ZEBRIUM “ROOT CAUSE AS A SERVICE” USER GUIDE

Table of Contents

INTRODUCTION 2

KEY CONCEPTS 4

- ML GENERATED “SUGGESTED ALERTS” (ALSO CALLED RCA REPORTS) 4
- KEY EVENTS AND LOG EVENTS OF INTEREST 4
- SERVICE GROUP 4
- DASHBOARD INTEGRATION: 4

HOW IT WORKS (THEORY OF OPERATIONS) 6

NAVIGATION 7

HOW TO GET THE BEST RESULTS 11

INGEST COMPLETE LOGS THAT CONTAIN A REAL PROBLEM 11
BE MINDFUL OF ELAPSED TIME 11
REVIEW SERVICE GROUP SETUP 11
REVIEW RCA SETTINGS 12
USE INTEGRATIONS TO SEPARATE HIGH PRIORITY ALERTS 12
MANAGE ALERT DESTINATIONS 13
USE ROUTING RULES TO CLASSIFY AND ROUTE ALERTS/RCAS 14
ENSURE ZEBRIUM’S ML HIGHLIGHTS SIGNIFICANT EVENTS WHEN THEY HAPPEN NEARBY 15
ENSURE ZEBRIUM’S ML IGNORES SPAMMY EVENTS WHEN THEY HAPPEN NEARBY 16
EDIT RCA FINGERPRINT ALERT KEYS 16
MANUALLY DEFINED ALERTS 18

Introduction

You'll see right away that Zebrium is different than typical observability tools. Monitoring tools are great for dashboards, setting up alerts, and understanding ***when*** things go wrong. Tracing tools help you narrow down ***where*** things are going wrong. But inevitably you have to look into logs to understand ***why*** things went wrong. Traditional log managers are great for searching and alerting, but they rely on you to know what to search for and build alerts for.

But when you are troubleshooting a new problem, the root cause is most often found in a small number of new/rare events and related errors. In these cases you typically don't know exactly what to search for. Zebrium uses machine learning (ML) to automatically identify these correlated clusters of anomalies and errors, and puts them in RCA (root cause) reports.

All you need to do is connect a stream of logs to Zebrium's ML engine. You can do this using one of Zebrium's open source collectors (in a k8s environment this is simply one helm command), or by uploading logs using Zebrium's APIs or CLI. You can then consume Zebrium's ML generated RCA reports in one of the following ways:

1. ***(Recommended)*** Connect it to your observability dashboards – like DataDog, New Relic, Elastic, Grafana, Dynatrace, AppD or ScienceLogic, so that RCA reports from Zebrium will be shown on the same dashboards and can be visually correlated with any spikes or alerts at the same times.
2. Connect it to your incident management tool like OpsGenie, PagerDuty, VictorOps or Slack (check the product for the full list), so that an RCA report is automatically created and sent to the incident management tool. This can work in one of two ways depending on the exact tool -
    a. In some cases (Opsgenie, OpsRamp, VictorOps) Zebrium can actually create an incident in the incident management tool when a root cause report is autonomously detected by Zebrium's ML.
    b. In other cases (Opsgenie, PagerDuty), a bi-directional integration is supported, so that the incident management tool will query Zebrium any time a new incident is created (by another monitoring tool for example). In this scenario Zebrium automatically responds by adding the right root cause report back into the timeline of the incident in the incident management tool.
3. Or, evaluate the feed of auto-detect incident RCA reports, particularly around times where you know things went wrong. You can also force the ML engine to do a deep scan and create a report on demand by clicking the "Scan for Root Cause" button.

In most cases, the default settings will work well, however, Zebrium provides several controls to adjust the sensitivity of its machine learning engine so you can tune signal /

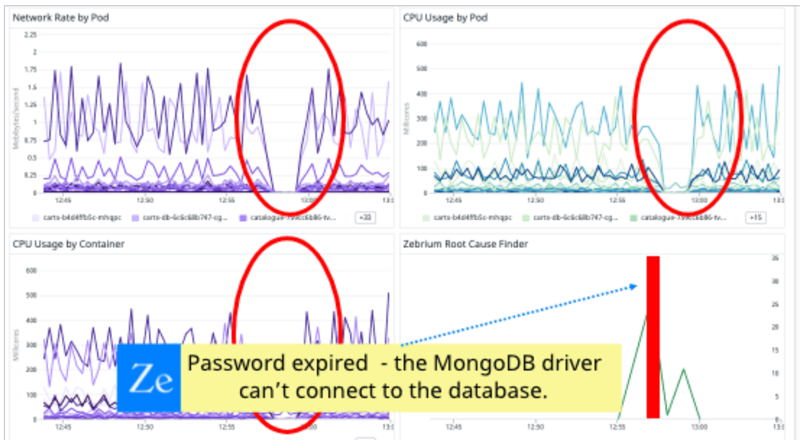
noise to best suit your environment. Zebrium also provides ways to group related log feeds so that the ML only correlates anomalies across related components of an application instance (the components that together define a failure domain). For details on changing the default sensitivity settings and managing deployments, please contact Zebrium.

