Zebrium Root Cause as a Service (RCaaS) Documentation

Release EA-90

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.

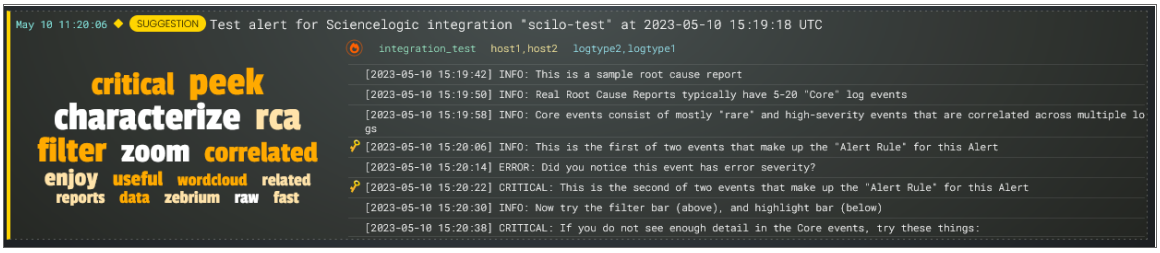
This chapter covers the following topics:

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 along with the existing alerts:



On the **Alerts** page, the summary report for a suggestion and an alert contains the following main elements:

- ***AI-generated title***. Displaying at the top of the summary pane, this title is generated using GPT Services that use new Generative AI models. You can enable or disable GPT services for a specific deployment of Zebrium by using the **GPT Services** column on the **Deployments** page (Settings ( )> Deployments).
- ***Word Cloud***. A set of relevant words chosen by the AI/ML engine from the log lines contained in the alert. Click a word in the cloud to highlight that word in the list of logs on the left.
- ***Significance icon***. Since not all suggestions that the AI/ML engine generates will relate to problems that actually impact users, the engine attempts to reason over the data and assess whether a problem actually requires attention. Hover over this icon at the top of the list of logs to view the confidence level of the AI/ML engine for this suggestion. A red icon ( ) means "High" confidence, and a yellow icon ( ) means "Medium" confidence.
- ***AI Assessment*** . Since not all suggestions that the AI/ML engine generates will relate to problems that actually impact users, the AI/ML engine attempts to reason over the data and assess whether a problem actually requires attention. Depending on the quality of the data, some suggestions might not include an AI Assessment. This value is shown in the Zebrium user interface as an **AI Assessment** value of one of the following:
    - "No Attention Needed" for content that the AI/ML engine assesses as unlikely to require immediate attention.
    - "Needs Your Attention" for content that the AI/ML engine believes should be looked into.
- ***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 Root Cause 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***.

You can click anywhere in the summary report for a suggestion or an alert to view a more detailed **Root Cause Report** page for that suggestion or alert. For more information, see *Root Cause Reports*.

> 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 time frame 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*.

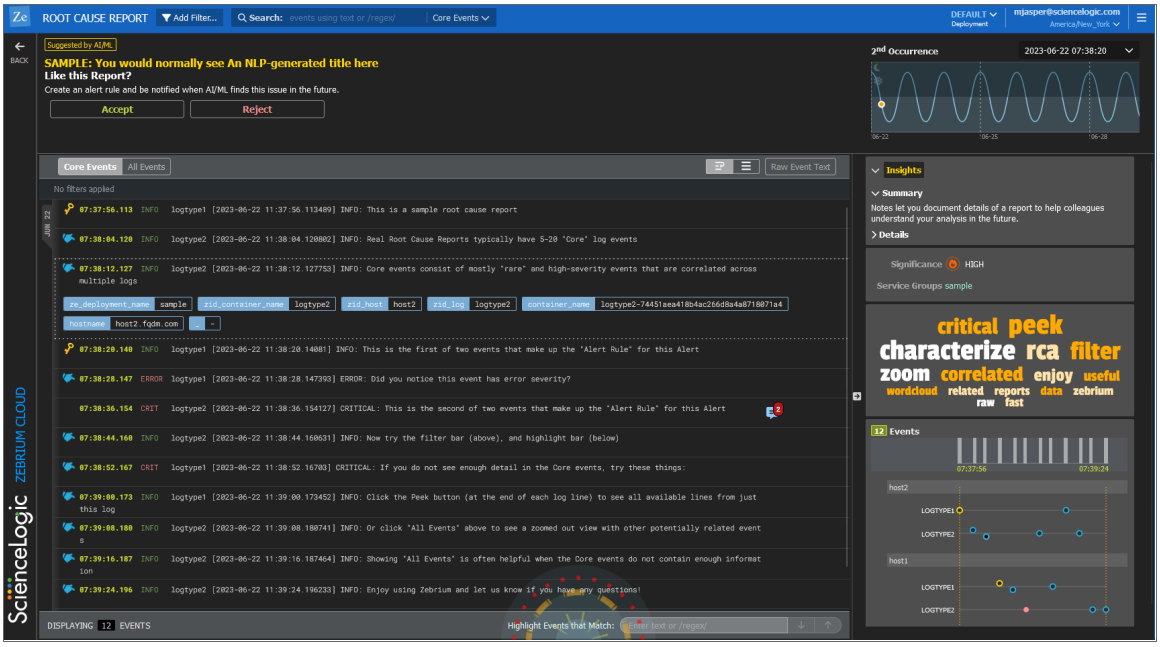
You can also decide on the action to take if the same kind of alert type occurs again, such as sending a notification to Slack, email, or another type of notification. For more information, see *Notification Channels*.

If you currently use SL1 from ScienceLogic, you can configure an integration that lets you view Zebrium suggestions in SL1 dashboards as well as on the SL1 **Events** page. For more information, see *ScienceLogic 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:



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.

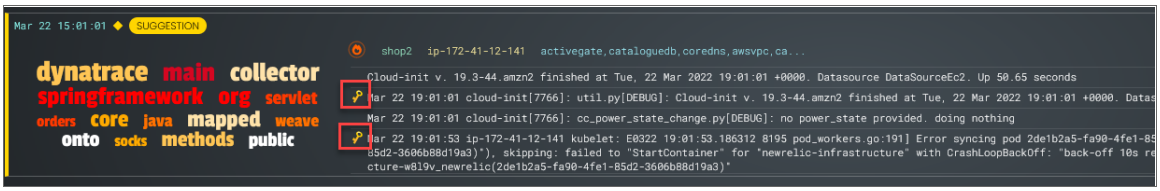
Each RCA report matches a particular "fingerprint" of log events. You can add notes, summaries, Jira links, and alert preferences to the alert rules for the RCA report so that future occurrences of the same type of problem will reflect these preferences and notes.

For more information, see *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:



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:

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

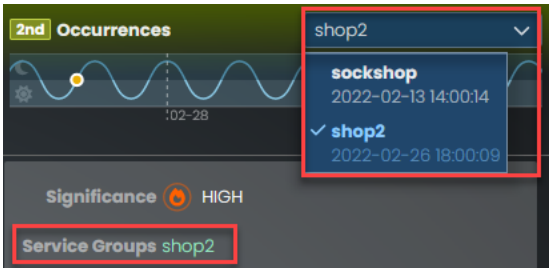
For more information, see *Log Collectors and File Uploads*.

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:



TIP: You can view a list of service groups by clicking the **[Filtering]** button on the **Alerts** page. The **Selected Filter** dialog contains a list of service groups in the **Service Groups** filter.

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
```

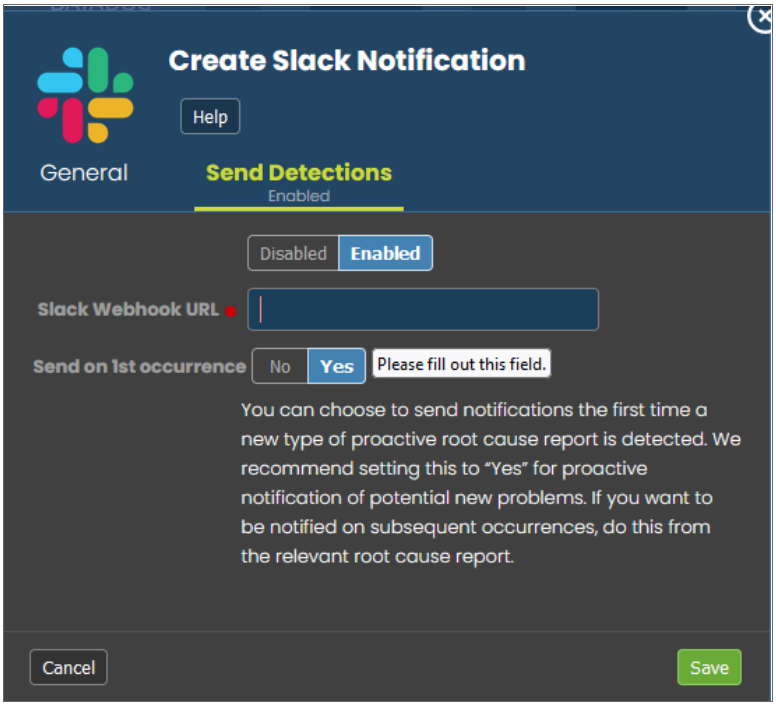
```
"app02" log events and default (i.e. database logs) but not across:
```

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

For more information, see *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 Slack, Microsoft Teams, and Webex Teams notifications.



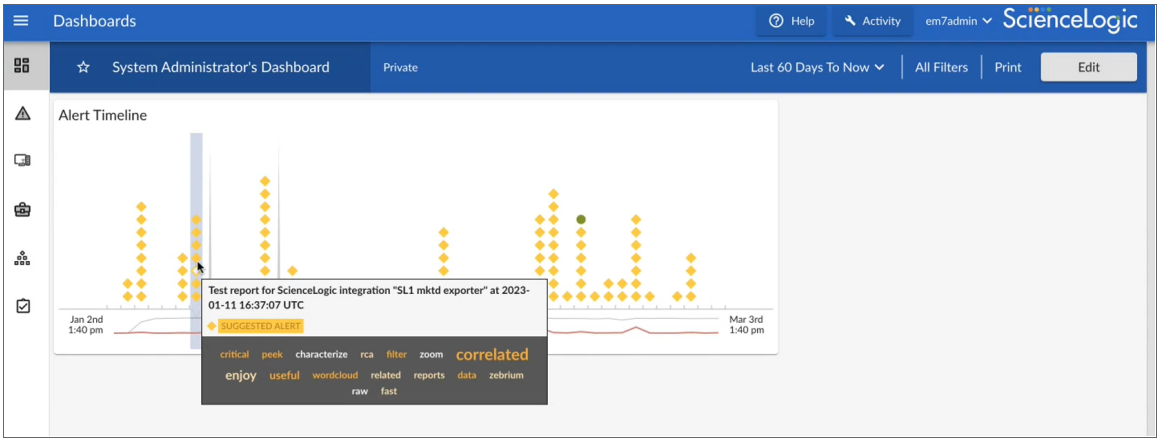
After you have created one or more notification channels, you can link any number of these to any RCA report created by the AI/ML engine. Linking a set of notification channels to a RCA report will send notifications of future RCA reports of the same type to those channels.

For more information, see *Notification Channels*.

ScienceLogic Integrations

You can integrate the Zebrium Root Cause service with the SL1 platform from ScienceLogic to send suggestions and alerts to the SL1 dashboards or to SL1 **Events**, **Devices**, and **Services** pages.

The following image shows the interactive Root Cause Timeline widget in an SL1 dashboard:



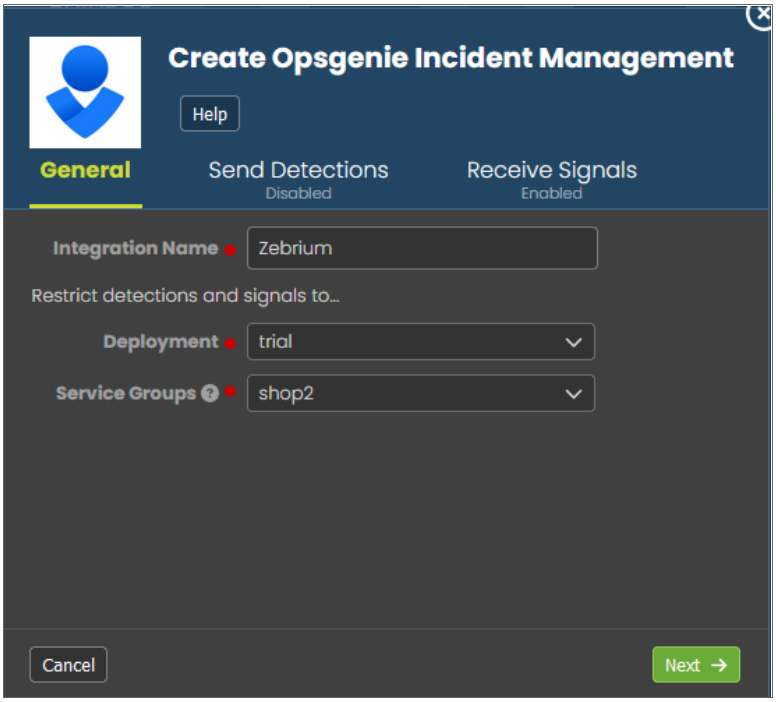
To enable a ScienceLogic integration, go to the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors), select an integration type, and follow the instructions for setting up that dashboard.

For more information, see *ScienceLogic 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.

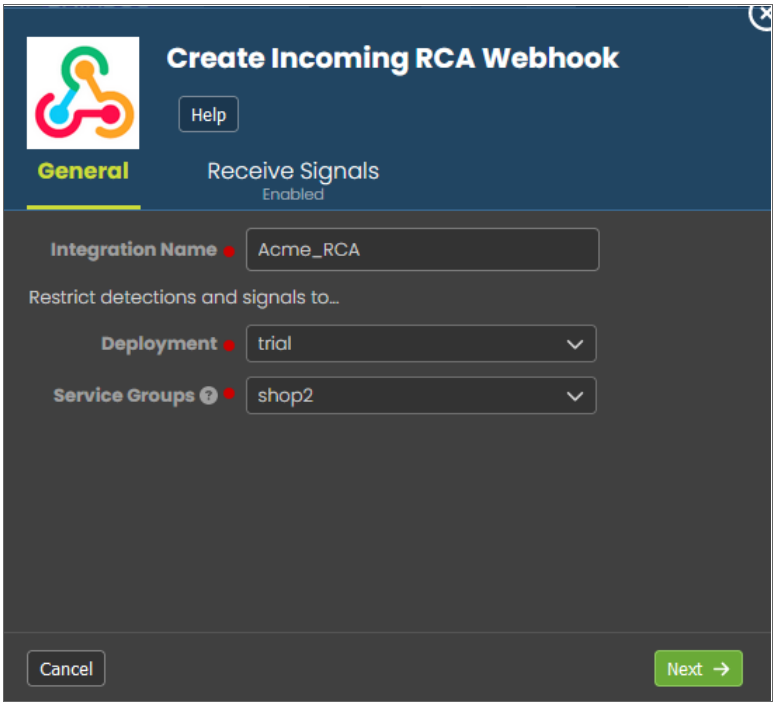
After you complete the configuration, you can can view details of root cause and direct the incident to the appropriate team. All of these features lead to faster Mean Time to Repair (MTTR) and less time manually hunting for root cause.



For more information, see *Incident Management Integrations*.

Integrations Using Webhooks

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

For more information, see *Using Webhooks to Create Integrations* .

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

Overview

This chapter provides an overview of how Zebrium works, and how to get started using Zebrium.

> IMPORTANT: Before you can start watching for suggestions and reviewing Root Cause reports, you will need to configure a method for gathering log data to send to Zebrium. For more information, see *Log Collectors and File Uploads*.

This chapter covers the following topics:

How Zebrium Works

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

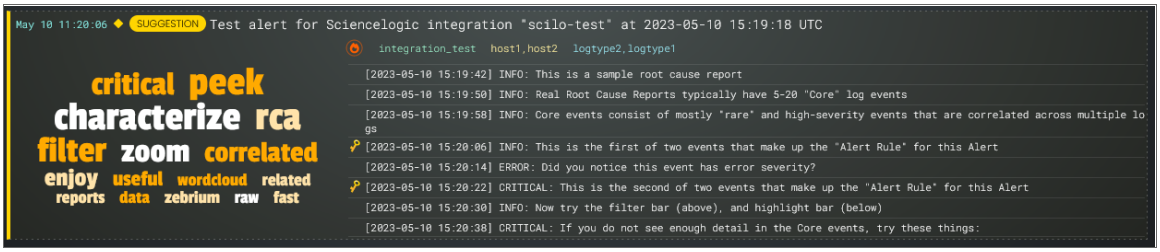
1. **Where are the problems or events occurring?** The events could be clusters of errors, warnings, stack traces, or other indicators of bad outcomes.
2. **Were there unusual events upstream that could help explain these bad outcomes?** This might be configuration changes, a new deployment, user actions,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



On the **Alerts** page, the summary report for a suggestion and an alert contains the following main elements:

- ***AI-generated title***. Displaying at the top of the summary pane, this title is generated using GPT Services that use new Generative AI models. You can enable or disable GPT services for a specific deployment of Zebrium by using the **GPT Services** column on the **Deployments** page (Settings ( )> Deployments).
- ***Word Cloud***. A set of relevant words chosen by the AI/ML engine from the log lines contained in the alert. Click a word in the cloud to highlight that word in the list of logs on the left.

- ***Significance icon***. Since not all suggestions that the AI/ML engine generates will relate to problems that actually impact users, the engine attempts to reason over the data and assess whether a problem actually requires attention. Hover over this icon at the top of the list of logs to view the confidence level of the AI/ML engine for this suggestion. A red icon () means "High" confidence, and a yellow icon () means "Medium" confidence.
- ***AI Assessment*** . Since not all suggestions that the AI/ML engine generates will relate to problems that actually impact users, the AI/ML engine attempts to reason over the data and assess whether a problem actually requires attention. Depending on the quality of the data, some suggestions might not include an AI Assessment. This value is shown in the Zebrium user interface as an **AI Assessment** value of one of the following:
    - "No Attention Needed" for content that the AI/ML engine assesses as unlikely to require immediate attention.
    - "Needs Your Attention" for content that the AI/ML engine believes should be looked into.
- ***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 Root Cause 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***.

You can click anywhere in the summary report for a suggestion or an alert to view a more detailed **Root Cause Report** page for that suggestion or alert. For more information, see *Root Cause Reports*.

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 time frame 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 *Notification Channels*.

If you currently use SL1 from ScienceLogic, you can configure an integration that lets you view Zebrium suggestions in SL1 dashboards as well as on the SL1 **Events** page. For more information, see *ScienceLogic 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 a ScienceLogic integration, such as the **SL1 Enhanced (12.x)** integration on the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors). After you configure the integration, data from the Root Cause reports from Zebrium will display in SL1 and you can correlate the reports with any spikes or alerts occurring at the same time. For more information, see *ScienceLogic Integrations*.

   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, or Slack. After you configure the incident management tool, an RCA report is automatically created and sent back to the incident management tool.

3. Evaluate the feed of auto-detected 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 ( ). Any Root Cause reports generated by that scan include a lightning bolt icon and the text "Result of RC Scan". For more information, see *Working with Suggestions and Root Cause Reports*.

Customizing Your ZebriumResults

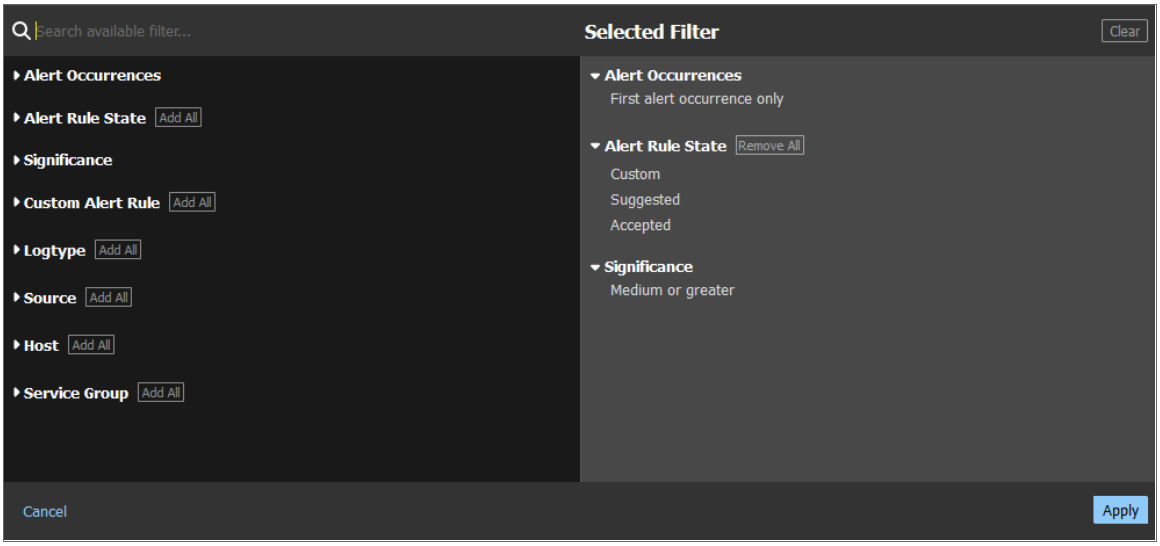
You can customize your Zebrium results on the **Alerts** page (the Zebriumhome page) by selecting one or more filters at the top of the page. You can use these filters to manage the number of suggestions and alerts that display on the **Alerts** page.

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.

You can also filter the list of suggestions by ***Significance***: the AI/ML engine assigns a value of Low, Medium, or High to each alert. Significance is a cumulative score for each suggestion, based on the rareness and "badness" (log severity level) of the log events within that alert. If you have a high Significance setting, the Root Cause events will have to be more rare and more "bad" to show up in the list of suggestions.

By default, only suggestions 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 edit the default Significance setting by editing the **Root Cause Significance** setting on the **Report Settings** page (Settings ( ) > Root Cause Settings.

These filters appear on the **Selected Filter** dialog, which displays when you click the **[Filtering]** button (Filtering...) on the **Alerts** page:



There is also a ***Search*** bar at the top of the **Alerts** page that you can use 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*.

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/QwlbihOOW5k.

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 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

Log Collectors and File Uploads

Overview

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

You can configure one or more ***log collectors*** to gather logs and send those logs to Zebrium for automated incident detection. You can also use a ***file upload*** method using the **ze** tool, the Zebrium command-line interface for uploading log events from files or streams.

The following pages explain how you can collect data from different sources, as well as file uploads:

- *AWS CloudWatch*
- *Azure Monitor OTel*
- *Docker (including ECS)*
- *File Upload (ze Command)*
- *Kubernetes*
- *Linux*
- *Logstash*
- *Syslog Forwarder*
- *VMware vSphere*
- *Windows OTel*

AWS CloudWatch Collector (Beta)

Legal

The AWS CloudWatch collector is provided by ScienceLogic with the following terms:

You may use, modify, reproduce, and distribute this freely and without restriction, provided as a condition of our provision to use the software you acknowledge that the software is provided as-is, and ScienceLogic will not have any monetary liability in association with the distribution of this software.

Overview

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.

NOTE: This feature is currently Beta. For access to this collector, contact Zebrium at support@zebrium.com.

Preparation

1. Download the 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. If not, you will get the following 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 **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.

Azure Monitor OTel Collector (Beta)

Legal

The Azure OTel collector is provided by ScienceLogic with the following terms:

You may use, modify, reproduce, and distribute this freely and without restriction, provided as a condition of our provision to use the software you acknowledge that the software is provided as-is, and ScienceLogic will not have any monetary liability in association with the distribution of this software.

NOTE: Additional information is coming soon for this collector.

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. The collector uses the Fluentd logging driver for Docker and the Zebrium Fluentd output plugin.

The GitHub repository for the collector is located at https://github.com/zebrium/ze-docker-log-collector.

Getting Started

When sending your logs from your docker daemon to Zebrium, there are two configuration options for where your log collector can be installed and configured. The collector can be installed within the docker daemon context that you are sending all the logs from, or it could be installed on an external host, and route the logs to it by each docker daemon.

Deploying the Collector

Regardless on the installation method, you will start the collector using the following command, substituting the token and URL in for the values found in your Zebrium **Integration and Collectors** page.

