ScienceLogic

Zebrium Root Cause as a Service (RCaaS) Documentation

Release EA84

Table of Contents

Chapter

1

Key Concepts

Overview

The following video explains how Zebrium can automatically show you the root cause of any kind of software or infrastructure problem, without any manual training or rules: https://www.youtube.com/watch?v=4jm108RXz1c.

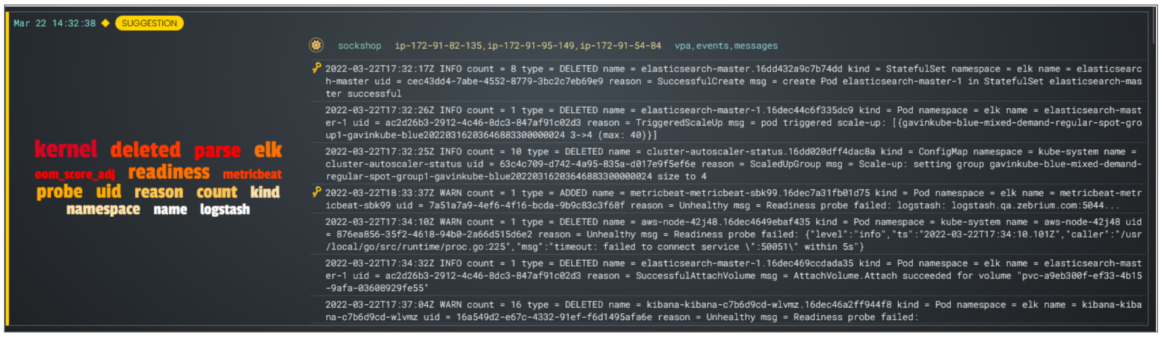
This chapter covers the following topics:

Zebrium Root Cause as a Service (RCaaS)

Zebrium Root Cause as a Service (RCaaS) uses unsupervised machine learning on logs to automatically find the root cause of software problems. It does not require manual rules or training, and it typically achieves accuracy within 24 hours.

As Zebrium ingests logs, the Zebrium artificial-intelligence machine-learning (AI/ML) engine analyzes the logs, looking for abnormal log line clusters that resemble problems, such as abnormally correlated rare and error events from across all log streams.

When the AI/ML engine detects one of these "abnormal" clusters, it generates a ***suggestion***, which appears on the **Alerts** page (the home page) of the Zebrium user interface:



```
2022-03-22T17:32:17Z INFO count = 8 type = DELETED name = elasticsearch-master.16dd432a9c7b74dd kind = StatefulSet namespace = elk name = elasticsearc
h-master uid = cec43dd4-7abe-4552-8779-3bc2c7eb69e9 reason = SuccessfulCreate msg = create Pod elasticsearch-master-1 in StatefulSet elasticsearch-mas
ter successful
2022-03-22T17:32:26Z INFO count = 1 type = DELETED name = elasticsearch-master-1.16dec44c6f335dc9 kind = Pod namespace = elk name = elasticsearch-mast
er-1 uid = ac2d26b3-2912-4c46-8dc3-847af91c02d3 reason = TriggeredScaleUp msg = pod triggered scale-up: [{gavinkube-blue-mixed-demand-regular-spot-gro
up1-gavinkube-blue20220316203646883300000024 3->4 (max: 40)}]
2022-03-22T17:32:25Z INFO count = 10 type = DELETED name = cluster-autoscaler-status.16dd020dff4dac8a kind = ConfigMap namespace = kube-system name =
cluster-autoscaler-status uid = 63c4c709-d742-4a95-835a-d017e9f5ef6e reason = ScaledUpGroup msg = Scale-up: setting group gavinkube-blue-mixed-demand-
regular-spot-group1-gavinkube-blue20220316203646883300000024 size to 4
2022-03-22T18:33:37Z WARN count = 1 type = ADDED name = metricbeat-metricbeat-sbk99.16dec7a31fb01d75 kind = Pod namespace = elk name = metricbeat-metr
icbeat-sbk99 uid = 7a51a7a9-4ef6-4f16-bcda-9b9c83c3f68f reason = Unhealthy msg = Readiness probe failed: logstash: logstash.qa.zebrium.com:5044...
2022-03-22T17:34:10Z WARN count = 1 type = DELETED name = aws-node-42j48.16dec4649ebaf435 kind = Pod namespace = kube-system name = aws-node-42j48 uid
= 876ea856-35f2-4618-94b0-2a66d515d6e2 reason = Unhealthy msg = Readiness probe failed: {"level":"info","ts":"2022-03-22T17:34:10.101Z","caller":"/usr
/local/go/src/runtime/proc.go:225","msg":"timeout: failed to connect service \":50051\" within 5s"}
2022-03-22T17:34:32Z INFO count = 1 type = DELETED name = elasticsearch-master-1.16dec469ccdada35 kind = Pod namespace = elk name = elasticsearch-mast
er-1 uid = ac2d26b3-2912-4c46-8dc3-847af91c02d3 reason = SuccessfulAttachVolume msg = AttachVolume.Attach succeeded for volume "pvc-a9eb300f-ef33-4b15
-9afa-03608929fe55"
2022-03-22T17:37:04Z WARN count = 16 type = DELETED name = kibana-kibana-c7b6d9cd-wlvmz.16dec46a2ff944f8 kind = Pod namespace = elk name = kibana-kiba
na-c7b6d9cd-wlvmz uid = 16a549d2-e67c-4332-91ef-f6d1495afa6e reason = Unhealthy msg = Readiness probe failed:
```

A suggestion is a summary report that contains the following main elements:

- ***AI-generated title***. This title is generated using the GPT-3 language model, which is trained on a large volume of public data. As a result, the titles might not always be accurate and should not be relied upon alone to make a decision about a suggestion.
- ***Word Cloud***. A set of relevant words chosen by the AI/ML engine from the log lines contained in the alert.
- ***Root Cause (RCA) Report Summary***. The report contains the actual cluster of anomalous log lines that was identified by the AI/ML engine. Up to eight of these log lines are shown in the summary view. You can click anywhere in the summary to view the full RCA report.
- ***Alert Key***. One or two log lines, denoted with a key icon (🔑), that are used to identify the suggestion if this type of suggestion occurs again. The alert keys make up an ***alert rule***.

IMPORTANT: Suggestions are generated when the AI/ML engine finds a cluster of correlated anomalies in your logs that resembles a problem. However, this does not mean that all suggestions relate to actual important problems. This is especially true during the first few days of using Zebrium, as the AI/ML engine learns the normal patterns in your logs.

When you start getting suggestions on the **Alerts** page, you can review the word clouds and event logs that display in the summary views for the Root Cause reports for the suggestions. As a best practice, identify a specific timeframe when a possible problem occurred, and then start looking at the reports that have the most interesting or relevant information related to the possible root cause of the problem.

You can choose to "accept" or "reject" a suggestion. For more information, see *Assessing Suggestions*.

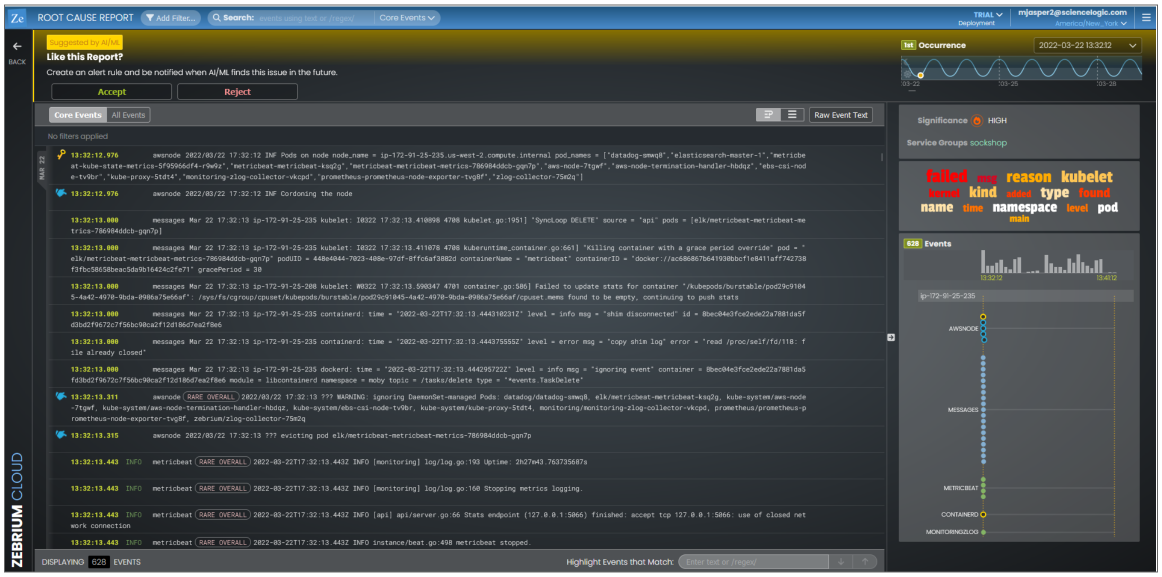
You can also decide on the action to take if the same kind of alert type occurs again, such as sending a notification to Slack, email, or another type of notification. For more information, see *Enabling Notification Channels*.

If you currently use a monitoring tool from ScienceLogic, Datadog, New Relic, Elastic, Dynatrace or AppDynamics, you can configure an observability dashboard integration that lets you view Zebrium suggestions on your existing monitoring dashboards. For more information, see *Configuring Observability Dashboard Integrations*.

Root Cause Reports (RCA Reports)

A ***Root Cause Report*** or ***RCA Report*** is a report generated by the AI/ML engine that consists of a group of log events that the AI/ML engine identified as being part of a problem.

A full **RCA Report** page (below) appears after you click the summary view for that report on the **Alerts** page:



```
13:32:12.976        awsnode 2022/03/22 17:32:12 INF Pods on node node_name = ip-172-91-25-235.us-west-2.compute.internal pod_names = ["datadog-smwq8","elasticsearch-master-1","metricbe
at-kube-state-metrics-5f95966df4-r9w9z","metricbeat-metricbeat-ksq2g","metricbeat-metricbeat-metrics-786984ddcb-gqn7p","aws-node-7tgwf","aws-node-termination-handler-hbdqz","ebs-csi-nod
e-tv9br","kube-proxy-5tdt4","monitoring-zlog-collector-vkcpd","prometheus-prometheus-node-exporter-tvg8f","zlog-collector-75m2q"]
13:32:12.976        awsnode 2022/03/22 17:32:12 INF Cordoning the node
13:32:13.000        messages Mar 22 17:32:13 ip-172-91-25-235 kubelet: I0322 17:32:13.410898 4708 kubelet.go:1951] "SyncLoop DELETE" source = "api" pods = [elk/metricbeat-metricbeat-me
trics-786984ddcb-gqn7p]
13:32:13.000        messages Mar 22 17:32:13 ip-172-91-25-235 kubelet: I0322 17:32:13.411078 4708 kuberuntime_container.go:661] "Killing container with a grace period override" pod = "
elk/metricbeat-metricbeat-metrics-786984ddcb-gqn7p" podUID = 448e4044-7023-408e-97df-8ffc6af3882d containerName = "metricbeat" containerID = "docker://ac686867b641930bbcf1e8411aff742738
f3fbc58658beac5da9b16424c2fe71" gracePeriod = 30
13:32:13.000        messages Mar 22 17:32:13 ip-172-91-25-208 kubelet: W0322 17:32:13.590347 4701 container.go:586] Failed to update stats for container "/kubepods/burstable/pod29c9104
5-4a42-4970-9bda-0986a75e66af": /sys/fs/cgroup/cpuset/kubepods/burstable/pod29c91045-4a42-4970-9bda-0986a75e66af/cpuset.mems found to be empty, continuing to push stats
13:32:13.000        messages Mar 22 17:32:13 ip-172-91-25-235 containerd: time = "2022-03-22T17:32:13.444310231Z" level = info msg = "shim disconnected" id = 8bec04e3fce2ede22a7881da5f
d3bd2f9672c7f56bc90ca2f12d186d7ea2f8e6
13:32:13.000        messages Mar 22 17:32:13 ip-172-91-25-235 containerd: time = "2022-03-22T17:32:13.443755555Z" level = error msg = "copy shim log" error = "read /proc/self/fd/118: f
ile already closed"
13:32:13.000        messages Mar 22 17:32:13 ip-172-91-25-235 dockerd: time = "2022-03-22T17:32:13.442957222Z" level = info msg = "ignoring event" container = 8bec04e3fce2ede22a7881da5
fd3bd2f9672c7f56bc90ca2f12d186d7ea2f8e6 module = libcontainerd namespace = moby topic = /tasks/delete type = "*events.TaskDelete"
13:32:13.311        awsnode (RARE OVERALL) 2022/03/22 17:32:13 ??? WARNING: ignoring DaemonSet-managed Pods: datadog/datadog-smwq8, elk/metricbeat-metricbeat-ksq2g, kube-system/aws-node
-7tgwf, kube-system/aws-node-termination-handler-hbdqz, kube-system/ebs-csi-node-tv9br, kube-system/kube-proxy-5tdt4, monitoring/monitoring-zlog-collector-vkcpd, prometheus/prometheus-p
rometheus-node-exporter-tvg8f, zebrium/zlog-collector-75m2q
13:32:13.315        awsnode 2022/03/22 17:32:13 ??? evicting pod elk/metricbeat-metricbeat-metrics-786984ddcb-gqn7p
13:32:13.443  INFO  metricbeat (RARE OVERALL) 2022-03-22T17:32:13.443Z INFO [monitoring] log/log.go:193 Uptime: 2h27m43.763735687s
13:32:13.443  INFO  metricbeat (RARE OVERALL) 2022-03-22T17:32:13.443Z INFO [monitoring] log/log.go:160 Stopping metrics logging.
13:32:13.443  INFO  metricbeat (RARE OVERALL) 2022-03-22T17:32:13.443Z INFO [api] api/server.go:66 Stats endpoint (127.0.0.1:5066) finished: accept tcp 127.0.0.1:5066: use of closed net
work connection
13:32:13.443  INFO  metricbeat (RARE OVERALL) 2022-03-22T17:32:13.443Z INFO instance/beat.go:498 metricbeat stopped.
```

The RCA report contains the actual cluster of anomalous log lines that was identified by the AI/ML engine. There are typically between ten and 100 log events in a report. Up to eight of these log lines are shown in the summary view. Clicking a summary on the **Alerts** page takes you to the full RCA report.

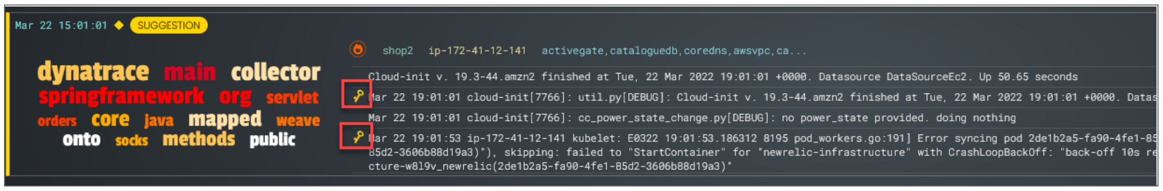
Each RCA report matches a particular "fingerprint" of log events. You can add notes, summaries, Jira links, and alert preferences to the alert rules for the RCA report so that future occurrences of the same type of problem will reflect these preferences and notes.

For more information, see *Working with Suggestions and Root Cause Reports*.

Alert Rules and Alert Keys

An ***alert rule*** is made up of one or two log events that best represent a specific type of problem that caused the event, and these events often provide clues as to the nature of the problem. These notable log events are called ***alert keys***, and the AI/ML engine uses these keys to trigger an alert when new log data is ingested.

A key icon (🔑) appears next to an alert key in the list of log events on the **Alerts** page and on the **RCA Report** page:


Mar 22 15:01:01
SUGGESTION
dynatrace main collector springframework org servlet orders core java mapped weave onto socks methods public
shop2 ip-172-41-12-141 activegate,cataloguedb,coredns,awsvpc,ca...

```
Cloud-init v. 19.3-44.amzn2 finished at Tue, 22 Mar 2022 19:01:01 +0000. Datasource DataSourceEc2. Up 50.65 seconds
Mar 22 19:01:01 cloud-init[7766]: util.py[DEBUG]: Cloud-init v. 19.3-44.amzn2 finished at Tue, 22 Mar 2022 19:01:01 +0000. Datas
Mar 22 19:01:01 cloud-init[7766]: cc_power_state_change.py[DEBUG]: no power_state provided. doing nothing
Mar 22 19:01:53 ip-172-41-12-141 kubelet: E0322 19:01:53.186312 8195 pod_workers.go:191] Error syncing pod 2de1b2a5-fa90-4fe1-85
85d2-3606b88d19a3)"), skipping: failed to "StartContainer" for "newrelic-infrastructure" with CrashLoopBackOff: "back-off 10s re
cture-w819v_newrelic(2de1b2a5-fa90-4fe1-85d2-3606b88d19a3)"
```

The AI/ML engine also uses the alert keys as a "signature" for a particular type of alert. There are typically two hallmark events:

- The first event in the sequence, which is usually a rare event or anomaly and often relates the root cause.
- A high severity event, either as determined by log severity, or other indicators, such as certain words or phrases indicating a problem, like "exception", "failed", "could not restart", and so on.

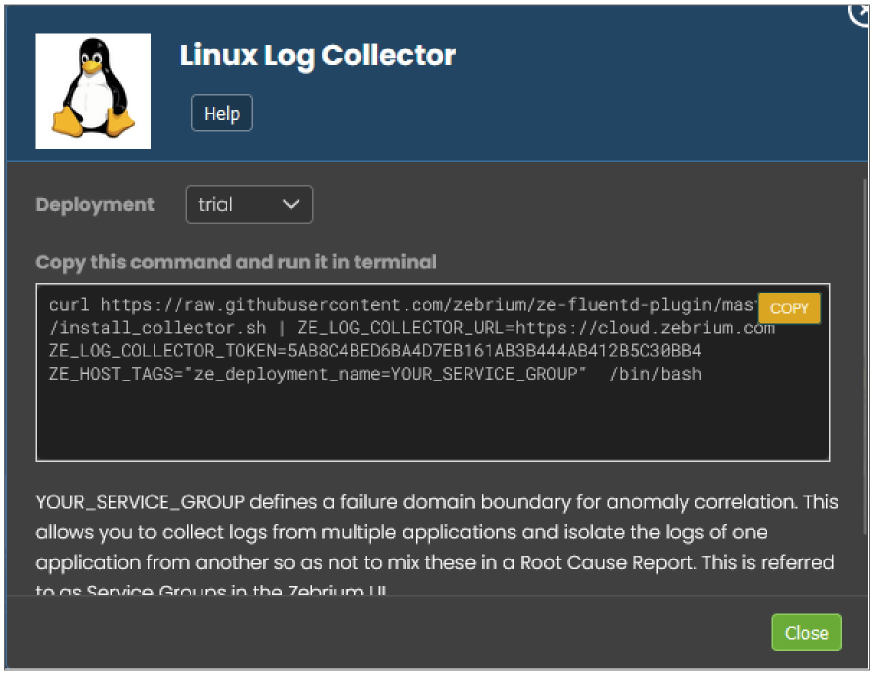
You can edit the alert keys of any Root Cause (RCA) report to select different log events if you believe those log events are more useful. Future matches of this type of RCA report will match against your user-defined alert keys, and carry forward your notes, summaries, Jira links, and alert preferences.

For more information, see *Editing Alert Keys.*

Log Collectors

When you are setting up your Zebrium system, one of the first tasks you need to do is configure a method for gathering log data to send to Zebrium so the AI/ML engine can begin to analyze the log data.

You would typically configure one or more ***log collectors*** to gather logs and send those logs to Zebrium for automated incident detection. For example, the following dialog explains how to set up a Linux log collector:



Deployment trial

Copy this command and run it in terminal

```
curl https://raw.githubusercontent.com/zebrium/ze-fluentd-plugin/mas
/install_collector.sh | ZE_LOG_COLLECTOR_URL=https://cloud.zebrium.com
ZE_LOG_COLLECTOR_TOKEN=5AB8C4BED6BA4D7EB161AB3B444AB412B5C30BB4
ZE_HOST_TAGS="ze_deployment_name=YOUR_SERVICE_GROUP"  /bin/bash
```

YOUR_SERVICE_GROUP defines a failure domain boundary for anomaly correlation. This allows you to collect logs from multiple applications and isolate the logs of one application from another so as not to mix these in a Root Cause Report. This is referred to as Service Groups in the Zebrium UI

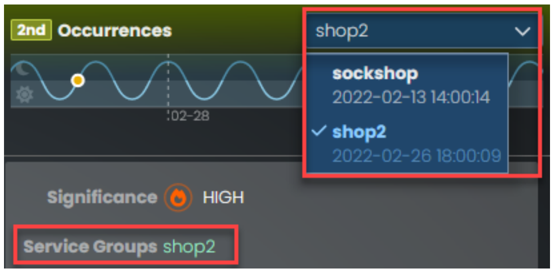
You can also use a ***file upload*** method using **ze**, the Zebrium command-line interface for uploading log events from files or streams.

For more information, see *Configuring Log Collectors and File Upload*.

Service Groups

A ***Service Group*** is the collection of log types, pods, hosts, and other items that are all part of a "failure domain". In other words, logs from the micro-services and processes that could all interact with each other to contribute to an incident should be part of a service group. The AI/ML engine will only attempt to correlate anomalies and errors across logs that fall within a service group. For more complex applications, you can have multiple service groups if there is more than one failure domain.

For example, in the following image, *sockshop* and *shop2* are two separate service groups where the same event occurred:



Using a service group allows you to collect logs from multiple applications or support cases and isolate the logs of one from another so as not to mix these in a RCA report.

If omitted, the service group is set to "default", which means that the service group represents shared services. For example, a database that is shared between two otherwise distinctly separate applications would be considered a shared service. In this example scenario, you would set the service group to "app01" for one application and "app02" for the other application. For the database logs, you would either omit the service group setting, or you could explicitly set it to "default".

With this configuration, RCA reports will consider correlated anomalies across the following:

```
"app01" log events and default (i.e. database logs) and
```

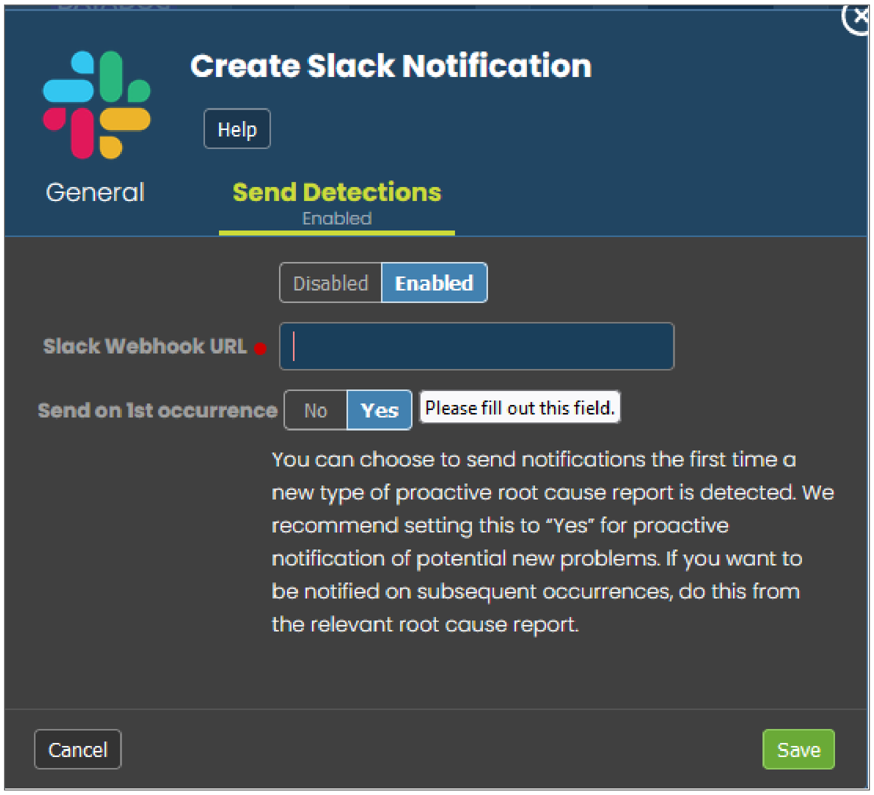
```
"app02" log events and default (i.e. database logs) but not across:
```

```
"app01" and "app02
```

For more information, see *Working with Suggestions and Root Cause Reports* .

Notification Channels

Notification Channels provide a mechanism to define the methods that Zebrium will use to send notifications from RCA reports. The supported types of notification channels include email, as well as Mattermost, Slack, Microsoft Teams, and Webex Teams notifications.



Send Detections: Disabled [✓ Enabled]

Slack Webhook URL ●

Send on 1st occurrence: No [✓ Yes]

Please fill out this field.

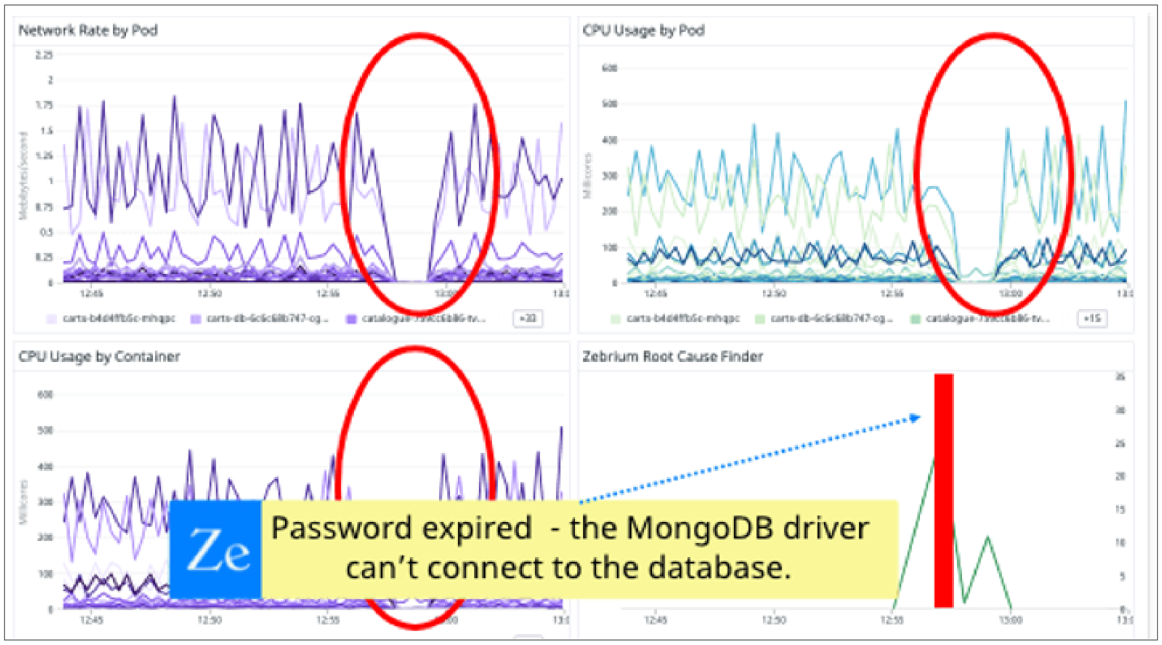
You can choose to send notifications the first time a new type of proactive root cause report is detected. We recommend setting this to "Yes" for proactive notification of potential new problems. If you want to be notified on subsequent occurrences, do this from the relevant root cause report.

After you have created one or more notification channels, you can link any number of these to any RCA report created by the AI/ML engine. Linking a set of notification channels to a RCA report will send notifications of future RCA reports of the same type to those channels.

For more information, see *Enabling Notification Channels*.

Observability Dashboard Integrations

You can integrate the Zebrium root cause service into your existing observability dashboards. For example, if you see symptoms of a potential problem in your metrics dashboard, you can have the root cause indicators for the problem surfaced right below that, as in the following image:


Network Rate by Pod
CPU Usage by Pod
CPU Usage by Container
Zebrium Root Cause Finder
Ze
Password expired - the MongoDB driver can't connect to the database.

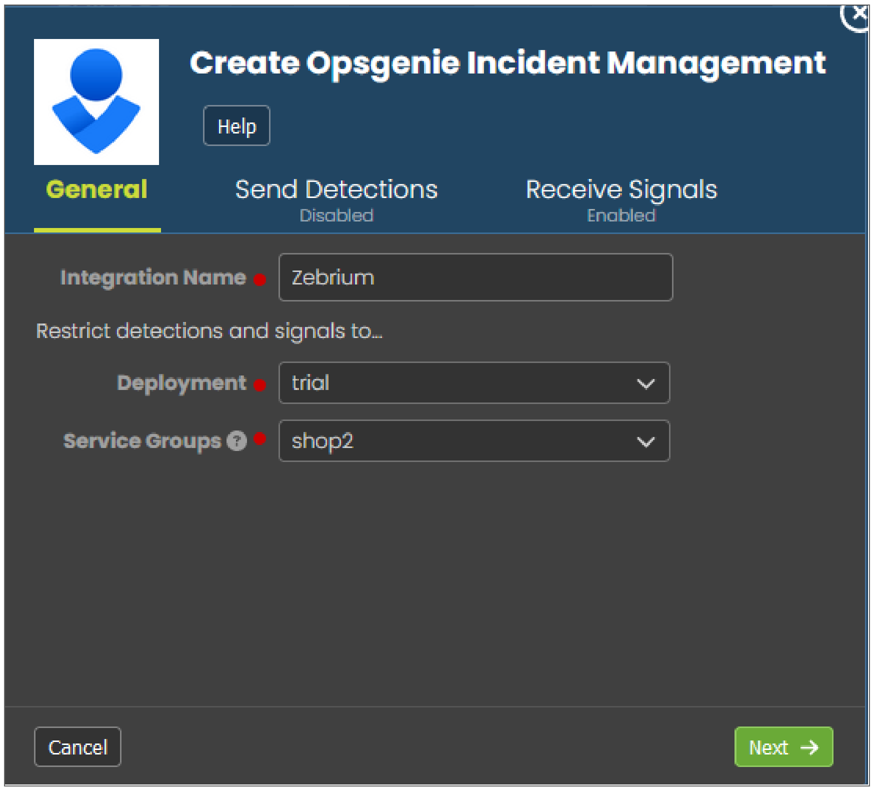
To enable this, go to the **Integrations & Collectors** page (Settings (☰) > Integrations & Collectors), select your preferred observability dashboard, and follow the instructions for setting up that dashboard.

For more information, see *Configuring Observability Dashboard Integrations*.

Incident Management Integrations

You can configure an integration between Zebrium and your third-party Incident Management application to automatically add Root Cause (RCA) reports to your incidents in the third-party application. Each Zebrium RCA report includes a summary, word cloud, and a set of log events display symptoms and root cause, along with a link to the full report in the Zebrium user interface.

After you complete the configuration, you can can view details of root cause and direct the incident to the appropriate team. All of these features lead to faster Mean Time to Repair (MTTR) and less time manually hunting for root cause.



Integration Name: Zebrium

Restrict detections and signals to...

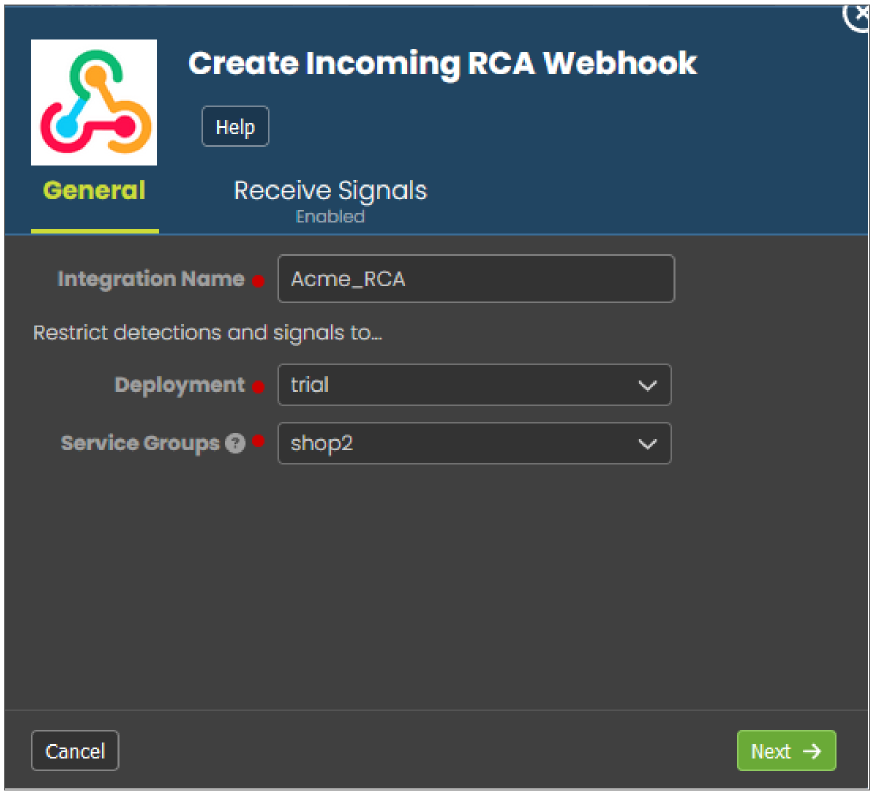
Deployment: trial

Service Groups: shop2

For more information, see *Configuring Incident Management Integrations*.

Integrations Using Webhooks

Zebrium provides support for using webhooks so you can build your own custom integrations.



Integration Name: Acme_RCA

Restrict detections and signals to...

Deployment: trial

Service Groups: shop2

Zebrium provides the following webhooks:

- Outgoing Root Cause Report Webhook
- Incoming Root Cause Report Incoming Webhook

For more information, see *Creating Integrations Using Webhooks*.

Zebrium On Prem

In additional to the standard option of a cloud configuration for Zebrium, you also have the option for a Zebrium on-premises (On Prem) configuration that is not located in the cloud.

For more information, see *Zebrium On Prem*.

Chapter 2

Getting Started with Zebrium

Overview

This chapter covers how Zebrium works and how to get started using Zebrium.

This chapter covers the following topics:

How Zebrium Works

When skilled engineers troubleshoot software, they typically ask the following questions:

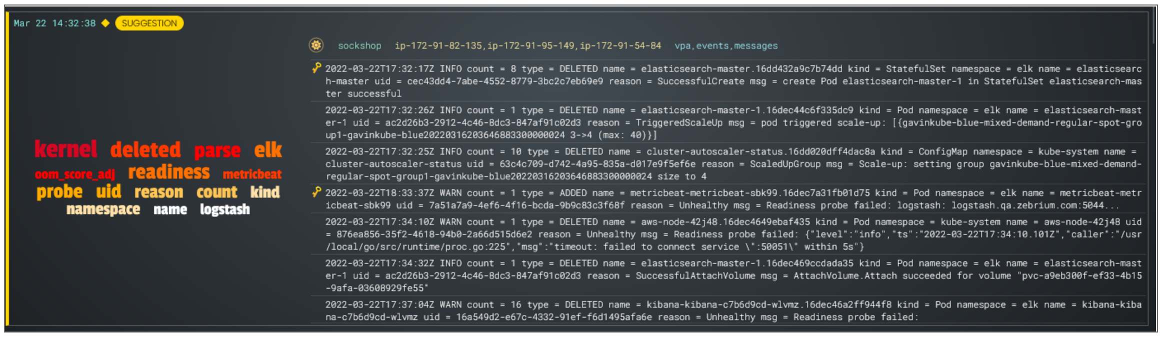
1. **Where are the problems or events occurring?** The events could be clusters of errors, warnings, stack traces, or other indicators of bad outcomes.
2. **Were there unusual events upstream that could help explain these bad outcomes?** This might be configuration changes, a new deployment, user actions, and so on.

In modern software, these events are often generated by different micro-services or software components, so you might have to switch between many log streams and then mentally correlate the events across them.

The Zebrium AI/ML engine emulates the workflow of a skilled engineer by performing the following actions:

1. Automatically build a catalog of all of the event types generated by the software.
2. Track the patterns of each event type in each log stream, such as the logs generated by a specific container, pod, or host.
3. Automatically identify unusual and "bad" events.
4. Identify unusually correlated clusters of rare and bad events that appear to be due to the same incident. The AI/ML engine scores each such collection based on a combination of how rare the underlying events are, and how bad the events are, such as how many warnings or errors are generated.
5. "Fingerprint" each cluster of such events as a unique type of issue. The events that rise above a specified threshold can be considered a potential Root Cause report, and they are summarized using Natural Language Processing (NLP) for Machine Learning.

When the AI/ML engine detects one of these "abnormal" clusters, it generates a ***suggestion***, which appears on the **Alerts** page (the home page) of the Zebrium user interface:


Mar 22 14:32:38
SUGGESTION
kernel deleted parse elk
oom_score_adj readiness metricbeat
probe uid reason count kind
namespace name logstash
sockshop ip-172-91-82-135,ip-172-91-95-149,ip-172-91-54-84 vpa,events,messages

```
2022-03-22T17:32:17Z INFO count = 8 type = DELETED name = elasticsearch-master.16dd432a9c7b74dd kind = StatefulSet namespace = elk name = elasticsearc
h-master uid = cec43dd4-7abe-4552-8779-3bc2c7eb69e9 reason = SuccessfulCreate msg = create Pod elasticsearch-master-1 in StatefulSet elasticsearch-mas
ter successful
2022-03-22T17:32:26Z INFO count = 1 type = DELETED name = elasticsearch-master-1.16dec44c6f335dc9 kind = Pod namespace = elk name = elasticsearch-mast
er-1 uid = ac2d26b3-2912-4c46-8dc3-847af91c02d3 reason = TriggeredScaleUp msg = pod triggered scale-up: [{gavinkube-blue-mixed-demand-regular-spot-gro
up1-gavinkube-blue20220316203646883300000024 3->4 (max: 40)}]
2022-03-22T17:32:25Z INFO count = 10 type = DELETED name = cluster-autoscaler-status.16dd020dff4dac8a kind = ConfigMap namespace = kube-system name =
cluster-autoscaler-status uid = 63c4c709-d742-4a95-835a-d017e9f5ef6e reason = ScaledUpGroup msg = Scale-up: setting group gavinkube-blue-mixed-demand-
regular-spot-group1-gavinkube-blue20220316203646883300000024 size to 4
2022-03-22T18:33:37Z WARN count = 1 type = ADDED name = metricbeat-metricbeat-sbk99.16dec7a31fb01d75 kind = Pod namespace = elk name = metricbeat-metr
icbeat-sbk99 uid = 7a51a7a9-4ef6-4f16-bcda-9b9c83c3f68f reason = Unhealthy msg = Readiness probe failed: logstash: logstash.qa.zebrium.com:5044...
2022-03-22T17:34:10Z WARN count = 1 type = DELETED name = aws-node-42j48.16dec4649ebaf435 kind = Pod namespace = kube-system name = aws-node-42j48 uid
= 876ea856-35f2-4618-94b0-2a66d515d6e2 reason = Unhealthy msg = Readiness probe failed: {"level":"info","ts":"2022-03-22T17:34:10.101Z","caller":"/usr
/local/go/src/runtime/proc.go:225","msg":"timeout: failed to connect service \":50051\" within 5s"}
2022-03-22T17:34:32Z INFO count = 1 type = DELETED name = elasticsearch-master-1.16dec469ccdada35 kind = Pod namespace = elk name = elasticsearch-mast
er-1 uid = ac2d26b3-2912-4c46-8dc3-847af91c02d3 reason = SuccessfulAttachVolume msg = AttachVolume.Attach succeeded for volume "pvc-a9eb300f-ef33-4b15
-9afa-03608929fe55"
2022-03-22T17:37:04Z WARN count = 16 type = DELETED name = kibana-kibana-c7b6d9cd-wlvmz.16dec46a2ff944f8 kind = Pod namespace = elk name = kibana-kiba
na-c7b6d9cd-wlvmz uid = 16a549d2-e67c-4332-91ef-f6d1495afa6e reason = Unhealthy msg = Readiness probe failed:
```

A suggestion is a summary report that contains the following main elements:

- ***AI-generated title***. This title is generated using the GPT-3 language model, which is trained on a large volume of public data. As a result, the titles might not always be accurate and should not be relied upon alone to make a decision about a suggestion.

- ***Word Cloud***. A set of relevant words chosen by the AI/ML engine from the log lines contained in the alert.
- ***Root Cause (RCA) Report Summary***. The report contains the actual cluster of anomalous log lines that was identified by the AI/ML engine. Up to eight of these log lines are shown in the summary view. You can click anywhere in the summary to view the full RCA report.
- ***Alert Key***. One or two log lines, denoted with a key icon (🔑), that are used to identify the suggestion if this type of suggestion occurs again. The alert keys make up an ***alert rule***.

IMPORTANT: Suggestions are generated when the AI/ML engine finds a cluster of correlated anomalies in your logs that resembles a problem. However, this does not mean that all suggestions relate to actual important problems. This is especially true during the first few days of using Zebrium, as the AI/ML engine learns the normal patterns in your logs.

When you start getting suggestions on the **Alerts** page, you can review the word clouds and event logs that display in the summary views for the Root Cause reports for the suggestions. As a best practice, identify a specific timeframe when a possible problem occurred, and then start looking at the reports that have the most interesting or relevant information related to the possible root cause of the problem.

You can choose to "accept" or "reject" a suggestion. For more information, see *Assessing Suggestions*.

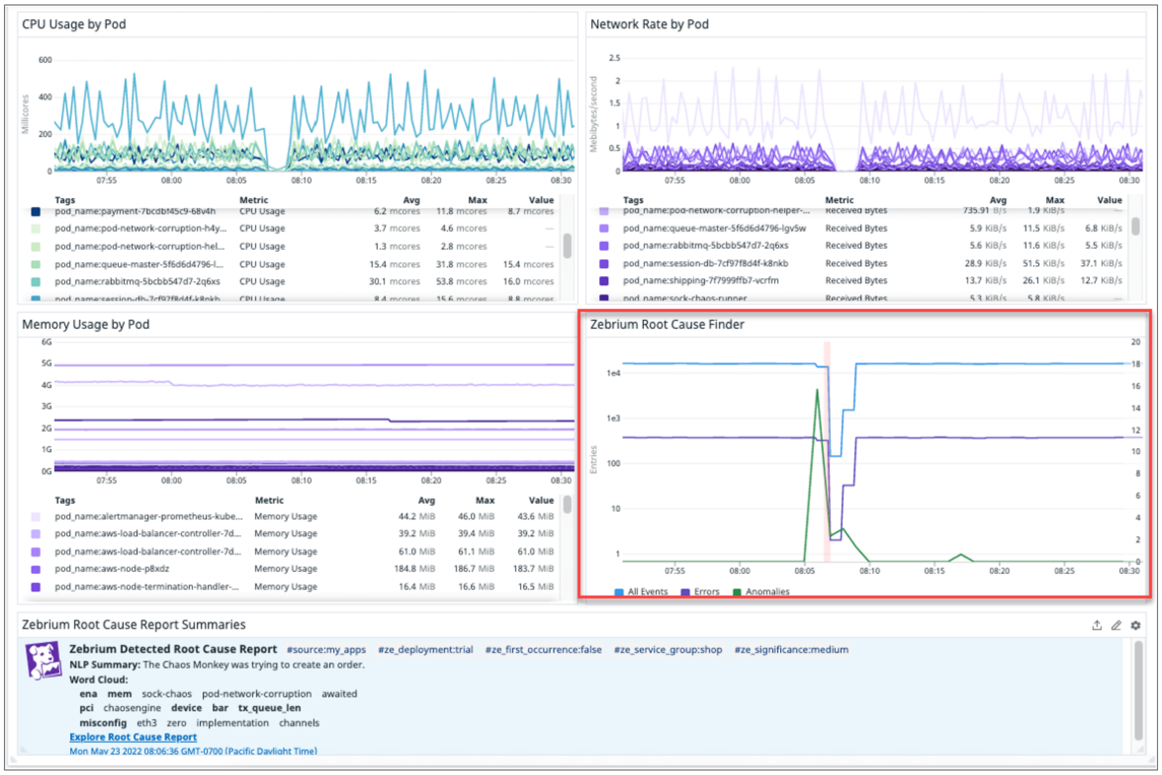
You can also decide on the action to take if the same kind of alert type occurs again, such as sending a notification to Slack, email, or another type of notification. For more information, see *Enabling Notification Channels*.

If you currently use a monitoring tool from ScienceLogic, Datadog, New Relic, Elastic, Dynatrace or AppDynamics, you can configure an observability dashboard integration that lets you view Zebrium suggestions on your existing monitoring dashboards. For more information, see *Configuring Observability Dashboard Integrations*.

Consuming Root Cause Reports

You can consume the AI/ML engine-generated Root Cause reports in one of the following ways:

1. **Recommended**. Connect Zebrium to one or more of your ***observability dashboards***, such as ScienceLogic, DataDog, New Relic, Elastic Stack, Grafana, Dynatrace, or AppDynamics. After you configure an observability dashboard, data from the Root Cause reports from Zebrium will display on that dashboard, and you can visually correlate the reports with any spikes or alerts occurring at the same time:



	Tags	Metric	Avg	Max	Value
■	pod_name:payment-7bcdbf45c9-68v4h	CPU Usage	6.2 mcores	11.8 mcores	8.7 mcores
■	pod_name:pod-network-corruption-h4y...	CPU Usage	3.7 mcores	4.6 mcores	—
■	pod_name:pod-network-corruption-hel...	CPU Usage	1.3 mcores	2.8 mcores	—
■	pod_name:queue-master-5f6d6d4796-l...	CPU Usage	15.4 mcores	31.8 mcores	15.4 mcores
■	pod_name:rabbitmq-5bcbb547d7-2q6xs	CPU Usage	30.1 mcores	53.8 mcores	16.0 mcores
■	pod_name:session-db-7cf97f8d4f-k8nkb	CPU Usage	8.4 mcores	15.6 mcores	8.8 mcores

	Tags	Metric	Avg	Max	Value
■	pod_name:pod-network-corruption-helper-...	Received Bytes	735.91 B/s	1.9 KiB/s	—
■	pod_name:queue-master-5f6d6d4796-lgv5w	Received Bytes	5.9 KiB/s	11.5 KiB/s	6.8 KiB/s
■	pod_name:rabbitmq-5bcbb547d7-2q6xs	Received Bytes	5.6 KiB/s	11.6 KiB/s	5.5 KiB/s
■	pod_name:session-db-7cf97f8d4f-k8nkb	Received Bytes	28.9 KiB/s	51.5 KiB/s	37.1 KiB/s
■	pod_name:shipping-7f7999ffb7-vcrfm	Received Bytes	13.7 KiB/s	26.1 KiB/s	12.7 KiB/s
■	pod_name:sock-chaos-runner	Received Bytes	5.3 KiB/s	5.8 KiB/s	—

	Tags	Metric	Avg	Max	Value
■	pod_name:alertmanager-prometheus-kube...	Memory Usage	44.2 MiB	46.0 MiB	43.6 MiB
■	pod_name:aws-load-balancer-controller-7d...	Memory Usage	39.2 MiB	39.4 MiB	39.2 MiB
■	pod_name:aws-load-balancer-controller-7d...	Memory Usage	61.0 MiB	61.1 MiB	61.0 MiB
■	pod_name:aws-node-p8xdz	Memory Usage	184.8 MiB	186.7 MiB	183.7 MiB
■	pod_name:aws-node-termination-handler-...	Memory Usage	16.4 MiB	16.6 MiB	16.5 MiB

Zebrium Root Cause Report Summaries

Zebrium Detected Root Cause Report #source:my_apps #ze_deployment:trial #ze_first_occurrence:false #ze_service_group:shop #ze_significance:medium
NLP Summary: The Chaos Monkey was trying to create an order.
Word Cloud:
ena **mem** sock-chaos pod-network-corruption awaited
pci chaosengine **device** **bar** **tx_queue_len**
misconfig eth3 zero implementation channels
Explore Root Cause Report
Mon May 23 2022 08:06:36 GMT-0700 (Pacific Daylight Time)

For more details, or to take action on one of these reports, click the URL to go directly to the detailed Root Cause report in the Zebrium user interface. For more information, see ***Working with Suggestions and Root Cause Reports***.