If you have any questions, please reach out to us via Slack or email hello@zebrium.com.

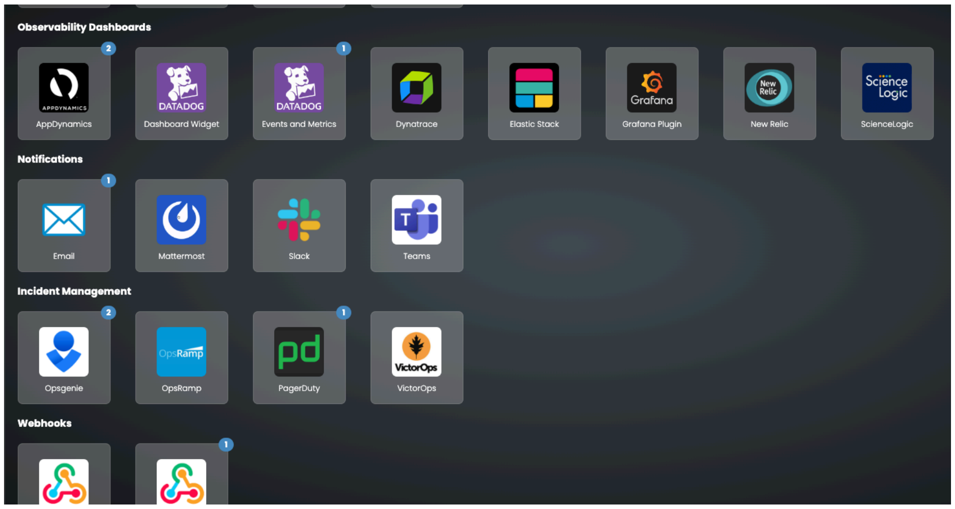
Key Concepts

- **ML Generated "Suggested Alerts" (also called RCA Reports)** – each ML Generated Alert (RCA report) consists of a group of log events that our ML identified as being part of a problem. Each RCA report matches a particular "fingerprint" of log events. It can be annotated with notes, summaries, Jira links, and alert preferences, so that future occurrences of the same type of problem will reflect these preferences and notes.
- **Key Events and Log Events Of Interest**: These events are shown in the summary view and often provide clues as to the nature of the problem. However, in some cases they may not contain anything obvious, and you will have to drill-down into the actual report to find relevant details. The Key Events are also used 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 (e.g. exception, died, failed, could not restart, etc.)
  - You can "edit" the fingerprint of any RCA type, to select different events if you believe they are more useful. Future matches of this type of RCA report will match against your user defined keys, and carry forward your notes, summaries, Jira links as well as alert preferences.
- **Service Group**: is the collection of log types, pods, hosts etc 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. Zebrium's ML will only attempt to correlate anomalies and errors across logs that fall within a Service Group. For more complex applications, it is possible to have multiple Service Groups if there is more than one failure domain.
- **Dashboard Integration:**

  Zebrium's root cause service can be easily integrated into your existing observability dashboards. So for example if you see symptoms of a potential problem in your metrics dashboard, you can have the root cause indicators for the problem surfaced right below that.



To enable this, simply go to the Integrations and Collectors menu under settings, pick your preferred observability dashboard and follow the instructions.



How It Works (Theory of Operations)

When skilled engineers troubleshoot software, they typically look for two things –

1. Where is bad "stuff" happening – i.e. clusters of errors, warnings, stack traces or other indicators of bad outcomes.
2. Were there unusual events upstream, which could help explain these bad outcomes? Examples might be config changes, a new deployment, user actions etc.

