Image Tag (6)

Container	cartserver
Deployment Attempt Time	January 17th 2023, 10:26:16 pm
Image Name	opscruiseindia/ocsouth
Image Tag	oc-simulator-0.16
Pod	cartserver-fb994b7cb-6pm8h

Figure 6

The full causal chain shown above is automatically detected by the RCA system when the initial SLO breach was detected as the decision engine searched dependencies, related alerts, events and information provided by the ML models. There was no need for Ops to dig through different alerts, events, metrics or flows, or try to construct the causal spatial dependencies, or find the contextual linking between all of them. Recall there were no traces available here either that are often the only way most Ops rely on to diagnose and solve performance problems.

Conclusion

We believe that troubleshooting performance issues in K8s applications requires an automated RCA system that can quickly identify and focus the Ops team into a few components or objects that are the likely cause—and surface the data and insights relevant to the fault domain. In effect, an automated RCA acts like expert SREs who build on their domain and diagnostic process knowledge and pull all information, telemetry from metrics to logs and events (and traces) besides configuration to isolate the cause. The benefit is a significant decrease in the manual effort ("toil") and time to resolution.

...an automated RCA acts like expert SREs who build on their domain and diagnostic process knowledge and pull all information, telemetry from metrics to logs and events (and traces) besides configuration to isolate the cause.

Learn More & Get Started

2

OpsCruise Intelligent Application Observability

OpsCruise's patented intelligent application observability platform is provided as a SaaS solution with gateways that sit on your infrastructure and collect metrics, logs, traces and configuration data from many popular open source monitoring tools (e.g., Prometheus, Elastic, Loki, Jaeger, etc.).

Our platform's deep understanding of Kubernetes, coupled with our unique ML-based behavior profiling empowers your entire team to predict performance degradations and instantly surface their cause. All at a third of the cost of the current monitoring stack and without the need to instrument code, deploy agents, or maintain open-source tools.

Want to see Automated RCA for yourself?

1. **Find your way to OpsCruise.com,**
2. **Sign up for our Free Forever offering,**
3. **Download the appropriate adapter(s) to plug into your telemetry environment or deploy our OSS components using standard Helm charts. You'll be up and running in 5 min, and within 24 hours, OpsCruise will learn your environment and start generating highly enriched alerts with causal analysis.**

GET STARTED