```
docker run -p 24224:24224 -e ZE_LOG_COLLECTOR_URL=<URL> -e ZE_LOG_
COLLECTOR_TOKEN=<TOKEN> --restart always zebrium/docker-log-col-
lector:latest
```

Additional ***environment variables*** can be specified to the collector to further extend the functionality.

Configuring the Docker Daemon

After the collector has been deployed and configured, modify the docker daemon configuration to start sending logs to the collector. For a complete list of configuration options, see the Docker documentation.

The docker daemon is located in **/etc/docker/daemon.json** on the Linux host and in **C:\ProgramData\docker\config\daemon.json** on the Windows host. For more about the docker daemon.json, see the Docker documentation.

Add the following configuration to your **daemon.json** file, substituting `<fluentd-address>` for the address of your log collector. If your log collector is deployed in the same docker daemon, then use 127.0.0.1:24224 as your address.

```
{
"log-driver": "fluentd",
  "log-opts": {
    "fluentd-address": "<fluentd-address>",
    "fluentd-async": "true"
  }
}
```

After the daemon file is updated, restart the docker daemon for the new changes to take effect. After this, you should be able to view the logs of the log collector and verify that it is receiving and forwarding logs to Zebrium.

Environment Variables

Below is a list of environment variables that are available for configuration of the Fluentd container:

Environment Variables	Default	Description	Required?
ZE_LOG_COLLECTOR_URL	""	Zebrium URL Endpoint for log ingestion.	Yes
ZE_LOG_COLLECTOR_TOKEN	""	Zebrium ZAPI token for log ingestion.	Yes
ZE_DEPLOYMENT_NAME	"default"	Zebrium *Service Group* Name.	No
FLUSH_INTERVAL	"60s"	Buffer Flush Interval.	No
ZE_LOG_LEVEL	"info"	Sets the log level for the output plugin.	No
VERIFY_SSL	"true"	Enables or disables SSL verification on endpoint.	No

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.

File Upload (ze Tool)

The **ze** tool is the Zebrium command-line interface for uploading log events from files or streams. For more information, see https://github.com/zebrium/ze-cli.

IMPORTANT: The **ze** tool was recently updated to version 2.0.0, and these documents refer to that version.

Getting Started

Prerequisites

- A collector token, which you can find by clicking the **[Other]** button under **Log Collectors** on the **Integrations & Collectors** page (Settings ( ) Integrations & Collectors) > in the Zebrium user interface.
- The URL to your instance of Zebrium, which you can also find by clicking the **[Other]** button under **Log Collectors** on the **Integrations & Collectors** page (Settings ( ) Integrations & Collectors) > in the Zebrium user interface.

Installing ze

1. Download the corresponding release from the ze-cli GitHub **Releases** page: https://github.com/zebrium/ze-cli/releases.
2. Set up your path in your shell config to include the new binary.
3. Start a new terminal and test your installation by running the following command:

```
ze -v
```

IMPORTANT: Before you start uploading log files with the **ze** tool, you will need to set ***Enable Historic Incident Detection*** to *Yes* in the Zebrium user interface. If you do not enable this setting, Zebrium cannot create Root Cause reports for logs that are older than a few hours.

1. In the Zebrium user interface, go to the **Root Cause Settings** page (Settings ( ) > Root Cause Settings).
2. Set ***Enable Historic Incident Detection*** to *Yes*, and then click **[Apply]**. An **Apply Change** dialog appears.
3. Click **[OK]**. Historic incident detection is enabled.

Configuration

The **ze** tool supports a variety of ways to set its parameters. You can set all parameters using arguments. To find out the arguments that are available and required for each call, use `ze -help` or `ze <subcommand> -`

`help`.

When leveraging the configuration file or environment variables, the **ze** tool uses the following precedence: configuration file > environment files > command-line arguments.

Configuration File

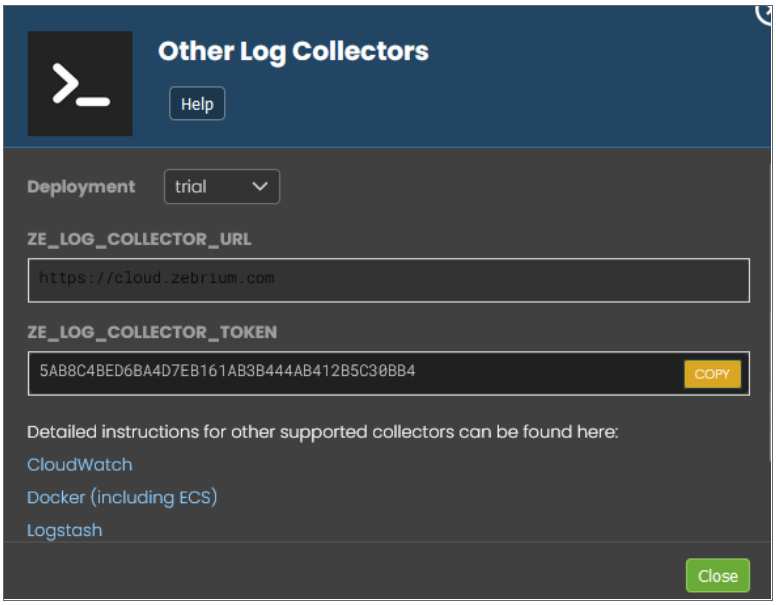
The **ze** tool supports setting global variables in a **.ze** file for easy configuration. The default location of this is **$HOME/.ze**; however, you can override this by passing a new path with the `--config` option.

The contents of the **.ze** file are as follows:

```
url: <ze_log_collector_url>
```

```
ze_token: <ze_log_collector_token>
```

where *<ze_log_collector_token>* is the collector token, which you can find by clicking the **[Other]** button under **Log Collectors** on the **Integrations & Collectors** page (Settings ( ) Integrations & Collectors) > in the Zebrium user interface:



Environment Variables

The **ze** tool supports setting the following environment variables:

```
ZE_AUTH: XXXXXXXXXXXX
```

```
ZE_URL: https://cloud.zebrium.com
```

Commands and Help

Run the following command to upload log event data to your Zebrium instance from a file or stream (STDIN) with appropriate metadata:

```
ze up
```

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

```
ze up -help
```

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

```
ze -help
```

Examples

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

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

Batch Uploads

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

Migrating from the Perl-based ze Tool (version 1.0.0)

The previous Perl-based application, version 1.0.0 of the **ze** tool, can be found at https://github.com/zebrium/ze-cli/tree/master/bin.

Replacing the .zerc File

Starting with version 2.0.0 of the **ze** tool, the **.zerc** file was replaced with a **.ze** file that accepts the configuration in **yaml**. This is described in the version 1.0.0 **Configuration** section: https://github.com/zebrium/ze-cli#configuration-file.

As a result, the configurations that were specified as:

```
url=<ZE_LOG_COLLECTOR_URL>
```

```
auth=<ZE_LOG_COLLECTOR_TOKEN>
```

will now need to be listed as the following:

```
url: <ZE_LOG_COLLECTOR_URL>
```

```
auth: <ZE_LOG_COLLECTOR_TOKEN>
```

Environment Variables

The **ze** tool now supports setting the following environment variables:

```
ZE_URL = <ZE_LOG_COLLECTOR_URL>
```

```
ZE_AUTH = <ZE_LOG_COLLECTOR_TOKEN>
```

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 that is used to identify the batch.

Batch Uploads vs Service Groups

Batch uploads are different from service groups in the following ways:

- **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.

Integrating Batch Uploads into the ze Tool

Batch uploads are integrated into the **ze** tool in the following main ways:

- A standalone upload, using `ze up` , automatically has a batch created for it.
- The batch ID is displayed when the upload is finished, so progress can be monitored using `ze batch state` and `ze batch show` , which are described below.
- A set of related uploads, using `ze up` , can be associated with a specific batch ID that has been created earlier using the `ze batch begin`.
- When all the logs for the batch are uploaded, the batch should be completed using `ze batch end`,
- 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.

ze batch Subcommand

The `ze batch` subcommand allows batch uploads to be created, completed, canceled, and monitored. The subcommand uses the following syntax:

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

ze batch end [--url=<url>] [--auth=<auth>] --batchId=<batchId>

ze batch cancel [--url=<url>] [--auth=<auth>] --batchId=<batchId>

ze batch state [--url=<url>] [--auth=<auth>] --batchId=<batchId>

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

TIP: Adding `-h` at the end of any of these commands will run the help menus.

Examples

Uploading a Large Log and Monitoring its Progress

When you successfully upload a log file, Zebrium displays a new batch ID, usually with a Processing state, which means that the log was accepted by the AI/ML engine and is being scanned for incidents:

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

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

```
Sent successfully
```

To monitor the batch until processing completes:

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

When the batch upload is completed, the state changes, typically to *Done*. For additional information, use the `ze batch show` option:

```
ze batch show ... --batchId=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.

To begin a new batch:

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

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

To upload several logs as part of the same batch, using the `--batchId` option:

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

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

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

To end the batch:

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

You can monitor the batch upload as in the previous example by using `ze batch state` and `ze batch show`.

Kubernetes Collector

The ***zlog-collector*** is the Zebrium log collector for Kubernetes.

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, configurations, and tags) and then including them in the Zebrium log data. You can enable this by providing a JSON mapping file to the log collector, as described in the repository at https://www.github.com/zebrium/ze-fluentd-plugin.

To use this functionality with the supplied Helm chart, you will need to complete a **customValues.yaml** file and supply that file to the Helm install command line with the following command:

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

A prototype **example_logPathMappings.yaml** file is provided in the repository 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

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

To use this functionality with the supplied Helm chart, complete a **customValues.yaml** file and supply that file 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	`{}`	
daemonset.priorityClassName	string	`""`	
daemonset.tolerateAllTaints	bool	`true`	
daemonset.tolerations	list	`[]`	set 'daemonset.tolerations[0].operator=Equal,daemonset.tolerations

Key	Type	Default	Description
			[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`	
zebrium.bufferRetryTimeout	string	`"1h"`	
zebrium.bufferRetryWait	string	`"10s"`	
zebrium.bufferTotalLimitSize	string	`"64GB"`	

Key	Type	Default	Description
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	string	`"/var/log/*.log,/var/log/syslog,/var/log/messages,/var/log/secure"`	
zebrium.pathMapFile	string	`""`	
zebrium.svcgrpMapFile	string	`""`	

Key	Type	Default	Description
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 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, or 20.04
- CentOS or Red Hat Enterprise Linux 7 or 8
- Amazon Linux 2

Installing the Collector

1. If the environment uses a proxy server, see *Operating with a Proxy Server*, below.
2. Determine which deployment name to use for the `<YOUR_SERVICE_GROUP>` value, below.
3. If your account has multiple deployments, go to the Zebrium user interface, click the ***Deployment*** drop-down in the top-right navigation bar, and switch to the deployment you want to use to collect Windows logs.
4. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings ( ) Integrations & Collectors).
5. Click the **[Linux]** button under **Log Collectors** and copy the command from the **Linux Log Collector** dialog. This command includes the Zebrium API server URL (`<ZAPI_URL>`) and authentication token (`<AUTH_TOKEN>`) values.
6. Update the command from step 4 with the relevant values and run it in a shell on the host. The command uses the following format:

```
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=<YOUR_SERVICE_GROUP>" /bin/bash
```

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

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

You can use the `ZE_USER_LOG_PATHS` environment variable 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

To exclude certain sensitive or noisy events from being sent to Zebrium, you can filter them at the source collection point:

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.

You can configure log path mapping using a JSON file, with the following 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.

Configuring Multiple Zebrium Service Groups Within a Single Collector

You can use a single td-agent to send log files to multiple Zebrium service groups. You should be familiar with advanced fluentd configuration for this feature. We recommended that you review the official documentation at https://docs.fluentd.org/configuration/config-file.

The following settings are required:

- Each service group needs to have its own source block and match block definitions.
- In each source block, the path should be as specific as possible.
- The paths in source blocks should not overlap.
- Each source block needs a unique **pos_file** (td-agent will create the file if it does not exist).
- Each source block should include a unique tag to specify which match block or service group should pick up the log events.
- Each match block should match on the tag in its corresponding source block.
- **ze_log_collector_url**, **ze_log_collector_token**, and **ze_log_collector_type** will probably be the same in all match blocks.
- **ze_host_tags** specifies the service group name with "ze_deployment_name=".
- each match block requires a unique buffer path, which will be created if the specified path does not exist.