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

Zebrium essentially emulates the workflow of a skilled engineer. It automatically builds a catalog of all the event types generated by the software. Then it tracks the patterns of each event type in each log stream (e.g. the logs generated by a specific container, pod or host), and automatically identifies unusual and bad events. Finally it identifies unusually correlated collections of rare and bad events that appear to be due to the same incident. Zebrium scores each such collection based on a combination of how rare the underlying events are, and how bad (e.g. how many warnings / errors). Each collection of such events is "fingerprinted" as a unique type of issue, and the ones that rise above a specified threshold can be considered a potential root cause report, and are summarized using NLP.

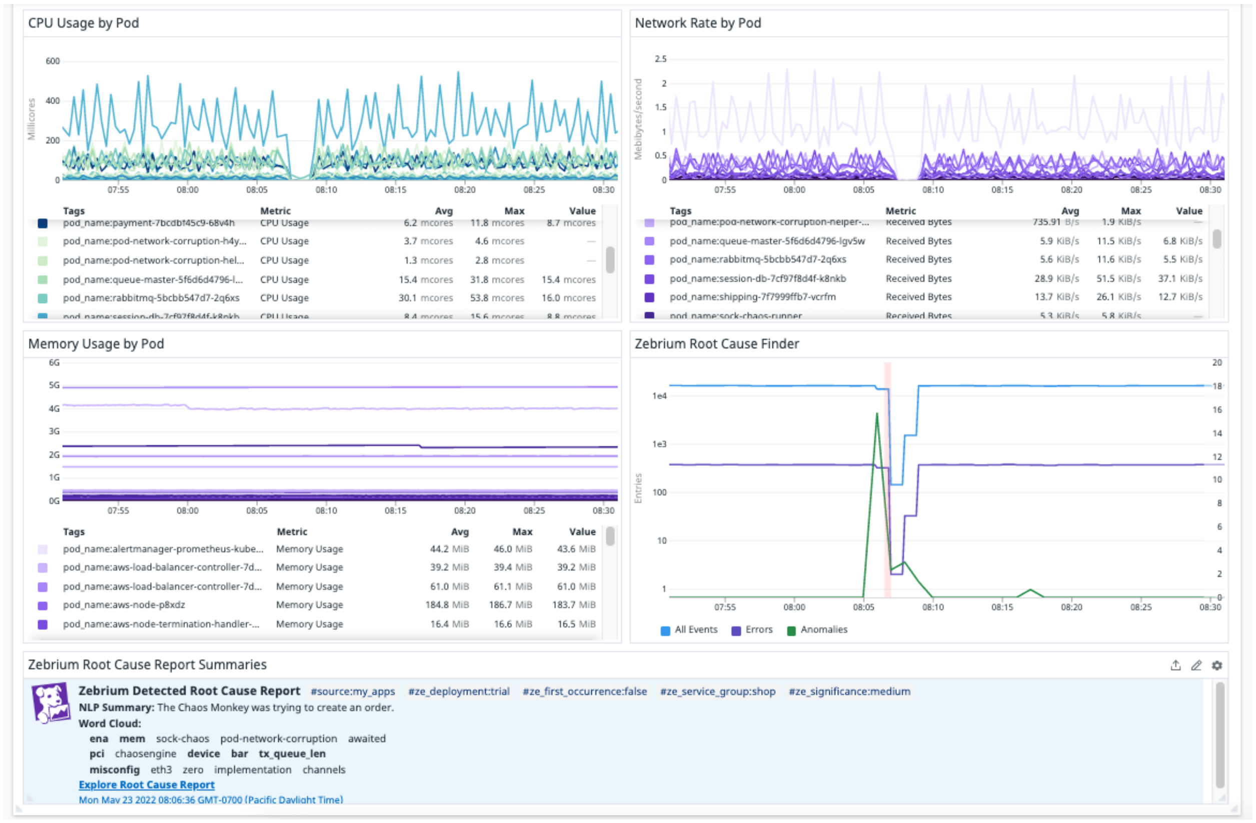
When the ML detects one of these unusual clusters, it will generate a suggested alert. A suggested alert contains metadata (e.g. title, description, etc.), a root cause report (set of correlated log lines that likely help to explain a problem) and a suggested alert rule (one or two log events types that form the signature for this type of alert). A user can choose to "accept" or "reject" a suggested alert. If a suggested alert is accepted, the user can edit the metadata and alert rule and can also decide upon the action to take if the same kind of alert type occurs again (e.g. to send a Slack, email or other notification).