2. Connect Zebrium to your ***incident management tool***, such as OpsGenie, PagerDuty, VictorOps, or Slack. After you configure the incident management tool, an RCA report is automatically created and sent back to the incident management tool.

   - In some cases (Opsgenie, OpsRamp, and VictorOps), Zebrium can create an incident in the incident management tool when a Root Cause report is autonomously detected by the AI/ML engine.

   - In other cases (Opsgenie and PagerDuty), a bi-directional integration is supported, so that the incident management tool will query Zebrium any time a new incident is created, such as by another monitoring tool. In this scenario, Zebrium automatically responds by adding the right Root Cause report back into the timeline of the incident in the incident management tool.

3. Evaluate the feed of auto-detect incident Root Cause reports on the **Alerts** page in the Zebrium user interface, particularly around times where you know things went wrong. You can also force the AI/ML engine to do a deep scan and create a report on demand by clicking the **[Scan for RC]** button on the **Settings** menu ().

For more information, see *Working with Suggestions and Root Cause Reports*.

Customizing Your Zebrium Results

You can customize your Zebrium results in the following ways:

- **Changing the sensitivity threshold**. You can edit the threshold to make the AI/ML engine more sensitive to catching subtle issues, or making it less sensitive to minimize noise. Note that in the latter case, the AI/ML engine will still detect the less significant issues and create reports for them, but will not display them until settings are changed.
- **Filtering the results**: page displays a list of filtering and search options at the top of the page. You can use these filters to manage the number of suggestions and alerts that display on the **Alerts** page. For example, by default only the first occurrence of each incident type is visible on dashboards and alert channel, unless you create filters that specify that the incident deserves an alert or suggestion. For more information about filtering, see *Using the Filters on the Alerts Page in Zebrium*.

In most cases, the default settings will work well. However, Zebrium provides several controls to adjust the sensitivity of its AI/ML engine so you can tune the signal-to-noise ratio to best suit your environment. Zebrium also provides ways to group related log feeds so that the AI/ML engine only correlates anomalies across related components of an application instance (the components that together define a failure domain).

For more information about changing the default sensitivity settings and managing deployments, contact support@zebrium.com.

Evaluating Zebrium

The best way to try Zebrium is on a system that is experiencing an actual problem. If there are no real problems, Zebrium will not find anything useful.

As an alternative, you can try Zebrium in an environment where you can simulate a real problem. You can also use this step-by-step guide to set up a demonstration online shopping application and cause a failure by using an open source chaos tool.

Signing Up for a New Account

To sign up for a new account and start sending your logs to Zebrium, watch this five-minute "Getting Started" video: https://youtu.be/QwIbihOOW5k.

The video covers how to :

1. Sign up for a new account by visiting https://www.zebrium.com/ and clicking the blue **[Get Started Free]** button.
2. Installing the Kubernetes log collector by using the customized Helm command found on the **Welcome** page. After you have configured the log collector, Zebrium can being reviewing your logs.

> **NOTE:** You will need to set your Timezone and Service Group (zebrium.deployment) when installing the collector.

What does Zebrium Do with Your Logs?

As logs are received by Zebrium, the Zebrium AI/ML engine automatically structures and categorizes each type of log event. This allows the AI/ML engine to identify anomalous log events. Many factors are used for anomaly detection, but the two most important are the rareness and the severity of each log line.

The AI/ML engine then looks for abnormal clusters of correlated anomalies across all the logs within a Service Group, also known as a failure domain. These clusters usually occur because of an actual problem.

If the AI/ML engine finds one of these clusters, it generates a ***Suggestion***. The suggestion contains a payload that includes the cluster of log lines.

Other than the log events that are contained in alerts, all other log data is discarded after a few hours.

Chapter

3

Configuring Log Collectors and File Uploads

Overview

When you are setting up your Zebrium system, one of the first tasks you need to do is configure a method for gathering log data to send to Zebrium so the AI/ML engine can begin to analyze the log data.

When you are setting up your Zebrium system, one of the first tasks you need to do is configure a method for gathering log data to send to Zebrium so the AI/ML engine can begin to analyze the log data.

You would typically configure one or more ***log collectors*** to gather logs and send those logs to Zebrium for automated incident detection. For example, the following dialog explains how to set up a Linux log collector:

You can also use a ***file upload*** method using **ze**, the Zebrium command-line interface for uploading log events from files or streams.

Zebrium lets you collect data from logs in a variety of ways, from a number of different sources. The following pages explain how you can collect data from these sources, as well as file uploads:

- *Kubernetes*
- *Linux*
- *File Upload (ze Command)*
- *CloudWatch*
- *Docker (including ECS)*
- *Logstash*
- *Syslog Forwarder*

Kubernetes Collector

This topic explains how to use ***zlog-collector***, the Zebrium log collector for Kubernetes.

Version 1.56.0 AppVersion 1.50.0

Installing the Helm Chart

To install the Helm chart with the release name **zebrium**, run the following commands:

```
helm repo add zebrium http://charts.zebrium.com
```

```
helm upgrade -i zlog-collector zebrium/zlog-collector --namespace zebrium
--create-namespace --set
zebrium.collectorUrl=<YOUR_ZE_API_URL>,zebrium.authToken=YOUR_ZE_API_AUTH_
TOKEN,zebrium.deployment=
<YOUR_DEPLOYMENT_NAME>,zebrium.timezone=<KUBERNETES_HOST_TIMEZONE>
```

where `<KUBERNETES_HOST_TIMEZONE>` is the time zone setting on Kubernetes host, such as `UTC` or `America/Los_Angeles`. If this option is not provided, the default value of `UTC` will be used.

Uninstalling the Helm Chart

To uninstall the Helm chart with the release name **zebrium**, run the following command:

```
helm delete zlog-collector -n zebrium
```

Additional Information

Log Path Mapping

Log path mapping is the process of detecting semantic items in log file paths (ids, configs and tags) then including them in the Zebrium log data. This is enabled by providing a JSON mapping file to the log collector, as described in the repo at https://www.github.com/zebrium/ze-fluentd-plugin.

To use this functionality with the supplied Helm chart a **customValues.yaml** file should be completed and supplied to the Helm install command line with:

```
helm install ... -f customValues.yaml ...
```

A prototype **example_logPathMappings.yaml** file is provided in the repo under the example directory, with the following format:

```
overridePMFConfig: true
zebrium:
  pathMapFile: "pathMapFile.json"
customPMFConfig: {
"mappings": {
  "patterns":["/var/log/remote_logs/(?<host>[^/]+)/.*"],
    "tags": [],
    "ids" : [
    "host"],
    "configs": []
  }
}
```

Custom Namespace to Service Group Mapping

Custom Namespace to Service Group Matching is the process of dynamically assigning a service group to a log stream based on the resources namesapce. This is enabled by providing a JSON mapping file to the log collector.

To use this functionality with the supplied Helm chart, a **customValues.yaml** file should be completed and supplied to the Helm install command line with the following command:

```
helm install ... -f customValues.yaml ...
```

A prototype **example_ns_svcgrp.yaml** file is provided in the repository under the example directory, with the following format:

```
overrideSVCGRPConfig: true
zebrium:
  svcgrpMapFile: "svcgrpMapFile.json"
customSVCGRPConfig: {
  "mynamespace1" : "svcgrp1",
  "mynamespace2" : "svcgrp1",
  "mynamespace3" : "svcgrp3"
}
```

Values

Key	Type	Default	Description
daemonset.dnsPolicy	string	`"ClusterFirst"`	
daemonset.nodeSelector	object	`{}`	

Key	Type	Default	Description
daemonset.priorityClassName	string	`""`	
daemonset.tolerateAllTaints	bool	`true`	
daemonset.tolerations	list	`[]`	set ‘daemonset.tolerations [0].operator=Equal,daemonset.tolerations [0].effect=NoSchedule,daemonset.tolerations [0].key=node-role.kubernetes.io/master’
extraEnv	list	`[]`	
image.name	string	`"zebrium/zlog-collector"`	
image.pullPolicy	string	`"Always"`	
image.tag	string	`"latest"`	
resources.limits.cpu	string	`"1000m"`	
resources.limits.memory	string	`"1Gi"`	
resources.requests.cpu	string	`"20m"`	
resources.requests.memory	string	`"500Mi"`	
ruby.gcHeapOldObjectLimitFactor	float	`1.2`	
secret.enabled	bool	`true`	
services.automountServiceAccountToken	bool	`true`	
services.automountServiceAccountTokenSupported	bool	`false`	
updateStrategy	string	`"OnDelete"`	
zebrium.authToken	string	`""`	
zebrium.autoupdate	string	`"1"`	
zebrium.bufferChunkLimitRecords	int	`40000`	
zebrium.bufferChunkLimitSize	string	`"8MB"`	
zebrium.bufferRetryMaxTimes	int	`360`	

Key	Type	Default	Description
zebrium.bufferRetryTimeout	string	`"1h"`	
zebrium.bufferRetryWait	string	`"10s"`	
zebrium.bufferTotalLimitSize	string	`"64GB"`	
zebrium.clusterName	string	`""`	Name of the Kubernetes Cluster that the zlog-collector is deployed into
zebrium.collectorUrl	string	`""`	
zebrium.deployment	string	`"default"`	
zebrium.disableEc2MetaData	string	`"true"`	
zebrium.ec2ApiClientTimeoutSecs	string	`"1"`	
zebrium.excludeNamespaceRegex	string	`""`	Regex String to Exclude Namespaces, such as: `^(?!.*(bar foo))` would exclude all namespaces except **foo** and **bar**
zebrium.excludePath	string	`"[\"/var/log/boot.log\", \"/var/log/lastlog\"]"`	
zebrium.excludePodRegex	string	`""`	Regex String to exclude pods, such as: `^fluentbit.*` would exclude all fluentbit pods from collection
zebrium.flushInterval	string	`"30s"`	
zebrium.flushThreadCount	string	`"4"`	
zebrium.handleHostAsConfig	bool	`false`	
zebrium.k8sApiSecretName	string	`""`	
zebrium.logFile	string	`""`	
zebrium.logLevel	string	`"info"`	
zebrium.name	string	`"zlog-collector"`	
zebrium.nodeLogsPath	strin	`"/var/log/*.log,/var/lo`	

Key	Type	Default	Description
	g	`g/syslog, /var/log/messages,/var/log/secure"`	
zebrium.pathMapFile	string	`""`	
zebrium.svcgrpMapFile	string	`""`	
zebrium.tailFromHead	string	`"true"`	
zebrium.timezone	string	`"UTC"`	
zebrium.useHostEtcHostnameFile	bool	`false`	
zebrium.verifyK8sApiSSL	bool	`true`	
zebrium.verifySSL	string	`"true"`	

Linux Collector

The Zebrium Fluentd output plugin, **ze-fluentd-plugin**, sends the logs you collect with Fluentd on Linux to Zebrium for automated anomaly detection. You can access the plugin at the Zebrium github repository, which is located at https://github.com/zebrium/ze-fluentd-plugin.

For instructions on deploying the Zebrium Fluentd collector for Docker environments, see the instructions in ***Docker Container Log Collectors.***

System Requirements

The following Linux operating system distributions are supported:

- Ubuntu 16.04/18.04/20.04
- CentOS or Red Hat Enterprise Linux 7/8
- Amazon Linux 2

Installing the Collector

1. If the environment uses a proxy server, see ***Operation with a Proxy Server***, below.
2. Get the Zebrium API server URL and authentication token from Zebrium.
3. Determine what deployment name to use.
4. Run the following command in a shell on the host:

```
curl https://raw.githubusercontent.com/zebrium/ze-fluentd-
plugin/master/install_collector.sh | ZE_LOG_COLLECTOR_URL=<ZAPI_URL>
ZE_LOG_COLLECTOR_TOKEN=<AUTH_TOKEN> ZE_HOST_TAGS="ze_deployment_
name=<deployment_name>" /bin/bash
```

The default system log file paths are defined by the ZE_LOG_PATHS environment variable. Its default value is:

```
"/var/log/*.log,/var/log/syslog,/var/log/messages,/var/log/secure"
```

The ZE_USER_LOG_PATHS environment variable can be used to add more user specific log file paths. For example, to add app log files at **/app1/log/app1.log** and **/app2/log/*.log**, you can set ZE_USER_LOG_PATHS to:

```
"/app1/log/app1.log,/app2/log/*.log"
```

Upgrading the Collector

The upgrade command is similar to the installation command:

```
curl https://raw.githubusercontent.com/zebrium/ze-fluentd-
plugin/master/install_collector.sh | ZE_LOG_COLLECTOR_URL=<ZAPI_URL> ZE_
LOG_COLLECTOR_TOKEN=<AUTH_TOKEN> ZE_HOST_TAGS="ze_deployment_
name=<deployment_name>" OVERWRITE_CONFIG=1 /bin/bash
```

Please note that setting OVERWRITE_CONFIG to 1 will cause **/etc/td-agent/td-agent.conf** to be upgraded to the latest version.

Uninstalling the Collector

To uninstall:

```
curl https://raw.githubusercontent.com/zebrium/ze-fluentd-
plugin/master/install_collector.sh | ZE_OP=uninstall /bin/bash
```

Installing on Hosts with Existing td-agent Configuration

You can add the Zebrium output plugin on a host with existing td-agent configuration without running the Zebrium log collector installer.

1. Download the Zebrium output plugin from https://github.com/zebrium/ze-fluentd-plugin/releases/download/1.37.2/fluent-plugin-zebrium_output-1.37.2.gem.
2. Run the following command in the same directory where fluent-plugin-zebrium_output-1.37.2.gem is saved:

   ```
   sudo td-agent-gem install fluent-plugin-zebrium_output
   ```

3. Add Zebrium output configuration to the **/etc/td-agent/td-agent.conf** file.
   The following is an example configuration that duplicates log messages and sends one copy to Zebrium:

```
<match **>
  @type copy
  # Zebrium log collector
  <store>
    @type zebrium
    ze_log_collector_url "ZE_LOG_COLLECTOR_URL"
    ze_log_collector_token "ZE_LOG_COLLECTOR_TOKEN"
    ze_host_tags "ze_deployment_name=#
{Socket.gethostname},myapp=test2"
    @log_level "info"
    <buffer tag>
      @type file
      path "/var/td-agent/zebrium"
      flush_mode "interval"
      flush_interval "60s"
    </buffer>
  </store>
  <store>
      @type OTHER_OUTPUT_PLUGIN
      ...
  </store>
</match>
```

Configuration for td-agent

The configuration file for td-agent is at **/etc/td-agent/td-agent.conf**. The following parameters must be configured for your instance:

Parameter	Description	Note
ze_log_collector_url	Zebrium log host URL	Provided by Zebrium after your account has been created.
ze_log_collector_token	Authentication token	Provided by Zebrium after your account has been created.
path	Log files to read	Both files and file patterns are allowed. Files should be separated by comma. The default value is "/var/log/*.log,/var/log/syslog,/var/log/messages,/var/log/secure"
ze_host_tags	Host meta data	This parameter is optional. You can pass meta data in key-value pairs, the format is: "key1=value1,key2=value2". We suggest at least set one tag for deployment name: "ze_deployment_name=<your_deployment_name>"

Parameter	Description	Note
ze_host_in_ logpath	Log path component for remote host name	This parameter is optional. For situations where a remote host name is embedded in the log file directory path structure, e.g. "/var/log/remote/<host>/...", this can be used as the originating host for the log by setting this parameter to the path component to be used for the hostname. The value should be an integer, 1-based. In this example the configuration would be "ze_host_in_logpath=4".
ze_forward_tag	Tag to specify log-forwarded sources	This parameter is optional. It can be used to indicate sources that are being used for remote log forwarding, by specifying a specific fluentd "tag" to one or more sources. The default tag value is "ze_forwarded_ logs".
ze_path_map_file	Path mapping file	This parameter is optional. It allows embedded semantic data (ids, tags,configs) in logfile paths to be parsed and added to Zebrium log data. Set to the full path of a JSON file containing mapping information. Default is empty string. See ***Log Path Mapping***, below.

User Log Paths

User log paths can be configured via **/etc/td-agent/log-file-map.conf**. During log collector service startup, if **/etc/td-agent/log-file-map.conf** exists, log collector service script writes log paths defined in **/etc/td-agent/log-file-map.conf** to **/etc/td-agent/conf.d/user.conf**. Please note any user log paths configured at installation time via ZE_USER_LOG_PATHS must be added to **/etc/td-agent/log-file-map.conf** to avoid being overwritten.

```
{
  "mappings": [
    {
      "file": "/app1/log/error.log",
      "alias": "app1_error"
    },
    {
      "file": "/app2/log/error.log",
      "alias": "app2_error"
    },
    {
      "file": "/var/log/*.log",
      "exclude": "/var/log/my_debug.log,/var/log/my_test.log"
    }
  ]
}
```

Filtering Specific Log Events

If you wish to exclude certain sensitive or noisy events from being sent to Zebrium, you can filter them at the source collection point by doing the following:

1. Add the following in **/etc/td-agent/td-agent.conf** after other `@include`:

   ```
   @include conf.d/log_msg_filters.conf
   ```

2. Create a config file **/etc/td-agent/conf.d/log_msg_filters.conf** that contains the following:

   ```
   <filter TAG_FOR_LOG_FILE>
     @type grep
     <exclude>
       key message
       pattern /<PATTERN_FOR_LOG_MESSAGES>/
   </exclude>
   </filter>
   ```

3. Restart the td-agent with the following command:

   ```
   sudo systemctl restart td-agent
   ```

Example

Below is an example **log_msg_filters.conf** file for filtering out specific messages from a Vertica log file at **/fast1/vertica_catalog/zdb/v_zdb_node0001_catalog/vertica.log**.

In this example, the Fluentd tag for file is **node.logs.<*FILE_NAME_REPLACE_/_WITH_DOT*>** (replace all slashes with dots in the file path):

```
<filter node.logs.fast1.vertica_catalog.zdb.v_zdb_node0001_cata-
log.vertica.log>
  @type grep
  <exclude>
    key message
    pattern /^[^2]|^.[^0]|TM Merge|Authenticat|[Ll]oad *[Bb]alanc[ei]|\
[Session\]
<INFO>|\[Catalog\] <INFO>|\[Txn\] <INFO>|Init Session.*<LOG>/
  </exclude>
</filter>
```

Log Path Mapping

Log path mapping allows semantic information (like "tags", "ids", and "configs") to be extracted from log paths and passed to the Zebrium backend. For example, this can include log-file specific host information or business-related tags that are embedded in the path of the log file can be extracted.

Log path mapping is configured using a JSON file, with format:

```
{
  "mappings": {
    "patterns": [
      "regex1", ...
    ],
    "tags": [
      "tag_name", ...
    ],
    "ids": [
      "id_name",...
    ],
    "configs": [
       "config_name",...
    ]
  }
}
```

Set "patterns" to regular expressions to match the log file path. Each regex named capture in a matching regular expression will be compared to the "tags", "ids", and "configs" sections and added to the corresponding record section(s). Use the `ze_path_map_file` configuration parameter to specify the path to the JSON file.

Environment Variables

If the environment is using a proxy server to access the Internet then standard variables (e.g. http_proxy) should be configured prior to installation. For more information, see ***Operation with a Proxy Server***.

Usage

Start and Stop Fluentd

The Fluentd agent can be started or stopped with the following command:

```
sudo systemctl <start | stop> td-agent
```

Testing Your Installation

Once the collector has been deployed in your environment, your logs and anomaly detection will be available in the Zebrium UI.

Troubleshooting

In the event that Zebrium requires the collector logs for troubleshooting, the logs are located here:

1. Collector installation log: **/tmp/zlog-collector-install.log.***
2. Collector runtime log: **/var/log/td-agent/td-agent.log**

In case of an HTTP connection error, please check the spelling of the Zebrium host URL. Also check that any network proxy servers are configured appropriately.

Please contact Zebrium at support@zebrium.com if you need any assistance.

Operation with a Proxy Server

If the agent environment requires a non-transparent proxy server to be configured this should be done at two points:

- The standard http_proxy and https_proxy environment variables must be set in the local environment when the installer is run. This allows the installer to access the Internet to download necessary components.
- After installation is run the system service also needs to have the same environment variables available. This allows the Zebrium agent to communicate with the log host to send logs.

Setting the Proxy Server in a systemd Environment

If the agent service is run from systemd and a proxy server is in use, the service needs to have the appropriate proxy configuration added to systemd. This may not be needed if your system is already configured so that all systemd services globally use a proxy.

To do this, after the installation is performed edit the file **/etc/systemd/service/td-agent.service.d/override.conf** to add environment configuration lines for the proxy server, for example:

```
Environment=http_proxy=myproxy.example.com:8080
```

After this is done, run the following commands to reload the systemd daemon and start the service:

```
sudo systemctl daemon-reload
```

```
sudo systemctl restart td-agent
```

File Upload (ze Command)

ze is Zebrium's command-line interface for uploading log events from files or streams.

Features

up (upload)

Upload log event data to your Zebrium instance from a file or stream (stdin) with appropriate meta data.

help

Display help on ze command usage.

help_adv

Display advanced help on ze command usage.

Getting Started

Prerequisites

- Perl
- Perl JSON module
- curl
- Collector token from Zebrium, available from the Log Collector Setup page in the Zebrium user interface
- URL to your instance of Zebrium, available from the Log Collector Setup page in the Zebrium user interface

Installing ze

1. Download **bin/ze** from the Zebrium GitHub repository here: https://github.com/zebrium/ze-cli.
2. Move **bin/ze** to the appropriate bin directory in your PATH.
3. Make sure that the following ze command is executable:
   ```
   chmod 755 <path_to_ze_command>
   ```

NOTE: ze requires the Perl JSON module.

To install the Perl JSON module on Linux (Ubuntu):

```
sudo apt-get install libjson-perl
```

To install the Perl JSON module on Mac OS:

```
brew install cpanm
```

```
sudo cpanm install JSON
```

Configuration

No configuration is required. All options can be specified as command-line arguments. However, see the Setup section below for information on configuring your **.zerc** file.

Setup

For convenience, the collector TOKEN and URL can be specified in your $HOME/.zerc file.

Your ZE_LOG_COLLECTOR_URL and ZE_LOG_COLLECTOR_TOKEN are available in the the Zebrium UI under the Log Collector Setup page.

Example .zerc file:

```
url=<ZE_LOG_COLLECTOR_URL>
auth=<ZE_LOG_COLLECTOR_TOKEN>
```

Environment Variables

None

Usage

Run the following command for a complete list of command options:

```
ze help
```

Command Syntax and Options

```
ze up
        \
    [--url=<url>] [--auth=<token>]
          \
    [--file=<path>] [--log=<logtype>] [--host=<hostname>] [--svc-
grp=<service-group>]

    --url      - Zebrium Log Collector URL <ZE_LOG_COLLECTOR_URL> (omit to
look for url=<url> line in $HOME/.zerc)
    --auth     - Zebrium Log Collector Token <ZE_LOG_COLLECTOR_TOKEN>
(omit to look for auth=<token> line in $HOME/.zerc)
    --file     - Path to file being uploaded (omit to read from STDIN)
    --log      - Logtype of file being uploaded (omit to use base name
from file=<path> or 'stream' if STDIN)
    --host     - Hostname or other identifier representing the source of
the file being uploaded
    --svcgrp   - Service Group defines a failure domain boundary for anom-
aly correlation. This allows you to collect logs from multiple
                 applications or support cases and isolate the logs of one
from another so as not to mix these
                 in a Root Cause Report. This is referred to as a Service
Group in the Zebrium UI.

                 If omitted, Service Group will be set to "default".
Default is used to denote a service group that
                 represents shared-services. For example, a database that
is shared between two otherwise distinctly separate applications
                 would be considered a shared-service. In this example
scenario, you would set the Service Group for one application to "app01"
                 and to "app02" for the other application. For the data-
base logs, you would either omit the --svcgrp setting or you could
                 explicitly set it do "default" using --svcgrp=default.

                 With this configuration, Root Cause Reports will consider
correlated anomalies across:

                     "app01" log events and default (i.e. database logs)
and
                     "app02" log events and default (i.e. database logs)

                 but not across:

                     "app01" and "app02"
```

Advanced Options

Run the following command for a complete list of advanced options:

```
ze help_adv
```

Batch Uploads

The **ze** tool supports batch uploads. For more information, see *Zebrium batch uploads and ze CLI*.

Examples

1. Ingest three log files associated with the same support case "sr12345" (does not assume a **.zerc** configuration file exists):

   ```
   ze up --file=/casefiles/sr12345/messages.log --svcgrp=sr12345 --
   host=node01 --log=messages --url=<ZE_LOG_COLLECTOR_URL> --auth=<ZE_
   LOG_COLLECTOR_TOKEN>
   ```

   ```
   ze up --file=/casefiles/sr12345/application.log --svcgrp=sr12345 --
   host=node01 --log=application --url=<ZE_LOG_COLLECTOR_URL> --
   auth=<ZE_LOG_COLLECTOR_TOKEN>
   ```

   ```
   ze up --file=/casefiles/sr12345/db.log --svcgrp=sr12345 --host=db01 -
   -log=db --url=<ZE_LOG_COLLECTOR_URL> --auth=<ZE_LOG_COLLECTOR_TOKEN>
   ```

2. Ingest a continuous tail of **/var/log/messages**. When reading from a stream, such as STDIN, rather than from a file, ze requires the `-log` flag (assumes a **.zerc** configuration file exists):

   ```
   tail -f /var/log/messages | ze up --log=varlogmsgs --svcgrp=monitor01
   --help=mydbhost
   ```

Zebrium Batch Uploads and ze Command-line Interface

Zebrium batch uploads provide a way for grouping one or more related uploads so that they can be monitored and managed later as a unit. Each batch has a unique ID used to identify the batch.

Batch Uploads vs Service Groups

Batch uploads are different from service groups:

- **Service groups** provide a semantic connection across the data in uploads when looking for incidents.
- **Batch uploads** manage the overall phases of uploading and processing data in related logs. For example: monitoring if a batch is completed, how many lines of data have been ingested for, the time taken, and so forth.

Integration into ze CLI

Batch uploads are integrated into the **ze** command-line interface (CLI) in the following main ways:

- A standalone upload, using the **ze up** CLI, automatically has a batch created for it. The batch ID is displayed when the upload is finished so progress can be monitored using the **ze batch state** and **ze batch show** CLIs, described below.
- A set of related uploads, using the **ze up** CLI, can be associated with a specific batch ID that has been created earlier using the **ze batch begin** CLI. When all the logs for the batch are uploaded, the batch should be completed using **ze batch end**, or if there are errors the batch can be canceled using **ze batch cancel**. When **ze batch end** is used, all the logs for that batch are processed together by Zebrium.

ze batch CLI subcommand

The ze batch CLI subcommand allows batch uploads to be created, completed, cancelled and monitored. It has the following syntax:

```
ze batch begin [--url=<url>] [--auth=<auth>] [--batch_ID=<batch_ID>]
```

```
ze batch end [--url=<url>] [--auth=<auth>] -batch_ID=<batch_ID>
```

```
ze batch cancel [--url=<url>] [--auth=<auth>] -batch_ID=<batch_ID>
```

```
ze batch state [--url=<url>] [--auth=<auth>] -batch_ID=<batch_ID>
```

```
ze batch show [--url=<url>] [--auth=<auth>] -batch_ID=<batch_ID>
```

Examples

Uploading a Large Log and Monitoring its Progress

Upload a log file, on success the new batch ID is displayed, usually with a Processing state, meaning the log has been accepted by Zebrium and is being scanned for incidents:

```
ze up ... --file=myfile.log
```

```
State for batch upload baxyz1748ca is Processing
```

```
Sent sucessfully
```

Monitor the batch until processing completes:

```
watch ze batch state ... --batch_ID=baxyz1748ca
```

When the batch upload is completed, the state will change, typically to *Done*. For additional information, the **ze batch show** option can be used:

```
ze batch show ... --batch_id=baxyz1748ca

         Batch ID: baxyz1748ca
            State: Done
          Created: 2022-06-08T22:58:18Z
  Completion Time: 2022-06-08T22:59:45Z
  Expiration Time: 2022-06-10T22:59:45Z
            Lines: 377943
  Bundles Created: 2
Bundles Completed: 2
      Upload time: 0:17 min:sec
  Processing time: 1:20 min:sec
```

In this output, the expiration time refers to the batch upload metadata, not the uploaded logs or any detected incidents.

Uploading Multiple Logs to be Processed Together

The **ze batch begin** and **ze batch end** subcommands can be used to create a batch upload that spans several linked files. This allows them to be processed as a unit.

Begin a new batch:

```
ze batch begin ...
```

```
New batch upload ID: baxyz7357473aac1
```

Upload several logs as part of the same batch, using the **--batch_ID** option:

```
ze up --batch_ID=baxyz7357473aac1 ... --file=file1.log
```

```
ze up --batch_ID=baxyz7357473aac1 ... --file=file2.log
```

```
ze up --batch_ID=baxyz7357473aac1 ... --file=file3.log
```

End the batch:

```
ze batch end ... --batch_ID=baxyz7357473aac1
```

The batch upload can be monitored as in the previous example, using the **ze batch state** and **ze batch show** subcommands.

Batch_upload.sh script

The **batch_upload.sh** script wraps multiple logs with a batch upload in a single step.

Download **bin/batch_upload.sh** from the Zebrium GitHub repository here: https://github.com/zebrium/ze-cli.

This example uploads all files from **/var/log** using a custom host option to **ze**:

```
batch_upload.sh -o '--host=myhost' /var/log/*.log
```

Use `batch_upload.sh -h` for full options.

NOTE: The upload URL and authentication must be supplied on the command line using **-u** and **-a** options if you are not using a **.zerc** file. Do not use the `ze --url` or `ze --auth` options.

CloudWatch Collectors

The Zebrium CloudWatch collector ***ze-cloudwatch*** (lambda function for Amazon Web Services) sends logs to Zebrium for automated Anomaly detection. The Zebrium GitHub repository is located here: https://github.com/zebrium/ze-cloudwatch.

Preparation

1. Download Zebrium CloudWatch Lambda function package from https://github.com/zebrium/ze-cloudwatch/releases/download/1.47.0/zebrium_cloudwatch-1.47.0.zip.
2. If you have an existing Lambda function associated with the log group to be set up, you must go to AWS CloudWatch page and delete the existing subscription filter, otherwise you will get this error message: "An error occurred when creating the trigger: The log group host-log already has an enabled subscription filter associated with it."
3. If you do not have an existing role with Lambda execution permission, you should got to the AWS IAM service to create a role for running Lambda functions.

Installation

You will need to create a new Lambda function and then edit the function details.

1. Create a new Lambda function by going to the to AWS Lambda page.
2. Select **Author from scratch**.
3. Provide the following base information:
   - Function Name: zebrium-cloudwatch
   - Runtime: Node.js.12.x
4. Click on **Create function**.
5. To edit the function details, go to the ***Code entry type*** drop-down menu and choose *Upload a .zip file*.
6. Upload the Zebrium Lambda function package file that you just downloaded.
7. Enter *index.handler* for ***Handler setting***.
8. Choose *Node.js.12.x* for ***Runtime***.
9. For ***Execution role***, choose an existing role with Lambda execution permission.
10. Click on **Designer** and click on **Add a trigger**.
11. Type *CloudWatch Logs* and choose your log group.
12. Set the following environment variables:
    - ZE_DEPLOYMENT_NAME: Deployment name (Required)
    - ZE_HOST: Alternative Host Name (Optional)

- ZE_LOG_COLLECTOR_URL: ZAPI URL
- ZE_LOG_COLLECTOR_TOKEN: Auth token

13. Click **[Save]** to save your new Lambda function. New logs should appear on Zebrium web portal in a couple of minutes.

Configuration

No additional configuration is required.

Setup

No additional setup is required

Testing Your Installation

After the collector has been deployed in your CloudWatch environment, your logs and anomaly detection will be available in the Zebrium user interface.

Docker Container Log Collector

The Zebrium Docker container log collector, ***ze-docker-log-collector***, collects container logs and sends logs to Zebrium for automated incident detection. Our github repository is located here: https://github.com/zebrium/ze-docker-log-collector.

NOTE: A non-containerized Docker log collector based on Fluentd is also available here: https://docs.zebrium.com/docs/setup/docker_fluentd.

Getting Started

Docker

Use the following command to create a Docker log collector container:

```
sudo docker run -d --name="zdocker-log-collector" --restart=always \
                -v=/var/run/docker.sock:/var/run/docker.sock \
                -e ZE_LOG_COLLECTOR_URL="<ZE_LOG_COLLECTOR_URL>" \
                -e ZE_LOG_COLLECTOR_TOKEN="<ZE_LOG_COLLECTOR_TOKEN>" \
                -e ZE_HOSTNAME="<HOSTNAME>" \
                -e ZE_DEPLOYMENT_NAME="YOUR_DEPLOYMENT_NAME_HERE" \
                zebrium/docker-log-collector:latest
```

The **ZE_DEPLOYMENT_NAME** label essentially defines a failure domain boundary for anomaly correlation. This allows you to collect logs from multiple applications and isolate the logs of one application from another application so as not to mix these in a Root Cause Report. This is referred to as Service Groups in the Zebrium user interface.

Docker Compose

Use the following configuration file to deploy via **docker-compose** command:

```
version: '3.5'

services:
  zdocker-log-collector:
    image: zebrium/docker-log-collector:latest
    restart: always
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      ZE_LOG_COLLECTOR_URL: "<ZE_LOG_COLLECTOR_URL>"
      ZE_LOG_COLLECTOR_TOKEN: "<ZE_LOG_COLLECTOR_TOKEN>"
      ZE_DEPLOYMENT_NAME: "<YOUR_DEPLOYMENT_NAME_HERE>"
      ZE_HOSTNAME: "<HOSTNAME>"
```

AWS Elastic Container Service (ECS)

Add the following service to ECS on EC2 cluster configuration.

```
services:
  zdocker-log-collector:
    image: zebrium/docker-log-collector:latest
    restart: always
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      ZE_LOG_COLLECTOR_URL: "<ZE_LOG_COLLECTOR_URL>"
      ZE_LOG_COLLECTOR_TOKEN: "<ZE_LOG_COLLECTOR_TOKEN>"
      ZE_DEPLOYMENT_NAME: "<YOUR_DEPLOYMENT_NAME_HERE>"
```

To collect container logs from all nodes in an ECS cluster, the **zdocker-log-collector** service must be configured to run as an ECS daemon task.

To configure the daemon task:

1. Log in to the AWS console and navigate to the **ECS Clusters** section. Click into your cluster you run the Agent on.
2. On the **[Service]** tab, click **[Create]**.
3. For ***launch type***, select *EC2*.
4. For ***service type***, select *DAEMON*.
5. Type a service name and click **[Next step]**.
6. For the ***Load balance type*** option, select *None* and click **[Next step]**.
7. On the next page, click **[Next step]** without configuring Auto Scaling.
8. Review and click **Create Service**.

NOTE: ECS tasks must be configured to use the "json-file Log Driver for Zebrium" log collector to receive container logs. If there is a special log configuration on ECS instances, such as using the UserData section on an instance to set log configuration, those configurations might need to be modified or deleted.

Environment Variables

The following environment variables are supported by the collector:

Environment Variables	Description	Default Value	Note
ZE_LOG_COLLECTOR_URL	Zebrium log host server URL	None. Must be set by user	Provided by Zebrium after your account has been created.
ZE_LOG_COLLECTOR_TOKEN	Authentication token	None. Must be set by user	Provided by Zebrium after your account has been created.
ZE_HOSTNAME	Hostname of docker host	Empty. Optional	If ZE_HOSTNAME is not set, container hostname is used as source host for logs.
ZE_MAX_INGEST_SIZE	Maximum size of post request for Zebrium log server	1048576 bytes. Optional	Unit is in bytes
ZE_FLUSH_TIMEOUT	Interval between sending batches of log data to Zebrium log server.	30 seconds. Optional	Unit is in seconds. Please note Zebrium output plugin sends data immediately to log server when accumulated data reaches ZE_MAX_ INGEST_SIZE bytes.
ZE_FILTER_NAME	Collect logs for containers whose names match filter name pattern. These can include wildcards, for example, **my_ container1***	Empty. Optional	
ZE_FILTER_LABELS	Collect logs for containers whose labels match the labels as defined in ZE_FILTER_ LABELS. The format is: **label1:label1_ value,label2:label2_ value**. These can include wildcards, for example, **my_ label:xyz***	Empty. Optional	

Testing your Installation

After the Docker log collector software has been deployed in your environment, your container logs and incident detection will be available in the Zebrium user interface.

Logstash Collector

Configuring Logstash to Send Log Data to Zebrium

In Zebrium, you will need to retrieve your Zebrium URL and Auth Token for to configuring the Logstash HTTP Output plugin:

1. Log in to your Zebrium portal user account.
2. From the User menu area in Zebrium, click the Settings menu (hamburger) at top right.
3. Select *Integrations & Collectors*.
4. In the **Log Collectors** section, click **Other**.
5. Make a note of the values in the ***ZE_LOG_COLLECTOR_URL*** and ***ZE_LOG_COLLECTOR_TOKEN*** fields, as you will use them configuring Logstash.

Next, you will need to log into Logstash to complete the fields required by Zebrium.

Zebrium requires certain fields (keys) to be defined for each log event. These definitions are part of the "filter" section in the logstash configuration.

There are four types of Zebrium fields that require definition in the Logstash filter configuration for proper Incident detection in Zebrium. An example Logstash configuration is shown below the table:

Type	Description	Key Name	Key Definition	Requirement
Time	Timestamp/time zone of each log event.	@timestamp	Timestamp of each log event (rather than the time the event was processed by Logstash if possible).	Required.
		@ze_timezone	Time zone of each log event. E.g. "America/Los_Angeles"	Optional. **Note:** UTC is the default.
Log Generator	Indicates the source of the log event.	@ze_deployment_name	Identifies the environment or application domain. In the Zebrium UI this is known as the Service Group (see Note on Service Groups below) E.g. "production", "dev", "acme_calendar_app"	Recommended.
		@ze_host	Host name identifier	Required.
		@ze_logtype	The basename of the log source. E.g. "access.log", "syslog". In the Zebrium UI, it will be the logtype. In the container world, this would probably be the app name.	Required.

Type	Description	Key Name	Key Definition	Requirement
Log Events Wrapped in JSON	If the application or host log events are simply wrapped in a JSON and contain a field like "message" : "2020-10-23 04:17:37 mars INFO systemd[1]: Stopped PostgreSQL RDBMS.", then these keys need to be defined.	@ze_msg	If the JSON contains a field representing a typical "log event" <PREFIX INFORMATION> <EVENT TEXT>, then this Zebrium key should be set to the value of that "log event". Zebrium's machine learning with then structure this field into an Event Type (etype) used for Incident detection.	Required (if your log events are wrapped in JSON).
		@ze_sev	If @ze_msg does not contain a severity, then this field can be used to explicitly set the severity based on some other criteria or field from the payload.	Optional.
External ID Mapping	Map events in Zebrium to corresponding events in Elasticsearch	@ze_xid	Assign a unique id (UUID) to every log event so that events in Zebrium can be mapped to corresponding events in Elasticsearch through a common UUID.	Required (if using Kibana/Elasticsearch to view Zebrium Incidents).

Service Groups

A Service Group defines a failure domain boundary for anomaly correlation. This allows you to collect logs from multiple applications and isolate the logs of one from another so as not to mix these in a Root Cause Report. This is referred to as a Service Group in the Zebrium user interface.

If you are uploading multiple logs from different services in the same application, you would specify the same Service Group for each log event from that application. For example, if you have a database log, an application log, and a middleware log for the Acme Calendar application. You would use an appropriate service group when uploading all files from that application, such as **acme_calendar_app**.

Configuring Logstash Filters for Zebrium Required Fields (in Logstash)

1. Edit the appropriate Logstash configuration file to define the required Zebrium with Elastic Stack filter definitions. All of these definitions are within the `filter { }` section of the configuration.
2. **TIME FIELDS**
   - @timestamp should contain the timestamp from the log event (not the timestamp when processed by Logstash). This is important for proper incident detection in Zebrium.

- Processing multi-line events should be enabled such that child log event lines are concatenated to the parent event with newlines.

- The following shows an example configuration for meeting these requirements:

```
#----------------------------------------------------------------
-------#
# Input Filter definition for processing multi-line events (if
needed) #
#----------------------------------------------------------------
-------#
codec => multiline {
  pattern => "^%{TIMESTAMP_ISO8601}"
  negate  => true
  what    => "previous"
}

#----------------------------------------------------------------
---------------------------#
# Grok and Date Filter for capturing log event timestamp in
@timestamp                     #
# If it is not possible to easily capture the event timestamp as
@timestamp as shown here, #
# it is OK to leave @timestamp as-is (i.e. use the logstash
generated timestamp)           #
#----------------------------------------------------------------
---------------------------#
grok {
  match => [ "message", "(?m)%{TIMESTAMP_ISO8601:logdate}" ] #
Note the multi-line capture pattern (?m)
}
date {
  # This will set @timestamp
  match        => [ "logdate", "yyyy-MM-dd HH:mm:ss,SSS", "yyyy-
MM-dd HH:mm:ss" ]
  timezone     => "America/Los_Angeles"
  remove_field => ["logdate"]
}

#---------------------------------------#
# Capture @ze_timezone                  #
# If not specified, UTC will be assumed #
#---------------------------------------#
mutate {
    add_field => { @ze_timezone => "America/Los_Angeles" }  #
Specify timezone (IANA TZ Names)
if your log timestamps are missing the timezone info, otherwise
UTC is assumed (optional).
}
```

3. **LOG GENERATOR FIELDS**

```
#------------------------------------------------------------------#
# Mutate Filter for capturing logtype, host and gid                #
# PLEASE READ CAREFULLY - YOU MUST SUBSTITUTE THE                  #
# RIGHT-HAND SIDE OF THE ASSIGNMENTS WITH YOUR FIELD NAMES/VALUES #
#------------------------------------------------------------------#
mutate {
   add_field => { "@ze_deployment_name" => "%{my_deployment}"  } #
assumes field "my_deployment"  is part of the payload (recommended)
   add_field => { "@ze_host"            => "%{host}"           } #
assumes field "host"           is part of the payload (required)
   add_field => { "@ze_logtype"         => "%{logtype}"        } #
assumes field "logtype"        is part of the payload (required)
}
```

4. **LOG EVENTS WRAPPED IN JSON FIELDS**
   This configuration is **required** if you have a "message" field in the JSON containing an unstructured log event. In that case, we will structure the message and create an Event-Type automatically for Incident Detection.

```
#------------------------------------------------------------------#
# Required if your log events are wrapped in JSON                  #
# PLEASE READ CAREFULLY - YOU MUST SUBSTITUTE THE                  #
# RIGHT-HAND SIDE OF THE ASSIGNMENTS WITH YOUR FIELD NAMES/VALUES #
#------------------------------------------------------------------#
mutate {
    add_field => { "@ze_msg"  => "%{message}"         } # Capture the
unstructured log event from the message field - Zebrium will
automatically structure this into an etype (required)
    add_field => { "@ze_sev"  => "%{[log][severity]}" } # Capture the
severity explicitly since "message" field does not contain severity
(optional)
    add_field => { "@ze_pfx"  => "%{[log][process]}"  } # Capture the
process name and add to the log event prefix so its part of the
automatic structuring (optional)
}
```

5. **EXTERNAL ID MAPPING FIELD**

NOTE: This is not part of a mutate filter.

```
uuid {
  target => "@ze_xid"  # Generate a Unique ID and assign to @ze_xid
}
```

6. **SAVE YOUR CONFIGURATION FILE.**

Configuring Log Event Output to Zebrium (in Logstash)

1. Edit the appropriate Logstash configuration file to define the required Zebrium with Elastic Stack output definition.

2. Add the following Output Filter definition for Zebrium and substitute ZE_LOG_COLLECTOR_URL and ZE_LOG_COLLECTOR_TOKEN with the values from step 5 of *Configuring Logstash to Send Log Data to Zebrium*, above.

```
output {
  if <SOME_CONDITION_IS_TRUE> {
    http {
      format      => "json_batch"
      http_method => "post"
      url         => "<ZE_LOG_COLLECTOR_URL>/log/api/v2/ingest?log_
source=logstash&log_format=json_batch"
      headers     => ["authtoken", "<ZE_LOG_COLLECTOR_TOKEN>"]
    }
  }
}
```

3. **SAVE YOUR CONFIGURATION FILE.**

Reload Logstash Configuration

Reload your Logstash configuration to pick up all changes. Data will now be ingesting into Zebrium.