The following example shows how this could be configured in **/etc/td-agent/td-agent.conf**:

```
<source>
  @type tail
```

```
  path "/var/log/auth.log"
  format none
  path_key tailed_path
  pos_file /var/log/td-agent/position_file_1.pos
  tag seamus1
  read_from_head true
</source>

<source>
  @type tail
  path "/var/log/syslog"
  format none
  path_key tailed_path
  pos_file /var/log/td-agent/position_file_2.pos
  tag seamus2
  read_from_head true
</source>

@include conf.d/user.conf
@include conf.d/containers.conf
@include conf.d/systemd.conf

<match seamus1>
  @type zebrium
  ze_log_collector_url "https://trial.zebrium.com"
  ze_log_collector_token "<your token here>"
  ze_log_collector_type "linux"
  ze_host_tags "ze_deployment_name=seamusfirstservicegroup"
  <buffer tag>
    @type file
    path /var/log/td-agent/buffer1/out_zebrium.*.buffer
    chunk_limit_size "1MB"
    chunk_limit_records "4096"
    flush_mode "interval"
    flush_interval "60s"
  </buffer>
</match>

<match seamus2>
  @type zebrium
```

```
  ze_log_collector_url "https://trial.zebrium.com"
  ze_log_collector_token "<your token here, should be the same as above>"
  ze_log_collector_type "linux"
  ze_host_tags "ze_deployment_name=seamussecondservicegroup"
  <buffer tag>
    @type file
    path /var/log/td-agent/buffer2/out_zebrium.*.buffer
    chunk_limit_size "1MB"
    chunk_limit_records "4096"
    flush_mode "interval"
    flush_interval "60s"
  </buffer>
</match>
```

You should set "patterns" to regular expressions to match the log file path. Each regex named captured in a matching regular expression will be compared to the "tags", "ids", and "configs" sections and added to the corresponding record sections. Use the `ze_path_map_file` configuration parameter to specify the path to the JSON file.

Usage

Start and Stop Fluentd

You can start or stop the Fluentd agent with the following command:

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

Testing Your Installation

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

Troubleshooting

In the event that Zebrium requires the collector logs for troubleshooting, the logs are located in the following locations:

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, check the spelling of the Zebrium host URL. Also check that any network proxy servers are configured appropriately.

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

Environment Variables

If the environment is using a proxy server to access the Internet then standard variables, such as **http_proxy**, should be configured prior to installation. For more information, see *Operating with a Proxy Server*.

Operating with a Proxy Server

If the agent environment requires a non-transparent proxy server to be configured, you should do this 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
```

Logstash Collector

Configuring Logstash to Send Log Data to Zebrium

> IMPORTANT: If you have upgraded from version 7.x of Logstash to version 8.x, ECS compatibility will be on by default. Depending on your environment and settings, you might need to turn off ECS compatibility. For more information, see https://www.elastic.co/guide/en/logstash/current/breaking-8.0.html#bc-ecs-compatibility.

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

1. If your account has multiple deployments, go to the Zebrium user interface, click the ***Deployment*** drop-down in the top-right navigation bar, and switch to the deployment you want to use to collect Windows logs.
2. Go to the **Integrations & Collectors** page (Settings ( ) Integrations & Collectors).
3. In the **Log Collectors** section, click **Other**.
4. 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 required (and one optional) Zebrium fields that you can use to define 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.

Type	Description	Key Name	Key Definition	Requirement
		`@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.
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).

Configuration metadata	Arbitrary name/value pairs associated with each log event.	`@ze_cfg_<name>`	Show as configuration metadata in the Zebrium user interface and in the Outgoing Webhook integration payload.For example:
			`mutate {`
			`add_field => { "@ze_cfg_myname1" => "myvalue1" }`
				Optional

Type	Description	Key Name	Key Definition	Requirement
			`}` Adds a metadata field called `myname1` with a value of `myvalue1`.	

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.

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, let's say that 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 code example 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 below:

```
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.

The 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. Make sure clients can reach the 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 the 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.

VMware vSphere Collector (Beta)

Legal

The VMware vSphere collector is provided by ScienceLogic with the following terms:

You may use, modify, reproduce, and distribute this freely and without restriction, provided as a condition of our provision to use the software you acknowledge that the software is provided as-is, and ScienceLogic will not have any monetary liability in association with the distribution of this software.

Overview

To collect logs for Zebrium, you will need to set up a Zebrium syslog forwarder within the vSphere environment and configure vCenter to forward syslog events to the Zebrium syslog forwarder.

NOTE: This feature is currently Beta. For access to this collector, contact Zebrium at support@zebrium.com.

Prerequisites

- VMware vSphere 6.7 or later
- VMware vCenter Server
- Linux VM for forwarding syslogs

Installation and Configuration

Installing the Zebrium Syslog Forwarder

1. Create a VM for hosting the Zebrium syslog forwarder. An Ubuntu 22.04 server with 1 CPU, 2G RAM, and 16G HDD is recommended.
2. Install the Zebrium syslog forwarder on the new VM by following the directions in the *Syslog Forwarder* topic.

Configuring vCenter Syslog Collection

Configure your vCenter Server to forward vCenter syslogs to your new VM by following the directions here: https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.monitoring.doc/GUID-9633A961-A5C3-4658-B099-B81E0512DC21.html

Configuring ESXi Host Syslog Collection

1. (Optional) For environments with fewer than 30 hosts, you can configure ESXi logs to be sent to the vCenter server, which in turn will forward the logs to Zebrium. For more information, see https://kb.vmware.com/s/article/2003322
2. (optional) For environments where forwarding ESXi logs to vCenter is not ideal, you can configure ESXi host logs to be sent directly to the Zebrium syslog forwarder. You can accomplish this in one of the following ways:
    - Configure the ESXi syslog settings via vSphere Inventory Advanced System Settings: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.esxi.upgrade.doc/GUID-9F67DB52-F469-451F-B6C8-DAE8D95976E7.html.
    - Configure the ESXi Syslog.global.logHost settings with the esxcli tool: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.esxi.install.doc/GUID-8981F5FA-BB2A-47FB-A59A-7FC5C523CFDE.html.

Collecting VM Logs

To send logs to Zebrium directly from VMs, please see the following topics:

- Linux-based VMs: ***Linux Collector***
- Windows-based VMs: ***Windows OTel Collector***
- Other VMs: ***Configuring Log Collectors and File Uploads***

> **NOTE:** Some VM operating systems might support forwarding syslogs. Forwarding the VM syslogs to the Zebrium syslog forwarder created in the ***Installing the Zebrium Syslog Forwarder*** topic might be an efficient VM log collection solution.

Windows OTel Collector (Beta)

Legal

The Windows OTel collector is provided by ScienceLogic with the following terms:

You may use, modify, reproduce, and distribute this freely and without restriction, provided as a condition of our provision to use the software you acknowledge that the software is provided as-is, and ScienceLogic will not have any monetary liability in association with the distribution of this software.

Overview

The following instructions explain how to install the Zebrium Open Telemetry (OTel) Collector on a Windows system using PowerShell.

NOTE: This feature is currently Beta. For access to this collector, contact Zebrium at support@zebrium.com.

Prerequisite

The Windows OTel collector requires you to install the Visual Studio 2015 or later Redistributable package. For more information, see https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170#visual-studio-2015-2017-2019-and-2022.

Zebrium Windows OTel Collector Installation

Before starting, you will need to obtain a copy of the Zebrium OTel Collector .**zip** file for Windows:

1. Unzip the **otelcol-sciencelogic-zebrium_Windows_x86_64.zip** archive:

   ```
   Expand-Archive ./otelcol-sciencelogic-zebrium_x86_64.zip
   ```

   ```
   cd otelcol-sciencelogic-zebrium_x86_64
   ```

2. If your account has multiple deployments, go to the Zebrium user interface, click the ***Deployment*** drop-down in the top-right navigation bar, and switch to the deployment you want to use to collect Windows logs.
3. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors).
4. In the **Log Collectors** section, click the **[Windows]** button. The **Windows Log Collector** dialog appears.
5. In the dialog, copy your Zebrium log collector URL and log collector token and click **[Close]**.

6. Edit the **otelcol.yaml** file, and paste the Zebrium log collector URL and token from step 5 into the appropriate place in the following two lines:

```
endpoint: <ze_log_collector_url>
```

```
ze_token: <ze_log_collector_token>
```

> **NOTE:** More advanced configuration options are suggested in the comments if needed.

7. Run the following install script as Administrator:

```
cmd.exe /c .\InstallSciencelogicOpenTelemetryCollector.bat
```

   This command installs the collector as a Windows Service that generates text logs in the **Logs** subdirectory.

8. After you complete these steps, logs will start streaming to your Zebrium account. In a few minutes, you can view log activity in the Zebrium user interface by going to the **Ingest History** page (Settings ( ) > Ingest History).

Uninstalling Zebrium Windows OTel Collector

To remove the Zebrium Windows OTel service, run the following command:

```
cmd.exe /c .\UninstallSciencelogicOpenTelemetryCollector.bat
```

Chapter 4

Suggestions and Root Cause Reports

Overview

This chapter explains suggestions in Zebrium and how to asses and disposition them, and it also explains how to use Root Cause reports to quickly address issues.

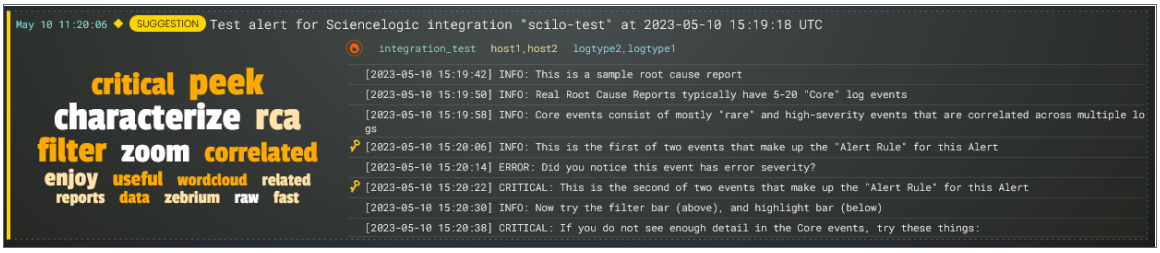
This chapter covers the following topics:

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 along with the existing alerts:



On the **Alerts** page, the summary report for a suggestion and an alert contains the following main elements:

- ***AI-generated title***. Displaying at the top of the summary pane, this title is generated using GPT Services that use new Generative AI models. You can enable or disable GPT services for a specific deployment of Zebrium by using the **GPT Services** column on the **Deployments** page (Settings ( )> Deployments).
- ***Word Cloud***. A set of relevant words chosen by the AI/ML engine from the log lines contained in the alert. Click a word in the cloud to highlight that word in the list of logs on the left.
- ***Significance icon***. Since not all suggestions that the AI/ML engine generates will relate to problems that actually impact users, the engine attempts to reason over the data and assess whether a problem actually requires attention. Hover over this icon at the top of the list of logs to view the confidence level of the AI/ML engine for this suggestion. A red icon ( ) means "High" confidence, and a yellow icon ( ) means "Medium" confidence.
- ***AI Assessment*** . Since not all suggestions that the AI/ML engine generates will relate to problems that actually impact users, the AI/ML engine attempts to reason over the data and assess whether a problem actually requires attention. Depending on the quality of the data, some suggestions might not include an AI Assessment. This value is shown in the Zebrium user interface as an **AI Assessment** value of one of the following:
    - "No Attention Needed" for content that the AI/ML engine assesses as unlikely to require immediate attention.
    - "Needs Your Attention" for content that the AI/ML engine believes should be looked into.
- ***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 Root Cause 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***.

You can click anywhere in the summary report for a suggestion or an alert to view a more detailed **Root Cause Report** page for that suggestion or alert. For more information, see *Root Cause Reports*.

> 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 time frame 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.

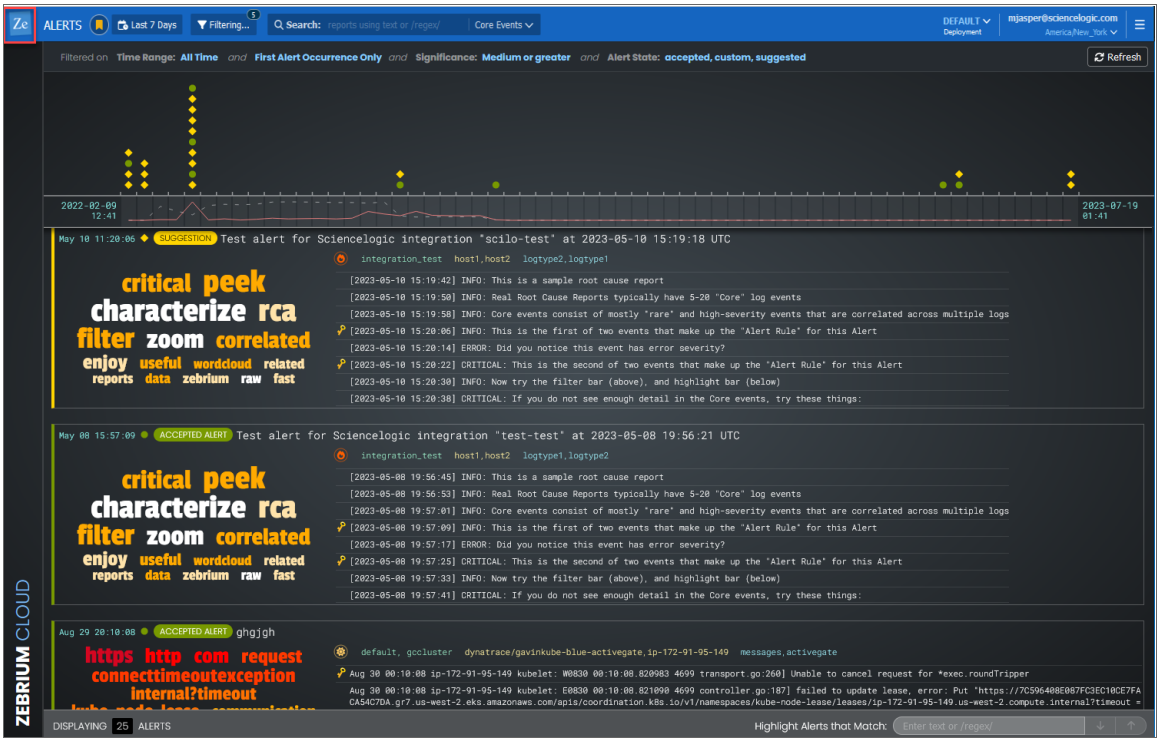
You can choose to "accept" or "reject" a suggestion. For more information, see *Assessing Suggestions*.

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 *Notification Channels*.

If you currently use SL1 from ScienceLogic, you can configure an integration that lets you view Zebrium suggestions in SL1 dashboards as well as on the SL1 **Events** page. For more information, see *ScienceLogic 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:



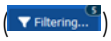
This page displays a list of filtering and search options at the top of the page. You can use these filters to manage the number of suggestions and alerts that display on the **Alerts** page. There is also a ***Search*** bar for text or regular expression (regex) searches, and a toggle for *Core Events*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

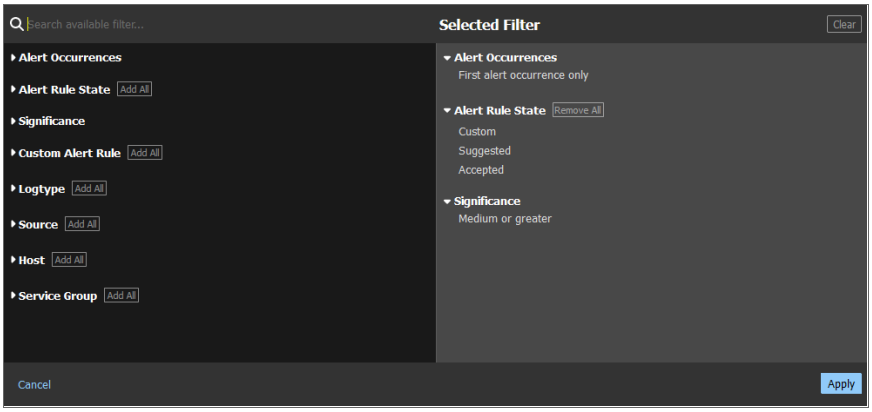
The Root Cause (RCA) reports that correspond to the items in the Timeline widget display in a summary view below the widget. If you click an icon in the Timeline widget, the RCA report for that icon moves to the top of the summary view below the widget. For more information about RCA reports, see *Root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

Using the Filters on the Alerts Page in Zebrium

At the top of the **Alerts** page, the **[Time Range]** button (Last 7 Days) 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.

TIP: You can click the **Views** icon () to change the view that is currently displayed on the **Alerts** page. A ***view*** is a predefined set of filters for the user interface. You can also create your own view based on the filters you use regularly. For example, if you set up your filters on the **Selected Filter** dialog to only see the most recent occurrence in a specific service group, for the past seven days, then after you set those filters, you can click **[Add view]** on the **Views** menu to create a view for those filters. Later you can select that new view from the **Views** menu to get your customized set of filters.

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 a suggestion 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 suggestion, based on how likely that suggestion is related to a problem. By default, only suggestions with a significance of Medium and High are shown on the **Alerts** page, so if you want to also see suggestions with Low significance, select *Low or greater* for this filter.
- ***AI Assessment*** . Since not all suggestions that the AI/ML engine generates will relate to problems that actually impact users, the AI/ML engine attempts to reason over the data and assess whether a problem actually requires attention. This value is displayed as the **AI Assessment**. You can filter by *Needs Your Attention* and *No Attention Needed*.

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. Any Root Cause reports generated by that scan include a lightning bolt icon and the text "Result of RC Scan".

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:

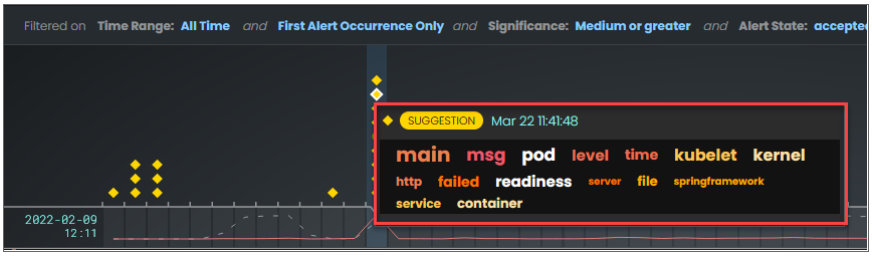
NOTE: The Timeline widget displays a list of the currently active filters at the top of the widget. For more information about filtering, see *Using the Filters on the Alerts Page in Zebrium*.

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. This icon only appears if you included *Rejected* as a filter for the **Alert Rule State** on the **Filtering** dialog.

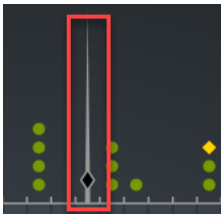
When you click an icon in the Timeline widget, the summary view for the corresponding RCA report for that suggestion or alert moves the top of list below the Timeline widget. Click anywhere in the summary view to open its **RCA Report** page.