Users can affect their results a few ways –

- They change the sensitivity threshold – making it more sensitive to catch subtle issues, or making it less sensitive to minimize "noise". Note that in the latter case the software will still detect the less significant issues and create reports for them, but will not show them to the user until settings are changed.
- They can filter the results: For example by default only the first occurrence of each "incident type" is visible on dashboards, alert channels etc, unless a user deems it alert worthy.

Navigation

- Assuming you have connected Zebrium to your observability dashboards, the simplest way to start is to look at the Zebrium widget when you have a problem, and click on the Zebrium detected root cause report that lines up with the same point in time.



CPU Usage by Pod

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

Network Rate by Pod

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

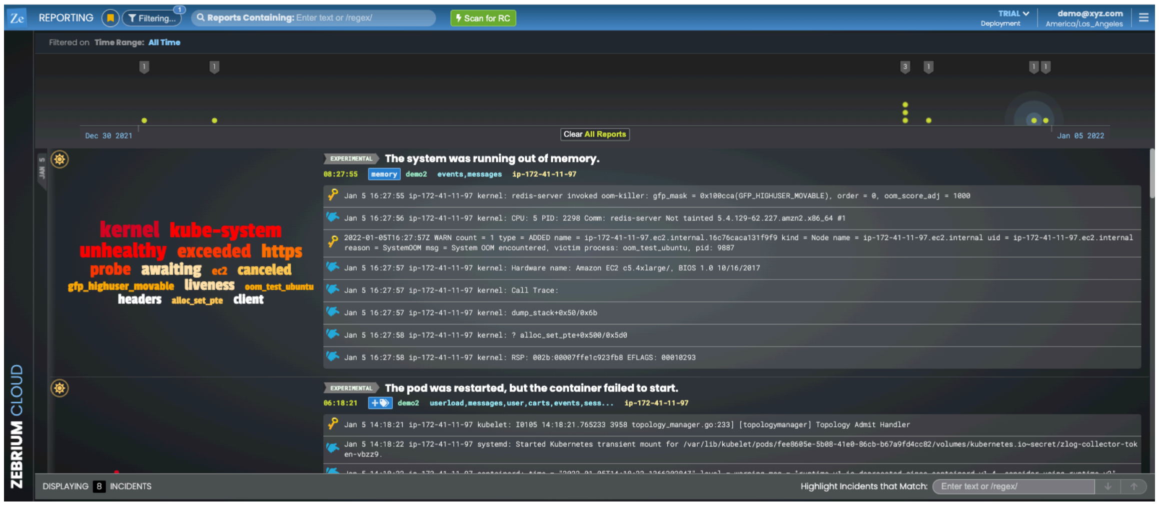
Memory Usage by Pod

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

Zebrium Root Cause Report Summaries

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

- For more details, or to take action on one of these reports, you can click on the URL to jump straight to the detailed report.
- Alternatively, you can start from the Zebrium UI (the main page showing a list of all known Alerts / RCA reports), and drill down on the report that matches the time frame (and other meta-data filters) when you suspect a problem.


REPORTING
Filtering...
Reports Containing: Enter text or /regex/
Scan for RC
TRIAL Deployment
demo@xyz.com America/Los_Angeles
Filtered on Time Range: All Time
Dec 30 2021
Clear All Reports
Jan 05 2022
ZEBRIUM CLOUD
kernel kube-system unhealthy exceeded https probe awaiting ec2 canceled gfp_highuser_movable liveness oom_test_ubuntu headers alloc_set_pte client
DISPLAYING 8 INCIDENTS
Highlight Incidents that Match: Enter text or /regex/