Complete Example for filebeat and winlogbeat Data

It is highly recommended you read this carefully and follow the sample provided.

```
input {
  beats {
    port => 5044
  }
}

filter {

  #--------------------------------------------#
  # Add the UUID to all events before          #
  # cloning a copy for the zebrium only fields #
  #--------------------------------------------#
  uuid {
    target => "@ze_xid"  # Generate a Unique ID and assign to @ze_xid
  }

  #---------------------------------------------#
  # Make a clone of the message so we only send #
  # Zebrium add-ons to Zebrium and not to other #
  # existing outputs like elastic               #
  #---------------------------------------------#
  clone {
    clones => ['zebrium']
  }

  #-----------------------------------#
  # Add Zebrium specifics to the clone #
  #-----------------------------------#
  if( [type] == 'zebrium' ) {
    #-------------------------------------------------------------#
    # Common attributes across filebeats, winlogbeats #
    #-------------------------------------------------------------#
    mutate {
      add_field => { "[@metadata][zebrium]" => true }
    }
    mutate {
      add_field => { "@ze_deployment_name" => "mydeployment01"  }
    }
    if( [host][hostname] ) {
      mutate {
        add_field => { "@ze_host" => "%{[host][hostname]}" }
      }
    } else if ( [host][name] ) {
      mutate {
        add_field => { "@ze_host" => "%{[host][name]}" }
      }
    }
    if( [@ze_host] ) {
      mutate {
        gsub => [ "@ze_host", "^([^\.]+)", "\1" ] # Use hostname without
fully qualified domain
      }
    } else {
      mutate {
        add_field => { "@ze_host" => "unknown" }
      }
    }
    #-------------------------------#
    # winlogbeat specific captures #
    #-------------------------------#
    if( [agent][type] and [agent][type] == "winlogbeat" ) {
      if( [log][level] ) {
        mutate {
          add_field => { "@ze_sev" => "%{[log][level]}" }
        }
      }
      if( [message] ) {
        mutate {
          add_field => { "@ze_msg"  => "%{[message]}"  }
          add_field => { "@ze_time" => "%{@timestamp}" }
        }
      }
      if( [event][provider] ) {
        mutate {
          add_field => { "@ze_logtype" => "%{[event][provider]}" }
        }
      } else if( [event][module] ) {
        mutate {
          add_field => { "@ze_logtype" => "%{[event][module]}" }
        }
      } else {
        mutate {
          add_field => { "@ze_logtype" => "winlogbeat" }
        }
      }
      if [@ze_logtype] and [@ze_logtype] =~ "^Microsoft\-Windows\-" {
        # Sometimes we see provider start with Microsoft-Windows-, so get
rid the that extraneous string and pickup the reaminder as the logtype
        mutate {
          gsub => [ "@ze_logtype", "^Microsoft\-Windows\-(.*)$", "\1" ]
        }
      }
    }
    #-----------------------------#
    # filebeat specific captures #
    #-----------------------------#
    if( [agent][type] and [agent][type] == "filebeat" ) {
      if( [message] ) {
        mutate {
          add_field => { "@ze_msg" => "%{[message]}" }
        }
      }
      if( [log][file][path] ) {
        grok {
          match => [ "[log][file][path]","%{GREEDYDATA}[\\/]%{GREEDYDATA:-
logtype}\.log" ]
        }
        mutate {
          add_field    => { "@ze_logtype" => "%{logtype}" }
          remove_field => [ "logtype" ]
        }
        mutate {
          # Sometimes the log filename starts with the hostname, remove
that so all logs of the same type are grouped together
          gsub => [ "@ze_logtype", "^%{@ze_host}([^\d]+).*$", "\1" ]
        }
      } else {
        mutate {
          add_field => { "@ze_logtype" => "filebeatlog" }
        }
      }
    }
  } # END OF ZEBRIUM
}

output {
  # SEND ZEBRIUM DATA TO ZEBRIUM ONLY
  if [@metadata][zebrium] {
    http {
        format      => "json_batch"
        http_method => "post"
        url         => "<ZE_LOG_COLLECTOR_URL>/log/api/v2/ingest?log_
source=logstash&log_format=json_batch"
        headers     => ["authtoken", "<ZE_LOG_COLLECTOR_TOKEN>"]
        proxy       => "<proxy>"
    }
  # THEN SEND DATA AS WAS DONE BEFORE ADDING ZEBRIUM
  } else if [@metadata][pipeline] {
    elasticsearch {
        hosts => ["https://localhost:9200"]
        index => "%{[@metadata][beat]}-%{[@metadata][version]}"
        pipeline => "%{[@metadata][pipeline]}"
        ssl => true
        ssl_certificate_verification => true
        cacert => '/etc/logstash/certs/ca.crt'
        user => elastic
        password => "${ES_PW}"
    }
  } else {
    elasticsearch {
        hosts => ["https://localhost:9200"]
        index => "%{[@metadata][beat]}-%{[@metadata][version]}"
        pipeline => beats
        ssl => true
        ssl_certificate_verification => true
        cacert => '/etc/logstash/certs/ca.crt'
        user => elastic
        password => "${ES_PW}"
    }
  }
}
```

Syslog Forwarder

The Zebrium Syslog Forwarder accepts both syslogs and raw logs and forwards them to Zebrium for automated Anomaly detection.

Our github repository is located here: https://github.com/zebrium/ze-log-forwarder.

Preparation

1. By default, the syslog forwarder container uses TCP and UDP port 5514 for syslog, and TCP port 5170 for TCP forwarding. Please make sure clients can reach host IP on those ports.
2. For syslog forwarding, make sure the host firewall does not block port 5514 for both TCP and UDP. For TCP forwarding, make sure TCP port 5170 is open.
3. Install Docker software if it is not installed.

Forward Syslog

Installation

1. To support syslog over TCP and UDP, run the following command as root, and be sure to replace items in `<BRACKETS>` with real values:

```
docker run -d --name="zlog-forwarder" --restart=always \
 -p 5514:5514/tcp \
 -p 5514:5514/udp \
 -e ZE_LOG_COLLECTOR_URL="<ZE_LOG_COLLECTOR_URL>" \
 -e ZE_LOG_COLLECTOR_TOKEN="<ZE_LOG_COLLECTOR_TOKEN>" \
 -e ZE_DEPLOYMENT_NAME="<DEPLOYMENT_NAME>" \
 zebrium/log-forwarder:latest
```

2. To support syslog over TLS and UDP, create or copy the root certificate, the host certificate, and the host private key files to a directory on the host that will be running log-forwarder container.

3. Run the following command as root:

```
docker run -d --name="zlog-forwarder" --restart=always \
 -p 5514:5514/tcp \
 -p 5514:5514/udp \
 -v <USER_SERVER_CERTS_KEY_DIR>:/fluentd/tls
 -e ZE_SYSLOG_PROTOCOL="tls" \
 -e ZE_LOG_COLLECTOR_URL="<ZE_LOG_COLLECTOR_URL>" \
 -e ZE_LOG_COLLECTOR_TOKEN="<ZE_LOG_COLLECTOR_TOKEN>" \
 -e ZE_DEPLOYMENT_NAME="<DEPLOYMENT_NAME>" \
 zebrium/log-forwarder:latest
```

Client Configuration

1. Use the host IP as the syslog server IP address, and port 5514 for syslog port.
2. To configure rsyslog:

   - To use UDP, add the following to the end of the rsyslog configuration file ***.* @<LOG_FORWARDER_HOST_IP>:5514**
   - To use TCP, add the following to the end of the rsyslog configuration file ***.* @@<LOG_FORWARDER_HOST_IP>:5514**
   - To use TLS:
     - Copy **client_configs/rsyslog/25-zebrium.conf** to **/etc/rsyslog.d/**.
     - Open the file, replace **CLIENT_SSL_CERT_PATH** with the real client SSL certificate path, change **SERVER_HOST** to the hostname running log-forwarder container, and change **SERVER_DOMAIN_NAME** to the domain of the host running the log-forwarder container.
     - Restart the rsyslog service.

Setup

No additional setup is required

Forward Log via TCP

Installation

Run the following command as root, and be sure to replace items in `<BRACKETS>` with real values:

```
docker run -d --name="zlog-forwarder" --restart=always \
    -p 5170:5170/tcp
    -e ZE_LOG_COLLECTOR_URL="<ZE_LOG_COLLECTOR_URL>" \
    -e ZE_LOG_COLLECTOR_TOKEN="<ZE_LOG_COLLECTOR_TOKEN>" \
    -e ZE_DEPLOYMENT_NAME="<DEPLOYMENT_NAME>" \
    -e ZE_TCP_HOSTNAME="<TCP_FORWARDER_HOSTNAME>" \
    -e ZE_TCP_LOGBASE="tcp_forwarder" \
    -e ZE_TIMEZONE="<TIME_ZONE>" \
    zebrium/log-forwarder:latest
```

where `<TIME_ZONE>` is timezone of the log messages, such as `"UTC"` or `"EDT"`.

Setup

No additional setup is required

Testing your installation

After the log forwarder software has been deployed in your environment, your logs and anomaly detection will be available in the Zebrium user interface.

Chapter 4

Working with Suggestions and Root Cause Reports

Overview

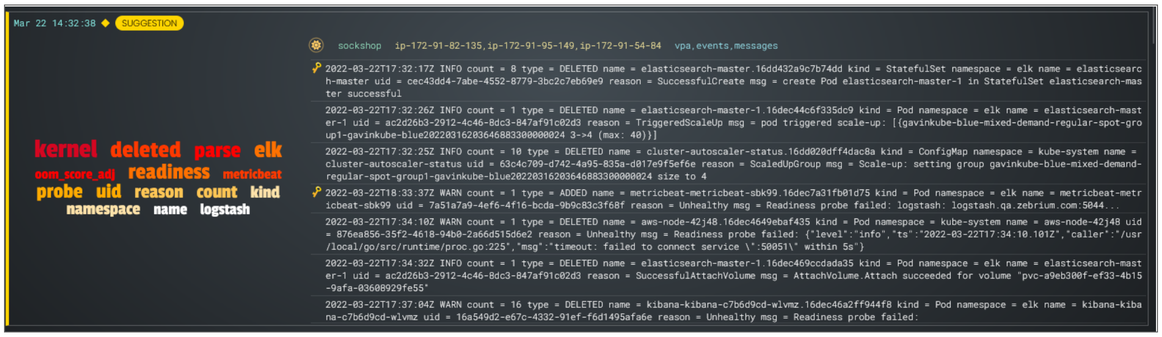
This chapter covers the following topics:

Suggestions in Zebrium

Zebrium Root Cause as a Service (RCaaS) uses unsupervised machine learning on logs to automatically find the root cause of software problems. It does not require manual rules or training, and it typically achieves accuracy within 24 hours.

As Zebrium ingests logs, the Zebrium artificial-intelligence machine-learning (AI/ML) engine analyzes the logs, looking for abnormal log line clusters that resemble problems, such as abnormally correlated rare and error events from across all log streams.

When the AI/ML engine detects one of these "abnormal" clusters, it generates a ***suggestion***, which appears on the **Alerts** page (the home page) of the Zebrium user interface:



```
sockshop   ip-172-91-82-135,ip-172-91-95-149,ip-172-91-54-84   vpa,events,messages
2022-03-22T17:32:17Z INFO count = 8 type = DELETED name = elasticsearch-master.16dd432a9c7b74dd kind = StatefulSet namespace = elk name = elasticsearc
h-master uid = cec43dd4-7abe-4552-8779-3bc2c7eb69e9 reason = SuccessfulCreate msg = create Pod elasticsearch-master-1 in StatefulSet elasticsearch-mas
ter successful
2022-03-22T17:32:26Z INFO count = 1 type = DELETED name = elasticsearch-master-1.16dec44c6f335dc9 kind = Pod namespace = elk name = elasticsearch-mast
er-1 uid = ac2d26b3-2912-4c46-8dc3-847af91c02d3 reason = TriggeredScaleUp msg = pod triggered scale-up: [{gavinkube-blue-mixed-demand-regular-spot-gro
up1-gavinkube-blue20220316203646883300000024 3->4 (max: 40)}]
2022-03-22T17:32:25Z INFO count = 10 type = DELETED name = cluster-autoscaler-status.16dd020dff4dac8a kind = ConfigMap namespace = kube-system name =
cluster-autoscaler-status uid = 63c4c709-d742-4a95-835a-d017e9f5ef6e reason = ScaledUpGroup msg = Scale-up: setting group gavinkube-blue-mixed-demand-
regular-spot-group1-gavinkube-blue20220316203646883300000024 size to 4
2022-03-22T18:33:37Z WARN count = 1 type = ADDED name = metricbeat-metricbeat-sbk99.16dec7a31fb01d75 kind = Pod namespace = elk name = metricbeat-metr
icbeat-sbk99 uid = 7a51a7a9-4ef6-4f16-bcda-9b9c83c3f68f reason = Unhealthy msg = Readiness probe failed: logstash: logstash.qa.zebrium.com:5044...
2022-03-22T17:34:10Z WARN count = 1 type = DELETED name = aws-node-42j48.16dec4649ebaf435 kind = Pod namespace = kube-system name = aws-node-42j48 uid
= 876ea856-35f2-4618-94b0-2a66d515d6e2 reason = Unhealthy msg = Readiness probe failed: {"level":"info","ts":"2022-03-22T17:34:10.101Z","caller":"/usr
/local/go/src/runtime/proc.go:225","msg":"timeout: failed to connect service \":50051\" within 5s"}
2022-03-22T17:34:32Z INFO count = 1 type = DELETED name = elasticsearch-master-1.16dec469ccdada35 kind = Pod namespace = elk name = elasticsearch-mast
er-1 uid = ac2d26b3-2912-4c46-8dc3-847af91c02d3 reason = SuccessfulAttachVolume msg = AttachVolume.Attach succeeded for volume "pvc-a9eb300f-ef33-4b15
-9afa-03608929fe55"
2022-03-22T17:37:04Z WARN count = 16 type = DELETED name = kibana-kibana-c7b6d9cd-wlvmz.16dec46a2ff944f8 kind = Pod namespace = elk name = kibana-kiba
na-c7b6d9cd-wlvmz uid = 16a549d2-e67c-4332-91ef-f6d1495afa6e reason = Unhealthy msg = Readiness probe failed:
```

A suggestion is a summary report that contains the following main elements:

- ***AI-generated title***. This title is generated using the GPT-3 language model, which is trained on a large volume of public data. As a result, the titles might not always be accurate and should not be relied upon alone to make a decision about a suggestion.
- ***Word Cloud***. A set of relevant words chosen by the AI/ML engine from the log lines contained in the alert.
- ***Root Cause (RCA) Report Summary***. The report contains the actual cluster of anomalous log lines that was identified by the AI/ML engine. Up to eight of these log lines are shown in the summary view. You can click anywhere in the summary to view the full RCA report.
- ***Alert Key***. One or two log lines, denoted with a key icon (🔑), that are used to identify the suggestion if this type of suggestion occurs again. The alert keys make up an ***alert rule***.

IMPORTANT: Suggestions are generated when the AI/ML engine finds a cluster of correlated anomalies in your logs that resembles a problem. However, this does not mean that all suggestions relate to actual important problems. This is especially true during the first few days of using Zebrium, as the AI/ML engine learns the normal patterns in your logs.

When you start getting suggestions on the **Alerts** page, you can review the word clouds and event logs that display in the summary views for the Root Cause reports for the suggestions. As a best practice, identify a specific timeframe when a possible problem occurred, and then start looking at the reports that have the most interesting or relevant information related to the possible root cause of the problem.

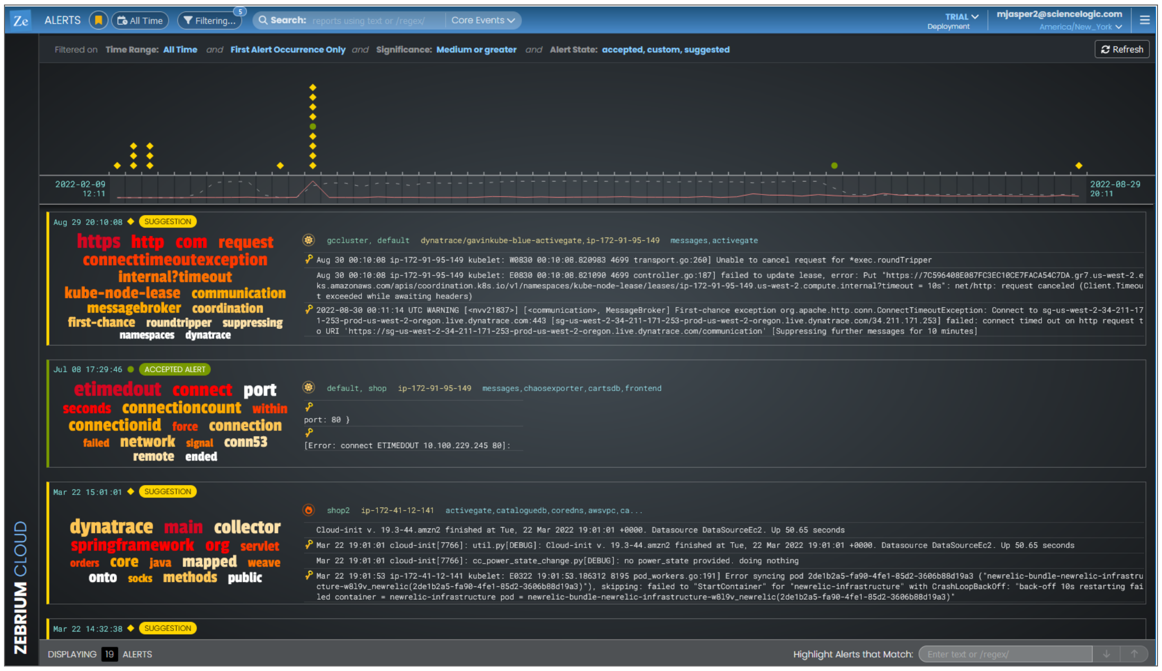
You can choose to "accept" or "reject" a suggestion. For more information, see *Assessing Suggestions*.

You can also decide on the action to take if the same kind of alert type occurs again, such as sending a notification to Slack, email, or another type of notification. For more information, see *Enabling Notification Channels*.

If you currently use a monitoring tool from ScienceLogic, Datadog, New Relic, Elastic, Dynatrace or AppDynamics, you can configure an observability dashboard integration that lets you view Zebrium suggestions on your existing monitoring dashboards. For more information, see *Configuring Observability Dashboard Integrations*.

Managing Suggestions in the Zebrium User Interface

The **Alerts** page is also the Zebrium home page, and you can get to this page by clicking the **Ze** icon (Ze) at the top left of any page in the Zebrium user interface:


Ze
ALERTS
All Time
Filtering...
5
Search: reports using text or /regex/
Core Events
TRIAL
Deployment
mjasper2@sciencelogic.com
America/New_York
Filtered on Time Range: All Time and First Alert Occurrence Only and Significance: Medium or greater and Alert State: accepted, custom, suggested
Refresh
2022-02-09 12:11
2022-08-29 20:11
Aug 29 20:10:08
SUGGESTION
https http com request connecttimeoutexception internal?timeout kube-node-lease communication messagebroker coordination first-chance roundtripper suppressing namespaces dynatrace
Jul 08 17:29:46
ACCEPTED ALERT
etimedout connect port seconds connectioncount within connectionid force connection failed network signal conn53 remote ended
Mar 22 15:01:01
SUGGESTION
dynatrace main collector springframework org servlet orders core java mapped weave onto socks methods public
Mar 22 14:32:38
SUGGESTION
ZEBRIUM CLOUD
DISPLAYING 19 ALERTS
Highlight Alerts that Match: Enter text or /regex/

```
gccluster, default  dynatrace/gavinkube-blue-activegate,ip-172-91-95-149  messages,activegate
Aug 30 00:10:08 ip-172-91-95-149 kubelet: W0830 00:10:08.820983 4699 transport.go:260] Unable to cancel request for *exec.roundTripper
Aug 30 00:10:08 ip-172-91-95-149 kubelet: E0830 00:10:08.821090 4699 controller.go:187] failed to update lease, error: Put "https://7C596408E087FC3EC10CE7FACA54C7DA.gr7.us-west-2.e
ks.amazonaws.com/apis/coordination.k8s.io/v1/namespaces/kube-node-lease/leases/ip-172-91-95-149.us-west-2.compute.internal?timeout = 10s": net/http: request canceled (Client.Timeou
t exceeded while awaiting headers)
2022-08-30 00:11:14 UTC WARNING [<nvv21837>] [<communication>, MessageBroker] First-chance exception org.apache.http.conn.ConnectTimeoutException: Connect to sg-us-west-2-34-211-17
1-253-prod-us-west-2-oregon.live.dynatrace.com:443 [sg-us-west-2-34-211-171-253-prod-us-west-2-oregon.live.dynatrace.com/34.211.171.253] failed: connect timed out on http request t
o URI 'https://sg-us-west-2-34-211-171-253-prod-us-west-2-oregon.live.dynatrace.com/communication' [Suppressing further messages for 10 minutes]
```

```
default, shop  ip-172-91-95-149  messages,chaosexporter,cartsdb,frontend
port: 80 }
[Error: connect ETIMEDOUT 10.100.229.245 80]:
```

```
shop2  ip-172-41-12-141  activegate,cataloguedb,coredns,awsvpc,ca...
Cloud-init v. 19.3-44.amzn2 finished at Tue, 22 Mar 2022 19:01:01 +0000. Datasource DataSourceEc2. Up 50.65 seconds
Mar 22 19:01:01 cloud-init[7766]: util.py[DEBUG]: Cloud-init v. 19.3-44.amzn2 finished at Tue, 22 Mar 2022 19:01:01 +0000. Datasource DataSourceEc2. Up 50.65 seconds
Mar 22 19:01:01 cloud-init[7766]: cc_power_state_change.py[DEBUG]: no power_state provided. doing nothing
Mar 22 19:01:53 ip-172-41-12-141 kubelet: E0322 19:01:53.186312 8195 pod_workers.go:191] Error syncing pod 2de1b2a5-fa90-4fe1-85d2-3606b88d19a3 ("newrelic-bundle-newrelic-infrastru
cture-w819v_newrelic(2de1b2a5-fa90-4fe1-85d2-3606b88d19a3)"), skipping: failed to "StartContainer" for "newrelic-infrastructure" with CrashLoopBackOff: "back-off 10s restarting fai
led container = newrelic-infrastructure pod = newrelic-bundle-newrelic-infrastructure-w819v_newrelic(2de1b2a5-fa90-4fe1-85d2-3606b88d19a3)"
```

This page displays a list of filtering and search options at the top of the page. You can use these filters to manage the number of suggestions and alerts that display on the **Alerts** page. There is also a ***Search*** bar for text or regular expression (regex) searches, and a toggle for *Core Events* and *All Events*. For more information about filtering, see *Using the Filters on the Alerts Page in Zebrium*.

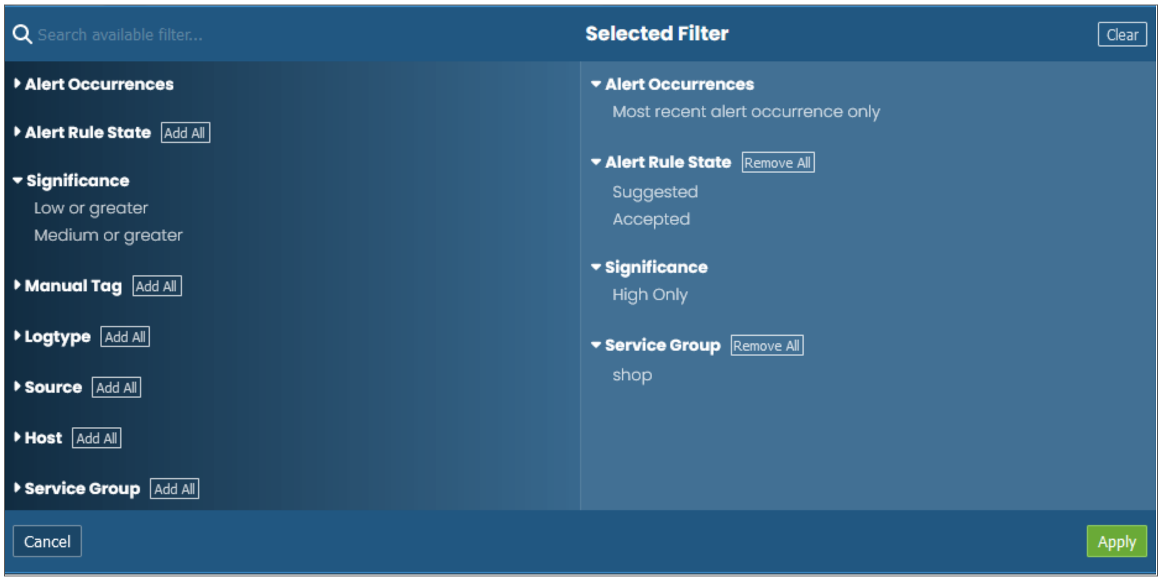
Below the filters is a Timeline widget that displays a set of icons organized by time. These icons represent all known suggestions, accepted alerts, and custom alerts for a specific period of time. For more information about the Timeline widget, see *Using the Timeline Widget on the Alerts Page*.

The Root Cause (RCA) reports that correspond to the items in the Timeline widget display in a summary view below the widget. If you click an icon in the Timeline widget, the RCA report for that icon moves to the top of the summary view below the widget. For more information about RCA reports, see *Root Cause Reports*.

Using the Filters on the Alerts Page in Zebrium

At the top of the **Alerts** page, the **[Time Range]** button (Today) lets you change the time frame of the alerts. The default time frame for displaying alerts is the last 7 days.

In addition, you can click the **[Filtering]** button to select filters that will control which RCA reports display on the **Alerts** page. The **Selected Filter** dialog appears:



You can filter by log types (which typically match container names), service groups, hosts, tags, and more. Any RCA reports that match these attributes will be shown in the filtered view.

Most of the filters on the **Selected Filter** dialog are self-explanatory. However, you should pay attention to the following filters, especially if you are not seeing the reports you want to see on the **Alerts** page:

- ***Alert Occurrences***. By default, only the first occurrence of an alert will be shown in the list, so that if the same type of alert occurs more than once, you will only see its first instance. You can change this if you wish to see all alert occurrences, the most recent alert occurrences, or other options.
- ***Alert Rule State***. You can filter by some or all custom alerts, suggestions, accepted alerts, or rejected alerts.
- ***Significance***. The AI/ML engine assigns a value of Low, Medium, or High to each alert, based on how likely that alert is related to a problem. By default, only alerts with a significance of Medium and High are shown on the **Alerts** page, so if you want to also see alerts with Low significance, select *Low or greater* for this filter.

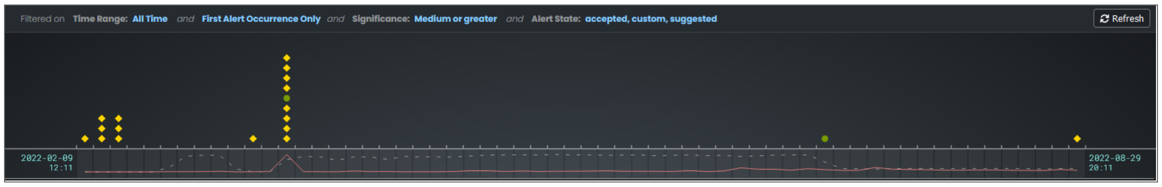
You can further filter the log events by typing a text string or a PCRE2-compliant regular expression into the **Search** field at the top of the page. Regular expression filters should use the syntax "`/regex/`". You can also change the search scope by toggling between *Core Events* and *All Events* on the **Search** field.

TIP: You can also highlight any desired alphanumeric strings within the visible log events by typing text or a regular expression in the ***Highlight Events that Match*** field at the bottom right of the **Alerts** page. This field also displays on the **RCA Report** pages.

If you do not see a report in a time of interest where you believe a problem occurred, the AI/ML engine might have suppressed it by the existing ***Significance*** filter settings. You can also force the AI/ML engine to do a deep scan and create a report on demand by clicking the **[Scan for RC]** button on the **Settings** menu (☰) and specifying a time of interest.

Using the Timeline Widget on the Alerts Page

The Timeline widget displays at the top of the **Alerts** page, and it lets you control which RCA report summaries display in the lower portion of the page:


Filtered on **Time Range:** All Time *and* First Alert Occurrence Only *and* **Significance:** Medium or greater *and* **Alert State:** accepted, custom, suggested

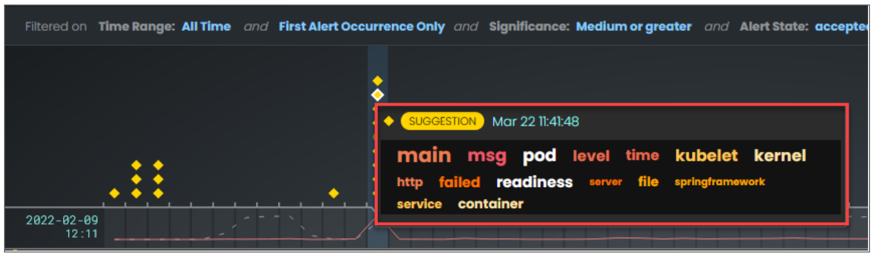
NOTE: The Timeline widget displays a list of the currently active filters at the top of the widget. For more information about filtering, see *Using the Filters on the Alerts Page in Zebrium*.

The main section of the Timeline widget contains a time-based chart with different icons that represent the following Zebrium elements:

- **Suggestion** (◆). A yellow diamond represents a potential problem found by the AI/ML engine. If you go to the **RCA Report** page for that suggestion, you can choose to accept or reject that suggestion.
- **Accepted Alert** (●). A green circle represents a suggestion that you or another Zebrium user has accepted.
- **Custom Alert** (▲). A blue triangle represents a custom alert, which you or another user defined by writing a regular expression in Zebrium that searches for a specific pattern.
- **Rejected Alert** (▼). A red triangle represents a suggestion that you or another Zebrium user has rejected as not relevant to your environment.

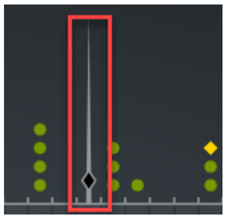
When you click an icon in the Timeline widget, the summary view for the corresponding RCA report for that suggestion or alert moves the top of list below the Timeline widget. Click anywhere in the summary view to open its **RCA Report** page.

When you hover over an icon in the chart, a pop-up window appears with date and time information about that specific suggestion, along with a title and word cloud that contains suggestions and information about the likely root cause:


Filtered on Time Range: All Time and First Alert Occurrence Only and Significance: Medium or greater and Alert State: accepte
SUGGESTION Mar 22 11:41:48
main msg pod level time kubelet kernel
http failed readiness server file springframework
service container
2022-02-09
12:11

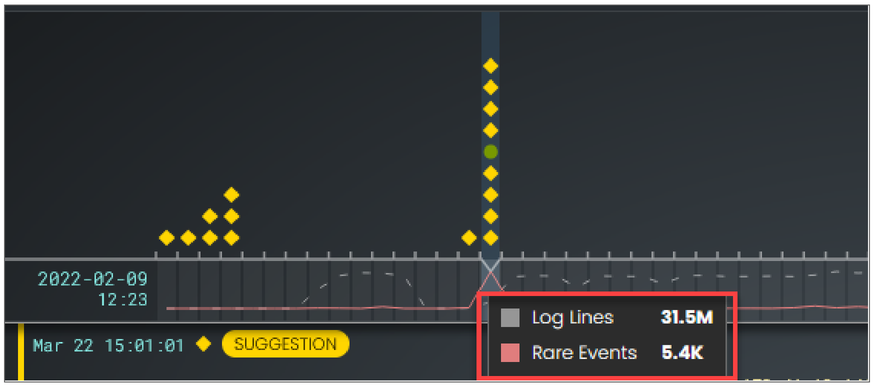
The Timeline widget also includes the following graphical elements:

- **Spike**. A gray vertical line appears on the widget if too many suggestions or alerts exist for a specific time for the user interface to show them all:



  You can click and drag the spike to the left or right to zoom in so you can see all of the suggestions for that specific time. Click **[Back]** to go back to the default view settings.

- **Log Lines timeline**. Hover over this gray line to view a pop-up window that displays the number of log lines that have been ingested within this time interval.
- **Rare Events timeline**. Hover over this red line to view a pop-up window that displays the number of events marked as rare, such as possible issues or problems, that have been ingested within this time interval. Rare events are often the most diagnostic anomalies in the logs.

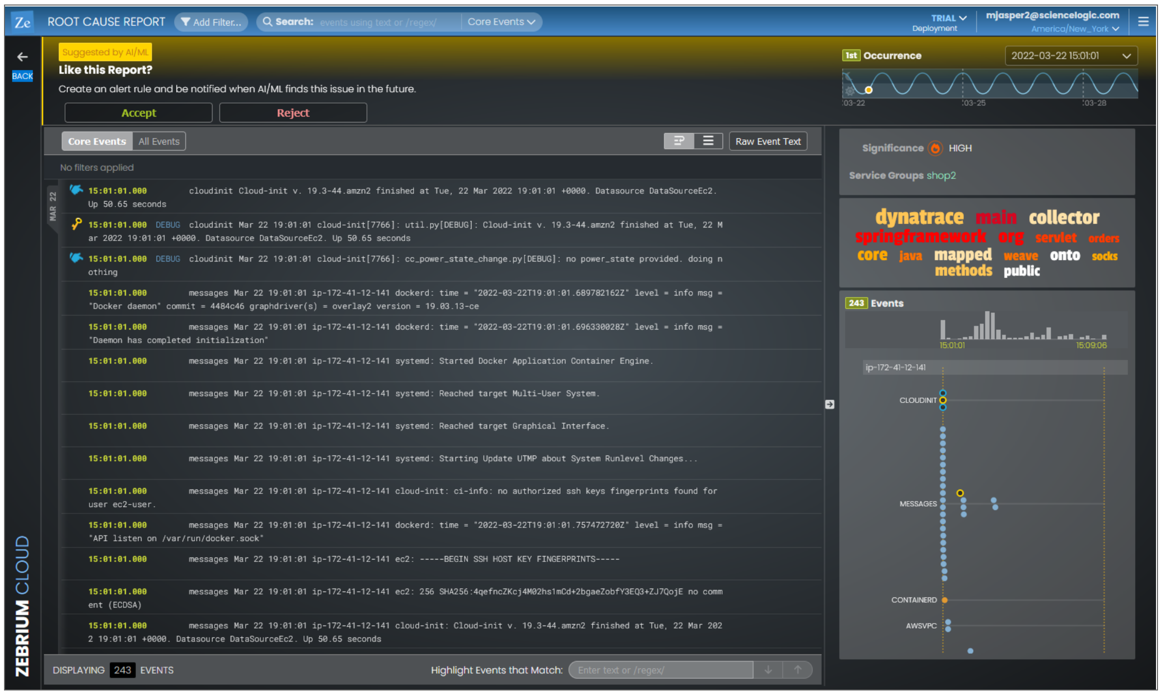


TIP: Click the **[Refresh]** button to get the most recently updated data for this page.

When you suspect a problem, you can drill down and view the RCA report from the timeline or the report summary view. The **RCA Report** page for that suggestion or alert appears. For more information, see *Root Cause Reports*.

Root Cause Reports

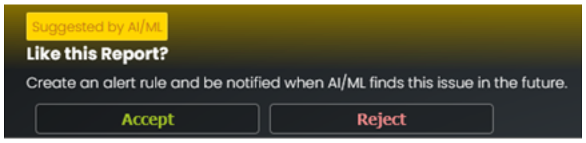
On the **Alerts** page, you can click anywhere in the summary view for a suggestion to open the **Root Cause Report** page. This page displays a more complete list of log events compiled by the AI/ML engine to describe this particular problem:



```
15:01:01.000        cloudinit Cloud-init v. 19.3-44.amzn2 finished at Tue, 22 Mar 2022 19:01:01 +0000. Datasource DataSourceEc2.
Up 50.65 seconds
15:01:01.000 DEBUG  cloudinit Mar 22 19:01:01 cloud-init[7766]: util.py[DEBUG]: Cloud-init v. 19.3-44.amzn2 finished at Tue, 22 M
ar 2022 19:01:01 +0000. Datasource DataSourceEc2. Up 50.65 seconds
15:01:01.000 DEBUG  cloudinit Mar 22 19:01:01 cloud-init[7766]: cc_power_state_change.py[DEBUG]: no power_state provided. doing n
othing
15:01:01.000        messages Mar 22 19:01:01 ip-172-41-12-141 dockerd: time = "2022-03-22T19:01:01.689782162Z" level = info msg =
"Docker daemon" commit = 4484c46 graphdriver(s) = overlay2 version = 19.03.13-ce
15:01:01.000        messages Mar 22 19:01:01 ip-172-41-12-141 dockerd: time = "2022-03-22T19:01:01.696330028Z" level = info msg =
"Daemon has completed initialization"
15:01:01.000        messages Mar 22 19:01:01 ip-172-41-12-141 systemd: Started Docker Application Container Engine.
15:01:01.000        messages Mar 22 19:01:01 ip-172-41-12-141 systemd: Reached target Multi-User System.
15:01:01.000        messages Mar 22 19:01:01 ip-172-41-12-141 systemd: Reached target Graphical Interface.
15:01:01.000        messages Mar 22 19:01:01 ip-172-41-12-141 systemd: Starting Update UTMP about System Runlevel Changes...
15:01:01.000        messages Mar 22 19:01:01 ip-172-41-12-141 cloud-init: ci-info: no authorized ssh keys fingerprints found for
user ec2-user.
15:01:01.000        messages Mar 22 19:01:01 ip-172-41-12-141 dockerd: time = "2022-03-22T19:01:01.757472720Z" level = info msg =
"API listen on /var/run/docker.sock"
15:01:01.000        messages Mar 22 19:01:01 ip-172-41-12-141 ec2: -----BEGIN SSH HOST KEY FINGERPRINTS-----
15:01:01.000        messages Mar 22 19:01:01 ip-172-41-12-141 ec2: 256 SHA256:4qefncZKcj4M02hs1mCd+2bgaeZobfY3EQ3+ZJ7QojE no comm
ent (ECDSA)
15:01:01.000        messages Mar 22 19:01:01 ip-172-41-12-141 cloud-init: Cloud-init v. 19.3-44.amzn2 finished at Tue, 22 Mar 202
2 19:01:01 +0000. Datasource DataSourceEc2. Up 50.65 seconds
```

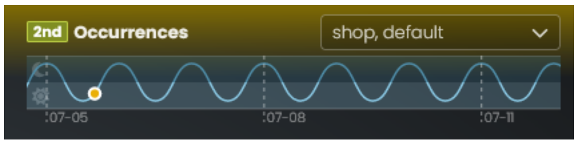
A typical **Root Cause Report** page contains the following elements:

- If this is a suggestion, the top pane states "Suggested by AI/ML", and you have the option of accepting or rejecting the suggestion:
  - If you ***accept*** the suggestion, Zebrium will create a rule for the settings for that suggestion in the future.
  - If you ***reject*** the suggestion, Zebrium will no longer show a suggestion with the same settings as that suggestion in the widget.


Suggested by AI/ML
Like this Report?
Create an alert rule and be notified when AI/ML finds this issue in the future.
Accept
Reject

For more information, see *Assessing Suggestions*.

- At the top right of the page is a panel that shows the number of occurrences of this type of event, a drop-down for each occurrence, and a sine wave depicting the time of each occurrence.



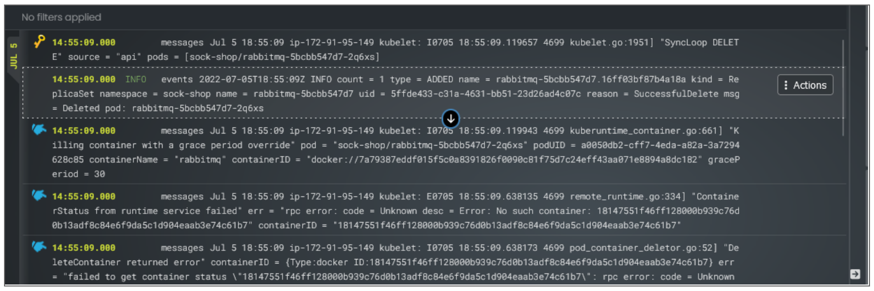
- The next pane down on the left contains a toggle for *Core Events* or *All Events*:

  - **Core Events** display by default, and they are the set of events that the AI/ML engine determined were the most likely events to explain the problem. Typically, the "core" list in an RCA report will contain somewhere between five and 25 log events.

  - **All Events** includes an much more expanded list of events that includes other surrounding anomalous log events, warnings, and errors surrounding this core list of events.

- On the same pane, you can also toggle between Wrap () and No Wrap () for displaying the logs in the pane below. You can also click **[Raw Event Text]** to view the log contents as text in a new dialog, in case you need to copy large amounts of text.

- The large pane on the left contains the list of log events that make up the report. You can think of these as the key log lines that explain a problem. You will usually see a combination root cause indicator and symptom log lines. There are typically 10-100 log lines in a report that span multiple log types.



```
14:55:09.000        messages Jul 5 18:55:09 ip-172-91-95-149 kubelet: I0705 18:55:09.119657 4699 kubelet.go:1951] "SyncLoop DELET
E" source = "api" pods = [sock-shop/rabbitmq-5bcbb547d7-2q6xs]
14:55:09.000 INFO   events 2022-07-05T18:55:09Z INFO count = 1 type = ADDED name = rabbitmq-5bcbb547d7.16ff03bf87b4a18a kind = Re
plicaSet namespace = sock-shop name = rabbitmq-5bcbb547d7 uid = 5ffde433-c31a-4631-bb51-23d26ad4c07c reason = SuccessfulDelete msg
= Deleted pod: rabbitmq-5bcbb547d7-2q6xs
14:55:09.000        messages Jul 5 18:55:09 ip-172-91-95-149 kubelet: I0705 18:55:09.119943 4699 kuberuntime_container.go:661] "K
illing container with a grace period override" pod = "sock-shop/rabbitmq-5bcbb547d7-2q6xs" podUID = a0050db2-cff7-4eda-a82a-3a7294
628c85 containerName = "rabbitmq" containerID = "docker://7a79387eddf015f5c0a8391826f0090c81f75d7c24eff43aa071e8894a8dc182" graceP
eriod = 30
14:55:09.000        messages Jul 5 18:55:09 ip-172-91-95-149 kubelet: E0705 18:55:09.638135 4699 remote_runtime.go:334] "Containe
rStatus from runtime service failed" err = "rpc error: code = Unknown desc = Error: No such container: 18147551f46ff128000b939c76d
0b13adf8c84e6f9da5c1d904eaab3e74c61b7" containerID = "18147551f46ff128000b939c76d0b13adf8c84e6f9da5c1d904eaab3e74c61b7"
14:55:09.000        messages Jul 5 18:55:09 ip-172-91-95-149 kubelet: I0705 18:55:09.638173 4699 pod_container_deletor.go:52] "De
leteContainer returned error" containerID = {Type:docker ID:18147551f46ff128000b939c76d0b13adf8c84e6f9da5c1d904eaab3e74c61b7} err
= "failed to get container status \"18147551f46ff128000b939c76d0b13adf8c84e6f9da5c1d904eaab3e74c61b7\": rpc error: code = Unknown
```

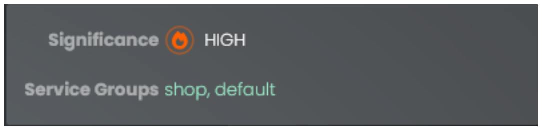
The columns in each log line show the event timestamp, a severity level, if available, the log type or service, and the text of the log. In addition, the following icons might appear to the left of some of the log events in the pane:

  - ***Alert Key*** (). One or two log events in the report might display this icon, which signifies that the AI/ML engine is using these event logs as a "signature" or alert rule to detect if the same type of alert occurs again in the future. Click the key icon () to view the definition of the key. To ensure accurate detection in the future, verify and edit the Alert Keys on the **Settings** menu () > *Alert Rules & Settings* page to match the one or two log events that best characterize this type of problem.
  - **Log line of interest** (). This icon appears next to any log events in the report that the AI/ML engine has identified as a possible event to explore. This is just an informational icon.

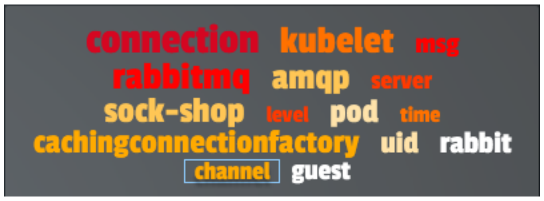
NOTE: You can hover over a log event to access the **Actions** button, which lets you perform additional actions related to that log event. For more information, see *Additional Actions on the RCA Report Page*.

- The bottom pane on the left lists the numbers of events that are currently being displayed. This number changes if you click a word in the word cloud, or if you type text or a regular expression in the ***Highlight Events that Match*** field.

- In the group of smaller panes to the right, the top pane displays the significance of the alert assigned by the AI/ML engine, from Low to High. The pane also includes the name of the *Service Group* impacted by the event.

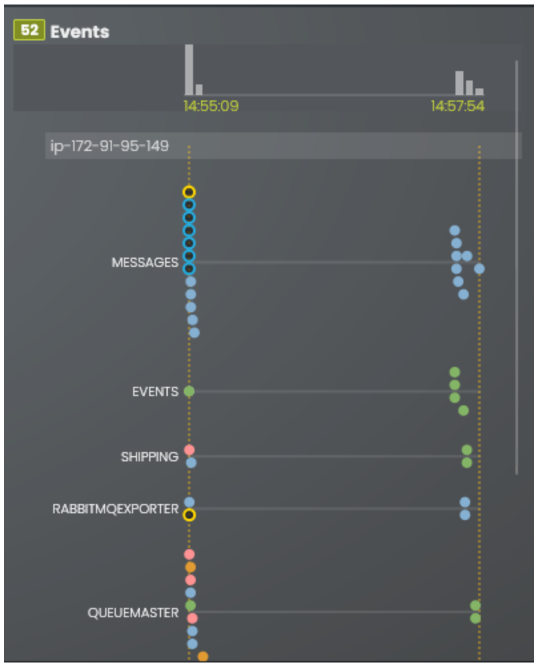


- The next pane displays the word cloud, which displays a set of keywords that the AI/ML engine selected from the report. For each word, the font size denotes how rare it is (smaller is more rare), and the color denoting how "bad" the underlying events were. For example, a word for a critical event displays in red.



TIP: Click a word in the cloud to filter the list of log events related to that word.

- Under the word cloud is a histogram that lists the number of events over time. You can click each gray rectangle in the histogram to see the number of events in each time period. Below the histogram are vertical rows of colored dots that represent the log events from the list on the left, arranged by micro-service and host name. The horizontal location of the dots are chronological, based on the histogram at the top of the pane. When you click a dot, the corresponding log event is highlighted on the left.



Additional Actions on the RCA Report Page

On the **RCA Report** page, you can hover over a log event to access the **Actions** button, which lets you perform the following actions related to that log event:

- *Peek*. Peek mode shows the surrounding log lines from the log type (log stream) itself, and you can drill down on logs from a particular host or pod. This is similar to looking at the log file for a single log generator. To exit Peek mode, click the **[Unpeek]** button.

- *Annotations*. You can add notes relevant to this event log. A note icon displays to the right of the event log, with a red badge listing the number of notes for that log.
- *Related Incidents*. Searches for other incidents that include this event. You can view the RCA report summaries for the related events for more information about the event.
- *Include this event type in future alerts*. Adds this event type to future alerts.
- *Exclude this event type in future alerts*. Excludes this event type from future alerts.
- *Create a custom alert rule using this event type*. Lets you create a custom alert rule using this event type.
- *Advanced*: These options let you create and use custom, include, and exclude regular expressions for this log event.

On the **RCA Report** page for an Accepted Alert, you can perform the following activities by clicking the **[Actions]** button at the top of the page:

- *Edit Alert Rule Metadata*. Opens the ***Edit Alert Metadata dialog*** so you can update the metadata of the alert rule.
- *Edit Alert Rule*. Opens the ***Edit Alert Rule Keys pane*** so you can change the alert keys, if needed.
- *Send One Time Alert*. Lets you send a one-time alert to the notification channel you specify here. For more information, see ***Enabling Notification Channels***.
- *Reject this Alert*. Changes the status of the accepted alert to rejected. For more information, see ***Rejecting a Suggestion***.
- *Revert to Suggested*. Changes the status of the accepted alert to a ***suggestion***.

Assessing Suggestions

On a regular schedule, you should assess (or disposition) your suggestions in Zebrium by accepting or rejecting them, as this will help improve the accuracy of the suggestions you will see in the future.