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:



The Timeline widget also includes the following graphical elements:

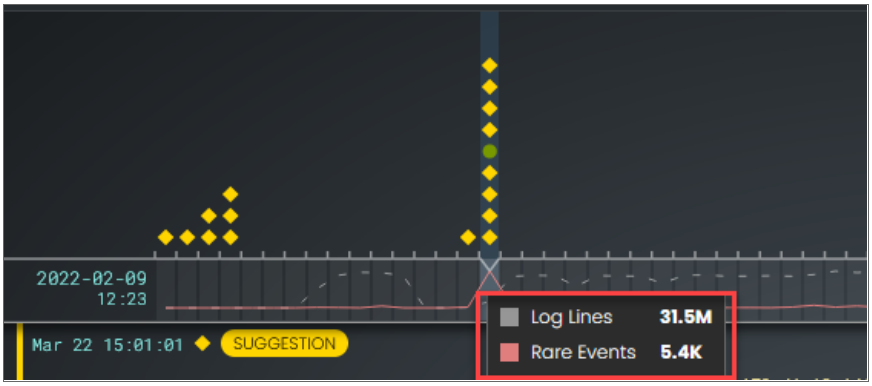
- **Spike**. A gray vertical line appears on the widget if too many suggestions or alerts exist for a specific time for the user interface to show them all:

  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.

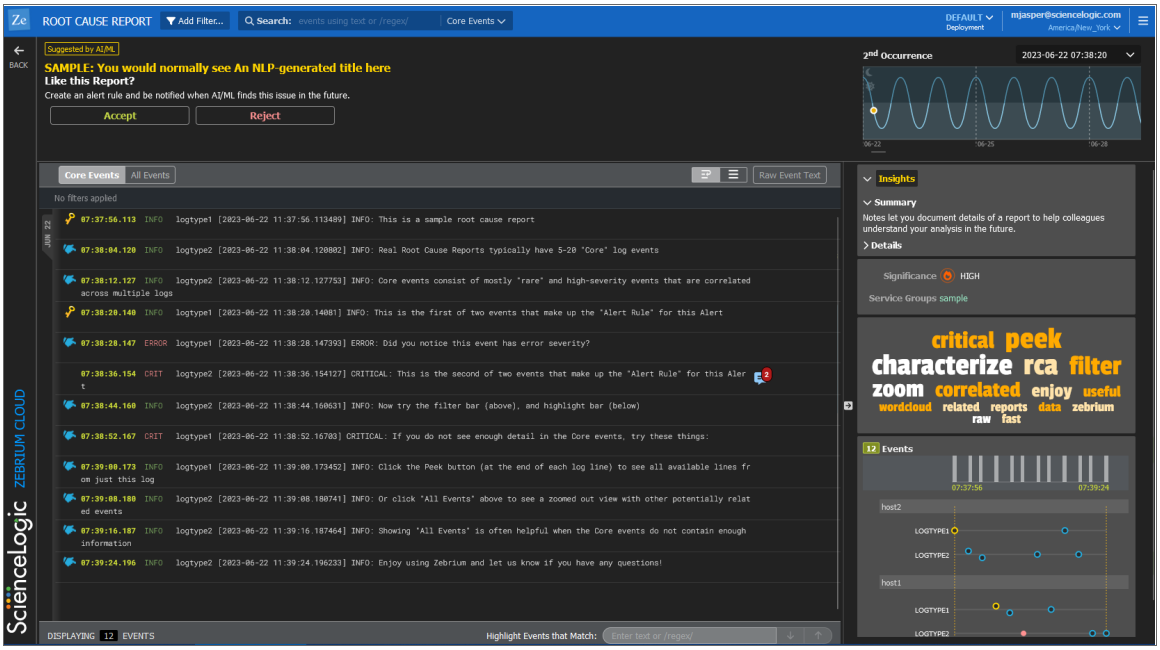


TIP: Click the **[Refresh]** button to get the most recently updated data for this page.

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:

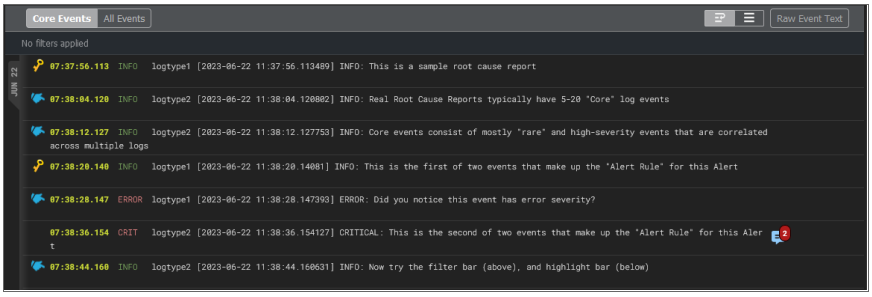


A typical **Root Cause Report** page contains the following elements:

- 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.

  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.



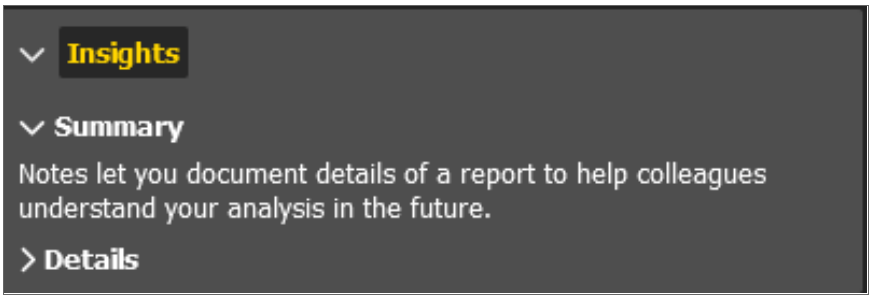
The columns in each log line show the event timestamp, a severity level, if available, the log type or service, and the text of the log. In addition, the following icons might appear to the left of some of the log events in the pane:

- ***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 possible events to explore. These events appeared in the report summary view on the **Alerts** page. This is just an informational icon.

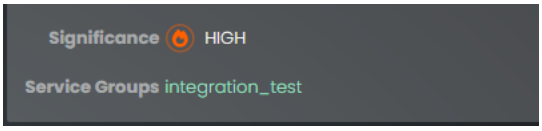
> **NOTE:** You can hover over a log event to access the **Actions** button, which lets you perform additional actions related to that log event. For more information, see *Additional Actions on the RCA Report Page*.

- 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.

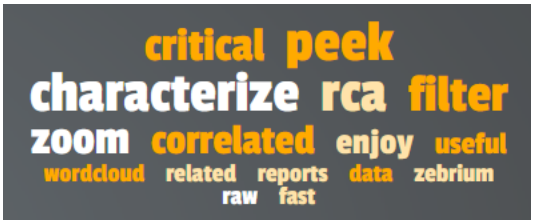
- In the group of smaller panes to the right, the top pane contains the **Insights** panel, which contains a Title, a Summary, and Details that are generated with GPT Services that use new Generative AI models. You can enable or disable GPT services for a specific deployment of Zebrium by using the **GPT Services** column on the **Deployments** page (Settings ( )> Deployments).



- The next pane displays an ***AI Assessment*** , where relevant (not all suggestions will include an AI Assessment, depending on the quality of the data). For this pane, the AI/ML engine attempts to reason over the data and assess whether a problem actually requires attention. This value is shown in the Zebrium user interface as an **AI Assessment** value of one of the following:
    - "No Attention Needed" for content that the AI/ML engine assesses as unlikely to require immediate attention.
    - "Needs Your Attention" for content that the AI/ML engine believes should be looked into.
- The next 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.

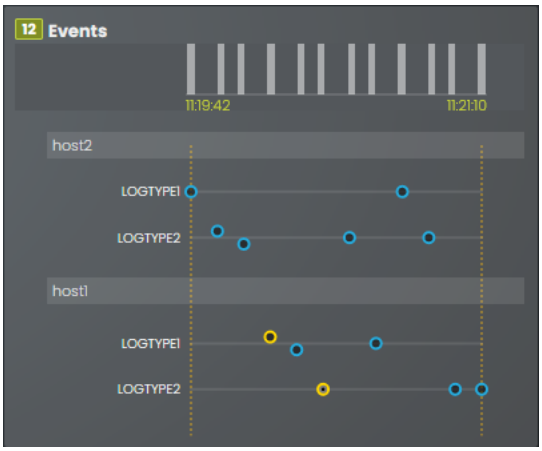


- The next pane displays the word cloud, which displays a set of keywords that the AI/ML engine selected from the report. For each word, the font size denotes how rare it is (smaller is more rare), and the color denoting how "bad" the underlying events were. For example, a word for a critical event displays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



> **TIP:** Click a word in the cloud to highlight the log events that contain that word in the list of logs on the left.

- 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 Root Cause Report Page

On the **Root Cause 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*. For an accepted alert, 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 **Root Cause 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 *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***.

> **TIP:** The **[Show Related Alerts and Suggestions]** button on the **Root Cause Report** page for a custom alert lets you augment the alert with related suggestions that the AI/ML engine uncovers in the surrounding log lines. You can use this button to help determine the root cause of a problem by showing a list of other alerts and suggestions that contain the same event.

Assessing Suggestions

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.

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

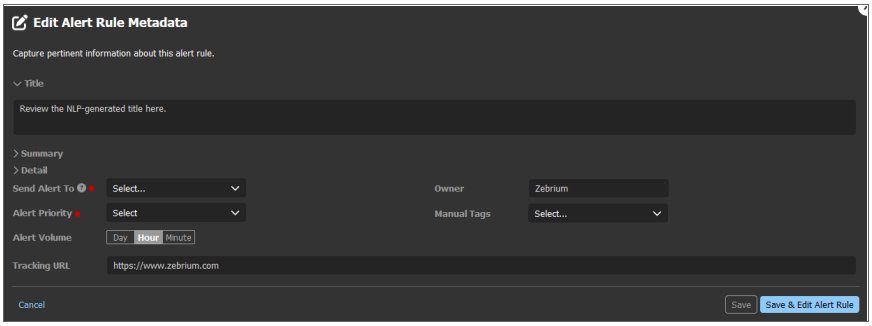
Accepting a Suggestion

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.

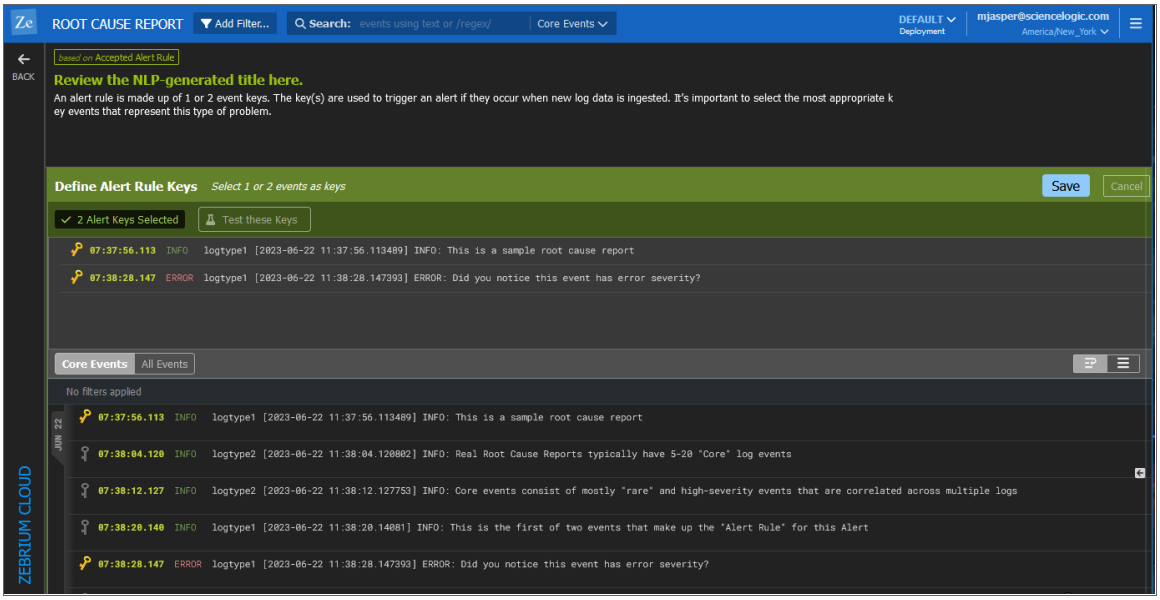
To accept a suggestion:

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



2. Complete the following fields:

    - ***Title***. Edit the name for this rule, or add a name if no name exists.
    - ***Summary***. Expand this field and edit the summary for this rule, or add a summary if none exists.
    - ***Detail***. Expand this field and edit the detail text for this rule, or add detail text if none exists.
    - ***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 *Notification Channels*. This field is required, but you can also click **[Select No one]** as an option.
    - ***Owner***. Type the name of the owner of this rule.
    - ***Alert Priority***. Set the priority from *P1* (highest) to *P5* (lowest). 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.
3. Click **[Save & Edit Alert Rule]**, the **Edit Alert Rule Keys** pane appears:



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 suggestion, 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.

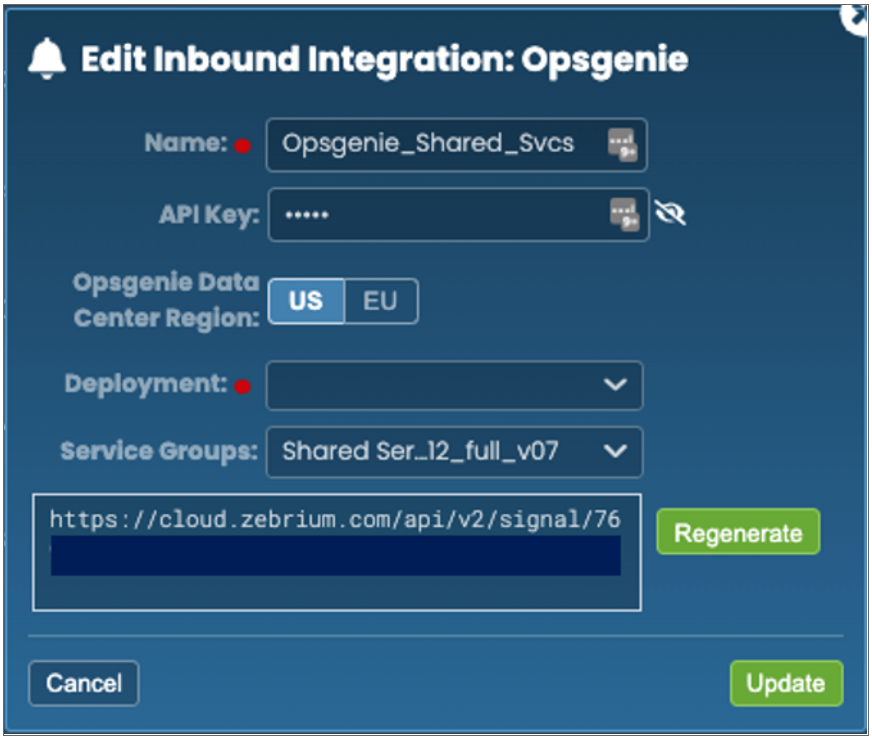
There are other settings that might be useful in rare cases, such as excluding a particular log type entirely if it is not useful from a diagnostics perspective.

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:



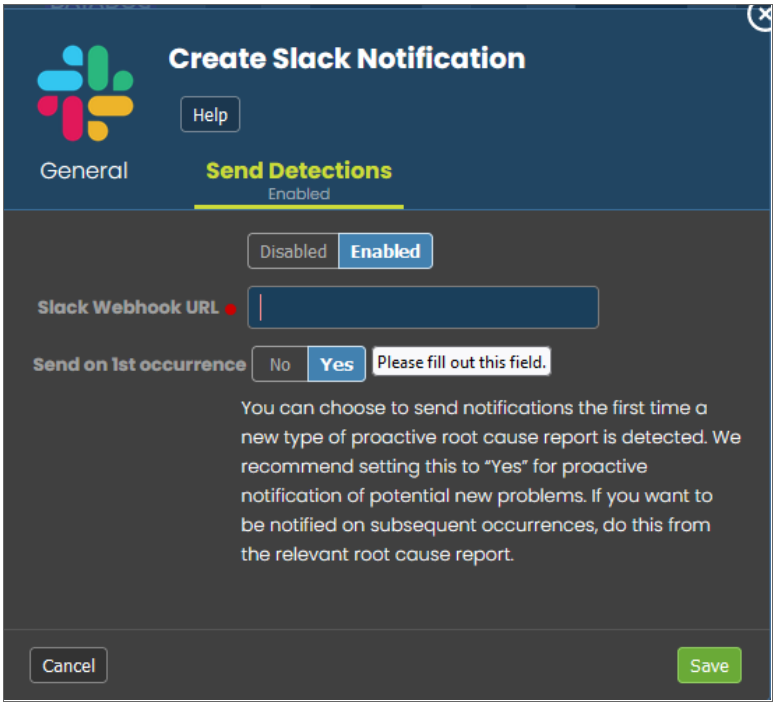
You can also use inbound integrations to route alerts rather than incidents to Zebrium. In this case Zebrium will not be able to update any incident fields, because it does not receive incident notifications. However, Zebrium will use the alerts as triggers to generate RCA reports, which will be sent 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 Slack, Microsoft Teams, and Webex Teams notifications.

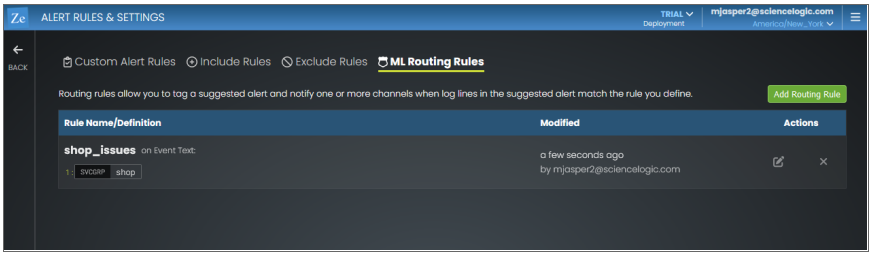


After you have created one or more notification channels, you can link any number of these to any RCA report created by the AI/ML engine. Linking a set of notification channels to a RCA report will send notifications of future RCA reports of the same type to those channels.

For more information, see *Notification Channels*.