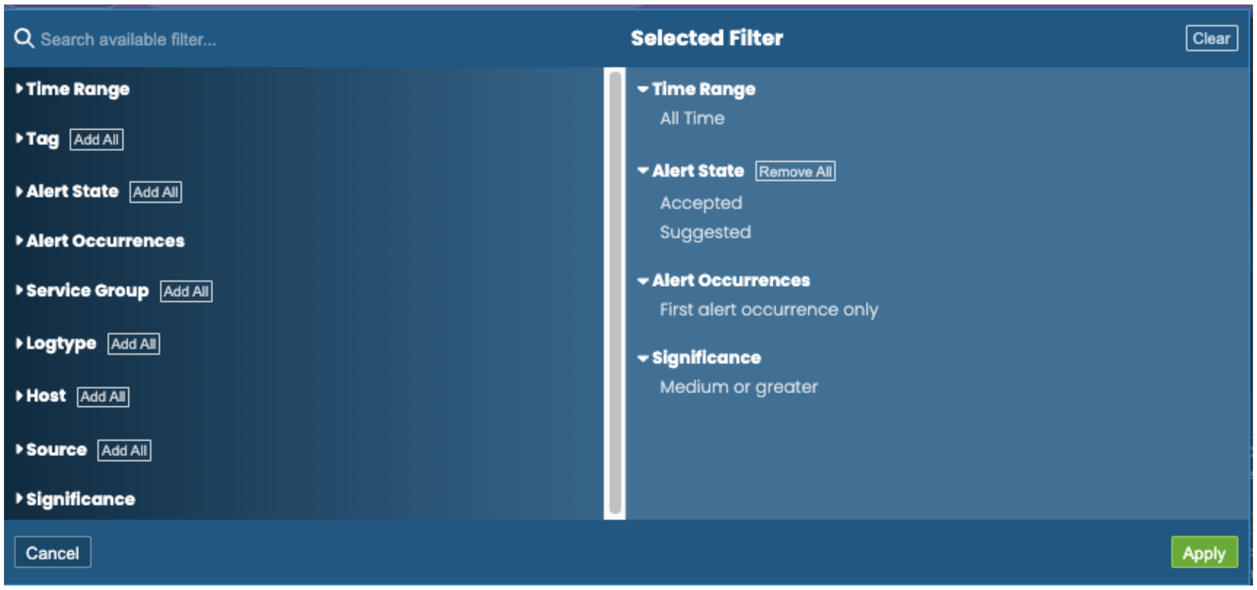
EXPERIMENTAL **The system was running out of memory.**
08:27:55 memory demo2 events,messages ip-172-41-11-97

```
Jan 5 16:27:55 ip-172-41-11-97 kernel: redis-server invoked oom-killer: gfp_mask = 0x100cca(GFP_HIGHUSER_MOVABLE), order = 0, oom_score_adj = 1000
Jan 5 16:27:56 ip-172-41-11-97 kernel: CPU: 5 PID: 2298 Comm: redis-server Not tainted 5.4.129-62.227.amzn2.x86_64 #1
2022-01-05T16:27:57Z WARN count = 1 type = ADDED name = ip-172-41-11-97.ec2.internal.16c76caca131f9f9 kind = Node name = ip-172-41-11-97.ec2.internal uid = ip-172-41-11-97.ec2.internal
reason = SystemOOM msg = System OOM encountered, victim process: oom_test_ubuntu, pid: 9887
Jan 5 16:27:57 ip-172-41-11-97 kernel: Hardware name: Amazon EC2 c5.4xlarge/, BIOS 1.0 10/16/2017
Jan 5 16:27:57 ip-172-41-11-97 kernel: Call Trace:
Jan 5 16:27:57 ip-172-41-11-97 kernel: dump_stack+0x50/0x6b
Jan 5 16:27:58 ip-172-41-11-97 kernel: ? alloc_set_pte+0x500/0x5d0
Jan 5 16:27:58 ip-172-41-11-97 kernel: RSP: 002b:00007ffe1c923fb8 EFLAGS: 00010293
```

EXPERIMENTAL **The pod was restarted, but the container failed to start.**
06:18:21 demo2 userload,messages,user,carts,events,sess... ip-172-41-11-97

```
Jan 5 14:18:21 ip-172-41-11-97 kubelet: I0105 14:18:21.765233 3958 topology_manager.go:233] [topologymanager] Topology Admit Handler
Jan 5 14:18:22 ip-172-41-11-97 systemd: Started Kubernetes transient mount for /var/lib/kubelet/pods/fee8605e-5b08-41e0-86cb-b67a9fd4cc82/volumes/kubernetes.io~secret/zlog-collector-tok
en-vbzz9.
```

- The list is sorted by time (most recent first), and can filtered by time range, log types, service groups, hosts, tags and significance. Any RCA reports that match these attributes will be shown in the filtered view.