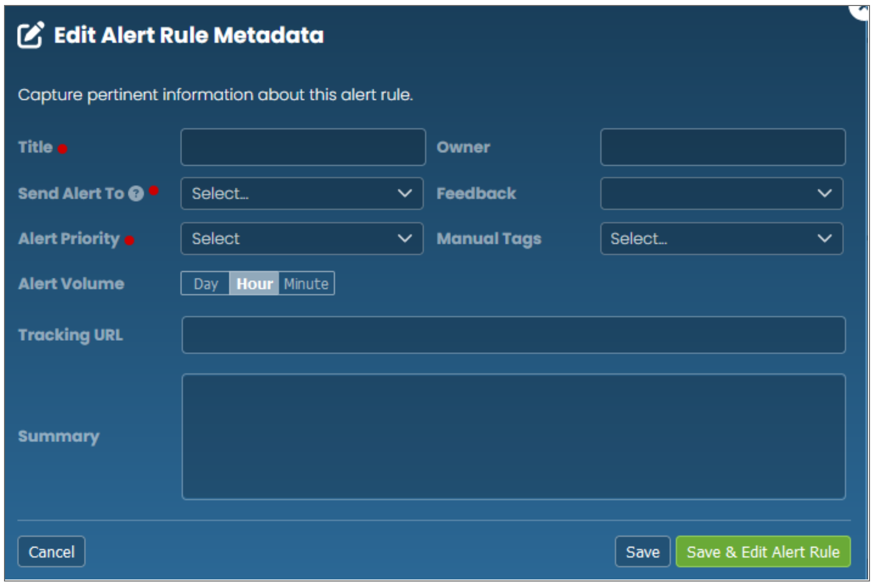
Accepting a Suggestion

You should *Accept* a suggestion if it relates to a real problem. If you accept the suggestion, Zebrium creates a rule for the settings for that suggestion in the future. Accepting a suggestion turns it into a ***Accepted Alert*** and creates an ***Accepted Alert Rule***.

NOTE: If you accept a suggestion but no longer want to use it as a rule, you can revert it to make the rule back into a suggestion again.

To accept a suggestion:

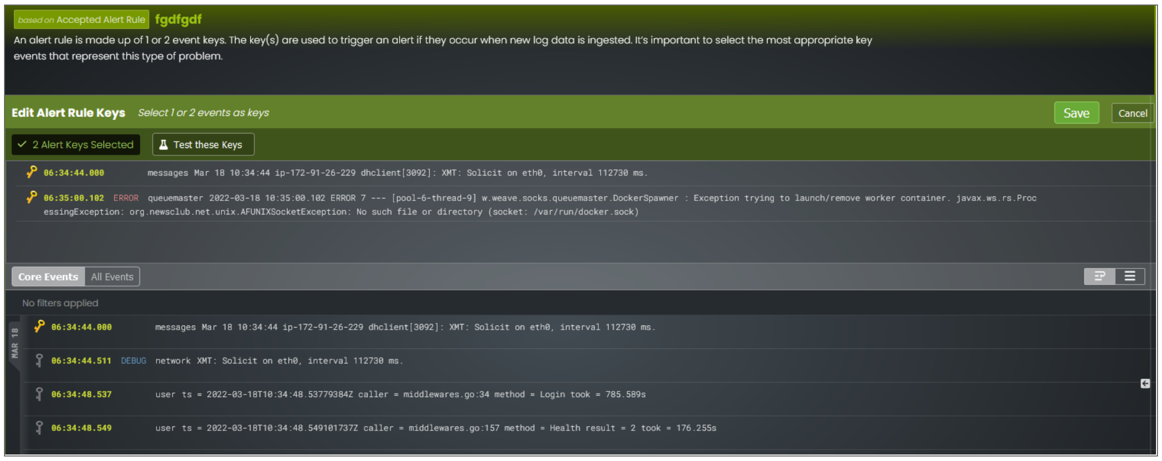
1. On the **RCA Report** page for the suggestion, click **[Accept]**. The **Edit Alert Rule Metadata** dialog appears:



2. Complete the following fields:

   - ***Title***. Type a unique name for this rule. Required.
   - ***Owner***. Type your name of the name of the owner of this rule.
   - ***Send Alert To***. Alerts will be sent to all dashboards that you have configured, along with any notification channel you specify here. You can set up notification channels in the **Integrations and Collectors** page. For more information, see *Enabling Notification Channels*. This field is required, but you can also click **[Select No one]** as an option.
   - ***Feedback***. Select a value from the drop-down, from *Poor* to *Excellent*.
   - ***Alert Priority***. Set the priority from P1 to P5. Required.
   - ***Manual Tags***. Select a tag as needed.
   - ***Alert Volume***. Select whether you want to alert at most once per day, once per hour, or once per minute.
   - ***Tracking URL***. Add a URL to use for tracking this rule.
   - ***Summary***. Add any more information related to this rule.

3. Click **[Save & Edit Alert Rule]**, the **Edit Alert Rule Keys** pane appears:



An alert rule is made up of 1 or 2 event keys. The key(s) are used to trigger an alert if they occur when new log data is ingested. It's important to select the most appropriate key events that represent this type of problem.

```
06:34:44.000        messages Mar 18 10:34:44 ip-172-91-26-229 dhclient[3092]: XMT: Solicit on eth0, interval 112730 ms.
06:35:00.102  ERROR  queuemaster 2022-03-18 10:35:00.102 ERROR 7 --- [pool-6-thread-9] w.weave.socks.queuemaster.DockerSpawner : Exception trying to launch/remove worker container. javax.ws.rs.Proc
essingException: org.newsclub.net.unix.AFUNIXSocketException: No such file or directory (socket: /var/run/docker.sock)
```

```
06:34:44.000        messages Mar 18 10:34:44 ip-172-91-26-229 dhclient[3092]: XMT: Solicit on eth0, interval 112730 ms.
06:34:44.511  DEBUG  network XMT: Solicit on eth0, interval 112730 ms.
06:34:48.537        user ts = 2022-03-18T10:34:48.53779384Z caller = middlewares.go:34 method = Login took = 785.589s
06:34:48.549        user ts = 2022-03-18T10:34:48.549101737Z caller = middlewares.go:157 method = Health result = 2 took = 176.255s
```

4. You can use the currently selected keys, or you can edit one or both keys.
5. To edit the alert keys, click a key from the top list to remove it. Click a key from the second list of keys to use that key instead.
6. Click **[Save]** and then click **[View Alert List]** to return to the **Alerts** page.

Rejecting a Suggestion

To reject a suggestion:

1. On the **RCA Report** page for the suggestion, click **[Reject]**. A dialog appears with the options to *Ignore* or *Reject*.
2. Click **[Ignore]** if you are not sure if it is a good suggestion, which gives other members of your team the option of reviewing the suggestion. The suggestion will still appear on the **Alerts** page, but will not generate a suggestion in the future.
3. Click **[Reject]** if you are sure that the suggestion is not helpful. Zebrium will hide the suggestion on the **Alerts** page, and will not notify you of future occurrences of the same suggestion type.

NOTE: You can restore a rejected alert by filtering for Rejected Alerts, navigating to the **RCA report** page for that alert, and clicking **[Restore & Accept]**. The alert is restored and marked as accepted, and Zebrium creates a rule based on the selected event keys. You can edit the alert metadata as needed before saving it.

Key Use Cases for Suggestions and Root Cause Reports

This section covers the main use cases and concepts related to using Zebrium, along with some tips and best practices.

Automated Root Cause Analysis Only

When you know a problem has occurred, you can look at Zebrium alerts around the time of the problem. As long as details of the problem are present in the logs, you should find that the AI/ML engine has generated a useful alert containing a report that explains the root cause of the problem. In this mode, the AI/ML engine typically identifies the root cause more than 90% of the time.

For more information, see https://www.zebrium.com/cisco-validation.

Proactive Detection and Root Cause Analysis

The AI/ML engine constantly scans logs for clusters of correlated anomalies that resemble problems. When it detects a potential problem, it proactively generates a suggestion. Be aware that while some suggestions will relate to important issues or problems, others will not be useful at all. As a result, do not think of suggestions in the same way that you normally think about *alerts* in other tools.

Instead of paging an operator with each new suggestion, as a best practice you should review suggestions at a convenient time periodically. When reviewing a suggestion, you can choose to:

- *Accept* the suggestion. This creates an alert rule that will detect if the same thing happens in the future.
- *Reject* the suggestion. This tells the AI/ML engine not to create such an alert in the future.
- *Ignore* the suggestion without doing anything more; you will need to click the **[Reject]** button for the suggestion first. Future occurrences will be filtered out by default.

TIP: Spending a few minutes each day reviewing suggestions from Zebrium will help to improve the signal-to-noise ratio of future suggestions.

Deterministic Detection of Known Problems

After you accept a suggestion, you can use it to deterministically notify you if the same problem occurs again. This is like having a robot that can generate alert rules for you.

You can also build your own custom rules to detect already known problems. When custom rules trigger, the AI/ML engine automatically generates a report with additional anomalies from the logs that can help to explain the root cause.

Getting the Best Results from Zebrium

The AI/ML engine will start working within a few minutes of logs arriving, detecting root causes for problems that occur in your environment, and presenting them as suggestions within the Zebrium user interface. The signal-to-noise ratio improves with time, and typically achieves a good level in about 24 hours.

If you are not satisfied with the quality of the results, there are a few things you can do. The next few topics address this situation.

Ingest Complete Logs That Contain a Real Problem

Sometimes users connect Zebrium to a software environment that is in a steady state, where nothing bad happens. In such cases, the logs do not actually contain any unusual events or significant errors. Naturally, in such cases, the AI/ML engine will not be able to generate a useful Root Cause report.

Also, sometimes users will upload a subset of the logs, or even a single log file, which also degrades the ability of the AI/ML engine to create meaningful root cause reports. For good results, connect Zebrium to a software environment where real problems occur, or where you can deliberately break things.

You can achieve equivalent results by uploading static log files from a real problem, but in this case, be sure to ensure that the log collection is complete; anything that a human would need for troubleshooting should be included. Also, make sure that the files are tagged with correct metadata, and that the logs cover a time range of 24 hours or more before the problem occurred.

Be Mindful of Elapsed Time

By default, Zebrium has a few settings that govern whether, and how well, a root cause report is created.

For instance, the AI/ML engine needs some history to build an event catalog, to learn normal patterns, and to learn the dependencies between log streams. If you connect Zebrium to a brand-new environment, for best results you should let it learn for about 24 hours before attempting tests. It is possible to get reasonable results much quicker, such as one to two hours after setup, but be prepared for noisier results.

Also, if the same kind of problem keeps occurring within a day, the AI/ML engine might consider it "typical", and not create a root cause report for it at all.

A common issue users encounter is that they induce the same problem more than once, and do not realize that default filter settings will only show the first occurrence of the problem. For more information, see ***Using the Filters on the Alerts Page in Zebrium***.

Review Service Group Setup

Service groups are a way to inform the AI/ML engine about the failure domains within your log streams. Only log streams or files coming from services, containers, and hosts that could affect each other should be placed in the same service group. If you see log events in a RCA report that originate from completely unrelated services, you can partition them by changing your log collector settings to place them in different service groups. Aside from assigning a Service Group label per daemonset, you can also map sets of k8s labels (like apps, or namespaces) into a particular Service Group by editing the YAML file for the log collector.

Review RCA Settings

A handful of the AI/ML engine settings are visible on the **Report Settings** page (Settings (☰) > Root Cause Settings.

The most common setting to consider adjusting is the **Root Cause Significance** setting. Think of this like a filter level; the higher the significance setting, the more selective the AI/ML engine will be in alerting. ***Significance*** is a cumulative score for each suggested alert, based on the rareness and "badness" (log severity level) of the constituent log events within that alert. The higher the significance setting, the more rare and bad the Root Cause events have to be to show up in an alert feed.

"Badness" is derived from the log severity level, but there are additional hidden settings that can optionally scan the log text, as well as add your own keywords or strings that have a special meaning for your software stack.

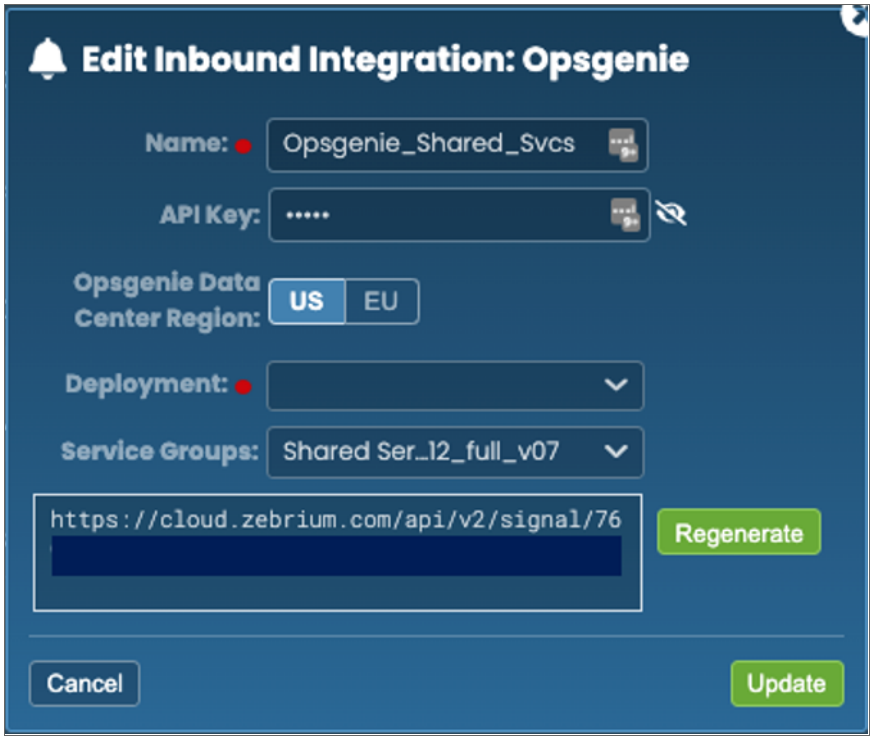
There are other settings that might be useful in rare cases, such as excluding a particular log type entirely if it is not useful from a diagnostics perspective.

Use Integrations to Separate High-priority Alerts

The AI/ML engine creates RCA reports when it identifies clusters of rare events and bad events, such as events with higher log severity, like warning or error, that are highly unlikely to occur by random chance. Nevertheless, all such clusters may not be due to high priority (P1 or P2) issues, and therefore may not need immediate attention.

One way to distinguish the high priority issues from others is to set up inbound integrations with tools such as PagerDuty, OpsGenie, and VictorOps. When an incident is created in one of these tools, due to an alert from some other observability tool, for example, the integration signals the AI/ML engine to analyze logs from the same environment and respond with a RCA report. The report is automatically appended to the incident, such as in the timeline or notes fields.

As a result, Zebrium RCA reports can be matched up with incident priorities that were already assigned based on other rules:


Edit Inbound Integration: Opsgenie

Name: Opsgenie_Shared_Svcs

API Key: •••••

Opsgenie Data Center Region: [✓ US] EU

Deployment:

Service Groups: Shared Ser_12_full_v07

https://cloud.zebrium.com/api/v2/signal/76
[redacted]

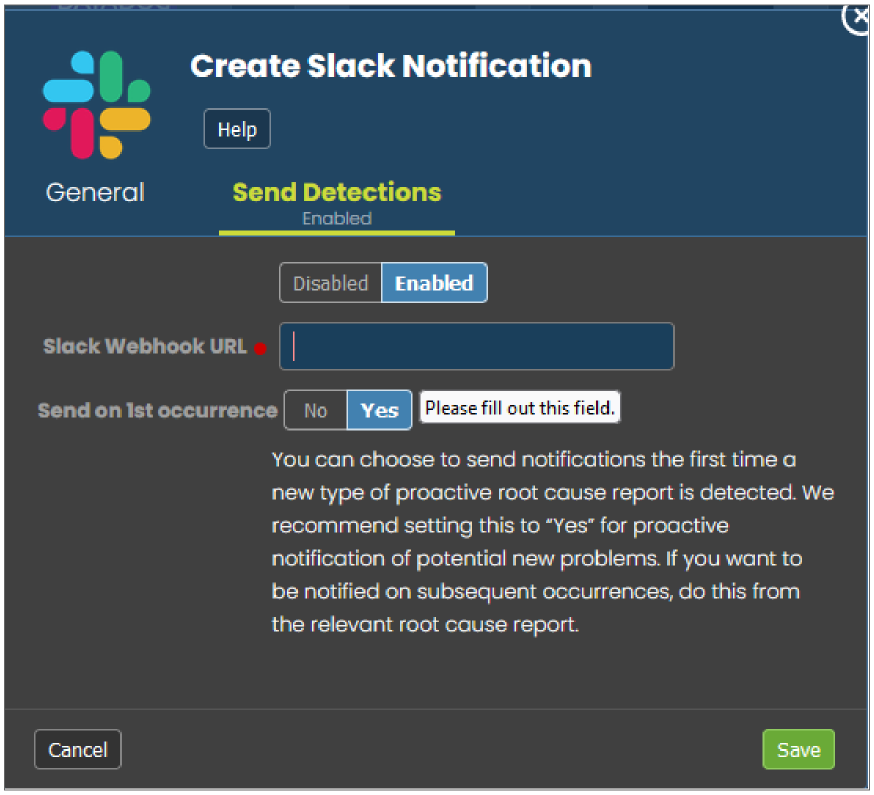
You can also use inbound integrations to route alerts rather than incidents to Zebrium. In this case Zebrium will not be able to update any incident fields, because it does not receive incident notifications. However, Zebrium will use the alerts as triggers to generate RCA reports, which will be sent to the outbound channels that are already configured.

Note that the AI/ML engine will continue to proactively detect alerts , even when there is no signal from a third-party tool like PagerDuty or OpsGenie, but these proactive alerts can now be routed to lower priority alert queues.

Manage Alert Destinations

There are multiple ways to manage and segregate alerts. The easiest way is to set up notification channels for every combination of deployments or service groups that you would like to route uniquely.

Notification Channels provide a mechanism to define the methods that Zebrium will use to send notifications from RCA reports. The supported types of notification channels include email, as well as Mattermost, Slack, Microsoft Teams, and Webex Teams notifications.



Slack Webhook URL

Send on 1st occurrence No Yes

Please fill out this field.

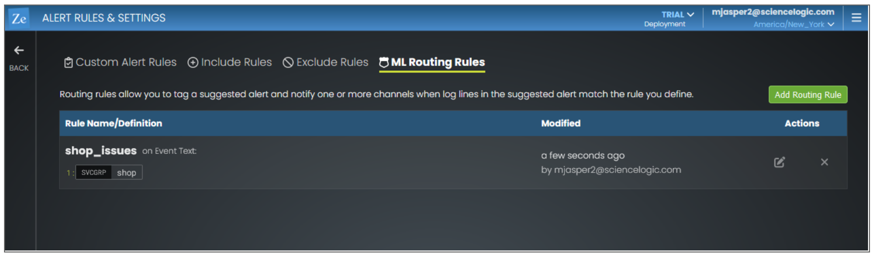
You can choose to send notifications the first time a new type of proactive root cause report is detected. We recommend setting this to "Yes" for proactive notification of potential new problems. If you want to be notified on subsequent occurrences, do this from the relevant root cause report.

After you have created one or more notification channels, you can link any number of these to any RCA report created by the AI/ML engine. Linking a set of notification channels to a RCA report will send notifications of future RCA reports of the same type to those channels.

For more information, see *Enabling Notification Channels*.

Use Routing Rules to Classify and Route Alerts

An even more powerful way to manage and route alerts is to set up routing rules on the **Alert Rules & Settings** page (Settings (☰) > Alert Rules & Settings), on the the **[ML Routing Rules]** tab:

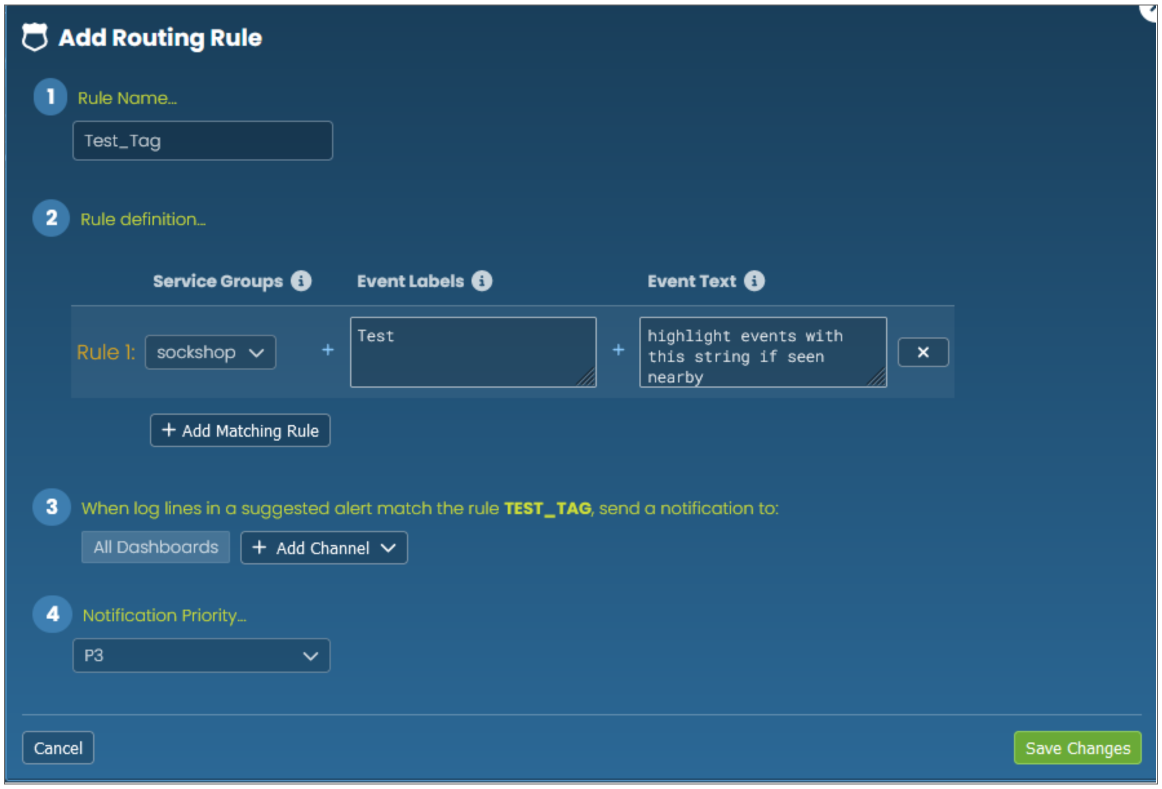


Routing rules allow you to tag a suggested alert and notify one or more channels when log lines in the suggested alert match the rule you define.

Rule Name/Definition	Modified	Actions
shop_issues on Event Text: 1 : SVCGRP shop	a few seconds ago by mjasper2@sciencelogic.com	

This allows you to set up rules regarding service group, event labels (such as the Kubernetes app or pod name), as well as string matches in the actual log event. Each routing rule lets you automatically triage alerts and RCA reports, and send them to the appropriate destination.

For example, you might want to create a "Networking" tag for alerts that involves logs from Kubernetes pods that affect networking services, or contain key words related to network issues, and send them to an email alias or Slack channel for the networking team:



Add Routing Rule

1 Rule Name...

Test_Tag

2 Rule definition...

	Service Groups		Event Labels		Event Text	
Rule 1:	sockshop	+	Test	+	highlight events with this string if seen nearby	✕

3 When log lines in a suggested alert match the rule **TEST_TAG**, send a notification to:

All Dashboards

4 Notification Priority...

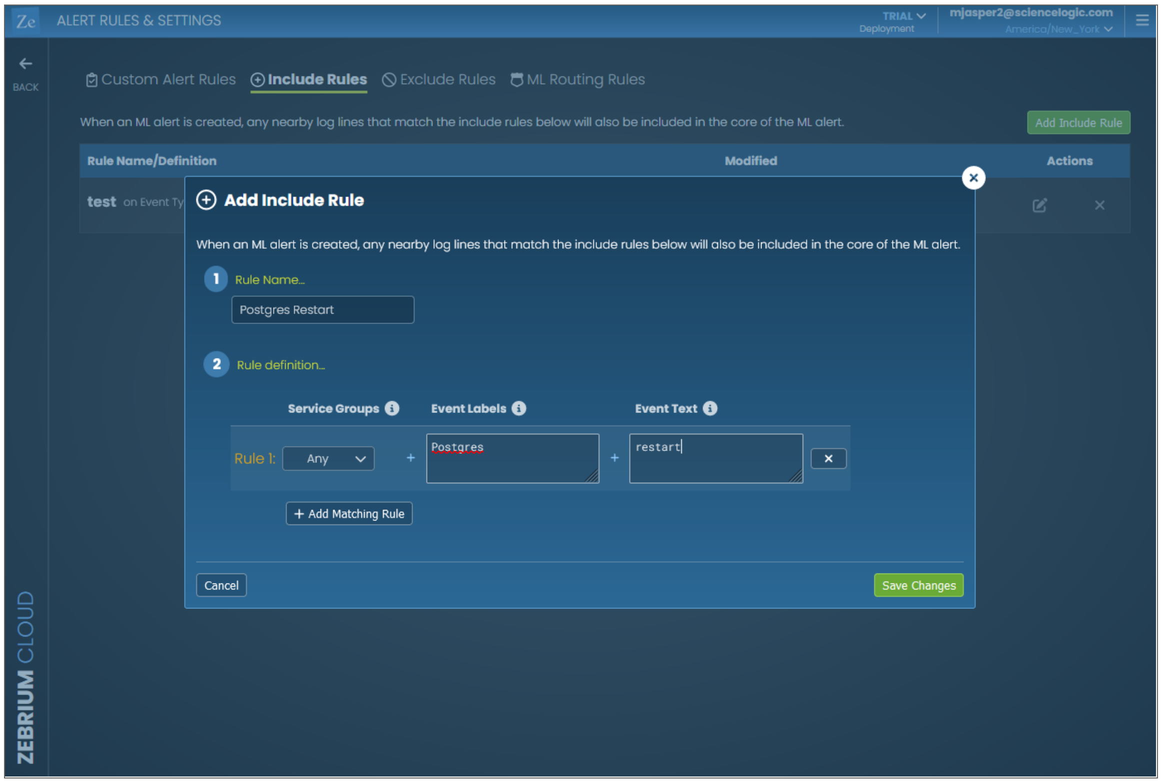
P3

Example: Ensure that the AI/ML Engine Highlights Significant Events When They Happen Nearby

As an example, let's say that your engineers know that a specific log event is useful from a troubleshooting perspective. If that event occurs in the vicinity of an auto-detected alert, you might want to ensure that it gets pulled into the core event list of any alert.

If you want this outcome, go to the **Alert Rules & Settings** page (Settings (☰) > Alert Rules & Settings), click the **[Include Rules]** tab, and define the pattern to match these events.

For example, the rule below will make sure any events coming from the Postgres log stream that contain the keyword "restart" will be pulled into an RCA report if the AI/ML engine detects unusual events within the vicinity of this restart event:



When an ML alert is created, any nearby log lines that match the include rules below will also be included in the core of the ML alert.

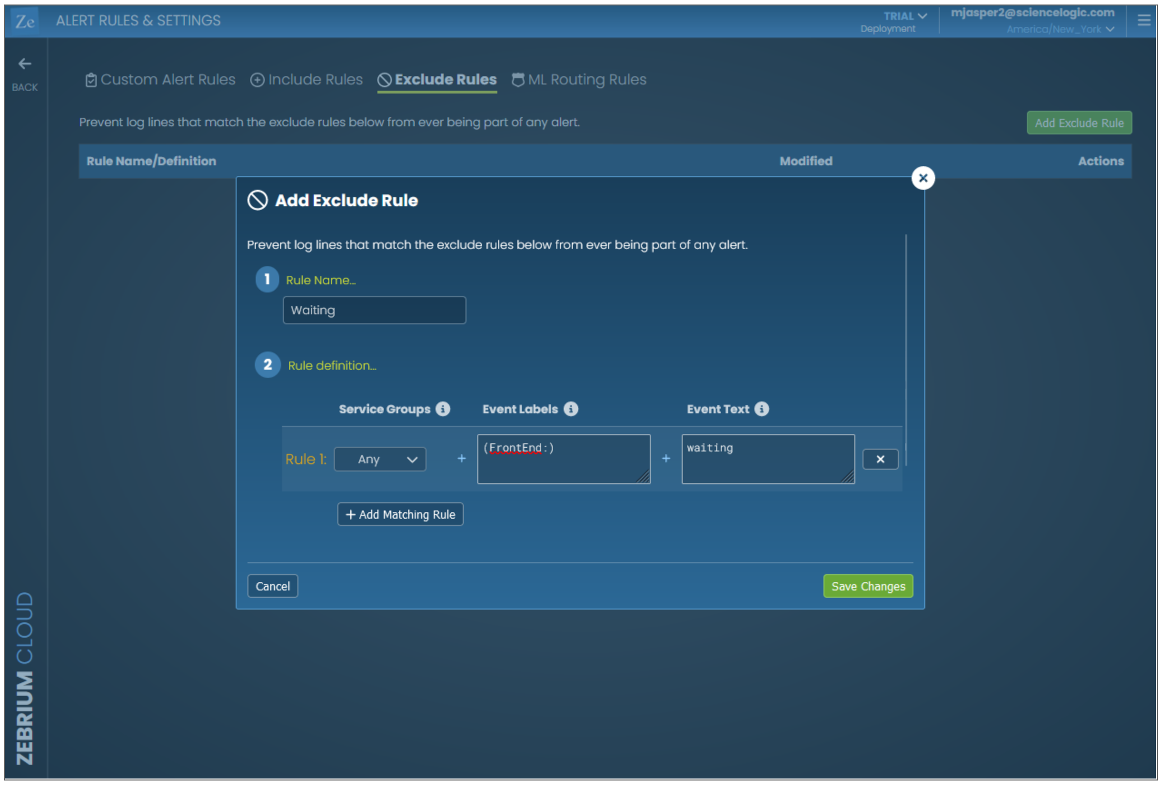
1 Rule Name...
Postgres Restart

2 Rule definition...
Rule 1: Any + Postgres + restart

Example: Ensure the AI/ML Engine Ignores Spam Events When They Happen Nearby

This configuration does the opposite of the previous feature. Let's say your engineers know that a specific log event is spam and low value from a troubleshooting perspective. If you want to keep it from showing up in RCA reports, simply specify the event label and pattern match to tell the AI/ML engine to exclude these events:

If you want this outcome, go to the **Alert Rules & Settings** page (Settings (☰) > Alert Rules & Settings), click the **[Exclude Rules]** tab, and define the pattern to exclude this kind of event:



Prevent log lines that match the exclude rules below from ever being part of any alert.

Rule Name: Waiting

Rule 1: Any + (FrontEnd:) + waiting

Chapter 5

Configuring Observability Dashboard Integrations

Overview

You can integrate the Zebrium root cause service into your existing observability dashboards. For example, if you see symptoms of a potential problem in your metrics dashboard, you can have the root cause indicators for the problem surfaced right below that, as in the following image:

To enable this, go to the **Integrations & Collectors** page (Settings (☰) > Integrations & Collectors), select your preferred observability dashboard, and follow the instructions for setting up that dashboard.

Zebrium offers the following Dashboard Integrations:

- *AppDynamics*
- *Datadog Dashboard Widget*
- *Datadog Events and Metrics*
- *Dynatrace*
- *Elastic Stack*
- *Grafana Plugin*
- *New Relic*
- *ScienceLogic*

AppDynamics

Features

- Automatically adds Root Cause Reports as Monitor Events in AppDynamics. This allows you to see the details of root cause on any AppDynamics dashboard.
- Each Zebrium RC Report includes a summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium UI.
- Leads to faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

How it Works

Our recommended mode of operation for Observability Dashboard Integrations is to use the Zebrium **Auto-Detect** mode as an accurate mechanism for explaining the reason something went wrong. In this mode, you continue to use your existing rules, alerts and metrics as the primary source of problem detection. You can then review Zebrium RC Report findings directly in your AppDynamics Dashboards alongside other metrics to explain the reason behind problems you were alerted on.

Augment mode is useful when you have Health Rule Violation or Anomaly-based Policy triggers defined in AppDynamics and you want a Root Cause Report automatically generated at the time of the alert. In this mode, Zebrium uses an AppDynamics webhook as a notification channel and will update your Dashboard with Root Cause Reports that coincide with the triggering monitor so they're immediately visible to you as you work the issue.

The two modes of operation are independent. You can configure Auto-Detect and/or Augment modes depending on your operational use-case.

Auto-Detect (Recommended): Send Root Cause Detections to your AppDynamics Dashboards

1. Zebrium continuously monitors all application logs and uses unsupervised machine learning to find anomalous log patterns that indicate a problem. These are automatically turned into Root Cause Reports highlighting details of any problems with over 95% accuracy.
2. Root Cause Report summaries are sent to AppDynamics using the **event** API and Root Cause details are visible on your AppDynamics Dashboards.
3. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Dashboard.

CLICK HERE to send Root Cause Detections to your AppDynamics Dashboards

Augment (Advanced Users): Receive Signals from AppDynamics Health Rule Violations

1. Any AppDynamics Health Rule Violation or Anomaly based Policy can trigger a webhook request for Root Cause Analysis from Zebrium.
2. Zebrium finds anomalous log patterns from your application that coincide with the event and creates a Root Cause Report.
3. Root Cause Report summaries are sent to AppDynamics using the **event** API and Root Cause details are visible on your AppDynamics Dashboards.
4. If you need to drill down further to look at correlated logs across your entire app, it’s just one click from your Dashboard.

CLICK HERE to receive Signals from AppDynamics Health Rule Violations

Sending Root Cause Detections to AppDynamics Dashboards

Integration Overview

1. Configure API access for creating Root Cause Reports as Monitor Events.
2. Create an AppDynamics Integration in Zebrium using the information from step 1.

Integration Details

STEP 1: Configure API Access for Creating Root Cause Reports as Monitor Events

To configure API Access:

1. From the **Gear** icon, click on **Administration**.
2. Click on the **[API Clients]** tab and click **Create**.
3. Enter a **Client Name** and make a note of that name for later use when configuring Zebrium.
4. Enter a **Description**.
5. Generate a **Secret** and make a note of the secret for later use when configuring Zebrium.
6. Add minimum **Roles** required to create a Monitor Event.
7. Click **[Save]**.

STEP 2: Create an AppDynamics Integration in Zebrium to Send Detections to AppDynamics

The following prerequisites are needed for the Zebrium configuration:

- **Controller URL** for the application being monitored (from the My AppDynamics Accounts page)
- **Account Name** (from the My AppDynamics Accounts page).
- **Application ID** for the application being monitored (this number can be found in your browser URL when viewing the Application).
- **API Client Name** (from step 3 under Configuring API Access, above).
- **Client Secret** (from step 5 under Configuring API Access, above).

To create the AppDynamics Integration in Zebrium:

1. From the User menu area in Zebrium, click on the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Observability Dashboards** section and click on **AppDynamics**.
4. Click the **[Create a New Integration]** button.

5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **[Send Detections]** tab.
10. Click the **[Enabled]** button.
11. Enter all the information listed above in the **Prerequisites**.
12. Click **[Save]**.

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

Receiving Signals from AppDynamics Health Rule Violations

Integration Overview

1. Configure API access for creating Root Cause Reports as Monitor Events.
2. Create an AppDynamics Integration in Zebrium to receive signals from AppDynamics.
3. Create an HTTP Request Template in AppDynamics to send signals to Zebrium.

Integration Details

STEP 1: Configure API Access for Creating Root Cause Reports as Monitor Events

1. From the **Gear** icon, click on **Administration**.
2. Click on the **[API Clients]**tab and click **Create**.
3. Enter a **Client Name** and save for later use when configuring Zebrium.
4. Enter a **Description**.
5. Generate a **Secret** and save for later use when configuring Zebrium.
6. Add minimum **Roles** required to create a Monitor Event.
7. Click **[Save]**.

STEP 2: Create an AppDynamics Integration in Zebrium to Receive Signals from AppDynamics

Prerequisites needed for Zebrium configuration:

- **Controller URL** for the application being monitored (from the My AppDynamics Accounts page)
- **Account Name** (from the My AppDynamics Accounts page).
- **Application ID** for the application being monitored (this number can be found in your browser URL when viewing the Application).
- **API Client Name** (from step 3 under Configuring API Access, above).
- **Client Secret** (from step 5 under Configuring API Access, above).

To create the AppDynamics Integration:

1. From the User menu area in Zebrium, click on the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Observability Dashboards** section and click on **AppDynamics**.
4. Click the **[Create a New Integration]** button.

5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **[Receive Signals]** tab.
10. Click the **[Enabled]** button.
11. Enter all the information listed above in the **Prerequisites**.
12. Click **[Save]**.
13. Copy the Webhook URL and make a note of it for the next procedure.
14. Click **[OK]**.

STEP 3: Create HTTP Request Template in AppDynamics to send Signals to Zebrium

1. Click the **[Alert & Respond]** tab at the top of the window.
2. Click on **HTTP Request Templates** from the navigation pane on the left side.
3. Click **[New]**.
4. Enter a **Name** for the template.
5. Under Request URL, select **POST** as the method.
6. Enter the **Inbound Webhook URL** from STEP 2.
7. Select **UTF-8** encoding.
8. Under **Authentication**, select **NONE** as the type.
9. Under **Payload**, select **application/json** as the **MIME Type**.
10. Select **UTF-8** as the **Payload Encoding**.
11. In the **Payload Text Box** enter:
    ```
    { "event_time" : "${latestEvent.eventTime}", "event_type" : "zebrium"
    }
    ```
12. Enter desired **Response Handling Criteria** and **Settings**.
13. Click **[Test]**.
14. Successful configuration will return: Response Status: **200 OK** Response Payload: **Processed signal: success**.
15. You can now create an Action to send an HTTP Request using the template you just created and use these Actions in your Health Rule Policies for your monitored Application.

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

Datadog Dashboard Widget

Please navigate to **Integrations** in your Datadog user interface and search for *Zebrium* to find complete documentation on installing the Zebrium dashboard widget for Datadog.

For more information, contact Zebrium at support@zebrium.com.

Datadog Events and Metrics

NOTE: In addition to the integrations described below, Zebrium also provides a custom Datadog Dashboard Widget. Select **Integrations** in your Datadog user interface and search for *Zebrium* for more details. For more information, contact Zebrium at support@zebrium.com.

Features

- Automatically adds Root Cause Reports as Events in Datadog. This allows you to see details of root cause on any Datadog dashboard.
- Automatically adds Log count metrics in Datadog.
- Each Zebrium RC Report includes a summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium user interface.
- Leads to faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

How it Works

Our recommended mode of operation for Observability Dashboard integrations is to use the Zebrium **Auto-Detect** mode as an accurate mechanism for explaining the reason something went wrong. In this mode, you continue to use your existing rules, alerts and metrics as the primary source of problem detection. You can then review Zebrium RC Report findings directly in your Datadog Dashboards alongside other metrics to explain the reason behind problems you were alerted on.

Augment mode is useful when you have monitors defined in Datadog and you want a Root Cause Report automatically generated at the time of the alert. In this mode, Zebrium uses a Datadog webhook as a notification channel and will update your Dashboard with Root Cause Reports that coincide with the triggering monitor so they're immediately visible to you as you work the issue.

The two modes of operation are independent. You can configure Auto-Detect and/or Augment modes depending on your operational use-case.

Auto-Detect (recommended): Send Root Cause Detections to your Datadog Dashboards

1. Zebrium continuously monitors all application logs and uses unsupervised machine learning to find anomalous log patterns that indicate a problem. These are automatically turned into Root Cause Reports highlighting details of any problems with over 95% accuracy.
2. Root Cause Report summaries are sent to Datadog using the **event** API and Root Cause details are visible on your Datadog Dashboards.
3. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Dashboard.
4. Log metrics are also sent to Datadog via the **series** API for visualization on your Datadog Dashboards.

CLICK HERE to send Root Cause Detections to your Datadog Dashboards

Augment (advanced users): Receive Signals from Datadog Triggered Monitors

1. Any Datadog Monitor can trigger a webhook request for Root Cause Analysis from Zebrium.
2. Zebrium finds anomalous log patterns from your application that coincide with the event and creates a Root Cause Report.
3. Root Cause Report summaries are sent to Datadog using the **event** API and Root Cause details are visible on your Datadog Dashboards.
4. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Dashboard.

CLICK HERE to receive Signals from Datadog Triggered Monitors

Sending Root Cause Detections to your Datadog Dashboards

NOTE: In addition to the integrations described below, Zebrium also provides a custom Datadog Dashboard Widget. Select **Integrations** in your Datadog user interface and search for *Zebrium* for more details. For more information, contact Zebrium at support@zebrium.com.

Integration Overview

1. Create an API Key in Datadog.
2. Create a Datadog Integration in Zebrium using the information from step 1.
3. Add Zebrium Root Cause Report events and Log metrics to your Datadog Dashboard.

Integration Details

STEP 1: Create an API Key in Datadog

1. From the Main Navigation panel in Datadog, hover over your Datadog Login Name and select **Organization Settings**.
2. Click on **API Keys**.
3. Click the **[+ New Key]** button.
4. Enter a **Name** for the API Key and click **Create Key**.
5. Copy and save the **Key** for use in STEP 2, below.

STEP 2: Create a Datadog Integration in Zebrium to Send Detections to Datadog

1. From the User menu area in Zebrium, click on the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Observability Dashboards** section and click on **Datadog Events and Metrics**.
4. Click **[Create a New Integration]**.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **[Send Detections]** tab.
10. Click **[Enabled]**.

11. Enter the **API Key** created in STEP 1, above.
12. Click **[Save]**.

STEP 3: Add Zebrium Root Cause Report Detections and Log Count Metrics to Your Datadog Dashboards

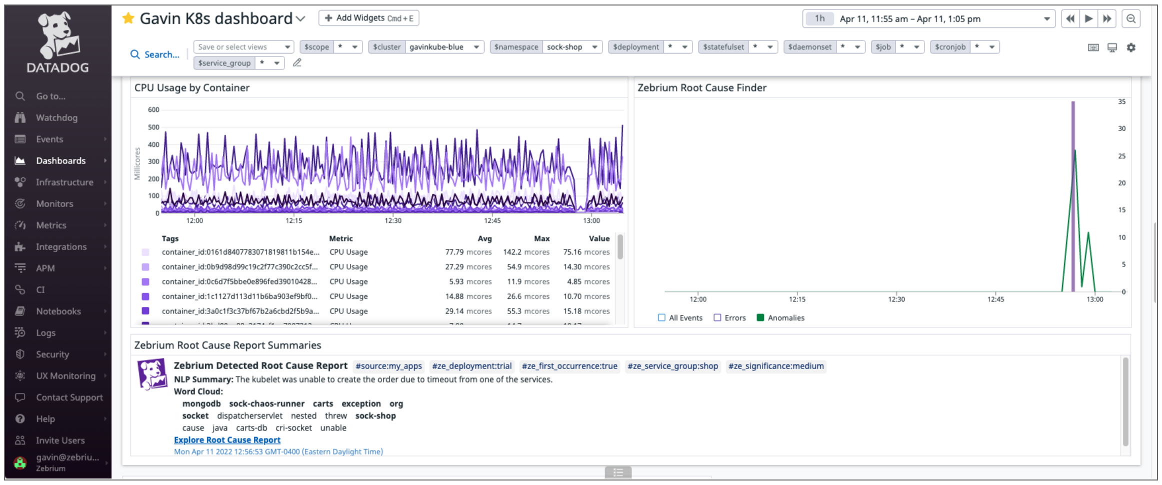
Zebrium sends events and metrics to Datadog as follows:

1. **Events** are sent each time a Zebrium Root Cause Report Detection occurs.
2. **Metrics** are sent for counts of all log events, error log events and anomaly log events

Visualizing Zebrium Data in Datadog

The following image displays a sample chart visualization showing:

1. A **Root Cause Finder** panel that displays a vertical bar whenever a Zebrium detection occurs. This allows you to easily see detections that are aligned with other metrics on your dashboards.
2. A **Root Cause Reports Summary** panel that list summary information for each Zebrium detection.



	Tags	Metric	Avg	Max	Value
	container_id:0161d8407783071819811b154e...	CPU Usage	77.79 mcores	142.2 mcores	75.16 mcores
	container_id:0b9d98d99c19c2f77c390c2cc5f...	CPU Usage	27.29 mcores	54.9 mcores	14.30 mcores
	container_id:0c6d7f5bbe0e896fed39010428...	CPU Usage	5.93 mcores	11.9 mcores	4.85 mcores
	container_id:1c1127d113d11b6ba903ef9bf0...	CPU Usage	14.88 mcores	26.6 mcores	10.70 mcores
	container_id:3a0c1f3c37bf67b2a6cbd2f5b9a...	CPU Usage	29.14 mcores	55.3 mcores	15.18 mcores

Zebrium Root Cause Report Summaries

Zebrium Detected Root Cause Report #source:my_apps #ze_deployment:trial #ze_first_occurrence:true #ze_service_group:shop #ze_significance:medium

NLP Summary: The kubelet was unable to create the order due to timeout from one of the services.

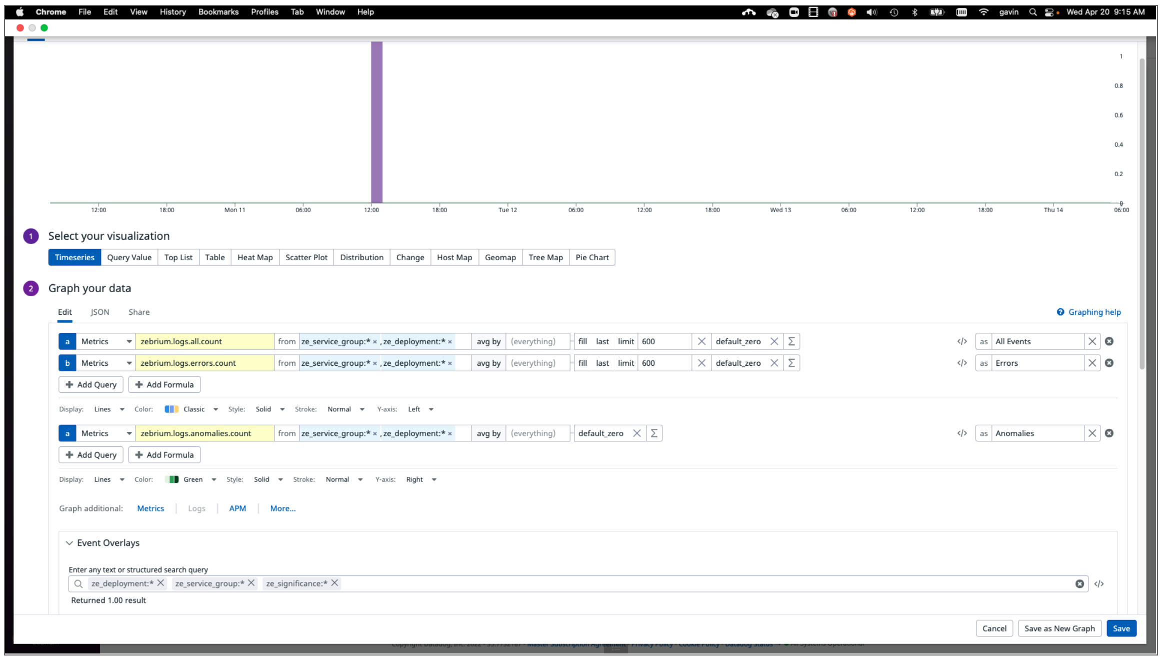
Word Cloud:

mongodb **sock-chaos-runner** **carts** **exception** **org**
socket dispatcherservlet nested threw **sock-shop**
cause java carts-db cri-socket unable

Explore Root Cause Report

Mon Apr 11 2022 12:56:53 GMT-0400 (Eastern Daylight Time)

The following image displays the definition of the **Root Cause Finder** panel:


1 Select your visualization

Timeseries | Query Value | Top List | Table | Heat Map | Scatter Plot | Distribution | Change | Host Map | Geomap | Tree Map | Pie Chart

2 Graph your data

Edit | JSON | Share

a Metrics zebrium.logs.all.count from ze_service_group:* , ze_deployment:* avg by (everything) fill last limit 600 default_zero as All Events

b Metrics zebrium.logs.errors.count from ze_service_group:* , ze_deployment:* avg by (everything) fill last limit 600 default_zero as Errors

Display: Lines Color: Classic Style: Solid Stroke: Normal Y-axis: Left

a Metrics zebrium.logs.anomalies.count from ze_service_group:* , ze_deployment:* avg by (everything) default_zero as Anomalies

Display: Lines Color: Green Style: Solid Stroke: Normal Y-axis: Right

Graph additional: Metrics | Logs | APM | More...