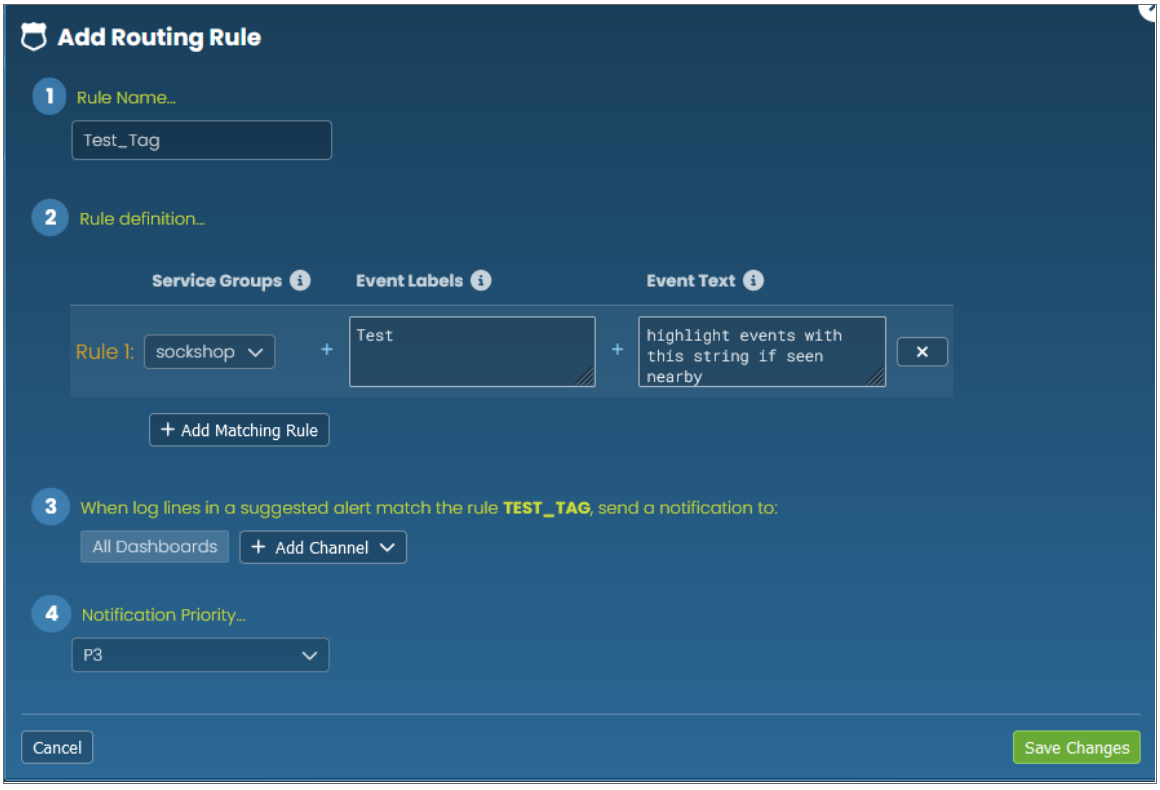
Use Routing Rules to Classify and Route Alerts

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:



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:



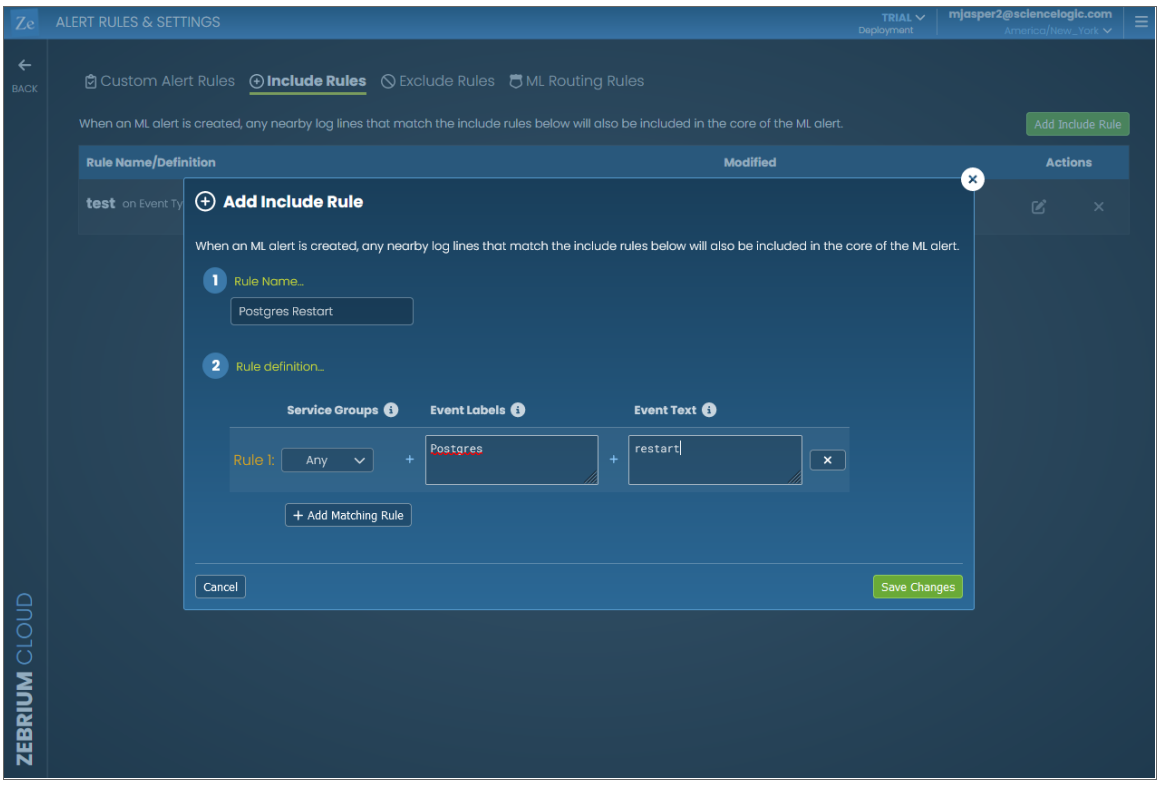
For more information about creating the rules for the ***Event Labels*** and ***Event Text*** fields, see *Defining Rules*.

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:

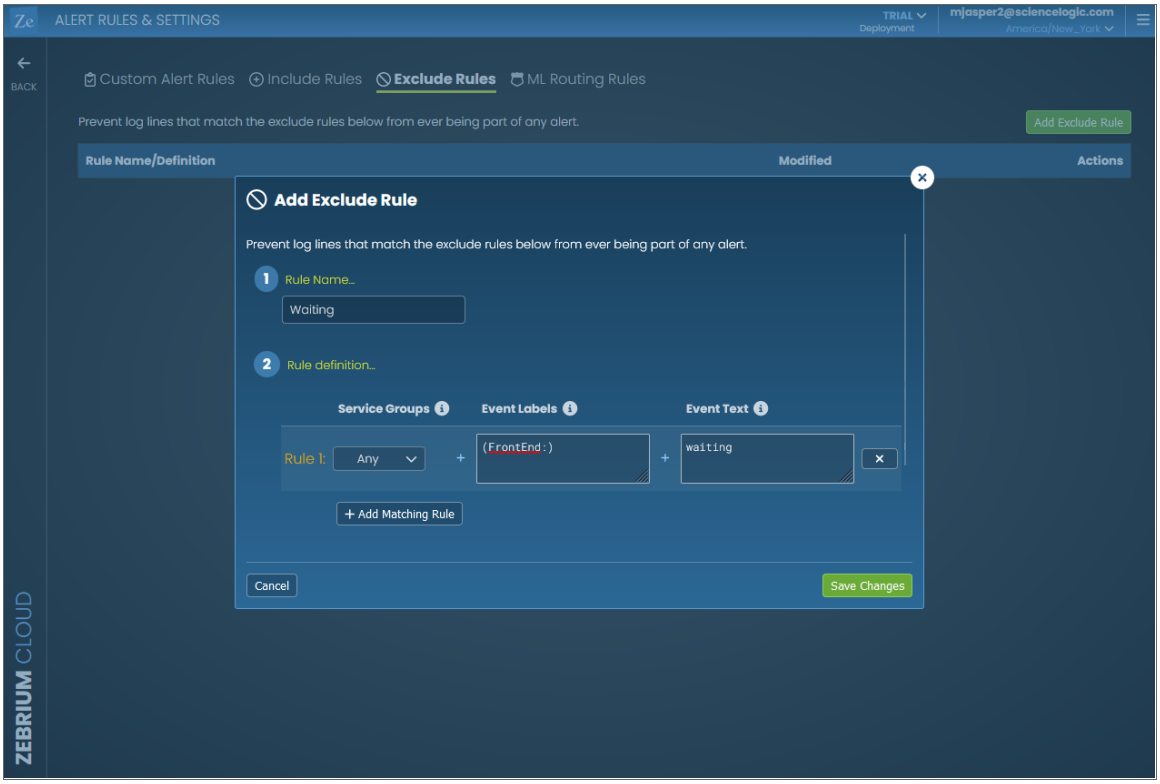


For more information about creating the rules for the ***Event Labels*** and ***Event Text*** fields, see *Defining Rules*.

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:



For more information about creating the rules for the ***Event Labels*** and ***Event Text*** fields, see *Defining Rules*.

Defining Rules

On the rules on the **Alert Rules & Settings** page (Settings ( ) > Alert Rules & Settings), you can set up detailed alert rules to help you manage the types of suggestions that the AI/ML engine creates. On the **Alerts** page, you can filter the list of suggestions and alerts based on the rules.

On the **Alert Rules & Settings** page, you can create the following types of rules for tags:

- ***Custom alert rules***. Custom alert rules deterministically create a suggestion or an alert and notify one or more channels when a log line matches the defined rule.
- ***Include rules***. When the AI/ML engine creates a suggestion, any nearby log lines that match the include rules below will also be included in the core of the suggestion.
- ***Exclude rules***. Prevent log lines that match the exclude rules from ever being part of any suggestion.
- ***Routing rules***. Routing rules allow you to tag a suggestion and notify one or more channels when log lines in the suggestion match the rule you define.

When you create any of these rules on the **Alert Rules & Settings** page, you will need to:

1. Select the ***service group or groups*** for the rule.
2. Specify the ***event label***, which requires one or more labels to match a corresponding Regular Express (regex) or case-sensitive substring.
3. Specify the ***event text***, which requires that the event text matches a corresponding regex or case-sensitive substring.

These three elements are described in detail in the following sections.

Service Groups

On the Add Rule dialog on the **Alert Rules & Settings** page, select one or more named service groups from the Service Groups drop-down.

If you do not select a service group, the rule can match an event from any service group. If you select one service group, the rule only matches an event from that service group, If you select more than one service group, the rule only matches an event from any *one* of the named service groups (a logical OR across the selections).

Event Labels

The ***Event Labels*** field requires one or more labels to match a corresponding case-sensitive substring, case-sensitive regex, or case-insensitive regex. All provided label conditions must be satisfied by the same event for that event to match the rule (a logical AND across the conditions).

The ***Event Labels*** field consists of a list of parenthesized conditions. Each parenthesized condition consists of the label name, followed by a colon, followed by one of a case-sensitive substring, a case-sensitive regex, and a case-insensitive regex.

As an example, to require the "app" label to start with the word *core*, case-insensitive; the "version" label to have a first digit of *6*; and the "State" label to have the value of *DONE*:

```
(app:/^core/i)(version:/^\D*6/)(State:DONE
```

Below are more details on entering each condition.

To require a case-sensitive substring match to the label value, enter something like this:

```
Exact Label Value Substring
```

To require a case-sensitive PCRE regex match to the label value, enter something like the following, surrounding the regex in forward slashes:

```
/Label Value With Digit\d/
```

It is generally recommended, but not required, to escape all non-alphanumeric literal characters within your regex as a best practice, such as:

```
/Label\ Value\ With\ Digit\d/
```

To require a case-sensitive PCRE regex match to the label value, enter something like this, surrounding the regex in forward slashes:

```
/Label Value With Digit \d/
```

It is generally recommended, but not required, to escape all non-alphanumeric literal characters within your regex as a best practice, such as:

```
/Label\ Value\ With\ Digit \d/
```

To require a case-insensitive PCRE regex match to the event text, enter something like this (note the "i" after the second forward slash):

```
/label value with digit \d/i
```

It is generally recommended, but not required, to escape all non-alphanumeric literal characters within your regex as a best practice, such as:

```
/label\ value\ with\ digit\ \d/i
```

In general, any parse errors related to ambiguous patterns can be resolved by escaping all non-alphanumeric literal characters.

Event Text

The ***Event Text*** field requires the event text to match a corresponding case-sensitive substring, case-sensitive regex, or case-insensitive regex. All provided label conditions must be satisfied by the same event for that event to match the rule (a logical AND across the conditions).

For example, the following rule matches any event containing the phrase *error code:* followed by a number.

```
/error code:\d+/i
```

To require a case-sensitive substring match to the event text, enter something like this:

```
Exact Text to Match
```

To require a case-sensitive PCRE regex match to the event text, enter something like this, surrounding the regex in forward slashes:

```
/Matches Any Digit\: \d/
```

It is generally recommended, but not required, to escape all non-alphanumeric literal characters within your regex as a best practice, such as:

```
/Matches\ Any\ Digit\:\ \d/
```

To require a case-insensitive PCRE regex match to the event text, enter something like this (note the "i" after the second forward slash):

```
/matches any digit\: \d/i
```

Chapter 5

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, 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*
- *Slack*
- *Microsoft Teams*
- *Webex Teams*

Email Notifications

Features

- You can configure Zebrium to automatically send Root Cause (RCA) reports to email recipients. This allows you to see details of root cause in your email client.
- Each Zebrium RCA report includes a summary, a 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.
- This means faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

Integration Details

To create an email integration in Zebrium to send suggestions to email recipients:

1. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings () > Integrations & Collectors).
2. In the **Notifications** section, click the **[Email]** button.
3. Click **[Create a New Integration]**. The **Create Email Notification Integrations** dialog appears.
4. On the **[General]** tab, enter an ***Integration Name*** for this integration.
5. In the ***Deployment*** drop-down, select a deployment for the integration.
6. In the ***Service Group(s)*** drop-down, select a service group for the integration.
7. On the **[Send Detections]** tab, click **[Enabled]**.
8. Enter the **Email Address List**. Add one email recipient per line.
9. You can choose to send notifications the first time the AI/ML engine detects a new type of proactive Root Cause report. We recommend setting the ***Send on 1st occurrence*** toggle 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.
10. Click **[Save]**.

Slack Notifications

Features

- You can configure Zebrium to automatically send Root Cause (RCA) reports to Slack channels. This allows you to see details of root cause in your Slack client.
- Each Zebrium RCA report includes a summary, a 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.
- This means faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

Integration Details

STEP 1: Create an incoming webhook in Slack:

1. Go 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, below.

STEP 2: Create a Slack integration in Zebrium to send suggestions to Slack:

1. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors).
2. In the **Notifications** section, click the **[Slack]** button.
3. Click **[Create a New Integration]**. The **Create Slack Notification Integration** dialog appears.
4. On the **[General]** tab, enter an ***Integration Name*** for this integration.
5. In the ***Deployment*** drop-down, select a deployment for the integration.
6. In the ***Service Group(s)*** drop-down, select a service group for the integration.
7. On the **[Send Detections]** tab, click **[Enabled]**.
8. In the ***Slack Webhook URL*** field, add the webhook that you created in STEP 1, above.
9. You can choose to send notifications the first time the AI/ML engine detects a new type of proactive Root Cause report. We recommend setting the ***Send on 1st occurrence*** toggle 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.
10. Click **[Save]**.

Microsoft Teams Notifications

Features

- You can configure Zebrium to automatically send Root Cause (RCA) reports to Microsoft Teams channels. This allows you to see details of root cause in your Microsoft Teams client.
- Each Zebrium RCA report includes a summary, a 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.
- This means faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

Integration Details

STEP 1: Create an incoming webhook in Microsoft Teams:

1. In Microsoft Teams, go to the **Channel** where 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 suggestions to Microsoft Teams:

1. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings () > Integrations & Collectors).
2. In the **Notifications** section, click the **[Microsoft Teams]** button.
3. Click **[Create a New Integration]**. The **Microsoft Teams Notification Integrations** dialog appears.
4. On the **[General]** tab, enter an ***Integration Name*** for this integration.
5. In the ***Deployment*** drop-down, select a deployment for the integration.
6. In the ***Service Group(s)*** drop-down, select a service group for the integration.
7. On the **[Send Detections]** tab, click **[Enabled]**.
8. In the ***Webhook URL*** field, add the webhook that you created in STEP 1, above.
9. You can choose to send notifications the first time the AI/ML engine detects a new type of proactive Root Cause report. We recommend setting the ***Send on 1st occurrence*** toggle 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.
10. Click **[Save]**.

Webex Teams Notifications

Features

- You can configure Zebrium to automatically send Root Cause (RCA) reports to Webex Teams spaces. This allows you to see details of root cause in your Webex Teams client.
- Each Zebrium RCA report includes a summary, a 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.
- This means faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.

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, below.

STEP 2: Create a Webex Teams integration in Zebrium to send suggestions to Webex Teams:

1. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors).
2. In the **Notifications** section, click the **[Webex Teams]** button.
3. Click **[Create a New Integration]**. The **Create Webex Teams Notification Integrations** dialog appears.
4. On the **[General]** tab, enter an ***Integration Name*** for this integration.
5. In the ***Deployment*** drop-down, select a deployment for the integration.
6. In the ***Service Group(s)*** drop-down, select a service group for the integration.
7. On the **[Send Detections]** tab, click **[Enabled]**.
8. In the ***Webhook URL*** field, add the webhook that you created in STEP 1, above.
9. You can choose to send notifications the first time the AI/ML engine detects a new type of proactive Root Cause report. We recommend setting the ***Send on 1st occurrence*** toggle 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.
10. Click **[Save]**.

Chapter 6

ScienceLogic Integrations

Overview

Zebrium offers the following ScienceLogic integrations:

- *SL1 Enhanced (12.x)*. This integration uses a Zebrium access token that you can use with the following methods of gathering and displaying data from Zebrium:
    - *ScienceLogic Root Cause Timeline Widget*. Requires SL1 version 12.1.0 or later. This integration adds a timeline view that shows Zebrium suggestions and alerts on any SL1 dashboard.
    - *ScienceLogic Events (Zebrium Connector for SL1)*. Requires SL1 version 12.2.0 or later. This integration adds Zebrium suggestions and alerts as enriched events (including summary and word cloud) within SL1. These suggestions and alerts can display on SL1 **Events**, **Devices**, and **Services** pages.
- *SL1 API*. This integration is the legacy integration with SL1, which in previous versions of Zebrium was called the "ScienceLogic Events" integration. This integration can be used with older versions of SL1, and supports sending text-only alerts and events.