Time Range
All Time

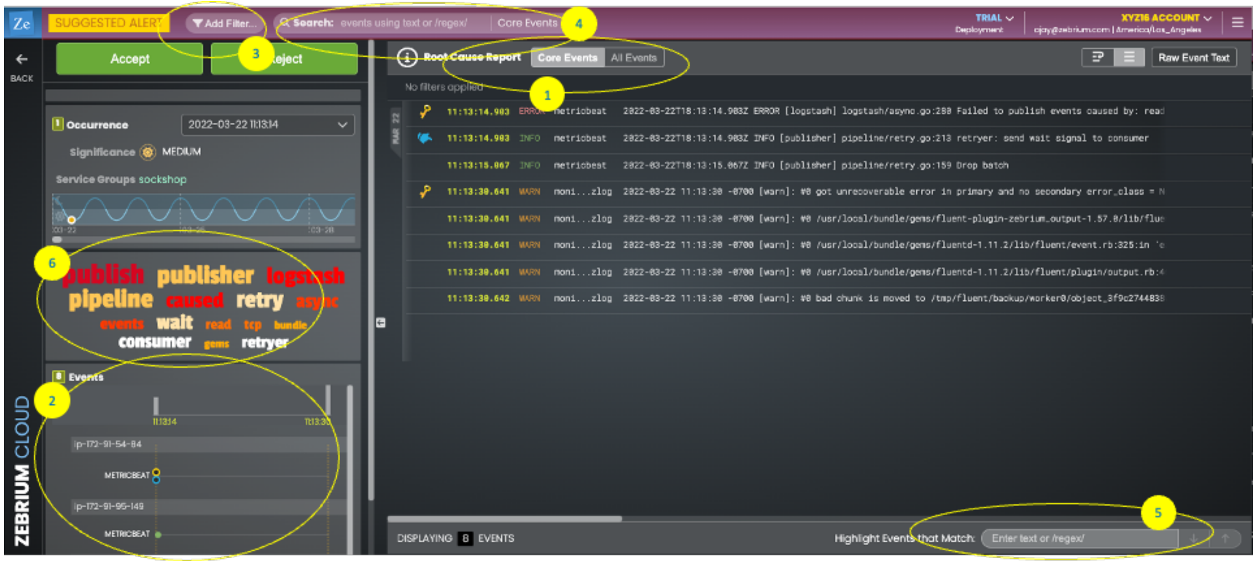
Alert State
Accepted
Suggested

Alert Occurrences
First alert occurrence only

Significance
Medium or greater

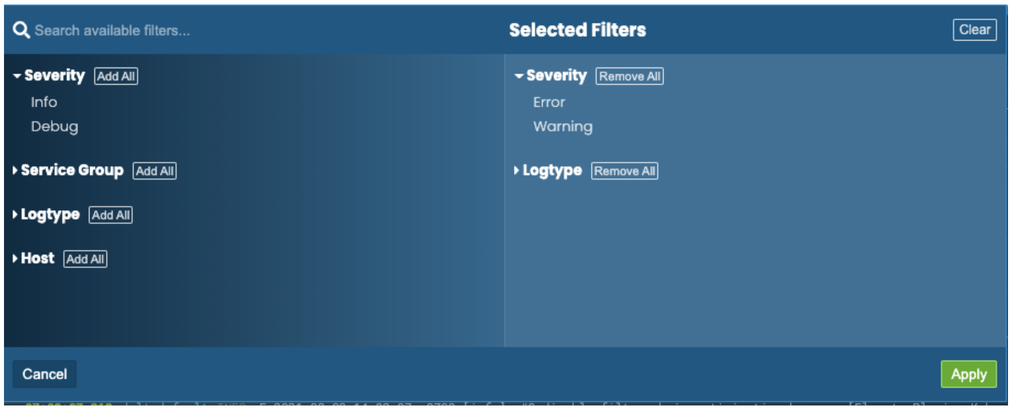
- The list shows the host, log types (which typically match container names), and the key events that describe a given RCA report
- **Important:** By default, the list only shows the first occurrence of a new type of RCA. If the same problem has occurred previously, you can check for repeat occurrences by un-filtering the "first occurrence only" field
- If you don't see a report a time of interest where you believe a problem occurred, it's likely because it was suppressed by the existing significance settings. You can force a report to be created via a deeper scan by clicking the "scan for root cause" button (#2 above), and specifying a time of interest.

Clicking on any of the reports will take you to a “RCA details” view. In this view, you’ll be shown a more complete list of log events compiled by our ML to describe this particular problem.