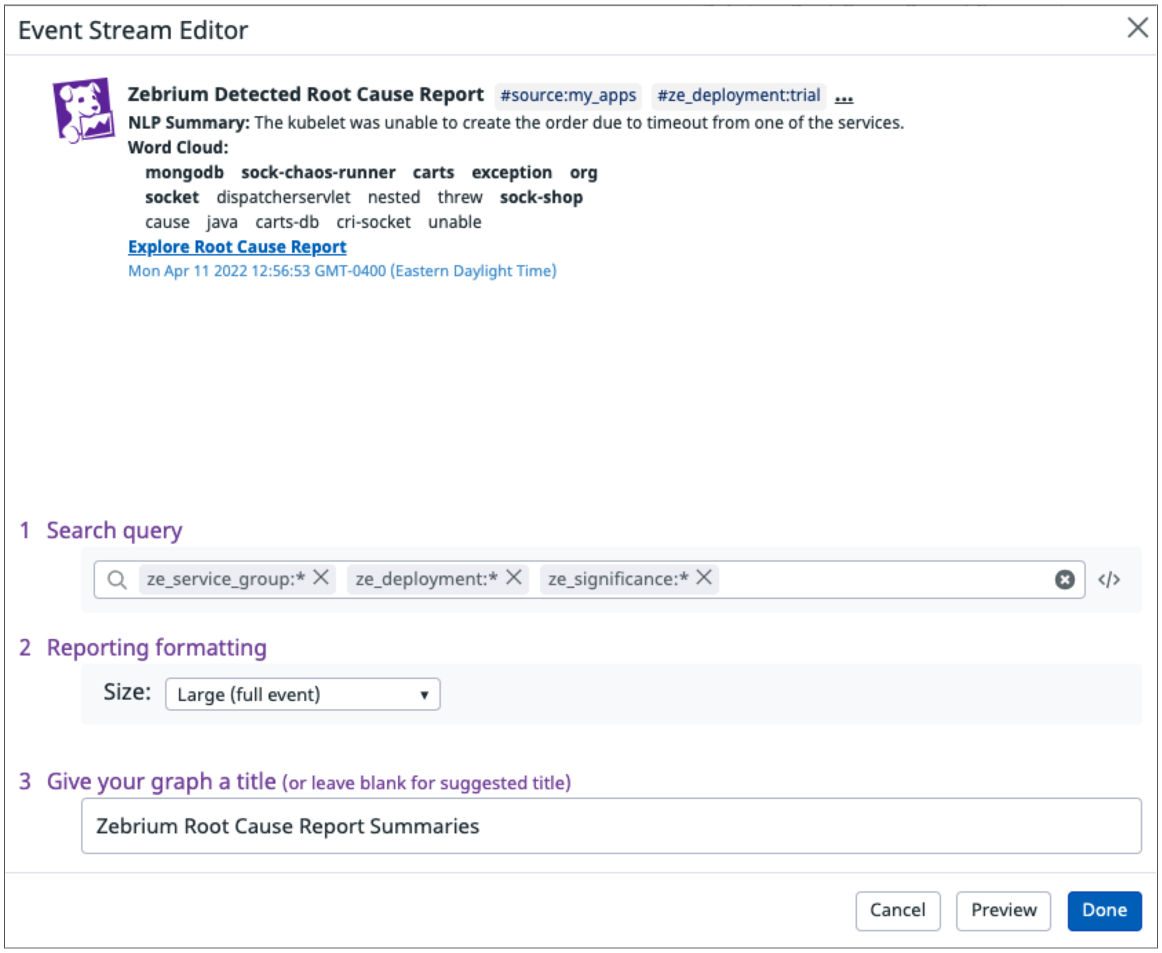
Event Overlays

Enter any text or structured search query

ze_deployment:* ze_service_group:* ze_significance:*

Returned 1.00 result

The following image displays the definition of the **Root Cause Reports Summary** panel:


Event Stream Editor

Zebrium Detected Root Cause Report #source:my_apps #ze_deployment:trial ...

NLP Summary: The kubelet was unable to create the order due to timeout from one of the services.

Word Cloud:

mongodb sock-chaos-runner carts exception org
socket dispatcherservlet nested threw sock-shop
cause java carts-db cri-socket unable

Explore Root Cause Report

Mon Apr 11 2022 12:56:53 GMT-0400 (Eastern Daylight Time)

1 Search query

ze_service_group:* ze_deployment:* ze_significance:*

2 Reporting formatting

Size: Large (full event)

3 Give your graph a title (or leave blank for suggested title)

Zebrium Root Cause Report Summaries

Important Metric Names

Metric Name	Description
zebrium.logs.all.count	Count of all log events received in a one-minute duration (per service_group and deployment).
zebrium.logs.anomalies.count	Count of anomaly log events received in a one-minute duration (per service_group and deployment).
zebrium.logs.errors.count	Count of error log events received in a one-minute duration (per service_group and deployment).
ze_service_group	Zebrium service group name for the corresponding metric or event.
ze_deployment	Zebrium deployment name for the corresponding metric or event.
ze_significance	Significance of the Root Cause Report (low, medium or high).

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

Receiving Signals from Datadog Triggered Monitors

Integration Overview

1. Create an API Key in Datadog.
2. Create a Datadog Integration in Zebrium using the information from step 1.
3. Create a Webhook Integration in Datadog using the information from step 2.
4. Add Webhook notifications to your Triggered Monitors in Datadog.
5. Add Zebrium Root Cause Reports to your Datadog Dashboard.

Integration Details

STEP 1: Create an API Key in Datadog

1. From the Main Navigation panel, hover over your Datadog Login Name and select **Organization Settings**.
2. Click on **API Keys**.
3. Click the **[+ New Key]** button.
4. Enter a **Name** for the API Key and click **Create Key**.
5. Copy and save the **Key** for use in STEP 2, below.

STEP 2: Create a Datadog Integration in Zebrium to Receive Signals from Datadog

1. From the User menu area in Zebrium, click the **Settings** menu (hamburger) .
2. Select **Integrations**.
3. Scroll to the **Observability Dashboards** section and click on **Datadog Events and Metrics**.
4. Click the **[Create a New Integration]** button.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **[Receive Signals]** tab.
10. Click the **[Enabled]** button.
11. Enter the **API Key** created in STEP 1, above.
12. Click **[Save]**.
13. Copy the **Webhook URL** and save it for use in STEP 3, below. Click **[OK]**.

STEP 3: Create a Webhook Integration in Datadog

1. From the Main Navigation panel, navigate to Integrations/Integrations.
2. Locate the Webhooks integration card and click Configure.
3. Click on the New button located in the Webhooks **section and enter a **Name and the URL saved in STEP 2.
4. In the Payload section, add: "alert_transition": "$ALERT_TRANSITION" after "event_type": "$EVENT_TYPE",
5. Click **[Save]**.

STEP 4: Add Webhook notifications to your Triggered Monitors in Datadog

1. From the Main Navigation panel, navigate to Monitors/Manage Monitors.
2. Click on the Monitor you wish to trigger Root Cause Reports.
3. Choose Edit from the gear icon on the Monitor page.
4. Add the webhook (from STEP 3) in the Notify your team list.
5. Click Save.

STEP 5: Add Zebrium Root Cause Report Detections to any of your Datadog Dashboards

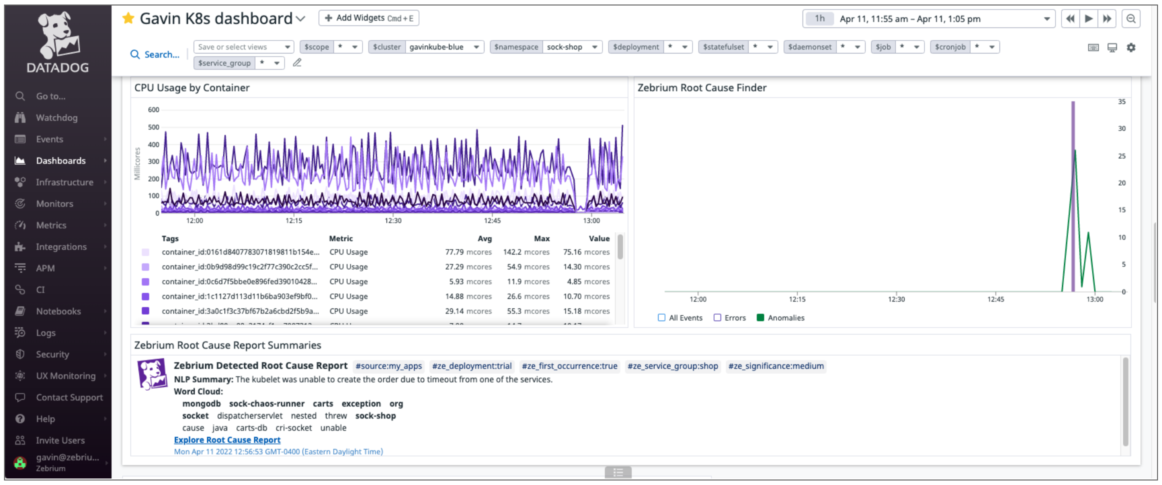
Zebrium sends events to Datadog as follows:

- **Events** are sent each time a Zebrium Root Cause Report Detection occurs.

Visualizing Zebrium Data in Datadog

The following image displays a sample chart visualization showing:

1. A **Root Cause Finder** panel that displays a vertical bar whenever a Zebrium detection occurs. This allows you to easily see detections that are aligned with other metrics on your dashboards.
2. A **Root Cause Reports Summary** panel that list summary information for each Zebrium detection.


Gavin K8s dashboard

Apr 11, 11:55 am – Apr 11, 1:05 pm

$scope * | $cluster gavinkube-blue | $namespace sock-shop | $deployment * | $statefulset * | $daemonset * | $job * | $cronjob * | $service_group *

CPU Usage by Container

Tags	Metric	Avg	Max	Value
container_id:0161d8407783071819811b154e...	CPU Usage	77.79 mcores	142.2 mcores	75.16 mcores
container_id:0b9d98d99c19c2f77c390c2cc5f...	CPU Usage	27.29 mcores	54.9 mcores	14.30 mcores
container_id:0c6d7f5bbe0e896fed39010428...	CPU Usage	5.93 mcores	11.9 mcores	4.85 mcores
container_id:1c1127d113d11b6ba903ef9bf0...	CPU Usage	14.88 mcores	26.6 mcores	10.70 mcores
container_id:3a0c1f3c37bf67b2a6cbd2f5b9a...	CPU Usage	29.14 mcores	55.3 mcores	15.18 mcores

Zebrium Root Cause Finder

Zebrium Root Cause Report Summaries

Zebrium Detected Root Cause Report #source:my_apps #ze_deployment:trial #ze_first_occurrence:true #ze_service_group:shop #ze_significance:medium

NLP Summary: The kubelet was unable to create the order due to timeout from one of the services.

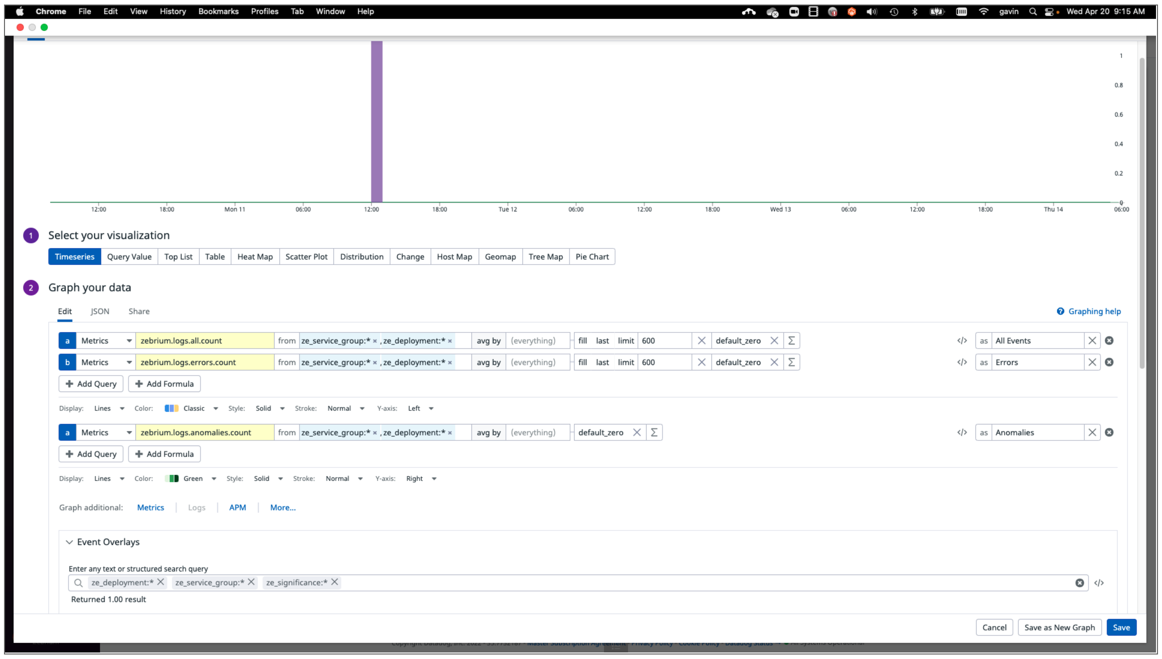
Word Cloud:

mongodb **sock-chaos-runner** **carts** **exception** **org**
socket dispatcherservlet nested threw **sock-shop**
cause java carts-db cri-socket unable

Explore Root Cause Report

Mon Apr 11 2022 12:56:53 GMT-0400 (Eastern Daylight Time)

The following image displays the definition of the **Root Cause Finder** panel:


1 Select your visualization

Timeseries | Query Value | Top List | Table | Heat Map | Scatter Plot | Distribution | Change | Host Map | Geomap | Tree Map | Pie Chart

2 Graph your data

Edit JSON Share

```
a  Metrics  zebrium.logs.all.count       from ze_service_group:* , ze_deployment:*  avg by (everything)  fill last limit 600  default_zero  as All Events
b  Metrics  zebrium.logs.errors.count    from ze_service_group:* , ze_deployment:*  avg by (everything)  fill last limit 600  default_zero  as Errors
Display: Lines  Color: Classic  Style: Solid  Stroke: Normal  Y-axis: Left

a  Metrics  zebrium.logs.anomalies.count from ze_service_group:* , ze_deployment:*  avg by (everything)  default_zero  as Anomalies
Display: Lines  Color: Green  Style: Solid  Stroke: Normal  Y-axis: Right
```

Graph additional: Metrics | Logs | APM | More...

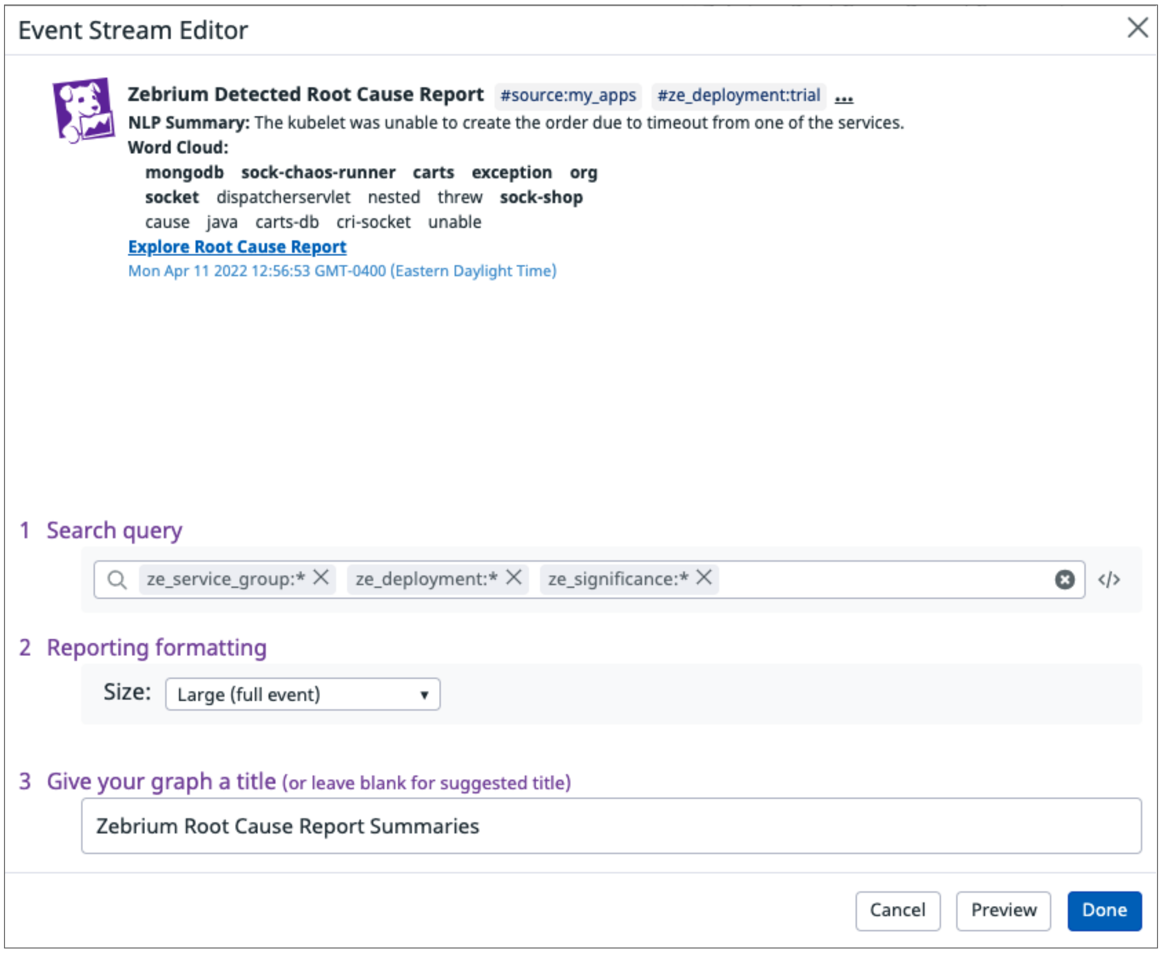
Event Overlays

Enter any text or structured search query

ze_deployment:* ze_service_group:* ze_significance:*

Returned 1.00 result

The following image displays the definition of the **Root Cause Reports Summary** panel:


Event Stream Editor

Zebrium Detected Root Cause Report #source:my_apps #ze_deployment:trial ...

NLP Summary: The kubelet was unable to create the order due to timeout from one of the services.

Word Cloud:

mongodb sock-chaos-runner carts exception org
socket dispatcherservlet nested threw sock-shop
cause java carts-db cri-socket unable

Explore Root Cause Report

Mon Apr 11 2022 12:56:53 GMT-0400 (Eastern Daylight Time)

1 Search query

ze_service_group:* ze_deployment:* ze_significance:*

2 Reporting formatting

Size: Large (full event)

3 Give your graph a title (or leave blank for suggested title)

Zebrium Root Cause Report Summaries

Important Metric Names

Metric Name	Description
zebrium.logs.all.count	Count of all log events received in a one-minute duration (per service_group and deployment).
zebrium.logs.anomalies.count	Count of anomaly log events received in a one-minute duration (per service_group and deployment).
zebrium.logs.errors.count	Count of error log events received in a one-minute duration (per service_group and deployment).
ze_service_group	Zebrium service group name for the corresponding metric or event.
ze_deployment	Zebrium deployment name for the corresponding metric or event.
ze_significance	Significance of the Root Cause Report (low, medium or high).

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

Dynatrace

Features

- Automatically adds Root Cause Reports as Events in Dynatrace. This allows you to see details of root cause on any Dynatrace dashboard.
- Automatically adds Log count metrics in Dynatrace.
- Each Zebrium RC Report includes a summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium user interface.
- Leads to faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

How it Works

Our recommended mode of operation for Observability Dashboard integrations is to use the Zebrium **Auto-Detect** mode as an accurate mechanism for explaining the reason something went wrong. In this mode, you continue to use your existing rules, alerts and metrics as the primary source of problem detection. You can then review Zebrium RC Report findings directly in your Dynatrace Dashboards alongside other metrics to explain the reason behind problems you were alerted on.

Augment mode is useful when you have monitors defined in Dynatrace and you want a Root Cause Report automatically generated at the time of the alert. In this mode, Zebrium uses a Dynatrace webhook as a notification channel and will update your Dashboard with Root Cause Reports that coincide with the triggering monitor so they're immediately visible to you as you work the issue.

The two modes of operation are independent. You can configure Auto-Detect and/or Augment modes depending on your operational use-case.

Auto-Detect (recommended): Send Root Cause Detections to Dynatrace Dashboards

1. Zebrium continuously monitors all application logs and uses unsupervised machine learning to find anomalous log patterns that indicate a problem. These are automatically turned into Root Cause Reports highlighting details of any problems with over 95% accuracy.
2. Root Cause Report summaries are sent to Dynatrace using the events API and Root Cause details are visible on your Dynatrace Dashboards.
3. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Dashboard.
4. Log metrics are also sent to Dynatrace via the metrics API for visualization on your Dynatrace Dashboards.

CLICK HERE to send Root Cause Detections to your Dynatrace Dashboards

Augment (advanced users): Receive Signals from Dynatrace Triggered Monitors

1. Any Dynatrace Monitor can trigger a webhook request for Root Cause Analysis from Zebrium.
2. Zebrium finds anomalous log patterns from your application that coincide with the event and creates a Root Cause Report.
3. Root Cause Report summaries are sent to Dynatrace using the events API and Root Cause details are visible on your Dynatrace Dashboards.
4. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Dashboard.

CLICK HERE to receive Signals from Dynatrace Triggered Monitors

Sending Root Cause Detections to Dynatrace Dashboards

Please Contact Zebrium for Early Access to this feature.

Receiving Signals from Dynatrace Triggered Monitors

Please Contact Zebrium for Early Access to this feature.

Elastic Stack

Features

- Automatically adds Root Cause Reports as Detection metrics in Elastic. This allows you to see details of root cause on any Kibana dashboard.
- Automatically adds Log metrics into Elastic.
- Each Zebrium RC Report includes a summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium user interface.
- Leads to faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

How it Works

Our recommended mode of operation for Observability Dashboard integrations is to use the Zebrium **Auto-Detect** mode as an accurate mechanism for explaining the reason something went wrong. In this mode, you continue to use your existing rules, alerts and metrics as the primary source of problem detection. You can then review Zebrium RC Report findings directly in your Kibana Dashboards alongside other metrics to explain the reason behind problems you were alerted on.

Auto-Detect: Send Root Cause Detections to your Kibana Dashboards

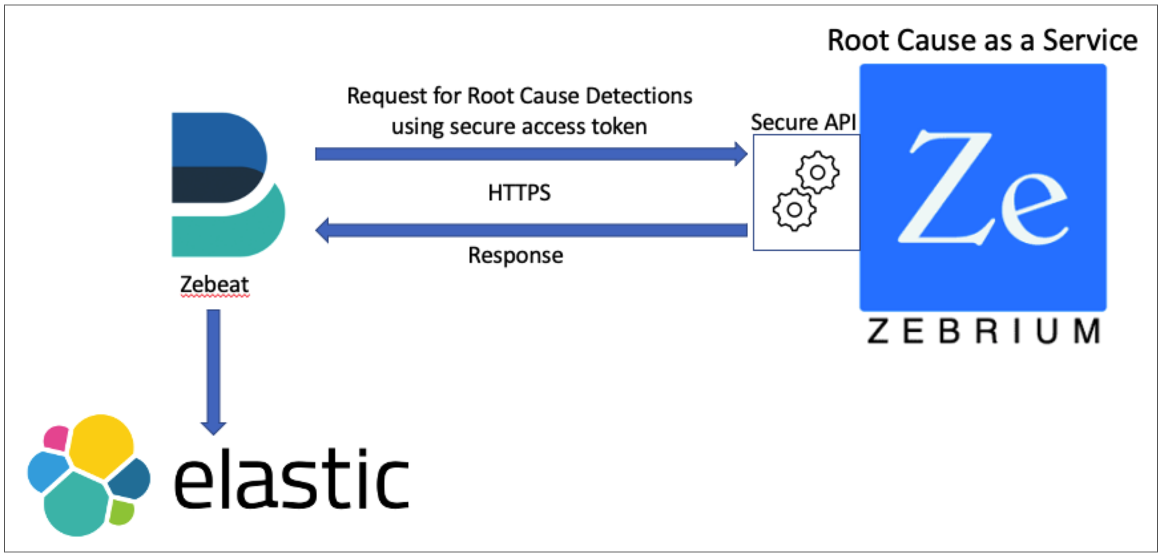
1. Zebrium continuously monitors all application logs and uses unsupervised machine learning to find anomalous log patterns that indicate a problem. These are automatically turned into Root Cause Reports highlighting details of any problems with over 95% accuracy.
2. Root Cause Report summaries are sent to Elastic via the Zebeat log shipper as metrics and Root Cause details are visible on your Kibana Dashboards.
3. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Dashboard.
4. Log metrics are also sent to Elastic via the Zebeat log shipper for visualization on your Kibana Dashboards.

CLICK HERE to send Root Cause Detections to your Kibana Dashboards

Sending Root Cause Detections to Your Kibana Dashboards

Integration Overview

1. Create a secure access token in Zebrium for the Zebeat collector.
2. Create Zebeat Override File and Deploy in your Kubernetes Environment using helm.
3. Create a visualization in your Kibana Dashboard using the Root Cause Report and log data provided by Zebeat.



Integration Details

STEP 1: Create a Secure Access Token in Zebrium

1. From the User menu area, click the **Settings** menu (hamburger).
2. Select **Access Tokens**.
3. Click **[+ Add Access Token]** button.
4. Enter a **Name** for the token.
5. Select *Viewer* for the **Role**.
6. Select the **Deployment** for the token.
7. Click the **[Add]** button.
8. Copy the **Access Token** that was just created and save for use in STEP 2.

STEP 2: Create Zebeat Override File and Deploy in your Kubernetes Environment

Create the Zebeat Override File:

1. Go to the Zebeat github repository at https://github.com/zebrium/helm-charts/tree/main/charts/zebeat.
2. Navigate to the **examples** directory.
3. Zebeat can send Root Cause Report data to Logstash or Elasticsearch directly. Choose one of the logstash or elasticsearch **.yaml** files as a template for the zebeat **override.yaml** file you will use when deploying the Zebeat chart.
4. Copy the contents of the **.yaml** file template to your local disk as **override.yaml** so you can customize for your environment.
5. Edit your local copy of the **override.yaml** file and make the following updates:

    - In the host parameter of the **metricbeat.modules** section, add the FQHN for your Zebrium instance where you generated the Access Token in STEP 1, above. For Zebrium SaaS, this will typically be: **https://cloud.zebrium.com**.
    - In the **access_tokens.yaml** parameter of the **accessTokens** section, add the FQHN for your Zebrium instance and the Access Token generated in STEP 1.
    - In the **output.elasticsearch** or **output.logstash** section, add the appropriate host for your Elastic deployment and any necessary credentials.
    - Save the override.yaml **file.**

Deploy Zebeat in your Kubernetes Environment

To install the chart with the release name **zebrium**, run the following commands:

```
helm repo add zebrium http://charts.zebrium.com
```

```
helm upgrade -i zebeat zebrium/zebeat --namespace zebrium --create-
namespace -f override.yaml
```

STEP 3: Create Visualizations in your Dashboard

Zebeat provides two metric sets for visualizing Zebrium RCaaS data in Elastic:

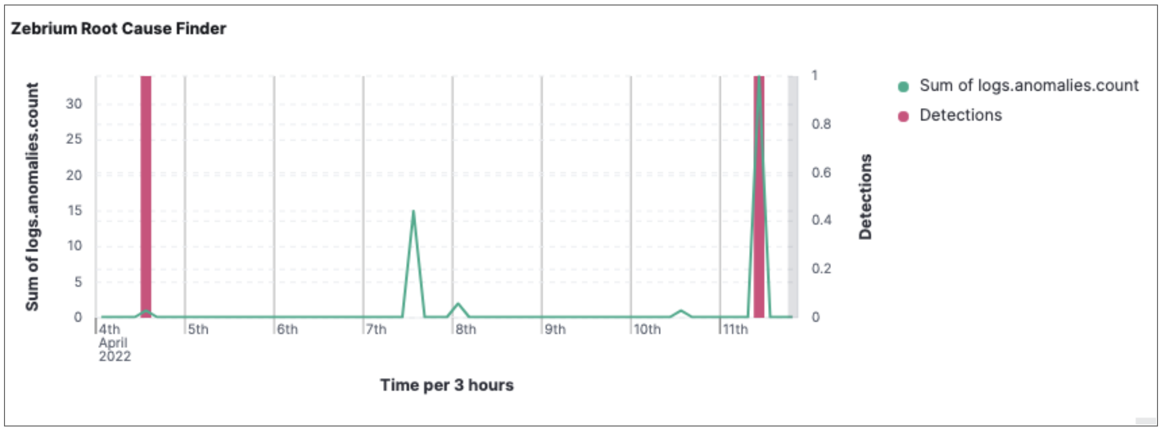
1. **Detections** provides Root Cause Report data.
2. **Logs** provides metrics on Log Event counts.

Visualizing in Kibana

The following image displays a sample chart visualization showing:

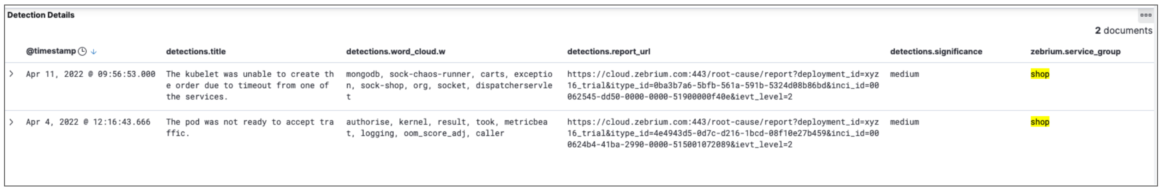
1. The sum Detections from the **detections** metric set using **detections.alwaysone.count** plotted as a bar chart with a Y-axis on the right-hand side.

2. Sum of Anomalies from the **logs** metricset using **logs.anomalies.count** plotted as a line chart with a Y-axis on the left-hand side.



Below is a sample Search visualization showing the following Root Cause Report details:

1. **detections.title**. NLP Summary.
2. **detections.word_cloud.w**. List of Word Cloud strings.
3. **detections.report_url**. Link for viewing full Root Cause Report details in the Zebrium portal.
4. **detections.significance**. Significance of the Root Cause analysis determined by Zebrium ML (low, medium, high).
5. **detections.service_group**. Service group where Root Cause detection was found.



Detection Details

2 documents

@timestamp	detections.title	detections.word_cloud.w	detections.report_url	detections.significance	zebrium.service_group
Apr 11, 2022 @ 09:56:53.000	The kubelet was unable to create the order due to timeout from one of the services.	mongodb, sock-chaos-runner, carts, exception, sock-shop, org, socket, dispatcherservlet	https://cloud.zebrium.com:443/root-cause/report?deployment_id=xyz16_trial&itype_id=0ba3b7a6-5bfb-561a-591b-5324d08b86bd&inci_id=00062545-dd50-0000-0000-51900000f40e&ievt_level=2	medium	shop
Apr 4, 2022 @ 12:16:43.666	The pod was not ready to accept traffic.	authorise, kernel, result, took, metricbeat, logging, oom_score_adj, caller	https://cloud.zebrium.com:443/root-cause/report?deployment_id=xyz16_trial&itype_id=4e4943d5-0d7c-d216-1bcd-08f10e27b459&inci_id=000624b4-41ba-2990-0000-515001072089&ievt_level=2	medium	shop

Important Metric Names

Metric Name	Description
logs.all.count	Count of all log events received in a one-minute duration (per service_group)
logs.anomalies.count	Count of anomaly log events received in a one-minute duration (per service_ group)
logs.errors.count	Count of error log events received in a one-minute duration (per service_group)
detections.alwaysone.count	Set to 1 each time there is a Zebrium Root Cause Report detection
detections.title	Title of the Root Cause Report (usually an NLP summary)
detections.word_cloud.w	List of words in the word cloud of the Root Cause Report (per service_group)
detections.report_url	URL of the Root Cause Report

Metric Name	Description
detections.significance	Significance of the Root Cause Report (low, medium or high)
zebrium.service_group	Zebrium service group name for the corresponding metric or detection

Sample Payloads for Detections and Logs Metricsets

Detections Metricset Payload

```
{
  "_index": ".ds-metricbeat-8.3.0-2022.04.07-000001",
  "_id": "u-aUGYABqSxIAr_l5fTX",
  "_version": 1,
  "_score": 1,
  "_source": {
    "@timestamp": "2022-04-11T16:56:53.000Z",
    "event": {
      "module": "zebrium",
      "duration": 292227850,
      "dataset": "detections"
    },
    "metricset": {
      "name": "detections",
      "period": 10000
    },
    "ecs": {
      "version": "8.0.0"
    },
    "host": {
      "name": "zebeat-67d8d6457b-8rblk"
    },
    "agent": {
      "type": "metricbeat",
      "version": "8.3.0",
      "ephemeral_id": "5c5a0778-b163-4187-916e-5fc1b730fbde",
      "id": "6c216ce2-16cc-4313-802d-2203a604159c",
      "name": "zebeat-67d8d6457b-8rblk"
    },
    "service": {
      "address": "https://cloud.zebrium.com",
      "type": "zebrium"
    },
    "zebrium": {
      "customer": "xyz16",
      "deployment": "trial",
      "service_group": "shop"
    },
    "detections": {
      "report_url": "https://cloud.zebrium.com:443/root-cause/re-
port?deployment_id=
xyz16_trial&itype_id=0ba3b7a6-5bfb-561a-591b-5324d08b86bd&inci_
id=00062545-dd50-0000-
0000-51900000f40e&ievt_level=2",
      "occurrence": {
        "count": 1
      },
      "word_cloud": [
        {
          "w": "mongodb",
          "b": 7,
          "s": 8
        },
        {
          "b": 8,
          "s": 7,
          "w": "sock-chaos-runner"
        },
        {
          "w": "carts",
          "b": 7,
          "s": 7
        },
        {
          "s": 6,
          "w": "exception",
          "b": 6
        },
        {
          "b": 6,
          "s": 3,
          "w": "sock-shop"
        },
        {
          "s": 6,
          "w": "org",
          "b": 5
        },
        {
          "b": 5,
          "s": 5,
          "w": "socket"
        },
        {
          "s": 5,
          "w": "dispatcherservlet",
          "b": 2
        }
      ],
      "alwaysone": {
        "count": 1
      },
      "includes_default": true,
      "title": "The kubelet was unable to create the order due to timeout
from one of the services.",
      "significance": "medium"
    }
  },
  "fields": {
    "zebrium.service_group": [
      "shop"
    ],
    "detections.includes_default": [
      true
    ],
    "zebrium.deployment": [
      "trial"
    ],
    "zebrium.customer": [
      "xyz16"
    ],
    "service.type": [
    "zebrium"
  ],
  "agent.type": [
    "metricbeat"
  ],
  "detections.occurrence.count": [
    1
  ],
  "logstash_stats.timestamp": [
    "2022-04-11T16:56:53.000Z"
  ],
  "event.module": [
    "zebrium"
  ],
  "detections.word_cloud.b": [
    7,
    8,
    7,
    6,
    6,
    5,
    5,
    2
  ],
  "agent.name": [
    "zebeat-67d8d6457b-8rblk"
  ],
  "host.name": [
    "zebeat-67d8d6457b-8rblk"
  ],
  "beats_state.timestamp": [
    "2022-04-11T16:56:53.000Z"
  ],
  "beats_state.state.host.name": [
    "zebeat-67d8d6457b-8rblk"
  ],
  "timestamp": [
    "2022-04-11T16:56:53.000Z"
  ],
  "detections.report_url": [
      "https://cloud.zebrium.com:443/root-cause/report?deployment_
id=xyz16_trial&itype_id=
0ba3b7a6-5bfb-561a-591b-5324d08b86bd&inci_id=00062545-dd50-0000-0000-
51900000f40e&ievt_level=2"
    ],
    "detections.word_cloud.w": [
      "mongodb",
      "sock-chaos-runner",
      "carts",
      "exception",
      "sock-shop",
      "org",
      "socket",
      "dispatcherservlet"
    ],
    "detections.title": [
      "The kubelet was unable to create the order due to timeout from one
of the services."
    ],
    "kibana_stats.timestamp": [
      "2022-04-11T16:56:53.000Z"
    ],
    "detections.alwaysone.count": [
      1
    ],
    "metricset.period": [
      10000
    ],
    "detections.word_cloud.s": [
      8,
      7,
      7,
      6,
      3,
      6,
      5,
      5
    ],
    "agent.hostname": [
      "zebeat-67d8d6457b-8rblk"
    ],
    "metricset.name": [
      "detections"
    ],
    "event.duration": [
      292227850
    ],
    "@timestamp": [
      "2022-04-11T16:56:53.000Z"
    ],
    "agent.id": [
      "6c216ce2-16cc-4313-802d-2203a604159c"
    ],
    "ecs.version": [
      "8.0.0"
    ],
    "service.address": [
      "https://cloud.zebrium.com"
    ],
    "agent.ephemeral_id": [
      "5c5a0778-b163-4187-916e-5fc1b730fbde"
    ],
    "agent.version": [
      "8.3.0"
    ],
    "event.dataset": [
      "detections"
    ],
    "detections.significance": [
      "medium"
    ]
  }
}
```

Logs Metricset Payload

```
{
  "_index": ".ds-metricbeat-8.3.0-2022.04.07-000001",
  "_id": "Xi5MG4ABTsyT1lUpY2dd",
  "_version": 1,
  "_score": 1,
  "_source": {
    "@timestamp": "2022-04-12T00:52:00.000Z",
    "event": {
      "dataset": "logs",
      "module": "zebrium",
      "duration": 144691043
    },
    "metricset": {
      "name": "logs",
      "period": 10000
    },
    "service": {
      "address": "https://cloud.zebrium.com",
      "type": "zebrium"
    },
    "zebrium": {
      "service_group": "default",
      "customer": "xyz16",
      "deployment": "trial"
    },
    "logs": {
      "errors": {
        "count": 0
      },
      "anomalies": {
        "count": 0
      },
      "all": {
        "count": 27
      }
    },
    "ecs": {
      "version": "8.0.0"
    },
    "host": {
      "name": "zebeat-67d8d6457b-8rblk"
    },
    "agent": {
      "version": "8.3.0",
      "ephemeral_id": "5c5a0778-b163-4187-916e-5fc1b730fbde",
      "id": "6c216ce2-16cc-4313-802d-2203a604159c",
      "name": "zebeat-67d8d6457b-8rblk",
      "type": "metricbeat"
    }
  },
  "fields": {
    "zebrium.service_group": [
      "default"
    ],
    "zebrium.deployment": [
      "trial"
    ],
    "zebrium.customer": [
      "xyz16"
    ],
    "service.type": [
      "zebrium"
    ],
    "agent.type": [
      "metricbeat"
    ],
    "logstash_stats.timestamp": [
      "2022-04-12T00:52:00.000Z"
    ],
    "event.module": [
      "zebrium"
    ],
    "agent.name": [
      "zebeat-67d8d6457b-8rblk"
    ],
    "host.name": [
      "zebeat-67d8d6457b-8rblk"
    ],
    "beats_state.timestamp": [
      "2022-04-12T00:52:00.000Z"
    ],
    "logs.anomalies.count": [
      0
    ],
    "beats_state.state.host.name": [
      "zebeat-67d8d6457b-8rblk"
    ],
    "timestamp": [
      "2022-04-12T00:52:00.000Z"
    ],
    "kibana_stats.timestamp": [
      "2022-04-12T00:52:00.000Z"
    ],
    "metricset.period": [
      10000
    ],
    "agent.hostname": [
      "zebeat-67d8d6457b-8rblk"
    ],
    "logs.errors.count": [
      0
    ],
    "metricset.name": [
      "logs"
    ],
    "event.duration": [
      144691043
    ],
    "@timestamp": [
      "2022-04-12T00:52:00.000Z"
    ],
    "agent.id": [
      "6c216ce2-16cc-4313-802d-2203a604159c"
    ],
    "ecs.version": [
      "8.0.0"
    ],
    "service.address": [
      "https://cloud.zebrium.com"
    ],
    "agent.ephemeral_id": [
      "5c5a0778-b163-4187-916e-5fc1b730fbde"
    ],
    "agent.version": [
      "8.3.0"
    ],
    "event.dataset": [
      "logs"
    ],
    "logs.all.count": [
      27
    ]
  }
}
```

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

Grafana Plugin

Features

- Automatically adds Root Cause Reports in Grafana. This allows you to see details of root cause on any Grafana dashboard.
- Automatically adds Log metrics in Grafana.
- Each Zebrium RC Report includes a summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium user interface.
- Leads to faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

How it Works

Our recommended mode of operation for Observability Dashboard integrations is to use the Zebrium **Auto-Detect** mode as an accurate mechanism for explaining the reason something went wrong. In this mode, you continue to use your existing rules, alerts and metrics as the primary source of problem detection. You can then review Zebrium RC Report findings directly in your Grafana Dashboards alongside other metrics to explain the reason behind problems you were alerted on.

Auto-Detect: View Root Cause Detections to your Grafana Dashboards

1. Zebrium continuously monitors all application logs and uses unsupervised machine learning to find anomalous log patterns that indicate a problem. These are automatically turned into Root Cause Reports highlighting details of any problems with over 95% accuracy.
2. Root Cause Report summaries and Log metrics are visible on your Grafana Dashboards using the Zebrium plugin.
3. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Dashboard.

***CLICK HERE* to View Root Cause Detections in your Grafana Dashboards**

Viewing Root Cause Detections in your Grafana Dashboards

Integration Overview

1. Create a secure access token in Zebrium for the Zebrium Datasource.
2. Download Plugins from GitHub.
3. Sign and Install Plugins.
4. Setup the Zebrium Datasource.
5. Install the Zebrium Root Cause Finder on a dashboard.

Prerequisites

- File and Web access to a Grafana Server running at minimum version 8.0. We recommend first downloading and installing a local copy of Grafana Server to test the setup procedures, before installing the plugins on your production Grafana instance. You can install a local copy of Grafana web server from here: https://grafana.com/grafana/download/8.3.6?pg=get&plcmt=selfmanaged-box1-cta1.
- Grafana Toolkit installed on your system. To install, run the **npm** command:

  ```
  npm i @grafana/toolkit
  ```

- Web access to your Zebrium instance.

Current Zebrium Plugins

- **Zebrium Root Cause Finder**. Plots counts of all log events, error events and rare events overlaid with detections found by Zebrium's machine learning.
- **Zebrium Data Source**. Provides the API access layer for Zebrium panel plugins.

Integration Details

STEP 1: Download Plugins from GitHub

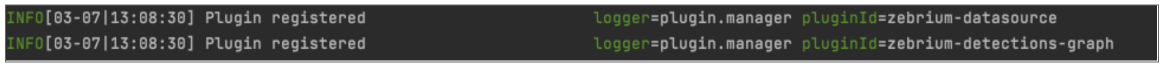
Use the following path to download the Zebrium Grafana plugins:

```
https://github.com/zebrium/grafana-plugin/releases
```

NOTE: This is a private repository in the current phase. For access, please contact Zebrium at support@zebrium.com.

STEP 2: Sign and Install Plugins

1. Stop your Grafana Server.
2. Move the plugins you downloaded from GitHub under your Grafana Server **plugins** directory. For example, if your Grafana Server is installed on the path **/Grafana-8.3.6**, you wuld install the downloaded plugins in ***/Grafana-8.3.6/data/plugins***.
3. Make sure that the Grafana Toolkit is installed on this system. If it isn't, run the **npm** command:
   ```
   npm i @grafana/toolkit
   ```
4. Follow the Grafana signing instructions for signing a private plugin: https://grafana.com/docs/grafana/latest/developers/plugins/sign-a-plugin/.
5. You must sign each plugin separately and you must sign them as a **private plugin**:
   - `cd /<PATH TO GRAFANA SERVER>/data/plugins/ze-datasource` and sign as a private plugin.
   - `cd /<PATH TO GRAFANA SERVER>/data/plugins/ze-detections` and sign as a private plugin.
6. Restart your Grafana Server. For example, if your server is in **/Grafana-8.3.6**, switch to that directory and run **./bin/grafana-server web**.
7. Check the Grafana server logs to verify that the new plugins have been registered:



```
INFO[03-07|13:08:30] Plugin registered                          logger=plugin.manager pluginId=zebrium-datasource
INFO[03-07|13:08:30] Plugin registered                          logger=plugin.manager pluginId=zebrium-detections-graph
```

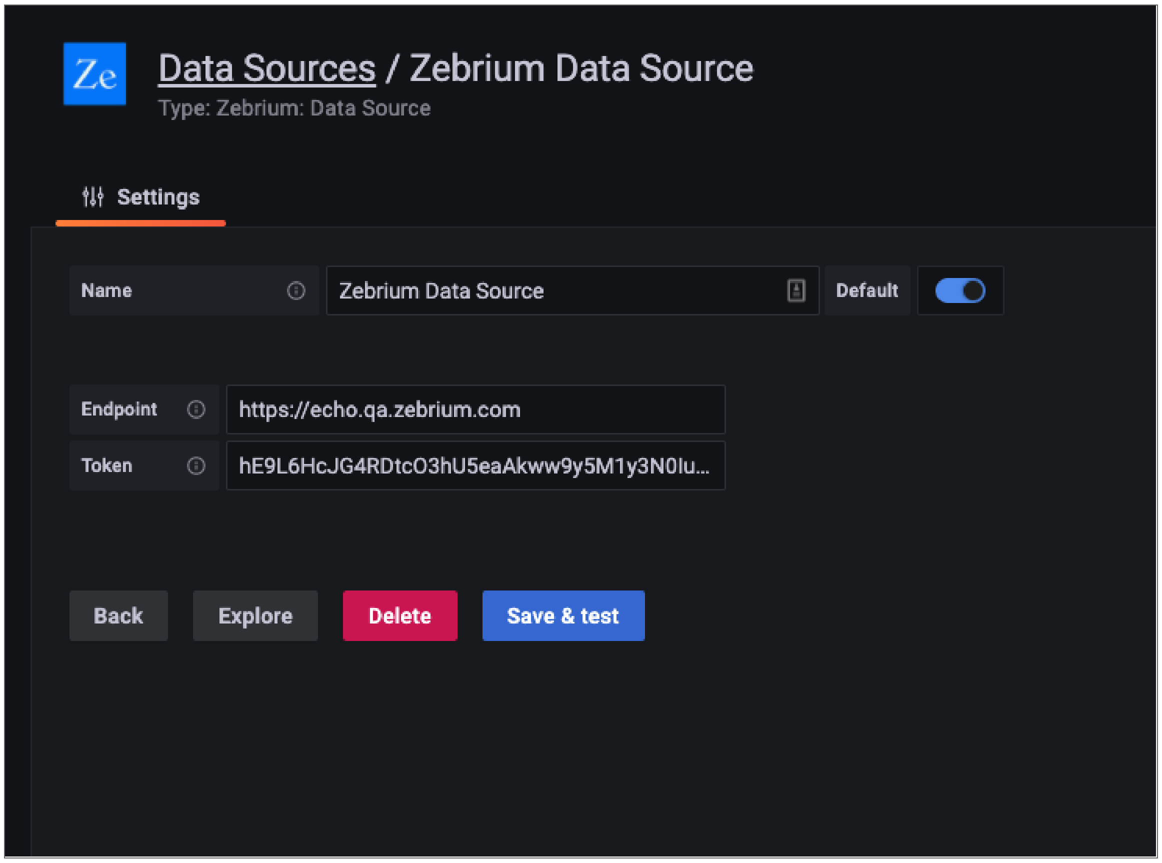
STEP 3: Create a Secure Access Token in Zebrium

1. From the User menu area, click the **Settings** menu (hamburger).
2. Select **Access Tokens**.
3. Click the **[+ Add Access Token]** button.
4. Enter a **Name** for the token.
5. Select *Viewer* for the **Role**.
6. Select the **Deployment** for the token.
7. Click the **[Add]** button.
8. Copy the **Access Token** that was just created and save it for use in STEP 4.