ScienceLogic Root Cause Timeline Widget

Features

- Automatically adds Root Cause reports in ScienceLogic SL1. This allows you to see details of root cause in any SL1 dashboard.
- The **Root Cause Timeline** widget in SL1 dashboards displays suggestions, accepted and custom alerts, and the Zebrium "word cloud" with summary root cause analysis (RCA) based on the relevant logs associated with the suggestions and alerts.
- This leads to faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.
- Requires SL1 12.1.0 or later.

How It Works

The 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 RCA report findings directly in your SL1 dashboards alongside other metrics to explain the reason behind the problems for which you were alerted.

Configuring the Root Cause Timeline Widget in SL1

For Zebrium users, a **Root Cause Timeline** visualization is available on the **Dashboards** page in SL1. This widget visualization lets you see when the AI/ML (machine learning) engine detects a possible or confirmed issue. When you hover over an icon for a suggestion or an alert in the widget, a pop-up displays a title and a word cloud that contains additional information about the likely root cause based on the relevant logs associated with the issue.

You can click the icon for a suggestion or an alert on the **Root Cause Timeline** visualization to go to the Zebrium user interface, where you can access further details and perform optional customizations on the **Root Cause Report** page.

IMPORTANT: The **Root Cause Timeline** widget is specific to "AIML Predictions" widget types only.

If you selected *Root Cause Timeline* as the visualization, complete the following fields:

- ***Title***. Enter a title for the widget.
- ***Zebrium Connection ID***. Enter the unique connection ID from Zebrium, which you can find by creating a service connection between SL1 and Zebrium. The value appears on the **Service Connections** page (Manage > Service Connections) in the SL1 user interface. For more information, see *Configuring a Zebrium Connection for the Root Cause widget*.

- ***Zebrium Service Groups***. Enter the name or names of the service groups in Zebrium that you want to monitor with this widget. If you have more than one service group, separate the names with commas. If you want to view sample alerts for troubleshooting purposes, include the "integration_test" service group here. If you leave this field blank, the widget will include all of the service groups. Optional.

> **NOTE:** If you try the sample alert feature, make sure to add the special *integration_test* service group to this field.

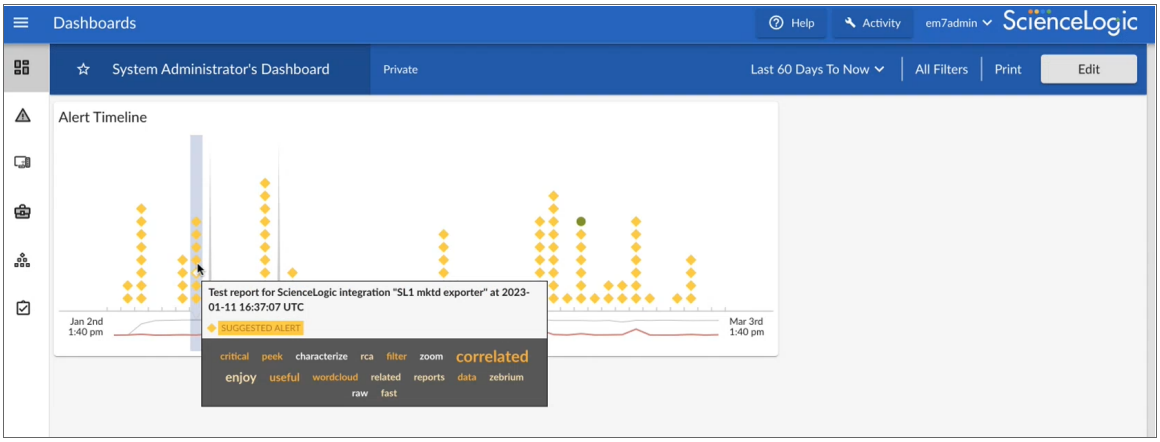
For more information about using the **Root Cause Timeline** visualization with "AIML Predictions" widget types, see *Using the Root Cause Timeline Widget*.

Configuring a Zebrium Connection for the Root Cause Timeline Widget in SL1

For Zebrium users, a **Root Cause Timeline** visualization is available on the **Dashboards** page in SL1. This widget visualization lets you see when the AI/ML (machine learning) engine detects a possible or confirmed issue. When you hover over an icon for a suggestion or an alert in the widget, a pop-up displays a title and a word cloud that contains additional information about the likely root cause based on the relevant logs associated with the issue.

You can click the icon for a suggestion or an alert on the **Root Cause Timeline** visualization to go to the Zebrium user interface, where you can access further details and perform optional customizations on the **Root Cause Report** page.

> IMPORTANT: The **Root Cause Timeline** widget is specific to "AIML Predictions" widget types only.



Connecting Your Zebrium Instance to the Root Cause Timeline Widget

To establish communication between Zebrium and the **Root Cause Timeline** widget in SL1, you will need to create a ***service connection***, which enables communication between SL1 and Zebrium.

This is a two-part process:

1. Create an "SL1 Enhanced (12.x)" integration in the Zebrium user interface.
2. Use the data from that integration to create the service connection in SL1.

Creating a Dashboard Widget Integration in Zebrium

You will need credentials for logging in to Zebrium to create the following integration.

To create an "SL1 Enhanced (12.x)" integration in Zebrium:

1. Log in to your Zebrium instance.
2. Go to the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors) and click the **[SL1 Enhanced (12.x)]** button in the **ScienceLogic** section. The **Integrations** dialog appears.
3. Click **[Create a New Integration]**. The **Create Integration** dialog appears.
4. On the **[General]** tab, complete the following fields:
    - ***Integration Name***. Type a name for the widget.
    - ***Deployment***. Select the Zebrium deployment that you want to monitor.
5. Click **[Save]**. The **Your Integration Info** dialog appears, with a summary of the key values for the widget integration.
6. Make a note of each value, as you will use all three values when creating the service connection in SL1. You can click each value to automatically copy that value.
7. Click **[OK]**. The new integration is added to the **ScienceLogic Integrations** dialog.

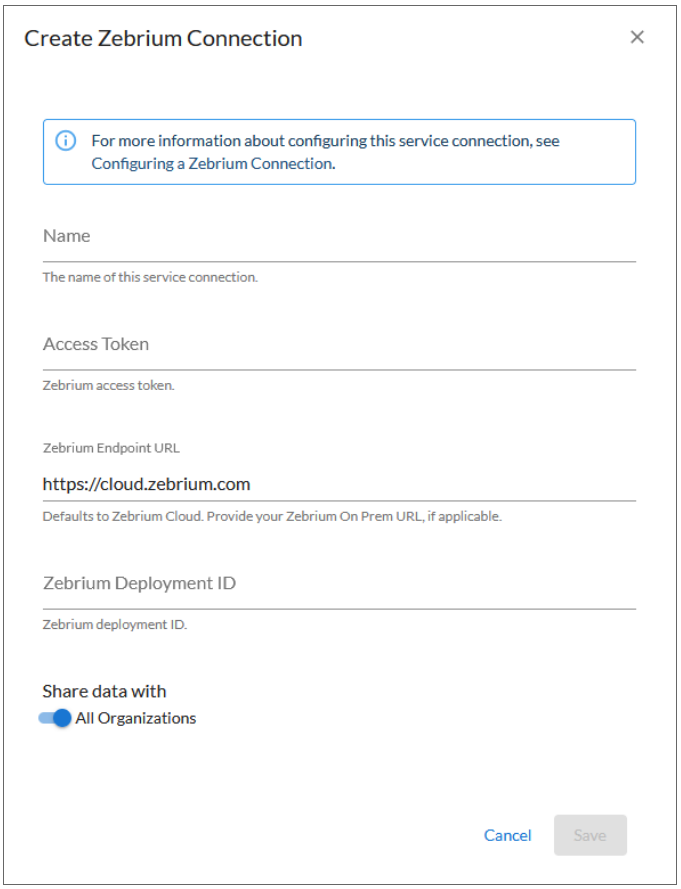
Creating a Service Connection in SL1

After you create the ScienceLogic integration in Zebrium, you will have the data you need to create the service connection in SL1.

IMPORTANT: To refer to this data in the Zebrium user interface, go to the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors) and click the **[SL1 Ehanced (12.x)]** button in the **ScienceLogic** section, and then click the edit icon ( ) for that integration. The **Edit** dialog displays all the relevant data you need for this procedure.

To create a Zebrium service connection in SL1:

1. In SL1, go to the **Service Connections** page (Manage > Service Connections).
2. Click **Add Service Connection**. The **Create Zebrium Connection** window appears.



3. Complete the following fields:
    - ***Name***. Type a name for this new service connection.
    - ***Access Token***. Add the *Access Token* value from the **Your Integration Info** dialog or the **Edit Integration** dialog.

      > **TIP:** You can also access this information on the **Access Tokens** page (Settings ( ) > Access Tokens) in the Zebrium user interface.

    - ***Zebrium Endpoint URL***. Add the *Endpoint URL* value from the **Your Integration Info** dialog or the **Edit Integration** dialog. Zebrium Cloud users can use the default value in this field, while Zebrium On Prem users will need to add the URL of their on-premises Zebrium instance.
    - ***Zebrium Deployment ID***. Add the *Deployment ID* value from the **Your Integration Info** dialog or the **Edit Integration** dialog.
    - ***Share data with***. Select the *All Organizations* toggle (turn it blue) to share with all existing and new organizations when you create them. Alternately, you deselect the *All Organizations* toggle (turn it gray) and select one or more organizations from the ***Selected Organizations*** drop-down to limit access to this connection to only the selected organizations.
4. Click **[Save]**.
5. On the **Service Connections** page, copy the *Service Connection ID* value from the **ID** column for the service connection you just created. You will use this value when you *create the Root Cause Timeline widget* for the AIML Predictions widget type.

Creating a Sample Alert for the Widget

To create a sample alert to display on the new widget, you will need to add the "integration_test" service group in the "SL1 Enhanced (12.x)" integration in the Zebrium user interface. You will not see the sample alert in SL1 unless you configure the connector or widget to include the "integration_test" service group.

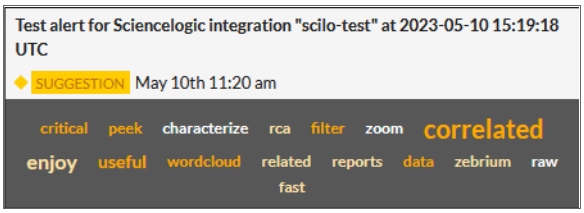
1. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors) and click the **[SL1 Enhanced (12.x)]** button in the **ScienceLogic** section. The **Integrations** dialog appears.
2. Click the edit button next to the integration with SL1 that you created earlier.
3. Make sure that one of the service groups in the ***Service Groups*** drop-down includes *integration_test*. The **[Create Sample Alert]** button creates an alert in the integration_test service group.
4. Click **[Save]**.
5. After you update the service group, you can click **[Create Sample Alert]** to test your settings. If your settings were correct, a sample alert will display on the **Alerts** page in the Zebrium user interface.

Using the Root Cause Timeline Widget

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

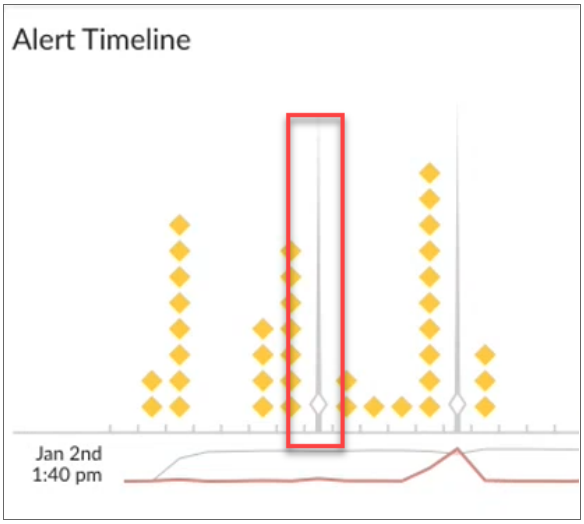
- **Suggestion** (). A yellow diamond represents a ***suggestion***, or a potential problem found by the AI/ML engine. When you click a yellow diamond, the **RCA Report** page for that suggestion opens in the Zebrium user interface. On that page, you can choose to accept or reject that suggestion.
    - 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.
- **Accepted Alert** (). A green circle represents an ***accepted alert***, 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.

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

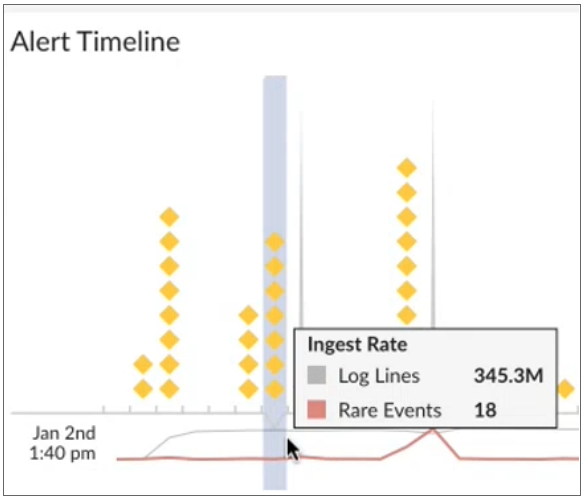


The Timeline widget also includes the following graphical elements:

- **Spike**. A gray vertical line appears on the widget if there are too many suggestions or alerts to show for a specific time. You can click and drag on the spike to zoom in so you can see all of the suggestions for that specific time. Click **[Reset zoom]** 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.



Working with Suggestions in the Zebrium User Interface

You can click the icon for a suggestion or an alert on the Timeline widget to go to the Zebrium user interface, where you can access further details and perform optional customizations on the **Root Cause Report** page.

For more information about what you can do on the **Root Cause Report** page, see *Root Cause Reports* in the Zebrium Documentation.

ScienceLogic Events (Zebrium Connector for SL1)

The Zebrium Connector, also called the **ze_connector** service, continually checks your Zebrium instance for suggestions and alerts. The Connector then looks for an SL1 device that matches the Zebrium alerts, and sends the Zebrium suggestions and alerts to that device in SL1.

As a result, the Zebrium Connector lets you view Zebrium suggestions and alerts in the following locations in SL1:

- The **Events** page
- The **Event Investigator**page for a Zebrium suggestion or alert
- The **[Investigator]** tab and the **[Events]** tab of the **Device Investigator** page
- The **Timeline** widget and the **[Log Insights]** tab of the **Service Investigator** page

The Zebrium Connector requires SL1 12.2.0 or later.

Workflow for Configuring the Connector

Before you can view Zebrium data on these SL1 pages, you will need to complete the following configuration steps in Zebrium and SL1:

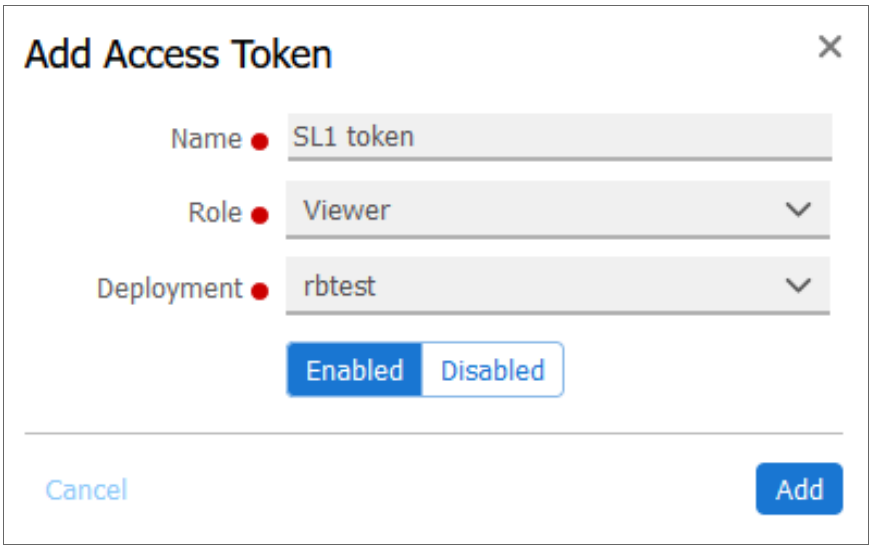
- Configure Zebrium:
    - *Create an authentication token in Zebrium*
- Configure SL1:
    - *Create a service connection in SL1*
    - *Create an SL1 authentication token*
    - *Create a default virtual device* (optional)
    - *Install the Zebrium Event Policies PowerPack*
- Configure the Zebrium Connector:
    - *Download and install the RPM file for the Zebrium Connector*
    - *Configure the config.yaml file*

Creating an Authentication Token in Zebrium

You first need to access the Zebrium user interface to get an authorization token, which you will use in the SL1 setup.

To create an authorization token in Zebrium:

1. In the Zebrium user interface, go to the **Access Tokens** page (Settings ( ) > Access Tokens).
2. Click **[Add Access Token]**. The **Add Access Token** dialog appears:



3. Complete the following fields:
   - ***Name***. Type a name for this token.
   - ***Role***. Select *Viewer*.
   - ***Deployment***. Select the deployment that you want to monitor.
4. Make sure the **Enabled** button is selected, and then click **[Add]**. The new token is added to the **Access Tokens** page. The token is in the format "Bearer <token>", such as *Bearer abcdefghijk*.
5. Hover over the **Name/Token** column of the new token and click the **[Copy]** button that appears.
6. Save the access token for the next set of steps.

Configuring SL1

Complete the following steps to configure SL1 so it can use the Zebrium Connector.

Create a Service Connection in SL1

To create a service connection in SL1:

1. In SL1, go to the **Service Connections** page (Manage > Service Connections).
2. Click **Add Service Connection**. The **Create Connection** window appears.
3. Complete the following fields:

- ***Name***. Type a name for this new service connection.
- ***Access Token***. Paste the access token you created in Zebrium into this field. You can view this information on the **Access Tokens** page (Settings ( ) > Access Tokens) in the Zebrium user interface.
- ***Zebrium Endpoint URL***. Add the endpoint URL for your Zebrium instance. Zebrium Cloud users can use the default value in this field, while Zebrium On Prem users will need to add the URL of their on-premises Zebrium instance.
- ***Share data with***. Select the *All Organizations* toggle (turn it blue) to share this connection with all existing and newly created organizations. Alternately, you deselect the *All Organizations* toggle (turn it gray) and select one or more organizations from the ***Selected Organizations*** drop-down to limit access to this connection to only those organizations.

4. Click **[Save]**. The service connection is added to the **Service Connections** page.

Create an SL1 Authentication Token

Next, you will need to encode your SL1 credentials to create an SL1 authentication token:

1. Go to a Base64 encoding site like https://www.base64encode.org and paste your SL1 username and password in the text box. Use the following format:

   `<username>:<password>`

   For example: `myuser:mypassword`

2. Use the default settings and click **[Encode]**. Your encoded credentials will look like the following:

   `bXl1c2VyOm15cGFzc3dvcmQ=`

   NOTE: The authentication token is in the format "Basic <token>".

3. Copy the newly encoded credentials, which will work as your SL1 authentication token.

Create a Default Virtual Device (optional)

The Zebrium Connector can send Zebrium suggestions and alerts to any device in SL1. If you do not have a specific device that you want to use for this purpose, you can optionally configure a "default" SL1 device. The Connector will send any Zebrium suggestions and alerts that do not map to existing SL1 devices to this default device.

For this purpose, you can create a virtual device in SL1 to receive all of these unassigned suggestions and alerts.

To create a default virtual device in SL1:

1. Ensure that SL1 includes a device class for virtual devices. These device classes must have a device category of "virtual" and a collection type of "virtual".
2. On the **Device Manager** page (Devices > Device Manager), click the **[Actions]** button and select *Create*

Virtual Device. The **Create Virtual Device** modal appears.

3. Complete the following fields:

    - ***Device Name***. Name of the virtual device.
    - ***Organization***. Organization to associate with the virtual device. Select from the drop-down list of all organizations in SL1.
    - ***Device Class***. The device class to associate with the virtual device. Select from the drop-down list of device classes. Only device classes with a device category of "virtual" and a collection type of "virtual" appear in the list.
    - ***Collector***. Specifies which instance of SL1 will perform auto-discovery and gather data from the device. Can also specify a "virtual" connector. Select from the drop-down list of all collectors in SL1.

4. Click **[Add]** to save the new virtual device. SL1 displays the new device ID after the text **Device Added**.
5. Before you close the modal, make a note of the ID for your new virtual device. You can sort for this ID on the **Devices** page in SL1 to quickly locate this new virtual device.

Install the Zebrium Event Policies PowerPack

To convert the API alerts sent by the Zebrium Connector into SL1 events, you will need the Zebrium event policies, which are available in the "Zebrium Event Policies" PowerPack. The event policies will be automatically enabled when you install the PowerPack.

To configure the Zebrium event policies:

1. Download and install the "Zebrium Event Policies" PowerPack. For more information, see Importing and Installing a PowerPack.
2. Go to the **Event Policies** page (Events > Event Policies) and sort by "Zebrium" in the **Name** column.
3. Make sure all of the Zebrium event policies from the PowerPack have a **Status** of *Enabled*. If not, check the boxes for the policies that are not enabled and click **[Enable]**.

Configuring the Zebrium Connector

The Zebrium Connector, also called the **ze_connector** service, continually checks your Zebrium instance for suggestions and alerts. The Connector then looks for an SL1 device that match the Zebrium alerts, and sends the Zebrium suggestions and alerts to that device in SL1.

You will need to install the Zebrium Connector RPM file on the SL1 server that you want to connect with Zebrium.

System Requirements

The SL1 server where you install this service must have the following:

- systemd
- Python 3.8
- `sudo` access to the server

- SL1 version 12.2.0 or later, running Oracle Linux 8 or later, with the "Zebrium Event Policies" PowerPack installed

IMPORTANT: ScienceLogic strongly recommends that you create a separate SL1 account for the Zebrium integration instead of using the default "em7admin" user account. For more information, see *Manually Creating a New User Account* in the SL1 Product Documentation.

Download and Install the RPM file for the Connector

You will need to download the RPM file from the ScienceLogic Support site, and then upload it to your SL1 system.

To download and install the RPM file:

1. Go to the ScienceLogic Support site at https://support.sciencelogic.com/s/.
2. Click the **[Product Downloads]** tab and select *SL1 Platform*. The **Platform Downloads** page appears.
3. Click the link for **SL1 Hollywood Platform 12.2**. The **Release Version** page appears.
4. In the **Release Files** section, click the RPM link for the **Zebrium Connector** RPM file. A **Release File** page appears.
5. Click **[Download File]** at the bottom of the **Release File** page.
6. SSH to the server where you are installing the RPM and run the following command to install the RPM:

```
sudo dnf install ze_connector-0.0.2-1.el8.noarch.rpm -y
```

7. *Configure the config.yaml file as needed:*

```
sudo vi /usr/bin/ze_connector/config.yaml
```

8. Restart the service and verify:

```
sudo systemctl restart zeconnector

sudo systemctl status zeconnector

sudo journalctl -u zeconnector

tail /usr/bin/ze_connector/out.log
```

Configure the config.yaml file

The **/usr/bin/ze_connector/config.yaml** file is supplied as part of the RPM install. You can use this sample configuration file to set up new jobs. This section explains the structure of the **config.yaml** file. You can copy this file and update it for the connector jobs.

NOTE: This schema will be overwritten to track the most recent Zebrium event found, specifically the `poll_timing.poll_start_time_iso` field.

Configuration Schema

- `jobs`: (array, required) - polling jobs to run
    - `name`: (str, required) - unique name of this job for log message readability
    - `sl1_api_config`: (obj, required)
        - `api_url`: (str, required) - URL endpoint for the SL1 API to query; do not include a "trailing slash" (/) at the end of the URL. Example: `api_url: https://127.0.0.1`
        - `api_auth`: (str, required) - Basic authentication token for the SL1 API (see *Create an SL1 Authentication Token* for format)
    - `poll_timing`: (obj, optional)
        - `poll_sleep_seconds`: (int, optional default:60) - number of seconds to sleep between polling requests
        - `poll_start_time_iso`: (str, optional default:now) - ISO 8601 timestamp for when to start querying for Zebrium alerts
    - `sl1_default_device_ids`: (array[str], optional default:[]) - list of SL1 device IDs to send alerts to if no device is matched automatically; omit to not send an alert if no device is matched
    - `ze_deployment_id`: (str, required) - Deployment ID of the Zebrium deployment to query. You can find this value in the **Deployment ID** column on the Deployments (Settings ( ) > Deployments) page of the Zebrium user interface.
    - `ze_service_groups`: (array[str], optional default:[]) - list of Zebrium service groups to query. You can view a list of service groups by clicking the **[Filtering]** button on the **Alerts** page of the Zebrium user interface. The **Selected Filter** dialog contains a list of service groups in the **Service Groups** filter. If you want to enable sample alerts, add `"integration_test"` under `ze_service_groups` in the **config.yaml** file.
    - `sl1_override_event_time`: (bool, optional default:False) - overrides using the Zebrium alert timestamp and instead uses now as when the alert occurred

Example Configuration

The following configuration will run two polling jobs:

- Job 1 will query **my1.sl1.com** using the defaults: poll every 60 seconds, starting from now
- Job 2 will query **my2.sl1.com** using overrides: poll every 120 seconds, starting from 09/05/2023, only query for Zebrium service groups **sg-1** and **sg-2**, send any unmatched events to SL1 device_id 1.

```
jobs:
  # minimal config required job
  # will default to all Zebrium Service Groups
  # will drop all alerts that don't match an SL1 device
  # polling will occur every 60s, starting from now
  - name: example_job_1
    ze_deployment_id: "sciencelogic_default"
    sl1_api_config:
      api_url: https://my1.sl1.com
      api_auth: "Basic dXNlcjpwYXNz"
  # maximal config job
  # will query only the 2 service groups provided
  # will send any alerts that don't match an SL1 device to device/1
  # will poll every 120 seconds from 9/5/2023 00:00:00 GMT to now
  - name: example_job_2
    sl1_default_device_ids:
      - "1"
    ze_service_groups:
      - "sg-1"
      - "sg-2"
      - "integration_test"
    ze_deployment_id: "some_other_deployment"
    sl1_api_config:
      api_url: https://my2.sl1.com
      api_auth: "Basic dXNlcjpwYXNz"
    poll_timing:
      poll_sleep_seconds: 120
      poll_start_time_iso: "2023-09-05 00:00:00"
    sl1_override_event_time: false
```

ScienceLogic SL1 API Integration

Features

- You can configure Zebrium to automatically add Root Cause (RCA) reports as events in ScienceLogic SL1.
- Each Zebrium RCA report includes a summary, a 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.
- This means faster Mean Time to Resolution (MTTR) and less time manually hunting for root cause.
- Requires SL1 11.2.0 or later.

> **NOTE:** The "SL1 API" integration is a legacy integration, and in previous releases it was called the "ScienceLogic Events" integration. To configure the newer Zebrium Connector (the **ze_connector** service), which sends Zebrium suggestions and alerts to the **Events** page, **Events Investigator** page, **Device Investigator**, and **Service Investigator** pages in SL1, see *Zebrium Connector for SL1*. This feature is available in SL1 version 12.2.0 or later.

How It Works

The 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 RCA report findings directly on the 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.

The Zebrium**Augment** mode is useful if you use a run book automation in SL1 to create a ticket based on an event from your alerts. In this mode, Zebrium updates the ticket directly with any Root Cause reports around the time of the event, so they are 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 the SL1 **Events** page.
3. With a single click on the SL1 **Events** page, you can drill down further into the Zebrium user interface to look at correlated logs across your entire application.

For details, see *Sending Root Cause Suggestions to the SL1 Events Page*.

Sending Root Cause Suggestions to the SL1 Events Page

Integration Overview

1. In ScienceLogic SL1, choose an existing Device 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 Device 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 in SL1 that is appropriate to use, you can create a virtual device for this purpose.

To 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). The DID also makes up part of the URL for the **Device Investigator** page for that device, such as https://*<SL1_IP_address>*/inventory/devices/**315**/investigator. You will use the DID when configuring the Zebrium integration.

To 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 **[Update]** 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

To 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:

```
Events: Event Note:Add/Rem

Events: Events/Event:View

Ticketing: Ticket:Notes:Add

Ticketing: Ticket:View
```

5. Click **[Save]**.

To 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]**.

To 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* for the ***Event Source***.
6. In the drop-down at the top of the next column, select *Regular Expression* (or [Regex Match] in the classic user interface).
7. In the first ***Match String*** field, type the following: `^Zebrium\s+(Detected|created).*`
8. Do not select ***Multi Match***.
9. Select ***Message Match***.
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 SL1 API Integration in Zebrium

1. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings () > Integrations & Collectors).
2. Scroll to the **ScienceLogic** section and select **ScienceLogic SL1 API**.
3. Click the **[Create a New Integration]** button.

4. On the **[General]** tab, enter an ***Integration Name*** for this integration.
5. Select the ***Deployment*** for the integration.
6. Select the ***Service Group(s)*** for the integration.
7. Go to the **[Send Detections]** tab.
8. Enter the ***Username*** and ***Password*** from STEP 2, above.
9. Enter the ***Device ID*** from STEP 1, above.
10. Enter the fully qualified ***Appliance URL*** to your instance of SL1 (**/api/*<api_endpoint>*** will be added automatically by the integration).
11. After you update this tab, you can click **[Create Sample Alert]** to test your settings. If your settings were correct, a sample alert will display on the **Alerts** page.
12. Click **[Save]**.

Chapter 7

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*
- *PagerDuty*

Opsgenie Incident Management Integrations

Features

- You can configure Zebrium to automatically add 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, a 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.
- This leads to faster Mean Time to Repair (MTTR) and less time manually hunting for root cause.

How it Works

The 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 RCA report findings directly in the incident that was created by Opsgenie to explain the reason behind the incident.

The Zebrium **Auto-Detect** mode is useful when you want to direct all Root Cause reports to Opsgenie for routing and dispositioning. You can also use **Auto-Detect** mode when you want to send only specific Root Cause reports to Opsgenie after first reviewing them 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 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. With a single click on your incident, you can drill down further into the Zebrium user interface to look at correlated logs across your entire application.

For details, see *Receiving Signals from Opsgenie*.

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

1. The Zebrium AI/ML engine 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, and the Root Cause details are visible as incidents in Opsgenie.

3. With a single click on your incident, you can drill down further into the Zebrium user interface to look at correlated logs across your entire application.

For details, see *Sending Root Cause Detections to Opsgenie as Incidents*.

Sending Root Cause Detections to Opsgenie as Incidents

This incident management integration automatically sends a Root Cause (RCA) report to Opsgenie so that the appropriate team is notified when the Zebrium AI/ML engine auto-detects an incident .

STEP 1: Add the Zebrium Integration to your Opsgenie Team

1. In the Opsgenie user interface, click 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. You will use this in STEP 2, below.
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. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors).
2. In the **Incident Management** section, click the **[Opsgenie]** button.
3. Click **[Create a New Integration]** button. The **Create Opsgenie Incident Management** dialog appears.
4. On the **[General]** tab, enter an ***Integration Name*** for this integration.
5. In the ***Deployment*** drop-down, select a deployment for the integration.
6. In the ***Service Group(s)*** drop-down, select a service group for the integration.
7. On the **[Send Detections]** tab, click **[Enabled]**.
8. Enter the **Opsgenie Webhook URL** that you created in STEP 1, above.
9. You can choose to send notifications the first time the AI/ML engine detects a new type of proactive Root Cause report. We recommend setting the ***Send on 1st occurrence*** toggle 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.

10. Click **[Save]**.

PagerDuty Event Management Integrations

Features

- You can configure Zebrium to automatically add Root Cause (RCA) reports to events in PagerDuty. This allows you to see details of root cause and direct the event to the appropriate team.
- Each Zebrium RCA report includes a summary, a 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.
- This leads to faster Mean Time to Repair (MTTR) and less time manually hunting for root cause.

How it Works

The recommended mode of operation for event 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 event creation. You can then review Zebrium RCA report findings directly in the event that was created by PagerDuty to explain the reason behind the event.

The Zebrium **Auto-Detect** mode is useful when you want to direct all Root Cause reports to PagerDuty for routing and dispositioning. You can also use **Auto-Detect** mode when you want to send only specific Root Cause reports to PagerDuty after first reviewing them 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 Events

1. Any PagerDuty event 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 PagerDuty using the **notes** API, and Root Cause details are visible in your PagerDuty Event.
4. With a single click on your event, you can drill down further into the Zebrium user interface to look at correlated logs across your entire application.

For details, see *Receiving Signals from PagerDuty*.

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

1. The Zebrium AI/ML engine 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, and the Root Cause details are visible as events in PagerDuty.

3. With a single click on your event, you can drill down further into the Zebrium user interface to look at correlated logs across your entire application.

For details, see *Sending Root Cause Detections to PagerDuty as Events*.

Receiving Signals from PagerDuty

STEP 1: Configure API Access for Zebrium in PagerDuty

1. In the PagerDuty user interface, go to the **Integrations** menu and select **API Access**.
2. Click the **[Create New API Key]** button.
3. Enter a description, such as "Zebrium Event 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 STEP 2. The key will not be visible in PagerDuty again.

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

1. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors).
2. In the **Event Management** section, click the **[PagerDuty]** button in the **Incident Management** section.
3. Click **[Create a New Integration]** button. The **Create PagerDuty Event Management** dialog appears.
4. On the **[General]** tab, enter an ***Integration Name*** for this integration.
5. In the ***Deployment*** drop-down, select a deployment for the integration.
6. In the ***Service Group(s)*** drop-down, select a service group for the integration.
7. On the **[Receive Signals]** tab, click **[Enabled]**.
8. Enter the ***Username*** for your PagerDuty portal.
9. Enter the ***API Key*** that you created in STEP 1, above.
10. Click **[Save]**. The **Your URL** dialog appears.
11. Copy the **Webhook URL** and save it for use in STEP 3, below.
12. Click **[OK]**.

STEP 3: Add the Zebrium Webhook to PagerDuty

1. n the PagerDuty user interface, go to the **Integrations** menu and select 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 *event.triggered*. Clear all other checkboxes.
8. Click the **[Add Webhook]** button.

How to Uninstall

Disable API Access in PagerDuty

1. In the PagerDuty user interface, go to the **Integrations** menu and 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. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors).
2. In the **Event Management** section, click the **[PagerDuty]** button.
3. Click the delete icon ( ) next to the Zebrium integration that you want to delete.
4. Click **[OK]** after confirming you wish to proceed.

Sending Root Cause Detections to PagerDuty as Events

This integration automatically sends Root Cause (RCA) reports to PagerDuty so that the appropriate team is notified when the Zebrium AI/ML engine auto-detects an event .

STEP 1: Create an Integration Key in PagerDuty

1. In the PagerDuty user interface, go to an existing or create a new **Event Orchestration** or **Event Rule** under the **Automation** menu item.
2. Under **Integrations** associated with the Event Orchestration or Rule, copy the corresponding *Integration Key* for STEP 2, below.

STEP 2: Create a PagerDuty Integration in Zebrium

1. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors).
2. In the **Incident Management** section, click the **[PagerDuty]** button.

3. Click **[Create a New Integration]** button. The **Create PagerDuty Event Management** dialog appears.
4. On the **[General]** tab, enter an ***Integration Name*** for this integration.
5. In the ***Deployment*** drop-down, select a deployment for the integration.
6. In the ***Service Group(s)*** drop-down, select a service group for the integration.
7. On the **[Send Detections]** tab, click **[Enabled]**. You might need to complete the **[Receive Signals]** tab before you can go to the next step. For more information, see *Create a PagerDuty Integration in Zebrium to Receive Signals from PagerDuty*, above.
8. In the ***Integration Key*** field, paste the Integration Key that you saved from STEP 1, above.
9. You can choose to send notifications the first time the AI/ML engine detects a new type of proactive Root Cause report. We recommend setting the ***Send on 1st occurrence*** toggle 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.
10. After you update this tab, you can click **[Create Sample Alert]** to test your settings. If your settings were correct, a sample alert will display on the **Alerts** page.
11. Click **[Save]**.

Chapter 8

Using Webhooks to Create Integrations

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.

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. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors).
2. In the **Webhooks** section, click the **[Outgoing RCA]** button.
3. Click **[Create a New Integration]** button. The **Create Outgoing RCA Webhook** dialog appears.
4. On the **[General]** tab, enter an ***Integration Name*** for this integration.
5. In the ***Deployment*** drop-down, select a deployment for the integration.
6. In the ***Service Group(s)*** drop-down, select a service group for the integration.
7. Enter the **Webhook URL** that will receive and handle the POST request.
8. On the **[Send Detections]** tab, click **[Enabled]**.
9. Enter the ***Webhook URL*** that will receive and handle the POST request.
10. Select the required **Authentication Method** for the endpoint and complete the necessary configuration using the information from STEP 1, above.
11. Click **[Save]**.