STEP 4: Set up the Zebrium Datasource

1. Open the Grafana server in a web browser. If you have installed a local Grafana Server, it is normally found at **http://localhost:3000**.
2. On the left toolbar, find the **Configure** menu (gear icon) and select data source.
3. Click **[Add a Datasource]**.
4. Scroll through the list of data sources and select *Zebrium Data Source*.

5. On the **Settings** page, enter the API endpoint. This is the FQDN for your Zebrium instance where you generated the Access Token in STEP 3. For Zebrium SaaS, this will typically be: **https://cloud.zebrium.com**.
6. Also on the **Settings** page, enter the access token you created from STEP 3.
7. Click **[Save & test]**.
8. Verify that you see the message *Successfully connected to Zebrium*. If you don't, review your entries above.


Data Sources / Zebrium Data Source

Type: Zebrium: Data Source

Settings

Name: Zebrium Data Source

Default: on

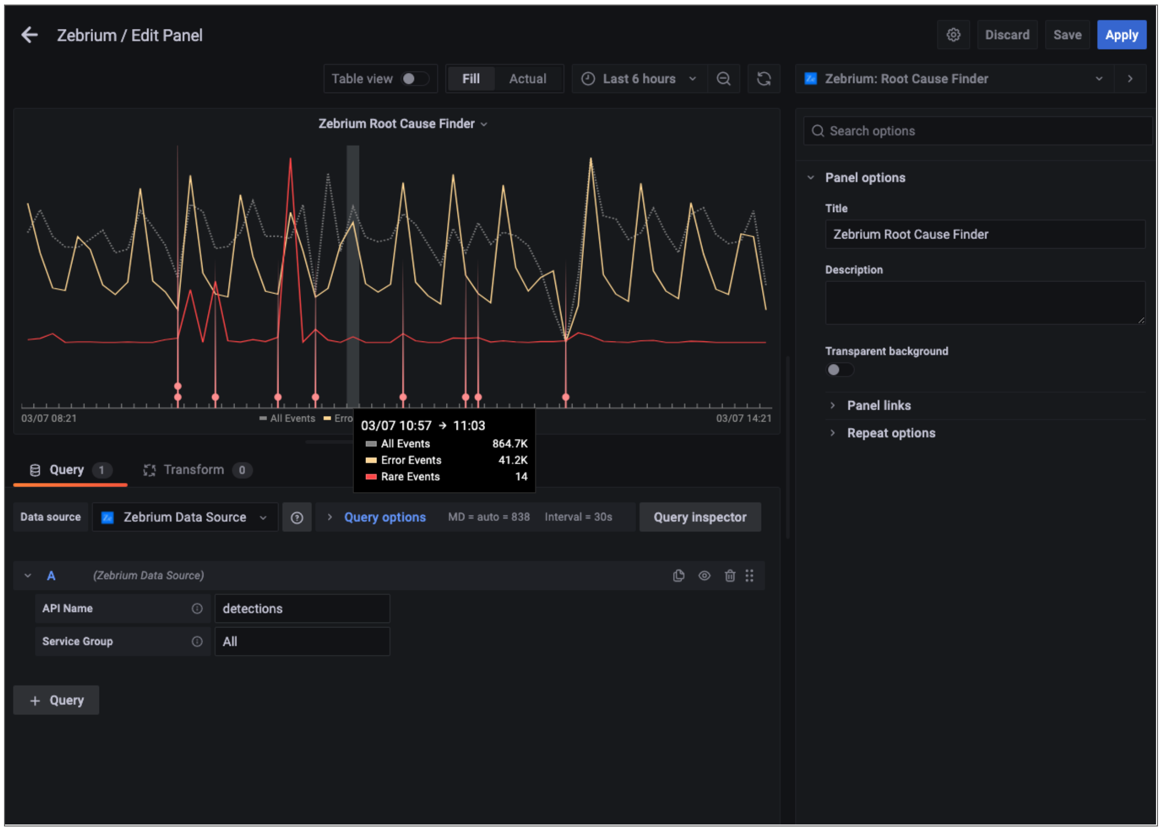
Endpoint: https://echo.qa.zebrium.com

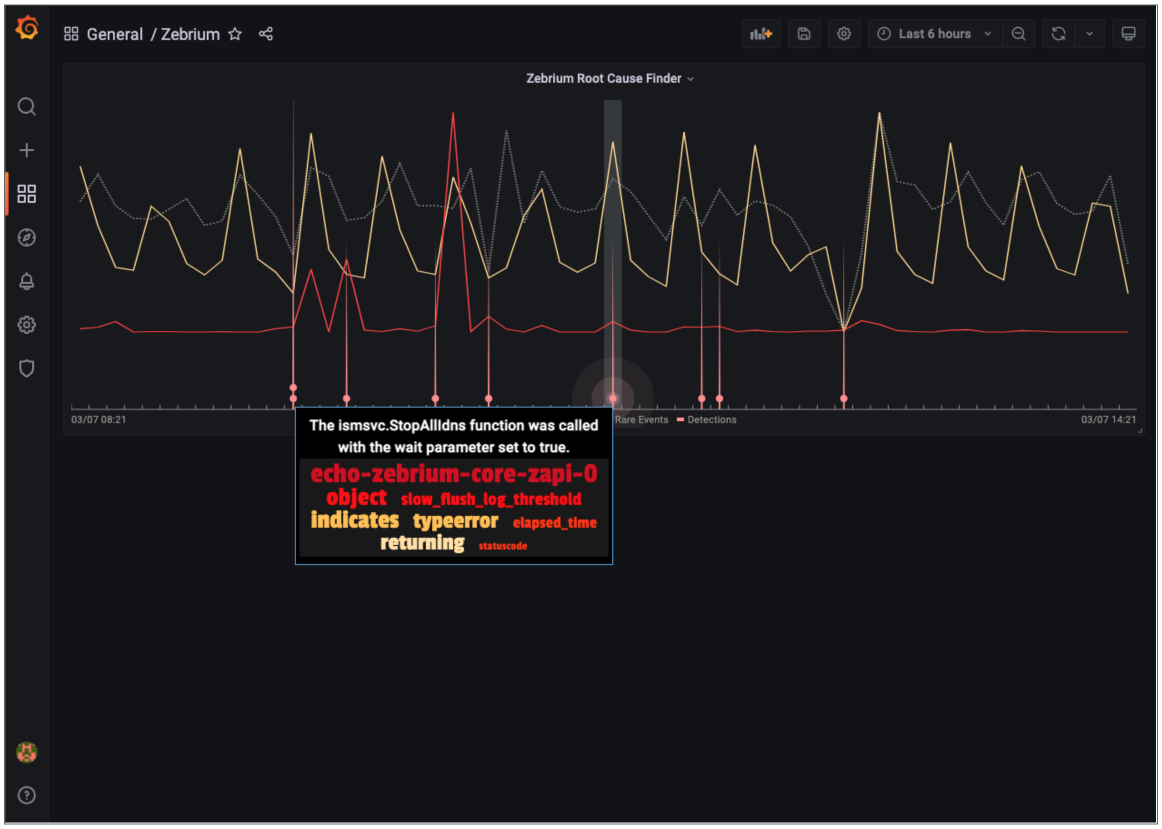
Token: hE9L6HcJG4RDtcO3hU5eaAkww9y5M1y3N0lu...

STEP 5: Install Zebrium Root Cause Finder on a Dashboard

1. Open the Grafana server in a web browser. If you have installed a local Grafana Server, it is normally found at **http://localhost:3000**.
2. On the left toolbar, find the **Create** menu (plus icon) and select *dashboard*.
3. On this new dashboard, click **[Add an Empty Panel]**. The panel setup screen appears.
4. In the upper right, open the list of visualizations and select the **Zebrium Root Cause Finder** plugin.
5. In the query panel, select **Zebrium Data Source**.
6. For ***API Name***, enter *detections*.
7. For **[Service Group]**, enter *All* to pull counts and detection data from all of your service groups. Alternatively, enter the name of a single service group to pull data from just that service group.

8. Give your panel a name; we suggest *Zebrium Root Cause Finder*. Then save your Dashboard and give it a name.
9. Verify that the **Zebrium Root Cause Finder** visualization shows up in the panel you just added:





The ismsvc.StopAllIdns function was called with the wait parameter set to true.

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

New Relic

Features

- Automatically adds Root Cause Reports as Events in New Relic. This allows you to see details of root cause on any New Relic dashboard.
- Automatically adds Log metrics in New Relic.
- Each Zebrium RC Report includes a clear summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium user interface.
- Leads to faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

How it Works

Our recommended mode of operation for Observability Dashboard integrations is to use the Zebrium **Auto-Detect** mode as an accurate mechanism for explaining the reason something went wrong. In this mode, you continue to use your existing rules, alerts and metrics as the primary source of problem detection. You can then review Zebrium RC Report findings directly in your New Relic Dashboards alongside other metrics to explain the reason behind problems you were alerted on.

Augment mode is useful when you have alerts defined in New Relic and you want a Root Cause Report automatically generated at the time of the alert. In this mode, Zebrium uses a New Relic webhook as a notification channel and will update your Dashboard with Root Cause Reports that coincide with the triggering alert so they're immediately visible to you as you work the issue.

The two modes of operation are independent. You can configure Auto-Detect and/or Augment modes depending on your operational use-case.

Auto-Detect (recommended): Send Root Cause Detections to your New Relic Dashboards

1. Zebrium continuously monitors all application logs and uses unsupervised machine learning to find anomalous log patterns that indicate a problem. These are automatically turned into Root Cause Reports highlighting details of any problems with over 95% accuracy.
2. Root Cause Report summaries are sent to New Relic using the events API and Root Cause details are visible on your New Relic Dashboards.
3. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Dashboard.
4. Log metrics are also sent to New Relic via the metric API for visualization on your New Relic Dashboards.

CLICK HERE to send Root Cause Detections to your New Relic Dashboards

Augment (advanced users): Receive Signals from New Relic Alert Policies

1. Any New Relic Alert Policy can trigger a webhook request for Root Cause Analysis from Zebrium.
2. Zebrium finds anomalous log patterns from your application that coincide with the Incident and creates a Root Cause Report.
3. Root Cause Report summaries are sent to New Relic using the event API and Root Cause details are visible on your New Relic Dashboards.
4. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Dashboard.

CLICK HERE to receive Signals from New Relic Alert Policies

Adding Zebrium Root Cause Reports to New Relic Dashboards

Integration Overview

1. Create an API Key in New Relic.
2. Create a New Relic Integration in Zebrium using the information from STEP 1.
3. Add Zebrium Root Cause Report events and Log metrics to your New Relic Dashboard.

Integration Details

STEP 1: Create an API Key in New Relic

1. From the ***Settings*** drop-down, select *API keys*.
2. Click **Create a key**.
3. Choose the appropriate Account and make a note of the **Account ID** for use in STEP 2.
4. Select ***Ingest - License*** as the Key type.
5. Enter a ***Name*** for the API key and click **Create a key**.
6. Copy and save the **Key** for use in STEP 2.

STEP 2: Create a New Relic Outbound Integration in Zebrium

1. From the User menu area, click on the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Observability Dashboards** section and click on **New Relic**.
4. Click the **[Create a New Integration]** button.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **[Send Detections]** tab.
10. Click the **[Enabled]** button.
11. Enter the **Account ID** from STEP 1, above.
12. Enter the **API Key** from STEP 1, above.
13. Click **[Save]**.

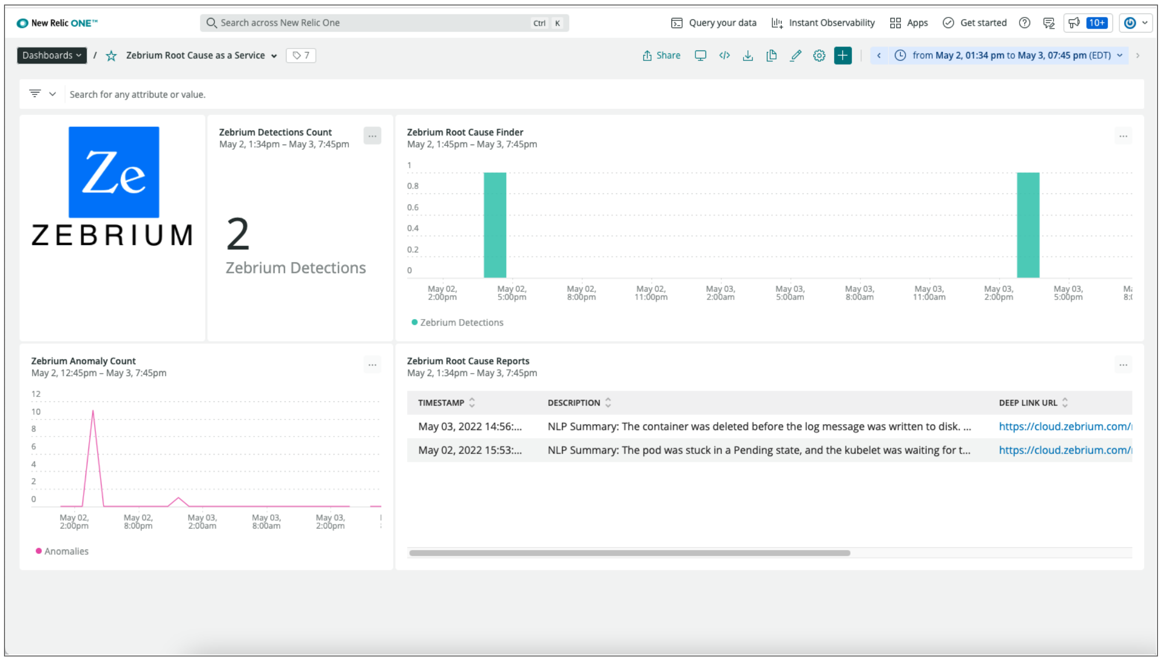
STEP 3: Add Zebrium Root Cause Report Detections and Log Metrics to your New Relic Dashboards

We recommend adding a few charts to your existing New relic dashboards so that you can see details of Zebrium root cause reports alongside your other observability data.

The image below shows a sample New Relic dashboard with the following Zebrium charts:

- A count of Zebrium Root Cause Report Detections
- A Zebrium Root Cause Finder chart with a green vertical bar wherever a detection occurs. This is useful when used on other dashboards as you can visually correlate what you are seeing in other data with what Zebrium has automatically detected.
- A chart showing log anomaly counts
- A chart containing a list of root cause report summaries

Sample New Relic Zebrium dashboard:

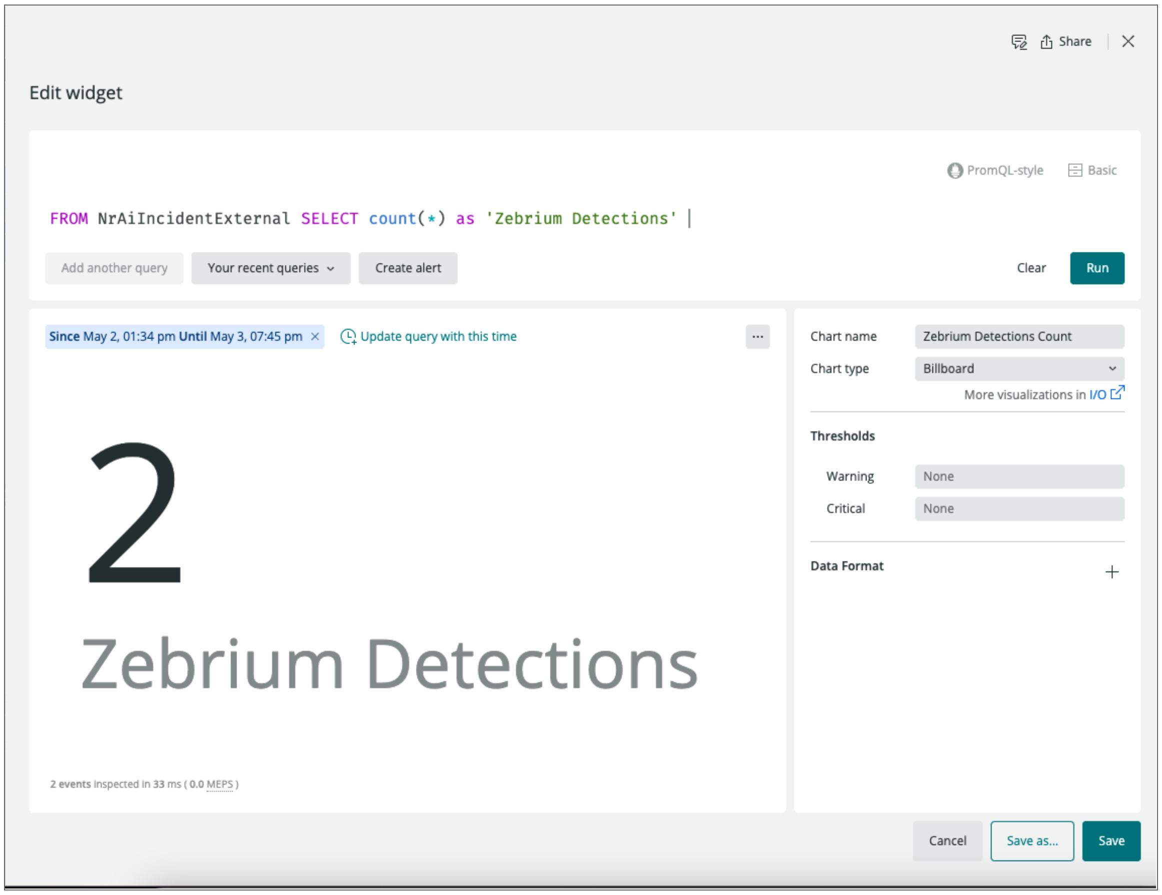

New Relic ONE
Search across New Relic One
Query your data
Instant Observability
Apps
Get started
Dashboards
Zebrium Root Cause as a Service
Share
from May 2, 01:34 pm to May 3, 07:45 pm (EDT)
Search for any attribute or value.
ZEBRIUM
Zebrium Detections Count
May 2, 1:34pm – May 3, 7:45pm
2
Zebrium Detections
Zebrium Root Cause Finder
May 2, 1:45pm – May 3, 7:45pm
Zebrium Detections
Zebrium Anomaly Count
May 2, 12:45pm – May 3, 7:45pm
Anomalies

Zebrium Root Cause Reports
May 2, 1:34pm – May 3, 7:45pm

TIMESTAMP	DESCRIPTION	DEEP LINK URL
May 03, 2022 14:56:...	NLP Summary: The container was deleted before the log message was written to disk. ...	https://cloud.zebrium.com/
May 02, 2022 15:53:...	NLP Summary: The pod was stuck in a Pending state, and the kubelet was waiting for t...	https://cloud.zebrium.com/

The following images show the definitions for each of the charts above.

Count of Zebrium Root Cause Report Detections:



```
FROM NrAiIncidentExternal SELECT count(*) as 'Zebrium Detections'
```

Since May 2, 01:34 pm Until May 3, 07:45 pm

2 events inspected in 33 ms (0.0 MEPS)

Chart name: Zebrium Detections Count

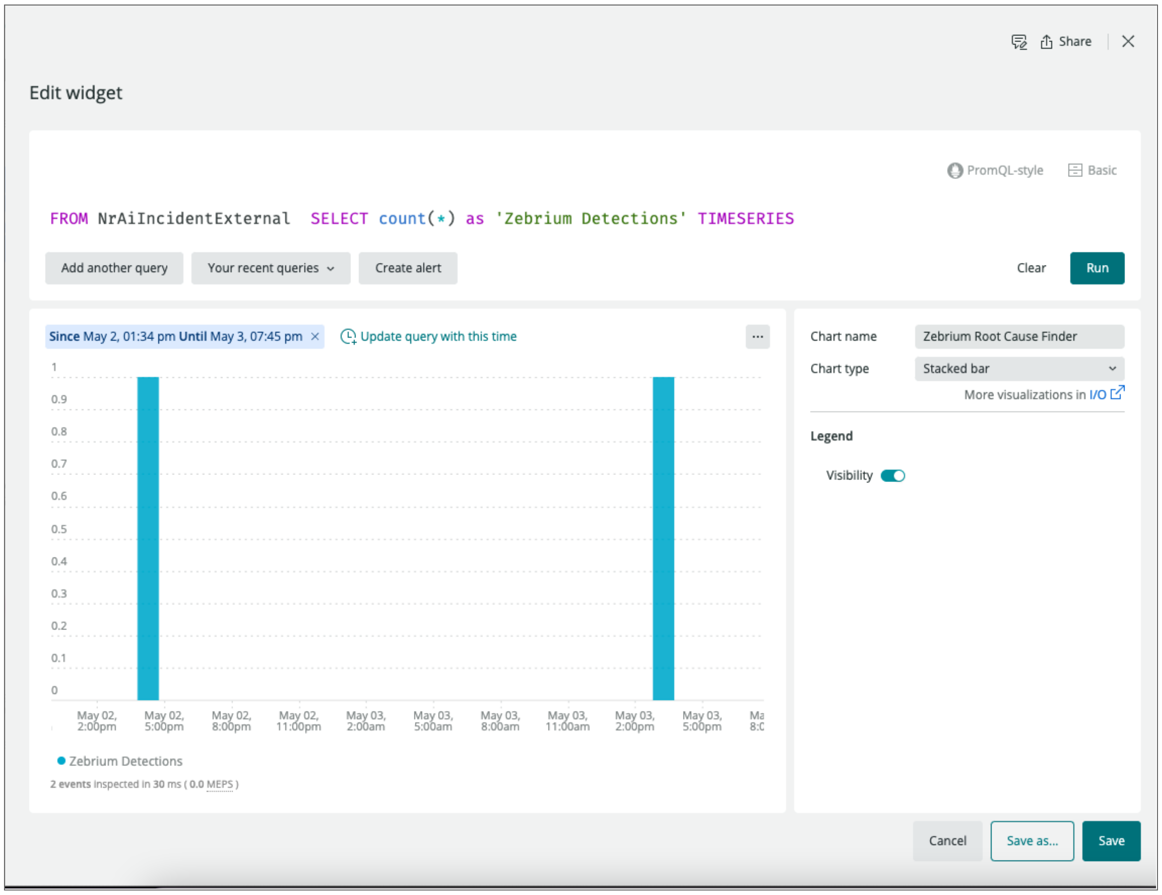
Chart type: Billboard

Thresholds

Warning: None

Critical: None

Root cause finder showing where detections occur:



```
FROM NrAiIncidentExternal  SELECT count(*) as 'Zebrium Detections' TIMESERIES
```

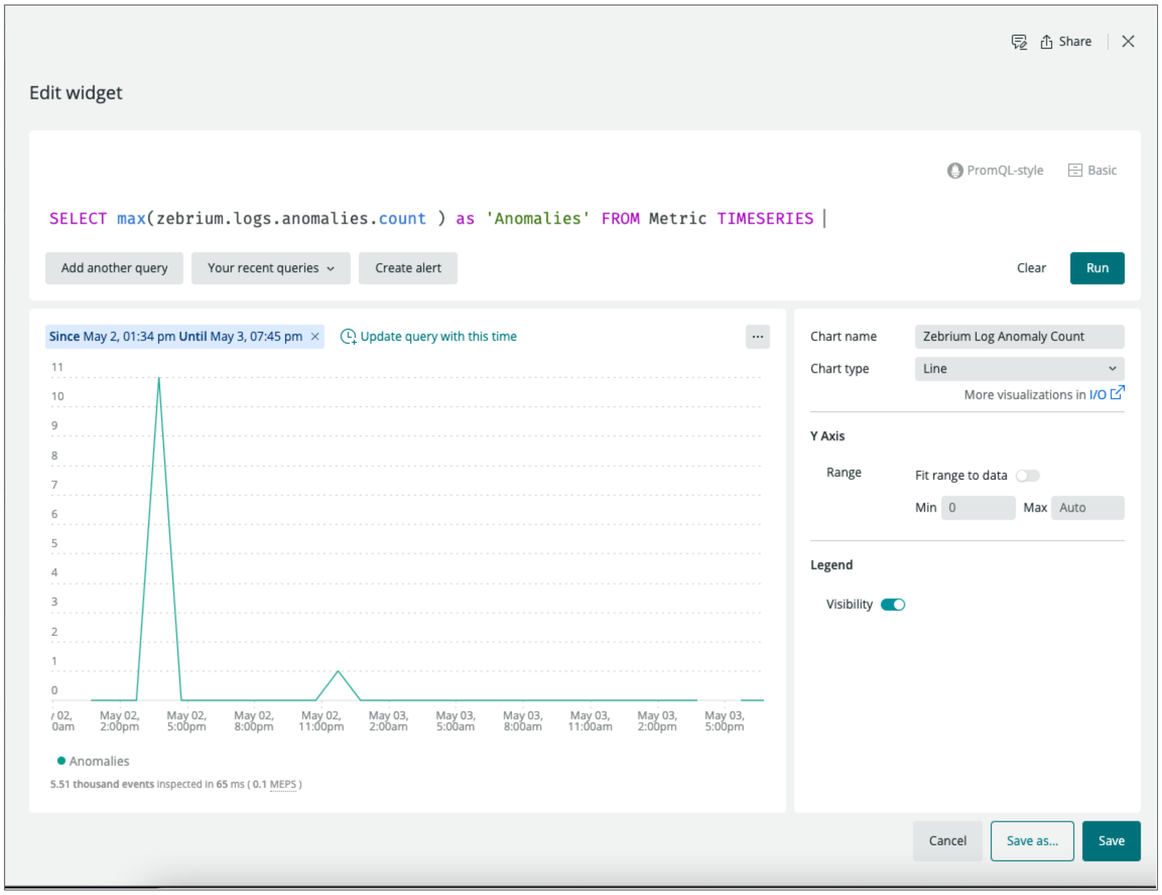
Since May 2, 01:34 pm **Until** May 3, 07:45 pm

2 events inspected in 30 ms (0.0 MEPS)

Chart name: Zebrium Root Cause Finder

Chart type: Stacked bar

Chart of log anomaly counts:



Edit widget

```
SELECT max(zebrium.logs.anomalies.count ) as 'Anomalies' FROM Metric TIMESERIES
```

Since May 2, 01:34 pm Until May 3, 07:45 pm

5.51 thousand events inspected in 65 ms (0.1 MEPS)

Chart name: Zebrium Log Anomaly Count

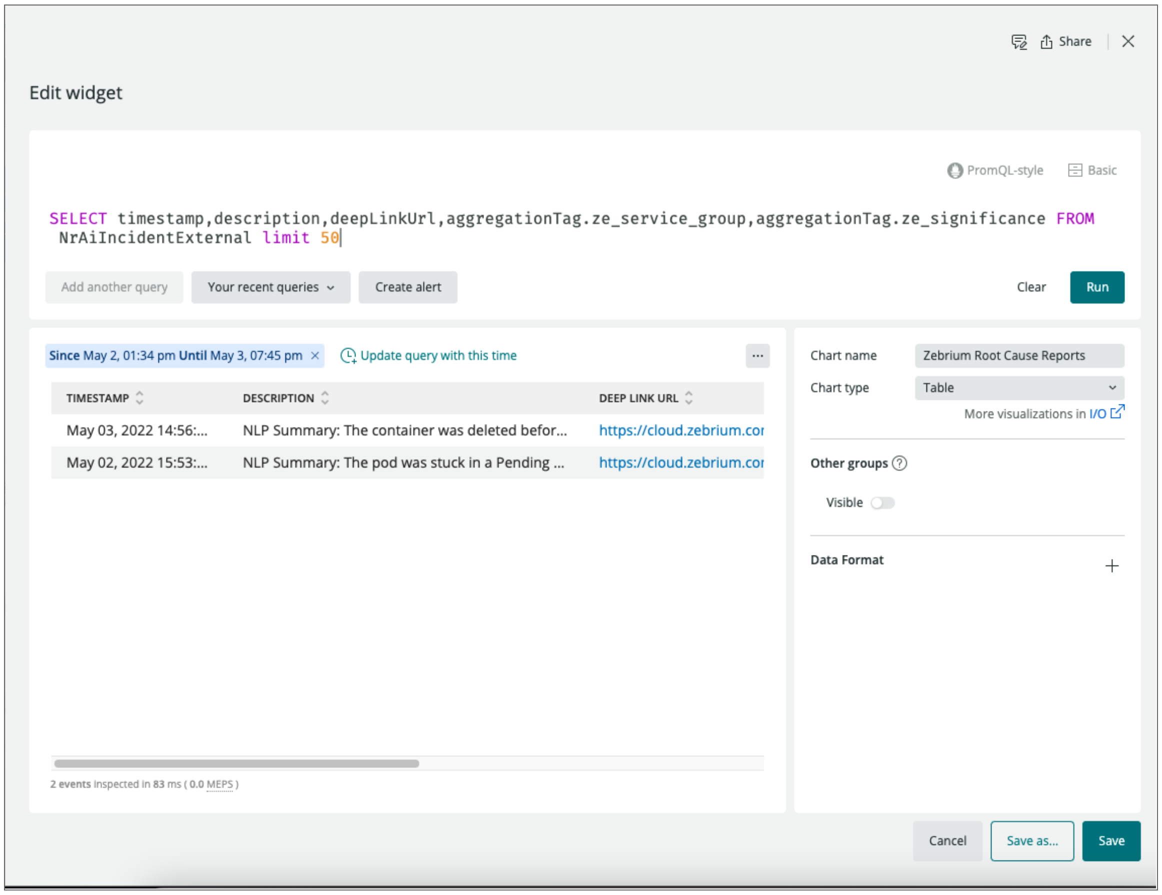
Chart type: Line

Y Axis

Range: Fit range to data; Min 0; Max Auto

Legend: Visibility on

List of root cause report summaries:



```
SELECT timestamp,description,deepLinkUrl,aggregationTag.ze_service_group,aggregationTag.ze_significance FROM
 NrAiIncidentExternal limit 50
```

Since May 2, 01:34 pm Until May 3, 07:45 pm

TIMESTAMP	DESCRIPTION	DEEP LINK URL
May 03, 2022 14:56:...	NLP Summary: The container was deleted befor...	https://cloud.zebrium.cor
May 02, 2022 15:53:...	NLP Summary: The pod was stuck in a Pending ...	https://cloud.zebrium.cor

2 events inspected in 83 ms (0.0 MEPS)

Chart name: Zebrium Root Cause Reports
Chart type: Table
Other groups

Important Metric Names

Metric Name	Description
zebrium.logs.all.count	Count of all log events received in a one-minute duration (per service_group).
zebrium.logs.anomalies.count	Count of anomaly log events received in a one-minute duration (per service_group).
zebrium.logs.errors.count	Count of error log events received in a one-minute duration (per service_group).

Zebrium Detections Event Payload

Zebrium Detections are sent to New Relic as Custom Events with **eventType** of **NrAiIncidentExternal**.

Here is a sample **NrAiIncidentExternal** payload for a Zebrium detection with descriptions of each field:

```
    {
      "aggregationTag.ze_deployment": "this is the deployment name",
      "aggregationTag.ze_first_occurrence": "set to true if this is the
```

```
first occurrence of this type of incident",
      "aggregationTag.ze_inci_id": "00062722-5f20-0000-0000-5190000353ee",
      "aggregationTag.ze_service_group": "this is the zebrium service
group name",
      "aggregationTag.ze_significance": "significance of the detection:
low, medium or high",
      "deepLinkUrl": "this is the URL in the Zebrium UI for this root
cause report",
      "description": "this is usually the NLP summary for the root cause
report",
      "entityName": "zebrium_detections",
      "source": "zebrium_detections",
      "state": "trigger",
      "timestamp": 1651647986000,
      "title": "Zebrium Detected Root Cause Report",
      "version": 1
    }
```

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

Augmenting New Relic with Root Cause Reports using Alert Policies

Integration Overview

1. Create an API Key in New Relic.
2. Create a New Relic Inbound Integration in Zebrium using the information from STEP 1.
3. Create a Webhook Notification Channel in New Relic using the information from STEP 2.
4. Add Webhook notifications to your Alert Policies in New Relic.
5. Add Zebrium Root Cause Reports to your New Relic Dashboard.

Integration Details

STEP 1: Create an API Key in New Relic

1. From the ***Settings*** drop-down, select *API keys*.
2. Click **Create a key**.
3. Choose the appropriate Account and make a note of the **Account ID** for use in STEP 2.
4. Select ***Ingest - License*** as the Key type.
5. Enter a ***Name*** for the API key and click **Create a key**.
6. Copy and save the **Key** for use in STEP 2.

STEP 2: Create a New Relic Inbound Integration in Zebrium

1. From the User menu area, click on the **Settings** menu (hamburger).
2. Select **Inbound Integrations**.
3. Click the **[+ Create Inbound Integration]** button.
4. Select **New Relic** as the Inbound Integration Type and click **Create**.
5. Enter a **Name** for this integration.
6. Select the **Deployment** for the integration.
7. Select the **Service Group(s)** for the integration.
8. Enter the **Account ID** from STEP 1, above.
9. Enter the **API Key** from STEP 1, above.
10. Click **[Create]**.
11. Copy the Webhook that was just created and save it for use in STEP 3.

STEP 3: Create a Webhook Notification Channel in New Relic

1. Click the **[Alerts & AI]** tab.
2. From the Main Navigation panel, click **Channels**.
3. Click the **[+ New notification channel]** button.
4. Select *Webhook* as the channel type.
5. Enter a **Channel name** for this Webhook.
6. Enter the **Base URL** generated by Zebrium in STEP 2.
7. Leave the ***Basic Auth***, ***Custom Headers***, and ***Use Custom Payload*** fields empty.
8. Click **Create channel**.

STEP 4: Add Webhook Notifications to your Alert Policies in New Relic

1. Click the **[Alerts & AI]** tab.
2. From the Main Navigation panel, click on **Alert conditions (Policies)**.
3. Click on any **Alert Policy** you wish to trigger a Root Cause Report request.
4. Click the **[Notification channels]** tab.
5. Click on **Add notification channels**.
6. Click on **Webhook**.
7. Select the Webhook notification channel created in STEP 3.

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

ScienceLogic

Features

- Automatically adds Root Cause Reports as Events in ScienceLogic SL1.
- Each Zebrium RCA Report includes a summary, word cloud, and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium user interface.
- Leads to faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

How it Works

Our recommended mode of operation for Observability Dashboard integrations is to use the Zebrium **Auto-Detect** mode as an accurate mechanism for explaining the reason something went wrong. In this mode, you continue to use your existing rules, alerts, and metrics as the primary source of problem detection. You can then review Zebrium RC Report findings directly on your ScienceLogic SL1 **Events** page (or **Events Console** in the classic user interface) alongside other metrics to explain the reason behind problems you were alerted on.

Augment mode is useful if you use run book automation to create a ticket based on an event from your alerts. In this mode, Zebrium will update the ticket directly with any Root Cause Reports around the time of the event so they're immediately visible to you as you work the case.

The two modes of operation are independent. You can configure Auto-Detect and/or Augment modes depending on your operational use case.

Auto-Detect (recommended): Send Root Cause Detections to your SL1 Events Page

1. Zebrium continuously monitors all application logs and uses unsupervised machine learning to find anomalous log patterns that indicate a problem. These are automatically turned into Root Cause Reports highlighting details of any problems with over 95% accuracy.
2. Root Cause Report Summaries are sent to ScienceLogic as Events, and Root Cause details are visible on your SL1 **Events** page.
3. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Event Console.

***CLICK HERE** to Send Root Cause Detections to your ScienceLogic Event Console*

Augment (advanced users): SL1 Tickets with Root Cause Reports

1. Any SL1 action policy can trigger a webhook request for Root Cause Analysis from Zebrium.
2. Zebrium finds anomalous log patterns from your application that coincide with the event and creates a Root Cause Report.
3. The report is sent to SL1 and updates the ticket associated with the event with Root Cause details.

4. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your **Events** page or ticket.

CLICK HERE ***to receive Signals from ScienceLogic Run Book Automation***

Sending Root Cause Detections to the SL1 Events Page

Integration Overview

1. In ScienceLogic SL1, choose an existing Device ID (DID) or create a new virtual device used to associate Root Cause reports from Zebrium.
2. Set up a user with restricted access to minimally required API access hooks.
3. Setup an event policy for the "Auto-Detected Root Cause Report" alert sent by Zebrium.
4. Create a ScienceLogic Integration in Zebrium using the information from STEPS 1 and 2.

Integration Details

STEP 1: Choose an Existing or Create a New Device

Because Zebrium is using logs from an application that may be spread across many hosts, containers, network devices, and more, there is no direct association of Root Cause Reports to a single hardware device. Instead, Zebrium associates Root Cause Reports to a "device" that represents the set of services that make up the application.

If you already have such a "device", like a Cloud Application, then Zebrium needs its Device ID (DID).

If you do not have an existing device that is appropriate to use, you can create a virtual device for this purpose.

Use an Existing Device

1. In SL1, go to the **Devices** page ( ). If you are using the classic user interface, go to Registry > Devices > Device Manager.
2. Locate the desired device from the list and make a note of the numeric Device ID (DID) in the ***ID*** column (or the ***DID*** column in the classic user interface). You will use the DID when configuring the Zebrium Integration.

Create a New Virtual Device

1. In SL1, go to the **Device Manager** page (Devices > Device Manager). If you are using the classic user interface, go to Registry > Devices > Device Manager.
2. Click **[Actions]** and select *Create Virtual Device*. The Create Virtual Device modal appears.

3. Complete the following fields:
    - ***Device Name***. Name of the virtual device. Can be any combination of alphanumeric characters, up to 32 characters in length.
    - ***Organization***. Organization to associate with the virtual device. Select from the drop-down list of all organizations in SL1.
    - ***Device Class***. Select ScienceLogic | Integration Service as the device class to associate with the virtual device.
    - ***Collector***. Specifies which instance of SL1 will perform auto-discovery and gather data from the device. Select the collector from the drop-down list of all collectors in SL1.
4. Click **[Add]** and close the modal.
5. Go to **Devices** page or the **Device Manager** page (Devices > Device Manager) and locate the newly created virtual device from the list.
6. Make a note of the numeric Device ID (DID) in the ***ID*** column (or the ***DID*** column in the classic user interface). You will use the DID when configuring the Zebrium Integration.

STEP 2: Create a User with Restricted API Access

Define a New Access Key for API Access

1. In SL1, go to the **Access Keys** page (System > Manage > Access Keys).
2. Click **[Key Manager]**. The Key/Hook Alignment Editor dialog appears.
3. Complete the following fields:
    - ***Name***. Name of the key, such as *API Access for Zebrium*.
    - ***Key Category***. Select *API Access*.
    - ***Key Description***. Enter an appropriate description for the key.
4. In the **Hook Alignment** section, select each of the following unaligned access hooks on the left-hand side and click » to move the selected hook to the **Aligned Access Hooks** on the right:

    ```
    Event Note:Add/Rem
    ```

    ```
    Events/Event:View
    ```

    ```
    Ticket:Notes:Add
    ```

    ```
    Ticket:View
    ```

5. Click **[Save]**.

Define a New User Policy using the New Access Key

1. In SL1, go to the **User Policies** page (Registry > Accounts > User Policies).
2. Click **[Create]**. A **Create New User Policy** dialog appears.

3. In the **Privilege Keys** section, select the access key that you created in the previous procedure. You might need to scroll down to the **API Access** section.
4. Complete the remaining fields according to your accepted policies.
5. Click **[Save]**.

Define a New User using the New User Policy

1. In SL1, go to the **User Accounts** page (Registry > Accounts > User Accounts).
2. Click **[Create]**. A **Create New Account** dialog appears.
3. Complete the following fields:
   - ***Require Password Reset***. Make sure *Next Login* is unchecked.
   - ***Account Type***. Select *Policy Membership*.
   - ***Policy Membership***. Select the new user policy created in the previous procedure.
4. Complete the remaining fields according to your accepted policies.
5. Make a note of the ***Username*** and ***Password*** for use in the next STEP.
6. Click **[Save]**.

STEP 3: Create an Event Policy for the Zebrium Alert

1. Go to the **Event Policies** page (Events > Event Policies). If you are using the classic user interface, go to Registry > Events > Event Manager.
2. Click **[Create Event Policy]**. If you are using the classic user interface, click **[Create]**.
3. In the ***Policy Name*** field at top left, type a name for the policy.
4. On the **[Policy Description]** tab, type a description of the policy, such as "Zebrium alert".
5. On the **[Match Logic]** tab (or the **[Policy]** tab in the classic user interface), select *API* fo the ***Event Source***.
6. In the ***Match Logic*** drop-down, select *Regular Expression* (or [Regex Match] in the classic user interface).
7. Do not select ***Multi Match***.
8. Select ***Message Match***.
9. In the first ***Match String*** field, type the following:
   ```
   ^Zebrium\s+(Detected|created).*
   ```
10. On the **[Event Message]** tab (or the **[Policy]** tab in the classic user interface), enter **%M** in the ***Event Message*** field.
11. Click **[Save]**.

STEP 4: Create a ScienceLogic Integration in Zebrium

1. From the User menu area in Zebrium, click on the ***Settings*** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Observability Dashboards** section and select **ScienceLogic**.

4. Click the **[Create a New Integration]** button.
5. Click the **[General]** tab.
6. Enter an ***Integration Name*** for this integration.
7. Select the ***Deployment*** for the integration.
8. Select the ***Service Group(s)*** for the integration.
9. Click the **[Send Detections]** tab.
10. Click the **[Enabled]** button.
11. Enter the ***Username*** and ***Password*** from STEP 2, above.
12. Enter the ***Device ID*** from STEP 1, above.
13. Enter the fully qualified ***Appliance URL*** to your instance of ScienceLogic (**/api/<*api_endpoint*>** will be added automatically by the integration).
14. Click **[Save]**.

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

Receiving Signals from a ScienceLogic Run Book Automation

Integration Overview

1. In ScienceLogic SL1, create a user with restricted access to minimally required API access hooks.
2. Set up webhook credentials and a HTTP action policy for sending a webhook to Zebrium.
3. Create a ScienceLogic Integration in Zebrium using the information from STEPS 1 and 2.
4. Set up a run book automation to augment tickets with Root Cause Reports.

Integration Details

STEP 1: Create a User with Restricted API Access

Define a new Access Key for API Access

1. In SL1, go to the **Access Keys** page (System > Manage > Access Keys).
2. Click **[Key Manager]**. The Key/Hook Alignment Editor dialog appears.
3. Complete the following fields:
    - ***Name***. Name of the key, such as *API Access for Zebrium*.
    - ***Key Category***. Select *API Access*.
    - ***Key Description***. Enter an appropriate description for the key.
4. In the **Hook Alignment** section, select each of the following unaligned access hooks on the left-hand side and click » to move the selected hook to the **Aligned Access Hooks** on the right:
    ```
    Event Note:Add/Rem
    ```
    ```
    Events/Event:View
    ```
    ```
    Ticket:Notes:Add
    ```
    ```
    Ticket:View
    ```
5. Click **[Save]**.

Define a New User Policy using the New Access Key

1. In SL1, go to the **User Policies** page (Registry > Accounts > User Policies).
2. Click **[Create]**. A **Create New User Policy** dialog appears.
3. In the **Privilege Keys** section, select the access key that you created in the previous procedure. You might need to scroll down to the **API Access** section.
4. Complete the remaining fields according to your accepted policies.
5. Click **[Save]**.

Define a New User using the New User Policy

1. In SL1, go to the **User Accounts** page (Registry > Accounts > User Accounts).
2. Click **[Create]**. A **Create New Account** dialog appears.
3. Complete the following fields:
    - ***Require Password Reset***. Make sure *Next Login* is unchecked.
    - ***Account Type***. Select *Policy Membership*.
    - ***Policy Membership***. Select the new user policy created in the previous procedure.
4. Complete the remaining fields according to your accepted policies.
5. Make a note of the ***Username*** and ***Password*** for use in the next STEP.
6. Click **[Save]**.

STEP 2: Set Up Webhook Credentials and HTTP Action Policy

Create Credentials

1. Go to the **Credentials** page (System > Manage > Credentials).
2. Click the **[Actions]** button and select *Create SOAP/XML Host Credential*.
3. Complete the following fields:
    - ***Profile Name***. Enter an appropriate name.
    - ***Context Encoding***. Select *text/xml*.
    - ***Method***. Select *[POST]*.
    - ***HTTP Version***. Select *[HTTP/1.1]*.
    - ***URL***. Because this is a required field, you will need to type a placeholder URL in this field, such as *https://text.com*. You will update the ***URL*** field with the actual URL after you create a ScienceLogic Integration in STEP 3, below.
    - ***HTTP Auth User*** and ***HTTP Auth Password***. Create a user name and password and save for use in STEP 3.
4. Click **[Save]**
5. On the **Credentials** page, locate the newly created credential from the list and note the numeric Credential ID from the ***ID*** column. You will use this number in the next step when you create a run nook HTTP action.

Create a HTTP Request Action

1. Go to the **Actions** page (Registry > Run Book > Actions).
2. Click **[Create]**. The **Action Editor** page appears.

3. Complete the following fields:
    - ***Action Name***. Enter an appropriate name.
    - ***Action Type***. Select *Make an HTTP Request (2.0)*.
4. In the ***Input Parameters***, enter:

```
{
 "credential_id" : REPLACE_WITH_CREDENTIAL_ID_FROM_STEP_2_ABOVE,
 "dynapp_guid": "",
 "url_override" : "",
 "relative_url": "",
 "payload": "{\"event_timestamp_first\": \"%d\", \"event_
id\":\"%e\", \"event_message\" : \"%M\", \"event_counter\" : \"%c\"
}",
 "command_label": "zebrium_rca"
}
```

5. Click **[Save]**.

STEP 3: Create a ScienceLogic Integration in Zebrium to Receive Signals from ScienceLogic

1. From the User menu area in Zebrium, click the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Observability Dashboards** section and click on **ScienceLogic**.
4. Click the **[Create a New Integration]** button.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **[Receive Signals]** tab.
10. Click the **[Enabled]** button.
11. Enter the **Username** and **Passwor**d created in STEP 1 above.
12. Enter the fully qualified Appliance URL to your instance of ScienceLogic (**/api/<*api_endpoint*>** will be added automatically by the integration).
13. Enter the **Username** and **Password** for Webhook Basic Authentication from STEP 2 above.
14. Click **[Save]**.
15. Copy the webhook URL and save for use in STEP 4 and click **[OK]**.
16. Return to the webhook credential you created in STEP 2 and add the Webhook to the ***URL*** field.

STEP 4: Set Up Run Book Automation to Augment Tickets with Root Cause Reports

You will now be able to use the Zebrium Signal Webhook HTTP Request Action, created in STEP 2, in your run book automation to instruct Zebrium to Augment an SL1 Ticket with a Root Cause Report.

The following is an example snippet from an Automation Policy Aligned Actions section:

```
1. Create Ticket [21]: My Ticket Template
```

```
2. Make an HTTP Request [102]: Zebrium Signal Webhook
```

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

Chapter 6

Configuring Incident Management Integrations

Overview

You can configure an integration between Zebrium and your third-party Incident Management application to automatically add Root Cause (RCA) reports to your incidents in the third-party application. Each Zebrium RCA report includes a summary, word cloud, and a set of log events display symptoms and root cause, along with a link to the full report in the Zebrium user interface.

After you complete the configuration, you can can view details of root cause and direct the incident to the appropriate team. All of these features lead to faster Mean Time to Repair (MTTR) and less time manually hunting for root cause.

Zebrium supports Incident Management integrations with the following third-party applications:

- *Opsgenie*
- *OpsRamp*
- *PagerDuty*
- *VictorOps*

Opsgenie

Features

- Automatically adds Root Cause (RCA) Reports to Incidents in Opsgenie. This allows you to see details of root cause and direct the incident to the appropriate team.
- Each Zebrium RCA report includes a summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium UI.
- Leads to faster Mean Time to Repair (MTTR) and less time manually hunting for root cause.

How it Works

Our recommended mode of operation for Incident Management integrations is to use the Zebrium **Augment** mode as an accurate mechanism for explaining the reason something went wrong. In this mode, you continue to use your existing rules as the primary source of problem detection and Incident creation. You can then review Zebrium RC Report findings directly in your Incident that was created by Opsgenie to explain the reason behind Incident.

Auto-Detect mode is useful when you want to direct all Root Cause Reports to Opsgenie for routing and dispositioning. Or, when you want to send only specific Root Cause Reports to Opsgenie after first reviewing in the Zebrium UI.

The two modes of operation are independent. You can configure Augment and/or Auto-Detect modes depending on your operational use-case.

Augment: Receive Signals from Opsgenie Incidents

1. Any Opsgenie Incident can trigger a webhook request for Root Cause Analysis from Zebrium.
2. Zebrium finds anomalous log patterns from your application that coincide with the Incident and creates a Root Cause Report.
3. Root Cause Report summaries are sent to Opsgenie using the notes API and Root Cause details are visible in your Opsgenie Incident.
4. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Incident.

CLICK HERE to receive Signals from Opsgenie Incidents

Auto-Detect: Send Root Cause Detections to Opsgenie as Incidents

1. Zebrium continuously monitors all application logs and uses unsupervised machine learning to find anomalous log patterns that indicate a problem. These are automatically turned into Root Cause Reports highlighting details of any problems with over 95% accuracy.

2. Root Cause Report summaries are sent to Opsgenie using the webhook interface so Root Cause details are visible as Incidents in Opsgenie.
3. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Incident.

CLICK HERE ***to send Root Cause Detections to Opsgenie as Incidents***

Receiving Signals from Opsgenie

Integration Overview

1. Configure API Access for Zebrium in Opsgenie
2. Create an Opsgenie Integration in Zebrium to Receive Signals from Opsgenie
3. Add the Zebrium Webhook to Opsgenie

Integration Details

STEP 1: Configure API Access for Zebrium in Opsgenie

1. From the **[Settings]** tab, click **API key management** in the left-hand navigation panel.
2. Click the **[Add new API key]** button.
3. Enter a name for the API, such as "Zebrium Incident Detection".
4. Enable **Read Access** and **Create and Update Access**. Delete Access can be disabled.
5. Make sure that **Enabled** is checked.
6. Copy the **API Key** and save it for later use when configuring Zebrium.
7. To view or manage your configured API Keys, go to the **[Settings]** tab and click **API key management** in the left-hand navigation panel.

STEP 2: Create an Opsgenie Integration in Zebrium to Receive Signals from Opsgenie

1. From the User menu area in Zebrium, click the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Incident Management** section and click **Opsgenie**.
4. Click the **[Create a New Integration]** button.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **Receive Signals** tab.
10. Click the **[Enabled]** button.
11. Select the **Region** of your Opsgenie portal.
12. Enter the **API Key** that you created in STEP 1, above.
13. Click **[Save]**.

14. Copy the **Webhook URL** and save it for use in STEP 3.
15. Click **[OK]**.

STEP 3: Add the Zebrium Webhook to Opsgenie

1. Go to the **Opsgenie Integration** page and click the **Zebrium integration** to modify or add a new Zebrium Integration.
2. Make sure that the **Send alert details to Zebrium for Opsgenie Alerts** option is enabled.
3. In the **Zebrium URL** text box, paste the **Zebrium Webhook URL** from STEP 2, above.
4. Click **[Save Integration]**.

How to Uninstall

Disable API Access

1. From the **[Settings]** tab in Opsgenie, click **API key management** in the left-hand navigation panel.
2. Click the **trash can** on the desired API Key.
3. Click **[OK]** after confirming you wish to proceed.

Delete the Zebrium Integration

1. From the User menu area in Zebrium, click the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Incident Management** section and click on **Opsgenie**.
4. Click the **red X** on the desired Zebrium integration.
5. Click **[OK]** after confirming you wish to proceed.

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

Sending Root Cause Detections to Opsgenie as Incidents

This integration automatically sends a Root Cause (RCA) Report to Opsgenie so that the appropriate team is notified when a Zebrium Incident is auto-detected.

Integration Overview

1. Add the Zebrium Integration to your Opsgenie Team
2. Create an Opsgenie Integration in Zebrium to Send Root Cause Detections to Opsgenie as Incidents

Integration Details

STEP 1: Add the Zebrium Integration to your Opsgenie Team

1. In Opsgenie, click on the **[Teams]** tab to access your Team dashboard.
2. Click the desired **Team** for the integration.
3. Click the **Integrations** section from the left-hand navigation pane.
4. Click the **[Add integration]** button.
5. Click the **[Add]** button under the Zebrium integration icon.
6. Make a note of the **Webhook URL** in the Zebrium section of the **Integration Setup** page.
7. In the **Settings** section, update the **Name** as desired.
8. Make sure that the **Enabled** checkbox is selected.
9. Click **Save Integration**.

STEP 2: Create an Opsgenie Integration in Zebrium to Send Root Cause Detections to Opsgenie as Incidents

1. From the User menu area in Zebrium, click the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Incident Management** section and click **Opsgenie**.
4. Click the **[Create a New Integration]** button.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **[Send Detections]** tab.
10. Click the **[Enabled]** button.
11. Select the **Region** of your Opsgenie portal.

12. Enter the **Opsgenie Webhook URL** that you created in STEP 1, above.
13. Click **[Save]**.

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

OpsRamp

Features

- Automatically adds Root Cause Reports to Incidents in OpsRamp. This allows you to see details of root cause and direct the incident to the appropriate team.
- Each Zebrium RCA Report includes a summary, word cloud, and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium user interface.
- Leads to faster Mean Time to Repair (MTTR) and less time manually hunting for root cause.

How it Works

Our recommended mode of operation for Incident Management integrations is to use the Zebrium **Augment** mode as an accurate mechanism for explaining the reason something went wrong. In this mode, you continue to use your existing rules as the primary source of problem detection and Incident creation. You can then review Zebrium RC Report findings directly in your Incident that was created by OpsRamp to explain the reason behind Incident.

Auto-Detect mode is useful when you want to direct all Root Cause Reports to OpsRamp for routing and dispositioning. Or, when you want to send only specific Root Cause Reports to OpsRamp after first reviewing in the Zebrium user interface.

The two modes of operation are independent. You can configure Augment and/or Auto-Detect modes depending on your operational use-case.

Augment: Receive Signals from OpsRamp Incidents

1. Any OpsRamp Incident can trigger a webhook request for Root Cause Analysis from Zebrium.
2. Zebrium finds anomalous log patterns from your application that coincide with the Incident and creates a Root Cause Report.
3. Root Cause Report summaries are sent to OpsRamp using the **notes** API and Root Cause details are visible in your OpsRamp Incident.
4. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Incident.

***CLICK HERE* to receive Signals from OpsRamp Incidents**

Auto-Detect: Send Root Cause Detections to OpsRamp as Incidents

1. Zebrium continuously monitors all application logs and uses unsupervised machine learning to find anomalous log patterns that indicate a problem. These are automatically turned into Root Cause Reports highlighting details of any problems with over 95% accuracy.

2. Root Cause Report summaries are sent to OpsRamp using the **webhook** interface so Root Cause details are visible as Incidents in OpsRamp.
3. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Incident.

CLICK HERE to send Root Cause Detections to OpsRamp as Incidents

Receiving Signals from OpsRamp

Please contact Zebrium for Early Access to this feature.

Sending Root Cause Detections to OpsRamp as Incidents

This integration automatically sends an Root Cause Reports to OpsRamp when a Zebrium Incident is auto-detected so that the appropriate team is notified.

Integration Overview

1. Add the Zebrium Integration to OpsRamp
2. Create an OpsRamp Integration in Zebrium to Send Root Cause Detections to OpsRamp as Incidents

Integration Details

STEP 1: Add the Zebrium Integration to OpsRamp

1. Click the **[Setup]** tab and select **Integrations/Integrations** from the main navigation panel.
2. Click **Other** under the **Available Integrations** section.
3. Click **Custom Integration**.
4. Enter a **Name** and **Description** for this integration.
5. Set **Category** to **Custom**.
6. Enter a **Logo** if desired.
7. Click **[Install]**.
8. Set **Authentication Type** to **OAUTH2**.
9. Set **Role** to **Client Administrator**.
10. Click **[Save]**.
11. Copy the **Tenant ID** and save it for later use when configuring the Zebrium Outbound Integration.
12. Copy the **Key** and save it for later use when configuring the Zebrium Outbound Integration.
13. Copy the **One Time Secret** and save it for later use when configuring the Zebrium Outbound Integration.
14. Copy the **Access Token** URLportion (from *https* through token) and save it for later use when configuring the Zebrium Outbound Integration.
15. Copy the **Incident Create** URL portion (from *https* through incidents) and save it for later use when configuring the Zebrium Outbound Integration.

STEP 2: Create an OpsRamp Integration in Zebrium to Send Root Cause Detections to OpsRamp as Incidents

1. From the User menu area in Zebrium, click the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Incident Management** section and click **OpsRamp**.

4. Click the **[Create a New Integration]** button.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **[Send Detections]** tab.
10. Click the **[Enabled]** button.
11. Enter the **Access Token URL** that you saved from STEP 1, above.
12. Enter the **Incident Create URL** that you saved from STEP 1, above.
13. Enter the **Tenant ID** that you saved from STEP 1, above.
14. Enter the **API Key** that you saved from STEP 1, above.
15. Enter the **API Secret** that you saved from STEP 1, above.
16. Click **[Save]**.

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

PagerDuty

Features

- Automatically adds Root Cause Reports to Incidents in PagerDuty. This allows you to see details of root cause and direct the incident to the appropriate team.
- Each Zebrium RCA Report includes a summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium UI.
- Leads to faster Mean Time to Repair (MTTR) and less time manually hunting for root cause.

How it Works

Our recommended mode of operation for Incident Management integrations is to use the Zebrium **Augment** mode as an accurate mechanism for explaining the reason something went wrong. In this mode, you continue to use your existing rules as the primary source of problem detection and Incident creation. You can then review Zebrium RC Report findings directly in your Incident that was created by PagerDuty to explain the reason behind Incident.

Auto-Detect mode is useful when you want to direct all Root Cause Reports to PagerDuty for routing and dispositioning. Or, when you want to send only specific Root Cause Reports to PagerDuty after first reviewing in the Zebrium user interface.

The two modes of operation are independent. You can configure Augment and/or Auto-Detect modes depending on your operational use-case.

Augment: Receive Signals from PagerDuty Incidents

1. Any PagerDuty Incident can trigger a webhook request for Root Cause Analysis from Zebrium.
2. Zebrium finds anomalous log patterns from your application that coincide with the Incident and creates a Root Cause Report.
3. Root Cause Report summaries are sent to PagerDuty using the **notes** API and Root Cause details are visible in your PagerDuty Incident.
4. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Incident.

CLICK HERE to receive Signals from PagerDuty Incidents

Auto-Detect: Send Root Cause Detections to PagerDuty as Incidents

1. Zebrium continuously monitors all application logs and uses unsupervised machine learning to find anomalous log patterns that indicate a problem. These are automatically turned into Root Cause Reports highlighting details of any problems with over 95% accuracy.

2. Root Cause Report summaries are sent to PagerDuty using the webhook interface so Root Cause details are visible as Incidents in PagerDuty.
3. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Incident.

***CLICK HERE** to send Root Cause Detections to PagerDuty as Incidents*

Receiving Signals from PagerDuty

Integration Overview

1. Configure API Access for Zebrium in PagerDuty
2. Create an PagerDuty Integration in Zebrium to Receive Signals from PagerDuty
3. Add the Zebrium Webhook to PagerDuty

Integration Details

STEP 1: Configure API Access for Zebrium in PagerDuty

1. From the **Integrations** menu, select **API Access**.
2. Click the **[Create New API Key]** button.
3. Enter a description, such as "Zebrium Incident Detection".
4. Make sure that the **Read-only API Key** option is not selected.
5. Click **[Create Key]**.
6. Copy the **API Key** and save it for later use when configuring Zebrium. The key will not be visible in PagerDuty again.

STEP 2: Create a PagerDuty Integration in Zebrium to Receive Signals from PagerDuty

1. From the User menu area in Zebrium, click the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Incident Management** section and click **PagerDuty**.
4. Click the **[Create a New Integration]** button.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **Receive Signals** tab.
10. Click the **[Enabled]** button.
11. Select the **Region** of your Opsgenie portal.
12. Enter the **API Key** that you created in STEP 1, above.
13. Click **[Save]**.

14. Copy the **Webhook URL** and save it for use in STEP 3.
15. Click **[OK]**.

STEP 3: Add the Zebrium Webhook to PagerDuty

1. From the **Integrations** menu, select **Generic Webhooks (v3)**.
2. Click the **[+ Add New Webhook]** button.
3. In the **WEBHOOK URL** area, paste the **Zebrium Webhook URL** that was copied in STEP 2 when configuring access for PagerDuty in Zebrium.
4. In the **SCOPE TYPE** drop-down, select *Service*.
5. In the **SCOPE** drop-down, select the desired service to which you want to add the Zebrium webhook.
6. Enter a **DESCRIPTION**, such as "Zebrium Signal".
7. In the **EVENT SUBSCRIPTION** field, select *incident.triggered*. Clear all other checkboxes.
8. Click the **[Add Webhook]** button.

How to Uninstall

Disable API Access in PagerDuty

1. From the **Integrations** menu, select **API Access**.
2. Click **Disable** or **Remove** on the API Access Key you want to delete.
3. Click the **[Save]** button after confirming you wish to proceed.

Delete the Zebrium Integration

1. From the User menu area in Zebrium, click the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Incident Management** section and click **PagerDuty**.
4. Click the **red X** on the Zebrium integration you want to delete.
5. Click **[OK]** after confirming you wish to proceed.

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

Sending Root Cause Detections to PagerDuty as Incidents

Please contact Zebrium for Early Access to this feature.

VictorOps

Features

- Automatically adds Root Cause Reports to Incidents in VictorOps. This allows you to see details of root cause and direct the incident to the appropriate team.
- Each Zebrium RCA Report includes a summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium user interface.
- Leads to faster Mean Time to Repair (MTTR) and less time manually hunting for root cause.

How it Works

Our recommended mode of operation for Incident Management integrations is to use the Zebrium **Augment** mode as an accurate mechanism for explaining the reason something went wrong. In this mode, you continue to use your existing rules as the primary source of problem detection and Incident creation. You can then review Zebrium RC Report findings directly in your Incident that was created by VictorOps to explain the reason behind Incident.

Auto-Detect mode is useful when you want to direct all Root Cause Reports to VictorOps for routing and dispositioning. Or, when you want to send only specific Root Cause Reports to VictorOps after first reviewing in the Zebrium user interface.

The two modes of operation are independent. You can configure Augment and/or Auto-Detect modes depending on your operational use-case.

Augment: Receive Signals from VictorOps Incidents

1. Any VictorOps Incident can trigger a webhook request for Root Cause Analysis from Zebrium.
2. Zebrium finds anomalous log patterns from your application that coincide with the Incident and creates a Root Cause Report.
3. Root Cause Report summaries are sent to VictorOps using the **notes** API and Root Cause details are visible in your VictorOps Incident.
4. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Incident.

***CLICK HERE** to receive Signals from VictorOps Incidents*

Auto-Detect: Send Root Cause Detections to VictorOps as Incidents

1. Zebrium continuously monitors all application logs and uses unsupervised machine learning to find anomalous log patterns that indicate a problem. These are automatically turned into Root Cause Reports highlighting details of any problems with over 95% accuracy.

2. Root Cause Report summaries are sent to VictorOps using the **webhook** interface so Root Cause details are visible as Incidents in VictorOps.
3. If you need to drill down further to look at correlated logs across your entire app, it's just one click from your Incident.

CLICK HERE ***to send Root Cause Detections to VictorOps as Incidents***

Receiving Signals from VictorOps

Please contact Zebrium for Early Access to this feature.

Sending Root Cause Detections to VictorOps as Incidents

This integration automatically sends a Root Cause Reports to VictorOps when a Zebrium Incident is auto-detected so that the appropriate team is notified.

Integration Overview

1. Create an Incoming Webhook in VictorOps.
2. Create a VictorOps Integration in Zebrium to Send Root Cause Detections to VictorOps as Incidents.

Integration Details

STEP 1: Create an Incoming Webhook in VictorOps

1. Follow the VictorOps (Splunk On-call) guides to create an incoming Webhook.
2. Save the webhook integration and note the **Webhook URL** for use in STEP 2, below.

STEP 2: Create a VictorOps Integration in Zebrium to Send Root Cause Detections to VictorOps as Incidents

1. From the User menu area in Zebrium, click the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Incident Management** section and click **VictorOps**.
4. Click the **[Create a New Integration]** button.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **[Send Detections]** tab.
10. Click the **[Enabled]** button.
11. Enter the **Webhook URL** that you saved in STEP 1, above.
12. Click **[Save]**.

Support

If you need help with this integration, please contact Zebrium at support@zebrium.com.

Chapter 7

Enabling Notification Channels

Overview

Notification Channels provide a mechanism to define the methods that Zebrium will use to send notifications from RCA reports. The supported types of notification channels include email, as well as Mattermost, Slack, Microsoft Teams, and Webex Teams notifications.

After you have created one or more notification channels, you can link any number of these to any RCA report created by the AI/ML engine. Linking a set of notification channels to a RCA report will send notifications of future RCA reports of the same type to those channels.

Supported notification channels include:

- *Email*
- *Mattermost*
- *Slack*
- *Microsoft Teams*
- *Webex Teams*

Email Notifications

Features

- You can configure Zebrium to automatically send Root Cause Reports to email recipients. This allows you to see details of root cause in your email client.
- Each Zebrium RC Report includes a summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium UI.
- This means faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

Integration Details

Create an Email Integration in Zebrium to Send Detections to Email Recipients

1. From the User menu area in Zebrium, click on the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Notifications** section and click on **Email**.
4. Click **[Create a New Integration]**.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **[Send Detections]** tab.
10. Click **[Enabled]**.
11. Enter the **Email Address List**. Add one email recipient per line.
12. Click **[Save]**.

Mattermost Notifications

Features

- Automatically send Root Cause Reports to Mattermost channels. This allows you to see details of root cause in your Mattermost client.
- Each Zebrium RC Report includes a summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium UI.
- This means faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

Integration Overview

1. Create an Incoming Webhook in Mattermost
2. Create a Mattermost Integration in Zebrium using the information from STEP 1.

Integration Details

STEP 1: Create an Incoming Webhook in Mattermost

1. In Mattermost, go to **Main Menu** > **Integrations** > **Incoming Webhook**.
2. Click **Add Incoming Webhook** and add name and description for the webhook.
3. Select the channel to receive webhook payloads, then click **Add** to create the webhook.
4. Copy and save the **Webhook URL** for use in STEP 2.

STEP 2: Create a Mattermost Integration in Zebrium to Send Detections to Mattermost

1. From the User menu area in Zebrium, click on the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Notifications** section and click on **Mattermost**.
4. Click **[Create a New Integration]**.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **[Send Detections]** tab.
10. Click **[Enabled]**.

11. Enter the **Webhook URL** created in STEP 1, above.
12. Click **[Save]**.

Slack Notifications

Features

- Automatically send Root Cause Reports to Slack channels. This allows you to see details of root cause in your Slack client.
- Each Zebrium RC Report includes a summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium UI.
- This means faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

Integration Overview

1. Create an Incoming Webhook in Slack
2. Create a Slack Integration in Zebrium using the information from STEP 1.

Integration Details

STEP 1: Create an Incoming Webhook in Slack

1. In a web browser, navigate to https://api.slack.com and log in to your workspace.
2. Click **Your Apps**, then the **[Create New App]** button, and then **From Scratch**.
3. Enter an **App Name**, select the appropriate **Workspace**, and then click **[Create App]**.
4. Click **Incoming Webhooks**.
5. Set **Activate Incoming Webhooks** to **On**.
6. Click **Add New Webhook to Workspace**.
7. Select the desired **Channel** and click **[Allow]**.
8. Copy and save the **Webhook URL** for use in STEP 2.

STEP 2: Create a Slack Integration in Zebrium to Send Detections to Slack

1. From the User menu area in Zebrium, click on the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Notifications** section and click on **Slack**.
4. Click **[Create a New Integration]**.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.

9. Click the **[Send Detections]** tab.
10. Click **[Enabled]**.
11. Enter the **Webhook URL** created in STEP 1, above.
12. Click **[Save]**.

Microsoft Teams Notifications

Features

- Automatically send Root Cause Reports to Microsoft Teams channels. This allows you to see details of root cause in your Microsoft Teams client.
- Each Zebrium RC Report includes a summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium UI.
- This means faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

Integration Overview

1. Create an Incoming Webhook in Microsoft Teams
2. Create a Microsoft Teams Integration in Zebrium using the information from STEP 1.

Integration Details

STEP 1: Create an Incoming Webhook in Microsoft Teams

1. In Microsoft Teams, go to the **Channel** that you want to receive notifications.
2. Click the ellipsis button (…) at the top right to open the configuration menu, and then select *Connectors*.
3. Click **Add/Configure Incoming Webhook**, add the ***Name***, and then click **[Create]**.
4. Copy and save the **Webhook URL** for use in STEP 2, below.
5. Click **[Done]**.

STEP 2: Create a Microsoft Teams Integration in Zebrium to Send Detections to Microsoft Teams

1. From the User menu area in Zebrium, click on the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Notifications** section and click on **Microsoft Teams**.
4. Click **[Create a New Integration]**.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **[Send Detections]** tab.
10. Click **[Enabled]**.

11. Enter the **Webhook URL** created in STEP 1, above.
12. Click **[Save]**.

Webex Teams Notifications

Features

- Automatically send Root Cause Reports to Webex Teams spaces. This allows you to see details of root cause in your Webex Teams client.
- Each Zebrium RC Report includes a summary, word cloud and a set of log events showing symptoms and root cause. Plus a link to the full report in the Zebrium UI.
- This means faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

Integration Overview

1. Create an Incoming Webhook in Webex Teams
2. Create a Webex Teams Integration in Zebrium using the information from STEP 1

Integration Details

STEP 1: Create an Incoming Webhook in Webex Teams

1. In Webex Teams, navigate to the **Space** where you want to receive notifications.
2. Click the **Gear** icon and select **Add Integrations and Bots...** to navigate to the **Webex App Hub** page.
3. Search for "webhooks" using the **Search apps** field on the **Webex App Hub** page.
4. Click on **Incoming webhooks**.
5. Scroll down and enter the **Webhook** name.
6. Select the desired **Space**.
7. Click **[Add]**.
8. Copy and save the **Webhook URL** for use in STEP 2.

STEP 2: Create a Webex Teams Integration in Zebrium to Send Detections to Webex Teams

1. From the User menu area in Zebrium, click on the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Notifications** section and click on **Webex Teams**.
4. Click **[Create a New Integration]**.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.

8. Select the **Service Group(s)** for the integration.
9. Click the **[Send Detections]** tab.
10. Click **[Enabled]**.
11. Enter the **Webhook URL** created in STEP 1, above.
12. Click **[Save]**.

Chapter 8

Creating Integrations Using Webhooks

Overview

Zebrium provides support for using webhooks so you can build your own custom integrations.

Zebrium provides the following webhooks:

- Outgoing Root Cause Report Webhook
- Incoming Root Cause Report Incoming Webhook

Root Cause Report Outgoing Webhook

Root Cause Report outgoing webhooks are sent when data is ingested and the AI/ML engine detects an incident comprised of anomalous events.

The frequency of Root Cause Report outgoing webhooks depend on data ingest and detection of root cause reports.

For more information, see ***Root Cause Report Outgoing Webhook.***

Root Cause Report Incoming Webhook

Signal incoming webhooks provide a generic mechanism for requesting Root Cause analysis for a specific time. This can be useful for integrating with third-party of custom solutions for which a specific integration is not currently available from Zebrium.

For more information, see ***Root Cause Report Incoming Webhook.***

Root Cause Report Outgoing Webhook

Features

- This section provides detailed information on webhook support provided by Zebrium so you can build your own custom integrations.
- Root Cause Report webhook payloads are sent when data is ingested and our machine learning detects an incident comprised of anomalous events.
- Frequency of Incident webhook depends on data ingest and detection of anomalies.

Integration Overview

1. Determine the destination endpoint and authentication requirements that will receive the Root Cause Report Outgoing Webhook.
2. Create a Root Cause Report Outgoing Webhook Integration in Zebrium using the information from STEP 1.

Integration Details

STEP 1: Determine the Destination Endpoint

The destination endpoint is the endpoint URL that will receive and process the content of the Root Cause Report Outgoing Webhook.

The authentication method for the endpoint can be one of the following:

- None
- Basic authentication
- Token (or Bearer) authentication

The authentication method and its associated configuration parameters will be used in STEP 2.

STEP 2: Create a Root Cause Report Outgoing Webhook Integration in Zebrium.

1. From the User menu area in Zebrium, click on the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Webhooks** section and click on **Outgoing RCA**.
4. Click **[Create a New Integration]**.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.

8. Select the **Service Group(s)** for the integration.
9. Click the **[Send Detections]** tab.
10. Click **[Enabled]**.
11. Enter the **Webhook URL** that will receive and handle the POST request.
12. Select the required **Authentication Method** for the endpoint and complete the necessary configuration using the information from STEP 1, above.
13. Click **[Save]**.

Webhook Payload Format

See ***Root Cause Report Outgoing Webhook Payload*** for a detailed description of the webhook payload.

Root Cause Report Outgoing Webhook Payload

Payload

Name	Type	Description
`account`	string	Zebrium account name for this customer_name
`customer_name`	string	Customer name of Zebrium instance
`deployment_name`	string	Name of the deployment where incident was raised
`event_type`	string	Always: “zebrium_incident”
`first_occurrence`	boolean	First time this incident has been seen
`incident_bad_level`	number	Numeric scale from 0-9 indicating the badness of the core events in the RC report (9 being very bad)
`incident_desc_alt`	string	**Unused**
`incident_desc`	string	Description of incident assigned by GPT3 or the user
`incident_epoch`	integer	UTC epoch of incident start
`incident_epoch_ts`	timestamp (yyyy-mm-ddThh:mm:ss.nnnnnnZ)	UTC timestamp of incident start
`incident_feedback`	number	1-5 Likert rating given to this incident type
`incident_fevent_gen`	string	Log generator name of the first event in the incident (Zebrium On-Prem only)
`incident_fevent_host`	string	Host of first event in the incident (Zebrium On-Prem only)
`incident_fevent_log`	string	Log name of first event in the incident (Zebrium On-Prem only)
`incident_fevent_ts`	string	Timestamp of the first event in the incident (Zebrium On-Prem only)
`incident_group`	string	Name of the incident group where incident was raised
`incident_hosts`	string	Comma separated list of hosts participating in this incident (Zebrium On-Prem only)
`incident_id`	uuid	Unique identifier for the incident
`incident_jira_url`	url encoded string	URL to the Jira Issue linked to this incident type
`incident_like`	url encoded string	API URL to "like" the incident
`incident_local_offset`	string	Local time offset from UTC as depicted in the log event
`incident_local_`	timestamp (yyyy-mm-	Local time of incident start

Name	Type	Description
`timestamp`	ddThh:mm:ss.nnnnnn)	
`incident_logs`	string	Comma separated list of logs participating in the incident (Zebrium On-Prem only)
`incident_mute`	url encoded string	API URL to "mute" the incident
`incident_name`	string	Title of incident assigned by GPT3 (abbreviated version) or the user
`incident_owner`	string	Owner assigned to this incident
`incident_priority`	string	Priority assigned to this incident (P1/P3)
`incident_rare_level`	number	Numeric scale from 0-9 indicating the rareness of the core events in the RC report (9 being very rare)
`incident_repeat_ct`	number	Number of times this incident type has been seen
`incident_repeat_idx`	number	Time ordered occurrence of this incident type
`incident_short_name`	string	System generated name for the incident type
`incident_spam`	url encoded string	API URL to tag incident as "spam"
`incident_state`	string	State of the incident (open, muted)
`incident_summary`	string	**Unused**
`incident_summary_feedback`	number	**Unused**
`incident_touches_agent`	boolean	Incident is related to a log or metrics collector vs. application
`incident_touches_k8s`	boolean	Incident is related to Kubernetes infrastructure
`incident_type`	uuid	Unique identifier for the incident type
`incident_url`	url encoded string	URL to view incident in the Zebrium UI
`incident_wevent_gen`	string	Log generator name of the worst event in the incident (Zebrium On-Prem only)
`incident_wevent_host`	string	Host of worst event in the incident (Zebrium On-Prem only)
`incident_wevent_log`	string	Log name of worst event in the incident (Zebrium On-Prem only)
`incident_wevent_ts`	string	Timestamp of the worst event in the incident (Zebrium On-Prem only)
`incident_words`	word object list	List of words (w) and their rareness/size (s) and badness (b) used in the word cloud

Name	Type	Description
`service_groups`	string list	List of service groups touched by this incident
`signal_association`	string	How is Incident associated to the signal (related or nearby)
`signal_initiated`	boolean	Incident is associated with a signal request
`signal_timestamp`	string	Timestamp of the signal request
`signal_type`	string	What initiated the signal. Could be USER, OPSGENIE, PAGERDUTY, SLACK
`incident_hallmark_event`	event object	Event determined to be the most severe indicator of the incident (**Unused**)
`incident_events`	event object list	All events in the core RC Report (level 0-2)
`key_events`	event object list	Key events (level 0) in RC Report
`interesting_events`	event object list	Interesting events (level 1) in RC Report
`nearby_events`	event object list	Nearby events (level 3-5) in RC Report

Event Object

Name	Type	Description
`app`	string	Application name from meta data
`container_name`	string	Container name from meta data
`epoch`	integer	UTC epoch of event
`epoch_ts`	timestamp (yyyy-mm-ddThh:mm:ss.nnnnnnZ)	UTC timestamp of event
`etype`	string	Name of the event type
`event_context_level`	integer	Event level: 0=key, 1=interesting, 2=core, 3,4,5=nearby
`event_meta_data`	set of name value pairs	Name value pairs derived from event meta data
`event_text`	string	Log event text
`event_uuid`	uuid	Unique identifier for the event
`hallmark`	boolean	True if this event is the hallmark event
`host`	string	Host on which event originated
`incident_group`	string	Name of the incident group where anomaly was raised
`local_offset`	string	Local time offset from UTC as depicted in the log event
`local_timestamp`	timestamp (yyyy-mm-ddThh:mm:ss.nnnnnn)	Local timestamp of event

Name	Type	Description
`log_name`	string	Name of log basename (e.g. syslog, error)
`namespace_name`	string	Namespace name from meta data
`root_cause`	boolean	True if this event is the root cause event
`severity_num`	integer	Severity number as defined by syslog
`severity`	string	Severity text as see in the log (e.g. INFO)
`ze_xid`	uuid	Unique external identifier for the event if provided by the log collector (otherwise empty)

Example Payload

```
{
  "incident_id": "00000000-0000-0000-0000-000000000000",
  "incident_type": "00000000-0000-0000-0000-000000000000",
  "incident_epoch_ts": "2021-10-15T21:07:13.813857Z",
  "incident_epoch": 1634332033813,
  "incident_state": "open",
  "incident_desc": "Notes let you document details of a report to help
colleagues understand your analysis in the future.",
  "incident_repeat_ct": 2,
  "incident_local_timestamp": "2021-10-15T21:07:13.813857Z",
  "incident_local_offset": "+0000",
  "incident_touches_k8s": false,
  "incident_touches_agent": false,
  "incident_name": "SAMPLE - You would normally see An NLP-generated title
here",
  "incident_short_name": "cfcd2",
  "incident_summary": "",
  "incident_owner": "Zebrium",
  "incident_feedback": 5,
  "incident_summary_feedback": "",
  "incident_jira_url": "https://www.zebrium.com",
  "incident_priority": "P3",
  "service_groups": [
    "sample"
  ],
  "signal_initiated": false,
  "signal_type": "",
  "signal_timestamp": "",
  "signal_association": "",
  "incident_repeat_idx": 2,
  "first_occurrence": false,
  "incident_hosts": "host1,host2,host3",
  "incident_logs": "logtype1,logtype2,zoom_log",
  "incident_fevent_host": "",
  "incident_fevent_log": "",
  "incident_fevent_ts": "",
  "incident_fevent_gen": "",
  "incident_wevent_host": "",
  "incident_wevent_log": "",
  "incident_wevent_ts": "",
  "incident_wevent_gen": "",
  "incident_bad_level": 5,
  "incident_rare_level": 5,
  "incident_words": [
    {
      "w": "critical",
      "s": 10,
      "b": 4
    },
    {
      "w": "peek",
      "s": 14,
      "b": 4
    },
    {
      "w": "characterize",
      "s": 14,
      "b": 1
    },
    {
      "w": "rca",
      "s": 14,
      "b": 2
    },
    {
      "w": "filter",
      "s": 12,
      "b": 4
    },
    {
      "w": "zoom",
      "s": 10,
      "b": 1
    },
    {
      "w": "correlated",
      "s": 8,
      "b": 4
    },
    {
      "w": "enjoy",
      "s": 6,
      "b": 2
    },
    {
      "w": "useful",
      "s": 4,
      "b": 4
    },
    {
      "w": "wordcloud",
      "s": 2,
      "b": 4
    },
    {
      "w": "related",
      "s": 2,
      "b": 2
    },
    {
      "w": "reports",
      "s": 2,
      "b": 2
    },
    {
      "w": "data",
      "s": 2,
      "b": 4
    },
    {
      "w": "zebrium",
      "s": 2,
      "b": 2
    },
    {
      "w": "raw",
      "s": 2,
      "b": 1
    },
    {
      "w": "fast",
      "s": 2,
      "b": 2
    }
  ],
  "account": "zebrium465_trial",
  "customer_name": "zebrium465",
  "deployment_name": "trial",
  "incident_group": "sample",
  "event_type": "zebrium_incident",
  "incident_url": "https://cloud.zebrium.com/root-cause/report?itype_
id=00000000-0000-0000-0000-000000000000&inci_id=00000000-0000-0000-0000-
000000000000&ievt_level=2",
  "incident_like": "https://cloud.zebrium.com /ap-
i/v2/incident/setstate/00000000-0000-0000-0000-
000000000000/liked/B316BB07D18F63B61AF62416BCD7A73B960D48DD",
  "incident_mute": "https://cloud.zebrium.com /ap-
i/v2/incident/setstate/00000000-0000-0000-0000-
000000000000/muted/B316BB07D18F63B61AF62416BCD7A73B960D48DD",
  "incident_spam": "https://cloud.zebrium.com /ap-
i/v2/incident/setstate/00000000-0000-0000-0000-
000000000000/spam/B316BB07D18F63B61AF62416BCD7A73B960D48DD",
  "incident_desc_alt": "Notes let you document details of a report to help
colleagues understand your analysis in the future.",
  "incident_hallmark_event": {
    "root_cause": false,
    "hallmark": true,
    "epoch_ts": "2021-10-15T21:07:29.833156Z",
    "epoch": 1634332049833,
    "etype": "line",
    "log_name": "logtype2",
    "severity_num": 2,
    "event_uuid": "00000000-0000-0000-0000-000000000008",
    "event_text": "[2021-10-15 21:07:29.833156] CRITICAL: This is the
second of two events that are used to characterize the report in the list
view",
    "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host1,zid_log=logtype2",
    "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host1.fqdm.com",
    "local_timestamp": "2021-10-15T21:07:29.833156Z",
    "local_offset": "+0000",
    "ze_xid": "",
    "event_context_level": 0,
    "host": "host1",
    "severity": "Critical",
    "app": null,
    "container_name": "logtype2",
    "namespace_name": null,
    "event_meta_data": {
      "ze_deployment_name": "sample",
      "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
      "hostname": "host1.fqdm.com"
    }
  },
  "incident_events": [
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:06:49.790742Z",
      "epoch": 1634332009790,
      "etype": "line",
      "log_name": "logtype1",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000003",
      "event_text": "[2021-10-15 21:06:49.790742] INFO: This is a sample
root cause report",
      "metadata_id": "ze_deployment_name=sample,zid_container_
```

```
name=logtype1,zid_host=host2,zid_log=logtype1",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host2.fqdm.com",
      "local_timestamp": "2021-10-15T21:06:49.790742Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host2",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype1",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host2.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:06:57.7982Z",
      "epoch": 1634332017798,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000004",
      "event_text": "[2021-10-15 21:06:57.7982] INFO: Real Root Cause
Reports typically have 5-20 \"Core\" log events",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host2,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host2.fqdm.com",
      "local_timestamp": "2021-10-15T21:06:57.7982Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host2",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host2.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:07:05.805105Z",
      "epoch": 1634332025805,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000005",
      "event_text": "[2021-10-15 21:07:05.805105] INFO: Core events con-
sist of mostly \"rare\" and high-severity events that are correlated
across multiple logs",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host2,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host2.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:05.805105Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host2",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host2.fqdm.com"
      }
    },
    {
      "root_cause": true,
      "hallmark": true,
      "epoch_ts": "2021-10-15T21:07:13.82029Z",
      "epoch": 1634332033820,
      "etype": "line",
      "log_name": "logtype1",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000006",
      "event_text": "[2021-10-15 21:07:13.82029] INFO: This is the first
of two events that are used to characterize the report in the list view",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype1,zid_host=host1,zid_log=logtype1",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host1.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:13.82029Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 0,
      "host": "host1",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype1",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host1.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:07:21.826703Z",
      "epoch": 1634332041826,
      "etype": "line",
      "log_name": "logtype1",
      "severity_num": 3,
      "event_uuid": "00000000-0000-0000-0000-000000000007",
      "event_text": "[2021-10-15 21:07:21.826703] ERROR: Did you notice
this event has error severity?",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype1,zid_host=host1,zid_log=logtype1",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host1.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:21.826703Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host1",
      "severity": "Error",
      "app": null,
      "container_name": "logtype1",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host1.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": true,
      "epoch_ts": "2021-10-15T21:07:29.833156Z",
      "epoch": 1634332049833,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 2,
      "event_uuid": "00000000-0000-0000-0000-000000000008",
      "event_text": "[2021-10-15 21:07:29.833156] CRITICAL: This is the
second of two events that are used to characterize the report in the list
view",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host1,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host1.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:29.833156Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 0,
      "host": "host1",
      "severity": "Critical",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host1.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:07:37.840903Z",
      "epoch": 1634332057840,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000009",
      "event_text": "[2021-10-15 21:07:37.840903] INFO: Now try the filter
bar (above), and highlight bar (below)",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host2,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host2.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:37.840903Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host2",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host2.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:07:45.851986Z",
      "epoch": 1634332065851,
      "etype": "line",
      "log_name": "logtype1",
      "severity_num": 2,
      "event_uuid": "00000000-0000-0000-0000-000000000010",
      "event_text": "[2021-10-15 21:07:45.851986] CRITICAL: If you do not
see enough detail in the Core events, try these things:",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype1,zid_host=host1,zid_log=logtype1",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host1.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:45.851986Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host1",
      "severity": "Critical",
      "app": null,
      "container_name": "logtype1",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host1.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:07:53.858345Z",
      "epoch": 1634332073858,
      "etype": "line",
      "log_name": "logtype1",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000011",
      "event_text": "[2021-10-15 21:07:53.858345] INFO: Click the Peek
```

```
button (at the end of each log line) to see all available lines from just
this log stream",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype1,zid_host=host2,zid_log=logtype1",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host2.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:53.858345Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host2",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype1",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host2.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:08:01.864572Z",
      "epoch": 1634332081864,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000012",
      "event_text": "[2021-10-15 21:08:01.864572] INFO: Or zoom out beyond
the Core events by clicking a Zoom level in Related Events (at the top)",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host2,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host2.fqdm.com",
      "local_timestamp": "2021-10-15T21:08:01.864572Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host2",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host2.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:08:09.871442Z",
      "epoch": 1634332089871,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000013",
      "event_text": "[2021-10-15 21:08:09.871442] INFO: Zooming is useful
when the Core events do not contain enough information",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host1,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host1.fqdm.com",
      "local_timestamp": "2021-10-15T21:08:09.871442Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host1",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host1.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:08:17.878258Z",
      "epoch": 1634332097878,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000014",
      "event_text": "[2021-10-15 21:08:17.878258] INFO: Enjoy using
Zebrium and let us know if you have any questions!",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host1,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host1.fqdm.com",
      "local_timestamp": "2021-10-15T21:08:17.878258Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host1",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host1.fqdm.com"
      }
    }
  ],
  "key_events": [
    {
      "root_cause": true,
      "hallmark": true,
      "epoch_ts": "2021-10-15T21:07:13.82029Z",
      "epoch": 1634332033820,
      "etype": "line",
      "log_name": "logtype1",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000006",
      "event_text": "[2021-10-15 21:07:13.82029] INFO: This is the first
of two events that are used to characterize the report in the list view",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype1,zid_host=host1,zid_log=logtype1",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host1.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:13.82029Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 0,
      "host": "host1",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype1",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host1.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": true,
      "epoch_ts": "2021-10-15T21:07:29.833156Z",
      "epoch": 1634332049833,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 2,
      "event_uuid": "00000000-0000-0000-0000-000000000008",
      "event_text": "[2021-10-15 21:07:29.833156] CRITICAL: This is the
second of two events that are used to characterize the report in the list
view",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host1,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host1.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:29.833156Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 0,
      "host": "host1",
      "severity": "Critical",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host1.fqdm.com"
      }
    }
  ],
  "interesting_events": [
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:06:49.790742Z",
      "epoch": 1634332009790,
      "etype": "line",
      "log_name": "logtype1",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000003",
      "event_text": "[2021-10-15 21:06:49.790742] INFO: This is a sample
root cause report",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype1,zid_host=host2,zid_log=logtype1",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host2.fqdm.com",
      "local_timestamp": "2021-10-15T21:06:49.790742Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host2",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype1",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host2.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:06:57.7982Z",
      "epoch": 1634332017798,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000004",
      "event_text": "[2021-10-15 21:06:57.7982] INFO: Real Root Cause
Reports typically have 5-20 \"Core\" log events",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host2,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host2.fqdm.com",
      "local_timestamp": "2021-10-15T21:06:57.7982Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host2",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host2.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:07:05.805105Z",
      "epoch": 1634332025805,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000005",
      "event_text": "[2021-10-15 21:07:05.805105] INFO: Core events con-
sist of mostly \"rare\" and high-severity events that are correlated
across multiple logs",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host2,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host2.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:05.805105Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host2",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host2.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:07:21.826703Z",
      "epoch": 1634332041826,
      "etype": "line",
      "log_name": "logtype1",
      "severity_num": 3,
      "event_uuid": "00000000-0000-0000-0000-000000000007",
      "event_text": "[2021-10-15 21:07:21.826703] ERROR: Did you notice
this event has error severity?",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype1,zid_host=host1,zid_log=logtype1",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host1.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:21.826703Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host1",
      "severity": "Error",
      "app": null,
      "container_name": "logtype1",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host1.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:07:37.840903Z",
      "epoch": 1634332057840,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000009",
      "event_text": "[2021-10-15 21:07:37.840903] INFO: Now try the filter
bar (above), and highlight bar (below)",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host2,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host2.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:37.840903Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host2",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host2.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:07:45.851986Z",
      "epoch": 1634332065851,
      "etype": "line",
      "log_name": "logtype1",
      "severity_num": 2,
      "event_uuid": "00000000-0000-0000-0000-000000000010",
      "event_text": "[2021-10-15 21:07:45.851986] CRITICAL: If you do not
see enough detail in the Core events, try these things:",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype1,zid_host=host1,zid_log=logtype1",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host1.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:45.851986Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host1",
      "severity": "Critical",
      "app": null,
      "container_name": "logtype1",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host1.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:07:53.858345Z",
      "epoch": 1634332073858,
      "etype": "line",
      "log_name": "logtype1",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000011",
      "event_text": "[2021-10-15 21:07:53.858345] INFO: Click the Peek
button (at the end of each log line) to see all available lines from just
this log stream",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype1,zid_host=host2,zid_log=logtype1",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host2.fqdm.com",
      "local_timestamp": "2021-10-15T21:07:53.858345Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host2",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype1",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host2.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:08:01.864572Z",
      "epoch": 1634332081864,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000012",
      "event_text": "[2021-10-15 21:08:01.864572] INFO: Or zoom out beyond
the Core events by clicking a Zoom level in Related Events (at the top)",
      "metadata_id": "ze_deployment_name=sample,zid_container_
```

```
name=logtype2,zid_host=host2,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host2.fqdm.com",
      "local_timestamp": "2021-10-15T21:08:01.864572Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host2",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host2.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:08:09.871442Z",
      "epoch": 1634332089871,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000013",
      "event_text": "[2021-10-15 21:08:09.871442] INFO: Zooming is useful
when the Core events do not contain enough information",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host1,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host1.fqdm.com",
      "local_timestamp": "2021-10-15T21:08:09.871442Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host1",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host1.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:08:17.878258Z",
      "epoch": 1634332097878,
      "etype": "line",
      "log_name": "logtype2",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000014",
      "event_text": "[2021-10-15 21:08:17.878258] INFO: Enjoy using
Zebrium and let us know if you have any questions!",
      "metadata_id": "ze_deployment_name=sample,zid_container_name-
e=logtype2,zid_host=host1,zid_log=logtype2",
      "metadata_cfg": "ze_deployment_name=sample,container_name=logtype1-
359f02372109b4222880d1c7932b717f,hostname=host1.fqdm.com",
      "local_timestamp": "2021-10-15T21:08:17.878258Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 1,
      "host": "host1",
      "severity": "Informational",
      "app": null,
      "container_name": "logtype2",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "logtype1-359f02372109b4222880d1c7932b717f",
        "hostname": "host1.fqdm.com"
      }
    }
  ],
  "nearby_events": [
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:06:25.77145Z",
      "epoch": 1634331985771,
      "etype": "line",
      "log_name": "zoom_log",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000000",
      "event_text": "[2021-10-15 21:06:25.77145] INFO: You are seeing this
event because you zoomed into Related Events level 3 (or because you
Peeked)",
      "metadata_id": "ze_deployment_name=sample,zid_container_name=zoom_
log,zid_host=host3,zid_log=zoom_log",
      "metadata_cfg": "ze_deployment_name=sample,container_name=zoom_log-
a32e129fccd92e3ab19e749655f152a7,hostname=host3.fqdm.com",
      "local_timestamp": "2021-10-15T21:06:25.77145Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 5,
      "host": "host3",
      "severity": "Informational",
      "app": null,
      "container_name": "zoom_log",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "zoom_log-a32e129fccd92e3ab19e749655f152a7",
        "hostname": "host3.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:06:33.778395Z",
      "epoch": 1634331993778,
      "etype": "line",
      "log_name": "zoom_log",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000001",
      "event_text": "[2021-10-15 21:06:33.778395] INFO: You are seeing
this event because you zoomed into Related Events level 2 (or because you
Peeked)",
      "metadata_id": "ze_deployment_name=sample,zid_container_name=zoom_
log,zid_host=host3,zid_log=zoom_log",
      "metadata_cfg": "ze_deployment_name=sample,container_name=zoom_log-
a32e129fccd92e3ab19e749655f152a7,hostname=host3.fqdm.com",
      "local_timestamp": "2021-10-15T21:06:33.778395Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 4,
      "host": "host3",
      "severity": "Informational",
      "app": null,
      "container_name": "zoom_log",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "zoom_log-a32e129fccd92e3ab19e749655f152a7",
        "hostname": "host3.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:06:41.784659Z",
      "epoch": 1634332001784,
      "etype": "line",
      "log_name": "zoom_log",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000002",
      "event_text": "[2021-10-15 21:06:41.784659] INFO: You are seeing
this event because you zoomed into Related Events level 1 (or because you
Peeked)",
      "metadata_id": "ze_deployment_name=sample,zid_container_name=zoom_
log,zid_host=host3,zid_log=zoom_log",
      "metadata_cfg": "ze_deployment_name=sample,container_name=zoom_log-
a32e129fccd92e3ab19e749655f152a7,hostname=host3.fqdm.com",
      "local_timestamp": "2021-10-15T21:06:41.784659Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 3,
      "host": "host3",
      "severity": "Informational",
      "app": null,
      "container_name": "zoom_log",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "zoom_log-a32e129fccd92e3ab19e749655f152a7",
        "hostname": "host3.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:08:25.885936Z",
      "epoch": 1634332105885,
      "etype": "line",
      "log_name": "zoom_log",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000015",
      "event_text": "[2021-10-15 21:08:25.885936] INFO: This is the last
event in the Related Events level 1 zoom out",
      "metadata_id": "ze_deployment_name=sample,zid_container_name=zoom_
log,zid_host=host3,zid_log=zoom_log",
      "metadata_cfg": "ze_deployment_name=sample,container_name=zoom_log-
a32e129fccd92e3ab19e749655f152a7,hostname=host3.fqdm.com",
      "local_timestamp": "2021-10-15T21:08:25.885936Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 3,
      "host": "host3",
      "severity": "Informational",
      "app": null,
      "container_name": "zoom_log",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "zoom_log-a32e129fccd92e3ab19e749655f152a7",
        "hostname": "host3.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:08:33.896882Z",
      "epoch": 1634332113896,
      "etype": "line",
      "log_name": "zoom_log",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000016",
      "event_text": "[2021-10-15 21:08:33.896882] INFO: This is the last
event in the Related Events level 2 zoom out",
      "metadata_id": "ze_deployment_name=sample,zid_container_name=zoom_
log,zid_host=host3,zid_log=zoom_log",
      "metadata_cfg": "ze_deployment_name=sample,container_name=zoom_log-
a32e129fccd92e3ab19e749655f152a7,hostname=host3.fqdm.com",
      "local_timestamp": "2021-10-15T21:08:33.896882Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 4,
      "host": "host3",
      "severity": "Informational",
      "app": null,
      "container_name": "zoom_log",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "zoom_log-a32e129fccd92e3ab19e749655f152a7",
        "hostname": "host3.fqdm.com"
      }
    },
    {
      "root_cause": false,
      "hallmark": false,
      "epoch_ts": "2021-10-15T21:08:41.903443Z",
      "epoch": 1634332121903,
      "etype": "line",
      "log_name": "zoom_log",
      "severity_num": 6,
      "event_uuid": "00000000-0000-0000-0000-000000000017",
      "event_text": "[2021-10-15 21:08:41.903443] INFO: This is the last
event in the Related Events level 3 zoom out",
      "metadata_id": "ze_deployment_name=sample,zid_container_name=zoom_
log,zid_host=host3,zid_log=zoom_log",
      "metadata_cfg": "ze_deployment_name=sample,container_name=zoom_log-
a32e129fccd92e3ab19e749655f152a7,hostname=host3.fqdm.com",
      "local_timestamp": "2021-10-15T21:08:41.903443Z",
      "local_offset": "+0000",
      "ze_xid": "",
      "event_context_level": 5,
      "host": "host3",
      "severity": "Informational",
      "app": null,
      "container_name": "zoom_log",
      "namespace_name": null,
      "event_meta_data": {
        "ze_deployment_name": "sample",
        "container_name": "zoom_log-a32e129fccd92e3ab19e749655f152a7",
        "hostname": "host3.fqdm.com"
      }
    }
  ]
}
```

Root Cause Report Incoming Webhook

Features

- This section provides detailed information on webhook support provided by Zebrium so you can build your own custom integrations.
- Root Cause Report incoming webhooks provide a generic mechanism for requesting Root Cause analysis for a specific time. This can be useful for integrating with third-party of custom solutions for which a specific integration is not currently available from Zebrium.