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	Summarization of the incident assigned by NLP 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_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`	timestamp (yyyy-mm-ddThh:mm:ss.nnnnnn)	Local time of incident start
`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 the incident assigned by NLP or the user
`incident_owner`	string	Owner assigned to this incident

Name	Type	Description
`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	Summarization of the incident assigned by NLP or the user
`incident_title`	string	Title of the incident assigned by NLP or the user
`incident_detail`	string	Full details of the incident assigned by NLP or the user
`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_words`	word object list	List of words (w) and their rareness/size (s) and badness (b) used in the word cloud
`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_`	event object list	Interesting events (level 1) in RC Report

Name	Type	Description
events		
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
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_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_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-
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": 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=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-
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"
      }
    }
  ],
  "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.

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

1. In the Zebrium user interface, go to the **Integrations & Collectors** page (Settings ( ) > Integrations & Collectors).
2. In the **Webhooks** section, click the **[Incoming RCA]** button.
3. Click **[Create a New Integration]** button. The **Create Incoming RCA Webhook** dialog appears.
4. On the **[General]** tab, enter an ***Integration Name*** for this integration.
5. In the ***Deployment*** drop-down, select a deployment for the integration.
6. In the ***Service Group(s)*** drop-down, select a service group for the integration.
7. On the **[Receive Signals]** tab, click **[Enabled]**.
8. Click **[Save]**.
9. Copy and save the contents of the **Your URL** text box for use in STEP 2, below.

STEP 2: Request a Root Cause Report from Zebrium

Send a POST request to the URL created in STEP 1 with the required payload:

```
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**, below, 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

User Management

Overview

User Management provides features for Role Based Access Controls (RBAC) that you can use to 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 that defines that user's permissions for accessing features and settings. Users are members of one or more groups to control which deployments the users can access.
- ***Groups***. Groups define which deployments are available to users in the group.
- ***Roles***: Roles are pre-defined user types, including Owner, Admin, Editor, and Viewer, that define the user permissions, such as Create, Read/View, Update, and Delete for each feature or application setting.

To access the user management pages in the Zebrium user interface, go to the **Users & Groups** page (Settings ( ) > **Users & Groups**).

Users

- Each user is assigned a role that defines that user's permissions for accessing features and settings.
- Users are members of one or more groups to control which deployments they can access.
- Users can be added, edited, and deleted by the Owner role.
- For more information, see ***Roles*** and ***Permissions***.

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, and deleted by the Owner role (see ***Roles*** below).

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 Editor 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 users of the role Member from previous releases.

Viewer

Users that are allowed read-only access to all but their own profile, such as changing their deployment selection or password.

Permissions

Setting or 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

Security

Overview

This chapter explains Zebrium security and how Zebrium protects your data.

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 Zebriumservice 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 an allow-listed 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 that are available from Zebrium:

- *Pre-installation*
- *Additional Configurations*
- *Installation*
- *Support*
- *Zebrium APIs*

Pre-installation

Before installation, there are several considerations to make when configuring your Kubernetes cluster for the Zebrium application. For the Zebrium software to be fully functional, the following software requirements must be met, and additional details and examples of these requirements are in the following topics:

- Kubernetes Version 1.19 or higher.
- Helm version 3 is required for installation of Zebrium.
- Kubernetes cluster availability meeting or exceeding the Zebrium sizing specifications. For more information, see *Sizing Considerations*.
- Ability to provision block storage. For more information, see *Storage Considerations*.
- Ingress Controller with https support for a Fully Qualified Host Name (FQHN). For more information, see *Ingress Considerations*.
- Access to the Zebrium registry. For more information, see *Helm Chart and Image Repository Access*.
- A Helm override file with your respective configurations.

After you have met these requirements, be sure to review the *Additional Configurations* topic. Otherwise, you are ready to move onto the *Installation* topic.

Sizing Considerations

Zebrium provides a sizing calculator at https://sizing.zebrium.com/ to assist with determining the vCPU and memory requirements of the destination Kubernetes Cluster. The sizing guide provides several key metrics for different Zebrium ingest volumes, which are detailed below:

- **Minimum Node Size (vCPU)**. Minimum node size for each node. This is based off the sizing of the Vertica container, as it is the largest requester of resources.
- **Minimum Node Size (Memory)**. Minimum node size for each node. This is based off the sizing of the Vertica container, as it is the largest requester of resources.
- **Total vCPUs required**. Total number of CPUs that need to be available in the cluster specifically allocated for the Zebrium deployment.
- **Total Memory required**. Total memory that needs to be available in the cluster specifically allocated for the Zebrium deployment.
- **Total SSD Storage required**. Total amount of disk space that is required to be able to provisioned within the cluster that is specifically allocated to the Zebrium deployment. While this is specified across the entire cluster as a sum, the storage requirements per node will vary with the containers that are deployed.

The Zebrium sizing calculator also exposes information about the major components that are deployed into the cluster, and their respective configurations are displayed under the **[Show Advanced Info]** button. You can use this information with the metrics above to more effectively determine the necessary resource requirements.

Example 1

ABC Corp is looking to install a Zebrium deployment in their local cluster with the intention to ingest up to 50 GB a day. After consulting the sizing guide, they provision a cluster that has 2 nodes attached to it. Each node has 12 vCPU and 24 GB of RAM attached to it. When installed, the Vertica pod is deployed to one node, requiring 280 GB of storage, while the zapi and core-engine pod was deployed to node 2, with 150 GB of storage required.

Example 2

CDE Corp is looking to install a Zebrium deployment in their local cluster with the intention to ingest up to 50 GB a day. After consulting the sizing guide, they provision a cluster that has 2 nodes attached to it. One node has 12 vCPU and 24 GB, and the other node has 6 vCPU and 18 GB of RAM attached. When they installed the Zebrium application, the Vertica and zapi pod is deployed to one node, requiring 380 GB of storage, 9 cores and 18 GB of RAM. However, they noticed that the core engine logs container is pending and will not start. This is due to the fact that while the node has 6 vCPU, the system pods deployed to that node has used 1.1 vCPUs. This left only 4.9 vCPUs available to core engine logs, which is requesting 5 vCPU.

> **NOTE:** While it is possible to meet the total resource requirements while disregarding the minimum node sizing, you run the risk of having pods being unable to schedule due to lack of resources. Please consult the **Advanced Info** section of the sizing guideline to ensure that all pods will have sufficient headroom to provision, while taking into account any other applications/pods deployed on the node.

Storage Considerations

The Zebrium deployment is made up of several statefulsets, all of which will require persistent volumes. Due to the constraints of our current database, Zebrium does not support NFS mounts, and all volumes provisioned will need to be physical or block storage. While the Zebrium deployment provides several ways to define the respective Kubernetes storage classes, it does not configure any external dependencies or permissions that may be needed to ensure provisioning happens correctly. It is the responsibility of the cluster operator to ensure that any and all external dependencies are met for your provisioner of choice.

The Zebrium application separates volumes into two different flavors: core and Vertica. This allows operators the flexibility of defining different retention and drive configurations for volumes that are mounted onto our central databases (Vertica), versus those used for our workers as buffers and working dir. Zebrium generally recommends that you configure the `reclaimPolicy` for your Vertica storage class to `Retain` to prevent any unintentional data loss if the corresponding statefulset or pvc is lost or deleted. This is not required on the core storage classes, as as loss of data stored in these claims will not be detrimental to the system.

Our application has two different options for how cluster operators can provide storage to satisfy the needs of the Zebrium Storage. Operators can bring their own storage classes, or use our Helm chart to define Zebrium specific classes. Both of these options are explored in more detail below.

Bring Your Own Storage Classes (BYOSC)

Cluster operators might choose to use existing storage classes instead of redefining these inside of the Zebrium Helm charts. To bring your own storage class, you will need add the following section to your override file. In the example below, we are using the `StorageClass` called `gp2` instead of the Zebrium-created ones.

```
zebrium-core:
  storageProvisioners:
    vertica:
      enabled: false
      customStorageClass: "gp2"
    core:
      enabled: false
      customStorageClass: "gp2"
```

This configuration disables the creation of the Zebrium storage classes, and instead instructs the pvcs to use the class `gp2`.

Using Zebrium Storage Classes

By default, Zebrium will provision two storage classes to be used for the Zebrium application. Configurations for these two storage classes are managed through the `storageProvisioners` section within the Helm chart. The default options for this settings are represented below:

```
storageProvisioners:
    vertica:
      enabled: true
      provisioner: kubernetes.io/no-provisioner
      reclaimPolicy: Retain
      parameters: {}
    core:
      enabled: true
      provisioner: kubernetes.io/no-provisioner
      reclaimPolicy: Retain
      parameters: {}
```

As we can see in the example above, we have two separate declarations of a storage provisioner, `vertica` and `core`. They both function the exact same way, so we will only focus on walking through `core` currently. As we can see, we have 4 configuration options available for the core storage provisioner and will dive into what each one does.

- `enabled`. Enables or disables the creation of the storage class within Helm. See ***BYOSC*** for an example of disabling this.
- `provisioner`. This configures which underlying storage class provisioner. We support any provisioner that is available to Kubernetes and that has been configured for your system.

- `reclaimPolicy`. Since these will be dynamically created volumes, we need to define a reclaim policy for when the resource is deleted. The available options are Retain or Delete. Read more here.
- `parameters`. Every provisioner provides a series of additional parameters that help to describe the volumes. These are unique to your selected provisioner. Read more here.

Dynamic vs Manual Volume Provisioning

While the Zebrium application will dynamically create persistent volume claims as the necessary pods are scheduled, cluster operators may choose to use a provisioner that does not support dynamic provisioning. An example of a provisioner that does not support dynamic provisioning would be the local. When using such provisioners, it is the responsibility of the cluster operator to ensure that any needed persistent volumes are created and available to the requesting persistent volume claims. A walkthrough of this can be found here.

Ingress Considerations

The Zebrium application leverages Kubernetes ingress as its preferred method for exposing its internal services and user interfaces to external consumers. Ingress resources are automatically created for each necessary route, and can be customized through the Helm chart parameters.

Helm Parameter Overrides

The Helm chart provides several level of configuration for modifying the ingress resources provisioned to tailor it to your desired ingress controller's requirements. Since ingress frequently uses annotations to configure some options depending on the controller, our Helm chart provides two ways to customize the ingress controller: through a global configuration or through application-level configurations. When the chart is templated, global values will override the resource-level configurations when both are set.

Global Overrides

Below are the available global configurations for all ingress resources. For annotations and TLS, the Helm chart will combine values defined in both the global and individual level. For the most update list of all options, please see the **values.yaml** file of the current Helm chart.

```
global:
  ingress:
    # -- Ingress Class to use for all objects.
    className:
    # -- Hostname to expose all ingress objects on.
    hostname: 'zebrium.example.com'
    # -- Global Annotations to add to all ingress objects
    annotations: {}
    tls: []
```

Resource Overrides

Below are the locations of the individual ingress resources, allowing you to modify only that particular ingress resource, instead of all resources. For the most update list of all options, please see the **values.yaml** file of the current Helm chart.

```
zebrium-core:
  zapi:
    ingress:
      path: '/api/v2'
      annotations: {}
      tls: []
  report:
    ingress:
      path: '/report/v1'
      annotations: {}
      tls: []
  mwsd:
    ingress:
      path: '/mwsd/v1'
      annotations: {}
      tls: []
zebrium-ui:
 ingress:
    path: '/'
    annotations: {}
    tls: []
zebrium-auth:
 ingress:
    path: '/auth'
    annotations: {}
    tls: []
```

Ingress Controllers

In order to expose the ingress resources defined by the Zebrium deployment, an ingress controller must be defined and configured by your cluster operator.

Packaged Ingress Controller

We do provide the option to install ingress-nginx as part of the Zebrium chart. If you wish to use the provided **ingress-nginx**, you can use the following configuration to get started:

```
ingress-nginx:
  enabled: true
global:
  ingress:
    className: nginx
    annotations:
      nginx.ingress.kubernetes.io/proxy-body-size: '0'
```

Hostname and DNS Resolution

Currently, the Zebrium deployment requires a dns hostname that allows access to the ingress endpoints. This endpoint needs to a be a fully qualified domain name (FQDN). Cluster operators should also ensure that this FQDN is added as a record to their DNS server and is resolvable from all systems intending to access the Zebrium installation. Network access from desired systems to the ingress endpoint should also be verified. To set the FQDN in the Zebrium Helm chart, use the following override:

```
global:
  ingress:
    hostname: ""
```

TLS

Due to browser security configurations, the Zebrium user interface must be served over HTTPS with a backing TLS certificate. Failure to do so will create a sign-in loop within the user interface, blocking the user from being able to access the internal system.

There are several ways to secure the ingress endpoint with a TLS certificate, including through the ingress resources themselves, through configuration of your ingress controller, using tools like cert-manager, a service mesh, or attaching certificates directly to provisioned resources, like cloud load balancers. It is at the discretion of the cluster operator to determine the best solution for their environment.

Helm Chart and Image Repository Access

Zebrium hosts its Helm charts and associated Docker images within its own registry. Zebrium will provide credentials (username/password) to access these resources as part of the on-prem onboarding process. As part of the installation process, you will create a Kubernetes image pull secret.

Additional Configurations

Zebrium On Prem allows for additional configurations to enable advanced features. Below is a list of these features and the necessary steps needed to configure them within the Zebrium Install Chart.

Enabling OpenAI Models

Zebrium supports leveraging OpenAI models to augment and enhance the summaries and titles of Zebrium Root Cause reports. Currently Zebrium supports the following OpenAI model providers:

- OpenAI
- Microsoft Azure OpenAI Services

Zebrium supports the following OpenAI models:

- Davinci
- GPT 3.5 Turbo
- GPT 4
- GPT 4 32k

To leverage these models, you will need to create and set up OpenAI services from one of the above providers. Zebrium supports multiple model configurations, using the following JSON format:

```
[
  {
    "name":  "gpt-3-davinci",
    "model": "gpt-3-davinci",
    "key":   "<KEY>",
    "url":   "<URL>",
    "default": true,
    "provider": "azure"
  },
  {
    "name":  "gpt-35-turbo",
    "model": "gpt-35-turbo",
    "key":   "<KEY>",
    "url":   "<URL>",
    "default": false,
    "provider": "azure"
  },
  {
```

```
    "name":  "gpt-4",
    "model": "gpt-4",
    "key":   "<KEY>",
    "url":   "<URL>",
    "default": false,
    "provider": "azure"
  },
  {
    "name":  "gpt-4-32k",
    "model": "gpt-4-32k",
    "key":   "<KEY>",
    "url":   "<URL>",
    "default": false,
    "provider": "azure"
  }
]
```

Prerequisites

- You have completed all assumptions and prerequisites from the installation.
- You have created an account in one of the supported OpenAI Providers.
- You have onboarded one or more supported models in your provider and have the appropriate URL and API keys.

Installation

1. Save the above JSON configuration into a JSON file on a machine with access to your Kubernetes cluster. For this example, we will be storing the file with the name of `ai-nlp-models.json`.
2. Create a configmap in the namespace that you are deploying your zebrium-onprem application into, using the following command:

   ```
   kubectl create configmap -n example ai-nlp-models --from-file ai-nlp-models.json
   ```

   In this example, we are naming our configmap `ai-nlp-models` and deploying it into the namespace example. When we created the configmap above, the contents of the file was stored in the configmap under a key corresponding to the filename. So in this example, the key of the configmap is `ai-nlp-models.json`. You can verify this by running the following command:

   ```
   kubectl describe configmap -n example ai-nlp-models
   ```

3. Update your helm override file and include the following section:

```
zebrium-core:
  additionalEnvs:
  - name: AI_NLP_MODELS
    valueFrom:
      configMapKeyRef:
        name: ai-nlp-models
        key: ai-nlp-models.json
```

   In this section, we set the new environment variable `AI_NLP_MODELS` to the value of the configmap we created in step 2. Be sure to update the `name` and `key` references to the appropriate values from step 2.

4. Add any more [configurations] (#additionalConfigurations) or continue with the ***installation process***.

Setting NLP Provider Limits

NLP providers OpenAI and Azure provide Usage Limit settings that allow you to :

1. Set a monthly budget, such as $300 USD per month.
2. Set an email notification threshold, such as $150 USD per month.

It is strongly recommended that you set these values on the NLP provider account to ensure that you stay within a well-defined and limited budget.

Installation

This section covers the steps for installing Zebrium On Prem.

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.
9. Any *additional configurations* have been made.

Installation Steps

Complete the following steps to install Zebrium On Prem.

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 repository 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.
5. The next time you log in to your account, the Zebrium user interface will enable the Deployments feature.

STEP 3: Ingesting Data into your Zebrium On Prem Instance

After you have your Zebrium instance fully initialized, you can configure collectors to send data into your Zebrium instance. Navigate to the **Integrations & Collectors** page from the Settings menu () and select the collector you would like to set up from the list. For more information on how to configure each collector, see *Configuring Log Collectors and File Uploads*.

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 how to contact Zebrium Support for additional help with Zebrium On Prem, when needed.

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://support.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": "000615d0-39a0-0000-0000-00ffff00004"
}
```

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": "000615d0-39a0-0000-0000-00ffff00004",
      "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"
}
```

Copyrights and Trademarks

ScienceLogic, the ScienceLogic logo, and EM7 are trademarks of ScienceLogic, Inc. in the United States, other countries, or both.

Below is a list of trademarks and service marks that should be credited to ScienceLogic, Inc. The ® and ™ symbols reflect the trademark registration status in the U.S. Patent and Trademark Office and may not be appropriate for materials to be distributed outside the United States.

- ScienceLogic™
- EM7™ and em7™
- Simplify IT™
- Dynamic Application™
- Relational Infrastructure Management™

The absence of a product or service name, slogan or logo from this list does not constitute a waiver of ScienceLogic's trademark or other intellectual property rights concerning that name, slogan, or logo.

Please note that laws concerning use of trademarks or product names vary by country. Always consult a local attorney for additional guidance.

Other

If any provision of this agreement shall be unlawful, void, or for any reason unenforceable, then that provision shall be deemed severable from this agreement and shall not affect the validity and enforceability of any remaining provisions. This is the entire agreement between the parties relating to the matters contained herein.

In the U.S. and other jurisdictions, trademark owners have a duty to police the use of their marks. Therefore, if you become aware of any improper use of ScienceLogic Trademarks, including infringement or counterfeiting by third parties, report them to Science Logic's legal department immediately. Report as much detail as possible about the misuse, including the name of the party, contact information, and copies or photographs of the potential misuse to: legal@sciencelogic.com. For more information, see https://sciencelogic.com/company/legal.

800-SCI-LOGIC (1-800-724-5644)

International: +1-703-354-1010