Integration Overview

1. Create a Root Cause Report Incoming Webhook Integration in Zebrium.
2. Send a request for a Root Cause Report to Zebrium using the webhook created in step 1 and the required payload.

Integration Details

STEP 1: Create a Root Cause Report Incoming Webhook Integration in Zebrium.

1. From the User menu area in Zebrium, click on the **Settings** menu (hamburger).
2. Select **Integrations**.
3. Scroll to the **Webhooks** section and click on **Incoming RCA**.
4. Click **[Create a New Integration]**.
5. Click the **[General]** tab.
6. Enter an **Integration Name** for this integration.
7. Select the **Deployment** for the integration.
8. Select the **Service Group(s)** for the integration.
9. Click the **[Receive Signals]** tab.
10. Click **[Enabled]**.
11. Enter the **Webhook URL** that will receive and handle the POST request.
12. Select the required **Authentication Method** for the endpoint and complete the necessary configuration using the information from STEP 1, above.
13. Click **[Save]**.
14. Copy and save the contents of the **Your URL** text box for use in STEP 2 when sending a request for a Root Cause Report to Zebrium.

STEP 2: Request a Root Cause Report from Zebrium

Send a POST request to the URL created in STEP 1 with the required payload (see the ***Webhook Payload*** page for payload details):

```
curl -X POST -H 'Content-type: application/json' --data '<REQUEST_JSON_
PAYLOAD>' <URL_FROM_STEP_1>
```

Webhook Payload Format

See ***Root Cause Report Incoming Webhook Payload*** for a detailed description of the webhook payload.

Root Cause Report Incoming Webhook Payload

Method URL	URL created for this integration
HTTP Method	POST
Content Type	application/json

Payload

Name	Type	Description	Required
zebrium.incident_ts	string	UTC Timestamp to perform RC Analysis, such as "2022-03-15T08:23:05Z"	Yes
zebrium.service_group	string	Zebrium service group to perform RC Analysis or 'All'	Yes

Example Payload

```
{
  "zebrium" : {
    "incident_ts" : "2022-03-15T08:23:05Z",
    "service_group" : "production"
  }
}
```

```
curl -X POST -H 'Content-type: application/json' --data '{ "zebrium" : {
"incident_ts" : "2022-03-15T08:23:05Z",
"service_group" : "production" } }' https://cloud.zebri-
um.com/api/v2/signal/E0D2C20624779984FADBE0D22E4125860A37299B
```

Chapter 9

Managing Users

Overview

User Management provides features for Role Based Access Controls whereby you can create groups, assign roles to users, and assign users to groups.

By default, nothing will change any user's access or roles that you have today so there is nothing you need to do unless desired. This means that All Users will be assigned the least restricted Owner role.

This chapter covers the following topics:

RBAC Component Definitions

- ***Users***: Each user is assigned a Role (permissions on features/settings) and Users are members of one or more Groups to control which deployments they can access.
- ***Groups***: Groups define which deployments are available to Users in the Group.
- ***Roles***: Pre-defined roles (Owner, Admin, Editor, Viewer) which define permissions (e.g. Create, Read/view, Update, Delete) for each feature or application setting.

Users

- Each user is assigned a Role (permissions on features/settings) and Users are members of one or more Groups to control which deployments they can access.
- Users belong to 1 or more groups.
- Users can be added/edited/deleted by the Owner role (see Roles and Permissions below).
- User Management is available under the Gear pull-down menu for Account Settings.

Groups

- Groups define which deployments are available to Users in the Group.
- The default group is All and has All deployments assigned to the group.
- Groups can be added/edited/deleted by the Owner role (see Roles and Permissions below).
- Group Management is available under the Gear pull-down menu for Account Settings.

Roles

The following role permissions are pre-defined and not configurable.

Owner

- Allows for billing and user management, including the creation and assignment of deployments in groups.
- Includes all permissions of the Admin and Member roles.
- Owner is the default role for a new user during initial account creation.
- All existing users are Owner roles until changed (by another Owner).

Admin

Day-to-day configuration including setting up integrations and various application customizations.

Editor

Users allowed to edit (create, update, delete) objects, particularly incident type metadata. This role will be assigned to user of the role Member in a previous release

Viewer

Users that are allowed read-only access to all but their own profile, e.g. to change their deployment selection or password

Permissions

Setting/Feature	Owner	Admin	Editor	Viewer
Report Notes and Alerting	Edit	Edit	Edit	View
Report Notes and Alerting	Edit	Edit	None	None
Report Notes and Alerting	Edit	Edit	None	None
Integrations	Edit	Edit	None	None
Root Cause Settings	Edit	Edit	None	None
User Management	Edit	Edit	None	None
Billing	Edit	None	None	None

Chapter 10

Zebrium Service Security

Overview

This chapter covers the following topics:

Culture Based on Data Security

Securing customer data is a critical part of our promise to customers. We understand how important data security and privacy are to our users.

The team behind Zebrium has decades of experience securely handling sensitive software logs and metrics for market leading enterprise products that are used by some of the most security conscious enterprises and government organizations. We have geared all aspects of our architecture, operations, and company culture to meet these expectations.

The purpose of this writeup is to provide our customers with a “plain English” description of some of the security protections we have in place. A more extensive, technical explanation is available in our infosec policy, which can be provided upon request.

Logical (and Optionally Physical) Separation of Customer Data

All customer data is tagged with a unique token identifier per organization, and each organization is assigned a unique schema within the underlying database. All read/write operations rigorously enforce the mapping of organization to assigned schema and data.

For customers with additional security restrictions, we offer the option of hosting your service in a dedicated virtual private cloud (VPC) instance assigned exclusively to you. Since your data never leaves the dedicated VPC, this provides an additional layer of protection over and above logical controls. Please contact us if you have more specific requirements for the location of the service.

Encryption

All customer interactions with the Zebrium service, including data upload, download and UI operations are encrypted using HTTPS and SSL.

All data at rest is encrypted using AES-256 encryption.

Single Sign-On Support

Zebrium supports most leading SSO providers via SAML including: Auth0, Azure, Duo, Jumpcloud, Okta.

Service Security

All inter-node communication within the Zebrium service is locked down by only allowing communication between white listed nodes over a private subnet. SSH access to the service is only enabled for white-listed IP addresses.

Every code deployment automatically updates Zebrium nodes to include security updates from the latest version of Ubuntu Linux currently available.

The service regularly undergoes penetration testing by 3rd parties, with no vulnerabilities unresolved.

All logs from software components of the Zebrium service are themselves fed into and analyzed by another instance of the Zebrium service in order to uncover anomalous patterns.

Handling of Sensitive Data

The Zebrium service supports the option of filtering out specific event types, for instance those containing sensitive fields such as IP addresses. One of the unique advantages of the Zebrium solution is the fact that all events in your logs are automatically and fully parsed, and all fields within them extracted and typed as variables. In the event that you accidentally upload customer sensitive data into our service, this capability means that we can support the clinical removal of such data.

Access by Zebrium Employees

Access to production systems running Zebrium software will be subject to the following conditions:

- Access to systems is only allowed by an explicitly defined group of Zebrium operations employees
- Access to systems is allowed only when there is a specific operational need
- SSH access to the Zebrium service is only enabled for a whitelisted set of IP addresses and ports
- Admin actions via management console, CLI, or access to underlying cloud services is audited, and audit logs are retained for retroactive review.

Access to data will be subject to the same conditions as above, plus some additional restrictions:

- Access will only be permitted for the purposes of troubleshooting, technical support or testing, tuning and quality assurance of our service.
- Additional access will only be permitted with customer consent and only on and as-needed basis.

Physical Security

- The Zebrium SaaS service is hosted in AWS datacenters with stringent security controls. Zebrium employees do not have physical access to these data centers.
- AWS data centers comply with the most rigorous security certifications including SOC 1, 2 and 3, PCI DSS 3.2 Level 1, ISO 27001, as well as FedRamp (select locations).

Customer Data

The customer retains full ownership of all customer data stored in Zebrium systems. Upon termination of the Zebrium service (or upon request), all copies of customer data will be deleted.

Reports and Third-party Audits

Extensive testing and auditing by internal and external security experts are part of our commitment to our customers. Reports are available upon request.

- CAIQv4
- SOC 2 Attestation
- Most recent third-party penetration test report

Chapter 11

Zebrium On Prem

Overview

In additional to the standard option of a cloud configuration for Zebrium, you also have the option for a Zebrium on-premises (On Prem) configuration that is not located in the cloud.

The following pages explain how to install a Zebrium On Prem configuration, how to contact Zebrium Support, and how to use the various APIs from Zebrium:

- *Getting Started*
- *Support*
- *Zebrium APIs*

Zebrium On Prem: Getting Started

The following pages describe how to get ready to install a Zebrium on-premises (On Prem) configuration, along with how to perform the installation and what to do after the installation:

- *Pre-installation*
- *Installation*

Pre-installation

This chapter covers the following topics:

Sizing Considerations

- Zebrium has a sizing calculator to determine vCPU and Memory requirements for your log ingest volume.
- Please contact Zebrium at support@zebrium.com for sizing calculations in preparation for deploying your Kubernetes cluster.

NOTE: For better supportability, Zebrium recommends that you install Zebrium On Prem in its own dedicated Kubernetes cluster rather than in a namespace on a shared cluster.

Software Requirements

The following list of requirements must be met for Zebrium software to be fully functional

- Kubernetes version 1.19 or higher required.
- Kubernetes cluster sizing must meet or exceed the Zebrium Sizing specifications. Please contact Zebrium to obtain these requirements.
- Helm version 3 is required for installation of Zebrium On Prem.
- Local storage is required. Shared storage such as NFS is not supported for the Zebrium database.
- Ingress Controller with https support for a Fully Qualified Host Name (FQHN).

Account Name

You must define the name of the account that will be used in your On Prem instance. You are currently limited to a single account. Recommended name format is *<company>_<deployment_name>*, such as **acme_onprem**.

Domain Name

You must define a Fully Qualified Host Name (FQHN) that will be used in the URL to view Root Cause Reports in the Zebrium UI.

Slack Channels

There are two Slack webhooks that can be configured in the Helm chart. Zebrium recommends configuring these two channels in your Slack instance and inviting Zebrium to the channels:

1. ***ZE_SLACK_WEBHOOK***. This channel will receive a summary of all Root Cause reports.
2. ***ZE_SLACK_DEBUG_WEBHOOK***. This channel will receive debug alerts on the operation of the Zebrium software. It is strongly recommended that Zebrium be invited to this Slack channel.

Helm Chart and Image Repository Access

- Zebrium will provide credentials (username/password) for access to your Helm chart and Zebrium container images managed in Harbor at: https://goharbor.io/
- You will be able to create a secret key in your Kubernetes deployment from your Harbor account which will be used in the Helm chart to pull images.

Helm Chart Overrides

Based on the sizing calculations, Zebrium will work with you to define the Helm chart specifications for scaling and required CPU and Memory for all Zebrium components.

Your FQHN will be configured in your Helm chart as:

- `global.hostname: ""`

The two Slack webhooks defined above will be configured in your Helm chart as the following:

- `zebrium-core.slack.enabled: "1"`
- `zebrium-core.slack.webhook: ""`
- `zebrium-core.slack.debugWebhook: ""`

The secret key used to pull images from Harbor will be configured in your Helm chart as:

- `global.imagePullSecret: ""`

Configuration Questions

1. Are you currently running your own ingress on your Kubernetes cluster?
2. Do you already have a storage class or provisioner for Kubernetes *PersistentVolumeClaims* (PVCs)?

Installation

This chapter covers the following topics:

Assumptions

1. All *pre-installation steps* have been completed.
2. Your Kubernetes cluster we will deploy to is sized to at least the minimum specification provided in our sizing guideline chart on the Zebrium sizing guide page for the log volume you plan to test. Click **Show Advanced Info** at the bottom of the sizing guideline chart for more details.
3. The cluster we will deploy to is running at minimum Kubernetes version 1.19.
4. Helm 3 can be configured and used with the cluster.
5. The cluster we will deploy to has local storage available (NFS is not supported).
6. A Fully Qualified Host Name is available for ingress with a SSL certificate (we only support https to our application UI)
7. Images can be pulled from **https://harbor.ops.zebrium.com/**, or other arrangements can be made to make the images available during install.
8. Any prerequisites for your preferred ingress controller have been gathered.

STEP 1: Installing the Helm Chart

1. Create a secret using the Harbor repository that was setup for you by Zebrium. Zebrium will provide you with your Harbor USERNAME and PASSWORD.
   ```
   kubectl create secret docker-registry regcred --docker-
   server=harbor.ops.zebrium.com --docker-username=<USERNAME> --docker-
   password=<PASSWORD> --docker-email=<EMAIL> --namespace <NAMESPACE>
   ```
2. Update your Helm chart **override.yaml** file with the secret from Step 1.
   ```
   global.imagePullSecret.Name="<SECRET>"
   ```
3. Add the Harbor repo to your Kubernetes cluster:
   ```
   helm repo add --username <USERNAME> --password <PASSWORD> <REPO_NAME>
   https://harbor.ops.zebrium.com/chartrepo/onprem
   ```
   ```
   helm repo update
   ```
4. Install the Zebrium On Prem Software:
   ```
   helm upgrade <RELEASE_NAME> -i --namespace <NAMESPACE> <REPO_
   NAME>/zebrium-onprem -f <override.yaml>
   ```

STEP 2: Configuring Your Account

Zebrium On Prem currently supports a single account where all data will be ingested. This account can have multiple users and logins.

Create your account and the first user:

IMPORTANT: You should only do this once.

1. In a browser, use the following format to type the URL for creating the account and first user:

   ```
   https://<Your_Zebrium_URL>/auth/sign-up?firstName=<first_
   name>&lastName=<last_name>&companyName=<company_name>&email=<user_
   email>
   ```

   For example:

   ```
   https://cloud.ze.com/auth/sign-
   up?firstName=Jane&lastName=Doe&companyName=Acme&email=JDoe@acme.com
   ```

2. Complete the remaining fields on the form and click the right arrow to continue.

3. Invite other users to your account by going to the **[User Management]** tab from the **Settings** menu.

4. Click **Add User** and complete the form

STEP 3: Configuring Outbound Notifications

If you would like Root Cause report notifications to be sent to a destination other than the Slack webhook defined in your Helm chart, you can configure additional notification channels by visiting the **Integrations** page from the **Settings** menu.

STEP 4: Configuring AutoSupport (optional)

When you register for Zebrium On Prem, Zebrium will provide a secure authentication token that can be used to send logs from the Zebrium On Prem software to Zebrium for remote monitoring (akin to Zebrium AutoSupport).

NOTE: Installing the Zebrium Kubernetes Log Collector will send logs from all namespaces in your Kubernetes cluster.

The log collector is deployed as a Kubernetes Helm chart as follows:

1. `kubectl create namespace zebrium`
2.
   ```
   helm install zlog-collector zlog-collector --namespace zebrium --repo
   https://raw.githubusercontent.com/zebrium/ze-kubernetes-
   collector/master/charts --set
   zebrium.collectorUrl=https://zapi03.zebrium.com,zebrium.authToken=<AUTH_
   TOKEN_FROM_ZEBRIUM>,zebrium.timezone=<KUBERNETES_HOST_TIMEZONE>
   ```

STEP 5: Ingesting Data into your Zebrium On Prem Instance

There are three supported methods to ingest data into Zebrium On Prem using your ZAPI Token and Endpoint:

- Zebrium CLI command
- Kubernetes Log Collector
- Elastic integration using Logstash

Obtaining your ZAPI Token and Endpoint

Instructions and commands for sending data to your On Prem instance is available under **Log Collector** in the settings menu.

Failure Domain Boundary

Because Zebrium On Prem currently supports a single account with only one deployment, if you intend to ingest data from unrelated services/applications it is important to specify the ze_deployment_name label which essentially defines a failure domain boundary for anomaly correlation.

You will see in the examples provided below, how to specify the ze_deployment_name label for each of the three methods that can be used to ingest data.

NOTE: The ze_deployment_name must be a single word lowercase characters.

Using the Command-line Interface to Ingest Data

For instructions about downloading, configuring, and using the Zebrium command-line interface, see ***File Upload (ze Command)***.

Here is an example that ingests a Jira log file into the atlassian failure domain (ze_deployment_name):

```
~/zapi/bin/ze up --file=jira.log --log=jira --ids=zid_host=jiraserver,ze_
deployment_name=atlassian --auth=97453627rDGSDE67FDCA77BCE44 --
url=http://34.72.193.228:443
```

Using the Kubernetes Log Collector to Ingest Data

If your application to be "monitored" is Kubernetes-based, this is the preferred method for sending logs to Zebrium On Prem.

The log collector is deployed as a Kubernetes Helm chart as follows:

1. `kubectl create namespace zebrium`
2. `helm install zlog-collector zlog-collector --namespace zebrium --repo https://raw.githubusercontent.com/zebrium/ze-kubernetes-collector/master/charts --set zebrium.collectorUrl=http://<ZAPI_ENDPOINT>:443,zebrium.authToken=<ZAPI_TOKEN>,zebrium.deployment=<ZE_DEPLOYMENT_NAME>,zebrium.timezone=<KUBERNETES_HOST_TIMEZONE>`

NOTE: Remember to substitute `<ZAPI_ENDPOINT>, <ZAPI_TOKEN>, <ZE_DEPLOYMENT_NAME>` and `<KUBERNETES_HOST_TIMEZONE>` with the relevant values for your system.

Using Logstash to Ingest Data

For instructions about configuring this integration, see *Logstash Collector*.

NOTE: Please contact Zebrium Support for assistance with configuration.

Zebrium On Prem: Support

This chapter covers the following topics:

Sending Operational Data to Zebrium Support

Slack Notifications

There are two Slack webhooks that can be configured in the Helm chart. Zebrium recommends configuring these two channels in your Slack instance and inviting Zebrium to the channels:

1. ***ZE_SLACK_WEBHOOK***. This channel will receive a summary of all Root Cause reports.
2. ***ZE_SLACK_DEBUG_WEBHOOK***. This channel will receive debug alerts on the operation of the Zebrium software. It is strongly recommended that Zebrium be invited to this Slack channel.

Log Data

When you register for Zebrium On Prem, Zebrium will provide a secure authentication token that can be used to send logs from the Zebrium On Prem software to Zebrium for remote monitoring (similar to Zebrium AutoSupport).

NOTE: Installing the Zebrium Kubernetes Log Collector will send logs from all namespaces in you Kubernetes cluster.

The log collector is deployed as a Kubernetes Helm chart as follows:

1. `kubectl create namespace zebrium`

2.
   ```
   helm install zlog-collector zlog-collector --namespace zebrium --repo
   https://raw.githubusercontent.com/zebrium/ze-kubernetes-
   collector/master/charts --set
   zebrium.collectorUrl=https://zapi03.zebrium.com,zebrium.authToken=<AUTH_
   TOKEN_FROM_ZEBRIUM>,zebrium.timezone=<KUBERNETES_HOST_TIMEZONE>
   ```

Contacting Zebrium Support

Slack (preferred)

If your company uses Slack, Zebrium will create a shared Slack channel and invite members of your company and team to join.

Email

Send email to: support@zebrium.com.

Support Hours

Day	Hours	Time Zone
Monday to Friday	6:00am - 6:00pm	Pacific
Saturday and Sunday	Limited	Pacific

Support SLAs

Contact Zebrium at support@zebrium.com.

Zebrium On Prem: API

The following pages describe the endpoints and APIs that are available from Zebrium, along with example request and response payloads:

- Incident API
    - *Create Incident Type*
    - *Read Incident*
- Signal API
    - *Create Signal*
    - *Read Signal*
- Batch Upload API
    - *Begin Batch*
    - *End Batch*
    - *Cancel Batch*
    - *Get Batch*
    - *List Batch*
    - *Listing Incidents for a Batch Upload*
    - *Usage*
- Etroot Vector API
    - *Get Etroot Vector*

Create Incident Type

Use this request to set attributes of an Incident Type.

Method URL	URL created for this integration
HTTP Method	POST
Content Type	application/json

Request Arguments

Required Arguments	Data Type	How To Use	Default
itype_id	string	Incident Type ID	None

Optional Arguments	Data Type	How To Use	Default
itype_title	string	Short title of the incident as seen in the RCA list and RCA report Notes section	None
itype_desc	string	Long description of the incident as seen in the RCA report Notes section	None
itype_tracking_url	string	URL pointing to additional information for the Incident as seen in the RCA report Notes section	None

Example Request Payload

```
{
    "itype_id": "00000000-0000-0000-0000-000000000000",
    "itype_title": "This is a short title",
    "itype_desc": "This is a longer description seen when viewing the RCA
report Notes",
    "itype_tracking_url": "https://sup-
port.acme.com/kb012345/instructions.html"
}
```

Example Response Payload

```
{
  "data": [
    {
      "itype_desc": "This is a longer description seen when viewing the
RCA report Notes",
      "itype_feedback_incident": 5,
      "itype_id": "00000000-0000-0000-0000-000000000000",
      "itype_keys": "",
      "itype_outbound_integration_ids": [
        "3ca42ef0-1510-4a61-aee3-9763bf008acf",
        "8a0d216e-ccbd-4cbf-9f16-c99b6701ffd4",
        "85b92f12-97f3-43d4-7d94-5ff9784a1a92"
      ],
      "itype_owner": "",
      "itype_priority_ts": "0001-01-01T00:00:00Z",
      "itype_title": "This is a short title",
      "itype_tracking_url": "https://sup-
port.acme.com/kb012345/instructions.html",
      "modify_user_name": "Zebrium",
      "ts": "2021-09-15T15:50:16.726916Z",
      "itype_outbound_priority": "P3"
    }
  ],
  "error": {
    "code": 200,
    "message": "200 OK"
  }
  "op": "create",
  "softwareRelease": "20210915074109"
}
```

Read Incident

Use this request to get attributes of an Incident based on specified filters.

Method URL	http://<mwsd_container_url>:<mwsd_container_port>/mwsd/v1/incident/read/list
HTTP Method	POST
Content Type	application/json

Request Arguments

Required Arguments	Data Type	How To Use	Default
time_from	integer	Include Incidents created after this epoch time (use 1 as beginning of time)	None
time_to	integer	Include Incidents created before this epoch time (use 999999999999 as all time)	None
timezone	string	Time zone name for time_from - time_to specification. Typically use “UTC”	None
repeating_incidents	string	Include “first” or “all” occurrence(s) of an Incident Type	None
occurrences	string	Always specify “none”	None
time_buckets	string	Always specify “none”	None

Optional Arguments	Data Type	How To Use	Default
inci_id	string	Return the Incident with this Incident ID	None
itype_id	string	Include only incidents of this Incident Type	None
itype_id	string	Return all Incidents created as a result of a signal with this SRID. Use the SRID returned from the Signal Create API	None
batch_ids	stringSlice	Return all Incidents associated with the Transactional Batch Upload. Use the Batch ID returned from the Begin Batch API	None

Example Request Payload

```
{
  "time_from": 1,
  "time_to": 999999999999,
  "repeating_incidents": "first",
  "occurrences": "none",
  "time_buckets": "none",
  "timezone": "UTC",
  "inci_signal": "000615d0-39a0-0000-0000-00fffff00004"
}
```

Example Response Payload

```
{
  "data": [
    {
      "inci_code": "5nDZv",
      "inci_fevt_etext": "Oct  5 18:50:17 ip-172-31-62-10 kernel:
[11128469.531293] nvme nvme2: rescanning",
      "inci_fevt_gen": "1ab751d74131e92b12ea357418a53d6ed4753583",
      "inci_fevt_host": "ip-172-31-62-10",
      "inci_fevt_log": "kern",
      "inci_fevt_ts": "2021-10-06T01:50:17.487Z",
      "inci_has_signal": true,
      "inci_hosts": "ip-172-31-59-106,ip-172-31-62-10,ip-172-31-62-236,ip-
172-91-93-128",
      "inci_id": "000615d0-0d97-6e58-0000-2f9000000c89",
      "inci_itype_occ": 1,
      "inci_itype_ttl": 1,
      "inci_logs": "kern,network,vertica",
      "inci_signal": "000615d0-39a0-0000-0000-00fffff00004",
      "inci_svc_grps": "portal03,qa-blue",
      "inci_ts": "2021-10-06T01:50:17.487Z",
      "inci_wevt_etext": "Oct  5 18:50:17 ip-172-31-62-10 kernel:
[11128469.531293] Permission expired : rescanning and calculating for
brady",
      "inci_wevt_gen": "1ab751d74131e92b12ea357418a53d6ed4753583",
      "inci_wevt_host": "ip-172-31-62-10",
      "inci_wevt_log": "kern",
      "inci_wevt_ts": "2021-10-06T01:50:17.487Z",
      "itype_code": "5nDZv",
      "itype_desc": "The first log message is a warning that the per-
mission voter for user brady has expired and will be recalculated.",
      "itype_feedback_incident": 0,
      "itype_id": "000615d0-0d97-6e58-0000-2f9000000c89",
      "itype_outbound_integration_ids": [],
      "itype_owner": "",
      "itype_title": "The first log message is a warning that the
permission voter for user brady has expired and will be recalculated.",
      "itype_tracking_url": ""
    }
  ],
  "error": {
    "code": 200,
    "data": null,
    "message": ""
  },
  "op": "read",
  "softwareRelease": "release-ea58_20211005201105"
}
```

Create Signal

Use this request to enter a time around which to search for interesting events to create a Root Cause report.

Method URL	http://<mwsd_container_url>:<mwsd_port>/mwsd/v1/incident/create/signal
HTTP Method	POST
Content Type	application/json

Request Arguments

Required Arguments	Data Type	How To Use
timestamp	string	Timestamp in RFC3339

Optional Arguments	Data Type	How To Use
service_group	string	Service group to scan for creating Root Cause Report. Default is all if not specified.

Example Request Payload

```
{
  "timestamp": "2020-12-11T00:53:04.451035Z",
  "service_group": "staging"
}
```

Example Response Payload

```
{
  "data": [
    {
      "customer": "zebrium466",
      "db_schema": "zebrium466_trial",
      "service_group": "ops-blue",
      "srid": "000615d0-39a0-0000-0000-00fffff00004"
    }
  ],
  "error": {
    "code": 200,
    "data": null,
    "message": ""
  },
  "op": "create",
  "softwareRelease": "20210412141334"
}
```

Read Signal

Use this request to get the status of a signal.

Method URL	http://<mwsd_container_url>:<mwsd_port>/mwsd/v1/signal/read
HTTP Method	POST
Content Type	application/json

Request Arguments

Required Arguments	Data Type	How To Use
filter	list	List of SRID filter strings of the format "srid=<SRID>" where SRID is the srid returned from incident/create/signal API call.

Optional Arguments	Data Type	How To Use
None		

Example Request Payload

```
{
  "filter": [ "srid=000615d0-39a0-0000-0000-00fffff00004" ]
}
```

Example Response Payload

- `bake_ct`. Number of bakes that have run since signal created.
- `expired`. Set to *True* after three bakes have run.
- `created_incident`. Set to *True* if an incident was created as a result of this signal request
- Use the **incident**/**read**/**list** API with the `inci_signal` filter set to the *`<SRID>`* to get the list of incidents created as result of this signal request.

```
{
  "data": [
    {
      "bake_ct": 1,
      "create_time": "2021-10-06T04:21:32.472795Z",
      "created_incident": true,
      "epoch": 1633485722000,
      "event_type": "zebrium_incident",
      "expired": false,
      "integration": "zebrium",
      "local_time": "Tue Oct  5 19:02:02 PDT 2021",
      "modify_time": "2021-10-06T04:22:35.043531Z",
      "payload_data": "{\"zebrium\": {\"epoch_ts\": \"2021-10-
06T02:02:02Z\", \"epoch_msec\": 1633485722000, \"epoch_local\": \"Tue Oct
5 19:02:02 PDT 2021\", \"deployment\": \"trial\", \"service_group\":
\"ops-blue\"}, \"slack\": null}",
      "service_group": "ops-blue",
      "siid": "765afdaa-e85b-43e5-baa4-16214de10296",
      "srid": "000615d0-39a0-0000-0000-00fffff00004",
      "ssid": "a3593b9e-7858-429e-981e-2d9e50dab43a",
      "ts": "2021-10-06T02:02:02Z"
    }
  ],
  "error": {
    "code": 200,
    "data": null,
    "message": ""
  },
  "op": "create",
  "softwareRelease": "20210412141334"
}
```

Begin Batch

The Begin Batch API is called to begin a new batch upload. It is called as the first step in performing a batch upload. See the ***Usage*** page for more information on batch uploads.

Method URL	POST http://<zapi_url>:<zapi_port>/api/v2/batch/
HTTP Method	POST
Content Type	application/json
Required Headers	Authorization (set to ZAPI token)

Request Arguments

Optional Arguments	Data Type	How To Use
processing_method	string	Set to **delay** or **opportunistic**
retention_hours	retention_hours	Minimum time to retain *batch status* after processing completes, in hours.
batch_id	string	Optional user specified batch Id. Must be unique.

Response Payload

Optional Arguments	Data Type	How To Use
batch_id	string	Upload id, use as ze_batch_id.

Example Request Payload

```
{
  "processing_method": "delay",
  "retention_hours" : 8
}
```

Example Response Payload

```
{
     "batch_id": "b1cc71aef9989ead80012ac"
}
```

End Batch

The End Batch API should be used when all files have been uploaded to ZAP for a batch upload. On success the batch upload will move into the Processing state.

Method URL	PUT http://<zapi_url>:<zapi_port>/api/v2/batch/<batch_id>
HTTP Method	PUT
Content Type	application/json
Required Headers	Authorization (set to ZAPI token)

Request Arguments

Arguments	Data Type	How To Use
uploads_complete	bool	Set to **true**

Response Payload

Optional Arguments	Data Type	How To Use
batch_id	string	
state	state	The new state

Example Request Payload

```
{
   "uploads_complete" : true
}
```

Example Response Payload

```
{
     "batch_id" : "b1cc71aef9989ead80012ac",
     "state" : "Processing"
}
```

Cancel Batch

The Cancel Batch API is called while uploading files to cancel a batch upload. Note that in some cases it may not be possible to cancel a batch upload. Use the returned **state** to check the new batch state.

Method URL	PUT http://<zapi_url>:<zapi_port>/api/v2/batch/<batch_id>
HTTP Method	PUT
Content Type	application/json
Required Headers	Authorization (set to ZAPI token)

Request Arguments

Arguments	Data Type	How To Use
cancel	bool	Set to **true**

Response Payload

Optional Arguments	Data Type	How To Use
batch_id	string	
state	state	The new state

Example Request Payload

```
{
   "cancel" : true
}
```

Example Response Payload

```
{
     "batch_id" : "b1cc71aef9989ead80012ac",
     "state" : "Cancelled"
}
```

Get Batch

The Get Batch API is called to get the status of a batch upload.

Method URL	GET http://<zapi_url>:<zapi_port>/api/v2/batch/<batch_id>
HTTP Method	GET

Content Type	application/json
Required Headers	Authorization (set to ZAPI token)

Example Response Payload

```
{
  "batch_id": "b1cc71aef9989ead80012ac",
  "state": "Done",
  "lines": 22000,
  "bundles": 3,
  "bundles_completed": 3,
  "created": "2022-10-12T07:20:50",
  "upload_time_secs": 250,
  "processing_time_secs": 45,
  "processing_method": "delay",
  "completion_time" : "2022-10-12T0755:17",
  "retention_hours" : 8,
  "expiration_time": "2022-10-12T15:55:17",
  "reason":""
  }
```

List Batches

The List Batches API is called to list current batch uploads. See the ***get_batch*** API to list a specific batch.

Method URL	GET http://<zapi_url>:<zapi_port>/api/v2/batch
HTTP Method	GET
Content Type	application/json
Required Headers	Authorization (set to ZAPI token)

HTTP Method GET

```
"batches" : [{
  "batch_id": "b1cc71aef9989ead80012ac",
  "state": "Done",
  "lines": 22000,
  "bundles": 3,
  "bundles_completed": 3,
  "created": "2022-10-12T07:20:50",
  "upload_time_secs": 250,
  "processing_time_secs": 45,
  "processing_method": "delay",
  "completion_time" : "2022-10-12T0755:17",
  "retention_hours" : 8,
  "expiration_time": "2022-10-12T15:55:17",
  "reason" :""
  },
  {
  "batch_id": "b2ef71aef9226ead80012ac",
  "state": "Uploading",
  "lines": 0,
  "bundles": 0,
  "bundles_bundles": 0,
  "created": "2022-10-14T08:23:34",
  "upload_time_secs": 10,
  "processing_time_secs": 0,
  "processing_method": "delay"
  "completion_time" : "",
  "retention_hours" : 8,
  "expiration_time": "2022-10-14T16:23:34",
  "reason" :""
  }
  ]
```

Listing Incidents for Batch Uploads

Listing Incidents for Batch Uploads

Incidents associated with batch uploads may be queried using the existing **read incident** and **find incident** *MWSD APIs*. The optional query parameter **batch_ids** will return only incidents for the specified batch ID(s).

For example, to query the incidents for batch id **bazo3aabb123ff**, the query would have the optional parameter:

```
...
"batch_ids" : [ "bazo3aabb123ff"],
...
```

If this batch had three incidents, then these would be reported as (using the **find** API):

```
{
  "data": [
    {
      "inci_code": "OdmrA",
      "inci_id": "0006266a-f550-0000-0000-01700000376e",
      "inci_ts": "2022-04-25T14:25:25Z",
      "itype_id": "ae64766d-36dd-419b-e62a-826675ec4a0d"
    },
    {
      "inci_code": "fI7GX",
      "inci_id": "0006266a-f550-0000-0000-1d800000ca3a",
      "inci_ts": "2022-04-25T14:25:25Z",
      "itype_id": "a1599bb9-ed24-d2ea-6a50-f624508a7423"
    },
    {
      "inci_code": "8ho3P",
      "inci_id": "0006266a-f550-0000-0000-84000004e23d",
      "inci_ts": "2022-04-25T14:25:25Z",
      "itype_id": "107d4f27-96d8-41ac-3528-9269bbe670da"
    }
  ],
  "error": {
    "code": 200,
    "data": null,
    "message": ""
  },
  "op": "read",
  "softwareRelease": "release-ea72_20220425101422"
}
```

Usage

The Batch Upload API allows a set of related logs to be grouped together when uploading to Zebrium. When compared to single file uploads, or the upload-status APIs, batch uplo***begin_batch.html***ads provide a more controlled and organized way to send groups of information to Zebrium. There can be multiple batch uploads concurrently underway.

The operational flow for batch uploads is:

1. Make an API call to Zebrium to begin a batch (begin_batch). A unique batch ID is returned on success that is used in subsequent steps while working with a batch. This API call creates the required Zebrium state for a batch and must be the first operation for each new batch.

2. The logs associated with a batch are uploaded, such as using **ze** or curl. These use the configuration variable **ze_batch_id** to notify Zebrium that the logs are part of a batch. This must be set to the **batch_id** used in step 1.
3. When all files have been uploaded make another API call to Zebrium to end the batch upload phase (***end_batch***). This tells Zeberium that all files for a batch are uploaded and processing can begin on the batch.
4. Check the state of a batch periodically (using the ***get_batch*** API) until processing has completed.

See the ***Example*** below for more information.

Additional operations that can be performed are:

- ***List Batches*** and their states.
- ***Get*** batch metrics.
- ***Cancel*** a non-finished batch.
- ***List incidents*** associated with a batch.

Batch IDs and Scope of Batches

Each batch upload is identified by a unique string, the batch ID. This is defined when the ***begin_batch*** API is called, and is valid for the lifetime of the batch upload.

Zebrium automaticaly returns a new batch ID from the ***begin_batch*** API by default. Alternatively, a user-defined batch ID may be supplied on the ***begin_batch*** API call. However, note that this cannot be reused until the batch has expired and been removed. Batch IDs are formed using 1-36 alphanumeric characters, plus '_' (underscore) and '-' (dash).

Batch ids are used as part of ZAPI uploads, along with a ZAPI token. They are associated with that ZAPI token at creation time, and may only be used with the same token in later upload calls.

The lifetime of a batch, or retention period, is set in hours. By default this is 8 hours. This can be overridden in the begin_batch API if desired. The retention period is used to extend the lifetime as a batch successfully proceeds through each state.

Batch States

Each batch upload exists in one of the following states:

State	Interpretation
Uploading	Files are being uploaded to the batch (step 1, 2 above)
Processing	All files have completed upload and are being processed. (triggered by step 3 above)
Done	Ingest and bake has completed on all uploads
Failed	The batch could not be uploaded and/or processed
Cancelled	The batch was cancelled by the user prior to step 3

Opportunistic or Delayed Batch Processing

When starting a new batch the API (step 1) allows the user to specify how to stage and process the batch, either delayed or opportunistic. The default is delay.

In both cases uploaded files for a batch are processed together in one or more bundles, with no other logs included in the bundles.

Type	Interpretation
Opportunistic	Zebrium may start processing uploaded files before the final commit (step 3). This can reduce the amount of temporary space needed for a batch, and spreads work out over a longer time.
Delayed	Zebrium will delay processing uploaded files until the final commit (step 3) occurs. This guarantees the batch is processed as a unit, although it may consume more temporary space and cause a burst of work when the batch ends.

If batches are typically small then using delay is appropriate. If batches are very large then using opportunistic may be appropriate.

Example

This example uses Curl to get a batch ID, uses the ze CLI to upload several files with the same batch ID, then uses Curl to advise Zebrium that all data for the upload has been sent. Finally, a check is made whether or not all the data in the upload has been processed.

Begin batch, get a batch ID:

```
curl --silent --insecure -H "Authorization: Token <authToken> " -H
"Content-Type: application/json" -X POST https://<ZapiHost>/api/v2/batch
```

```
BATCH_ID=<newBatchId>
```

Upload logs using **ze** CLI

```
ze up --url=https://mysite.example.com --auth=<authToken>--
file=syslog.syslog.log --log=syslog --ids=ze_deployment_name=case1 --
cfgs=ze_batch_id=$BATCH_ID
```

```
ze up --url=https://mysite.example.com --auth=<authToken> --
file=jira.jira.log --log=jira --ids=ze_deployment_name=case1 --cfgs=ze_
batch_id=$BATCH_ID
```

```
ze up --url=https://mysite.example.com --auth=<authToken> --
file=conflnc.conflnc.log --log=conflnc --ids=ze_deployment_name=case1 --
cfgs=ze_batch_id=$BATCH_ID
```

Indicate end of uploads:

```
curl --silent --insecure -H "Authorization: Token <authToken" -H "Content-
Type: application/json" -X PUT --data '{ "uploads_complete" : true }'
https://<zapi_host>/api/v2/batch/$BATCH_ID
```

Check the status of uploads is complete via the state that is returned in the response payload:

```
curl --silent --insecure -H "Authorization: Token <authToken" -H "Content-
Type: application/json" https://<zapi_host>/api/v2/batch/$BATCH_ID | grep
state
```

When the state becomes **Done**, the batch is successfully processed. While processing is underway other information from the get_batch API can be used to monitor progress, for example the number of bundles created for the batch, and completed so far:

```
...

"bundles": 8,

"bundles_completed": 3,

...
```

Note on Canceled and Failed Batches

A batch can be canceled while still performing uploads using the ***cancel_batch*** API. This causes the batch to transition to the Canceled state. Any uploaded files staged on Zebrium will be removed.

If a batch fails processing it transitions to the Failed state. The reason for the failure, if known, is available in the reason attribute. For example:

```
"state": "Failed",

...

"reason": "write bundle files failed"
```

would indicate insufficient temporary storage to process the batch.

Get Etroot Vector

Method URL	POST http://<mwsd_container_url>:<mwsd_container_port>/mwsd/v1/incidentevent/read/etroots
HTTP Method	POST
Content Type	application/json

Request Arguments

Required Arguments	Data Type	How To Use
inci_id	string	Incident ID to get etroot vector
ievt_level	integer	0 (first/worst), 1 (0 + other root cause events), 2 (0 + 1 + other core events)

Optional Arguments	Data Type	How To Use
None		

Example Request Payload

```
{
  "inci_id": "00061279-e560-0000-0000-013000000895",
  "ievt_level": 2
}
```

Example Response Payload

```
{
  "data": [
    {
      "ievt_etroot": "received_sigterm_from_systemd"
    },
    {
      "ievt_etroot": "read_domainname_sysconfig_network"
    },
    {
      "ievt_etroot": "mounted_message_queue_system"
    },
    {
      "ievt_etroot": "remount_root_kernel_systems"
    },
    {
      "ievt_etroot": "http_named_cookie_not_present"
    }
  ],
  "error": {
     "code": 200,
     "data": null,
     "message": ""
  },
  "op": "read",
  "softwareRelease": "20210903100945"
}
```

ScienceLogic

800-SCI-LOGIC (1-800-724-5644)

International: +1-703-354-1010