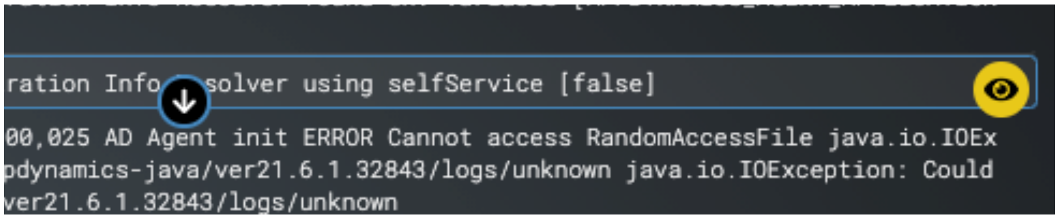


```
11:13:14.903 ERROR metricbeat  2022-03-22T18:13:14.983Z ERROR [logstash] logstash/async.go:280 Failed to publish events caused by: read
11:13:14.903 INFO  metricbeat  2022-03-22T18:13:14.983Z INFO [publisher] pipeline/retry.go:213 retryer: send wait signal to consumer
11:13:15.067 INFO  metricbeat  2022-03-22T18:13:15.067Z INFO [publisher] pipeline/retry.go:159 Drop batch
11:13:30.641 WARN  moni...zlog 2022-03-22 11:13:30 -0700 [warn]: #0 got unrecoverable error in primary and no secondary error_class = N
11:13:30.641 WARN  moni...zlog 2022-03-22 11:13:30 -0700 [warn]: #0 /usr/local/bundle/gems/fluent-plugin-zebrium_output-1.57.0/lib/flue
11:13:30.641 WARN  moni...zlog 2022-03-22 11:13:30 -0700 [warn]: #0 /usr/local/bundle/gems/fluentd-1.11.2/lib/fluent/event.rb:325:in `e
11:13:30.641 WARN  moni...zlog 2022-03-22 11:13:30 -0700 [warn]: #0 /usr/local/bundle/gems/fluentd-1.11.2/lib/fluent/plugin/output.rb:4
11:13:30.642 WARN  moni...zlog 2022-03-22 11:13:30 -0700 [warn]: #0 bad chunk is moved to /tmp/fluent/backup/worker0/object_3f9c2744838
```

- You will also see a visualization of the log events (#2 above), partitioned by micro-service and host name, with the horizontal location of the dots based on chronological sequence.
- You will also see a wordcloud of selected keywords picked out from the report – with the font size denoting how rare it is, and the color denoting how “bad” the underlying events were (for example a critical event is worse than an error).
- Typically, the “core” list in an RCA report will contain somewhere between 5-25 log events. However, you can then drill down by clicking on the “All Events” button (#1 above) to see a much longer list (typically a couple of hundred) of anomalies, warnings and errors surrounding this core list of events.
- You can filter events by severity (e.g. error, warning etc), log type, host etc – refer to #3 above and the filter view below.

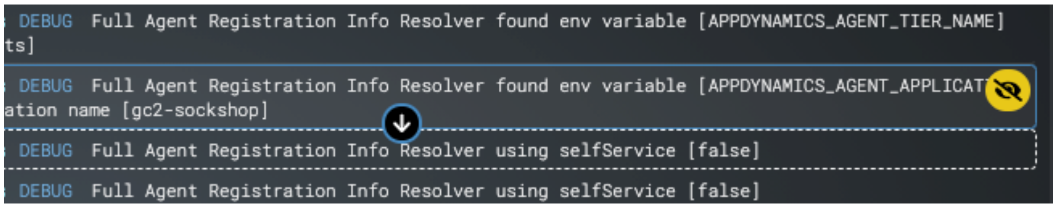


- You can further filter the log events at any zoom level, by matching against a text string or a (PCRE2 compliant) regex. *Note: regex filters should use the syntax "/regex/".*
- You can also highlight any desired alphanumeric strings within the visible log events using the "find" field at the bottom of the screen (#5 in the figure above).
- Alternatively, you can drill down on logs from a particular host or pod by clicking on the "peek" button. This is similar to looking at the log file for a single log generator.



```
ration Info  solver using selfService [false]
00,025 AD Agent init ERROR Cannot access RandomAccessFile java.io.IOEx
pdynamics-java/ver21.6.1.32843/logs/unknown java.io.IOException: Could
ver21.6.1.32843/logs/unknown
```

- To exit the peek mode, click the "clear" button.



```
 DEBUG  Full Agent Registration Info Resolver found env variable [APPDYNAMICS_AGENT_TIER_NAME]
ts]
 DEBUG  Full Agent Registration Info Resolver found env variable [APPDYNAMICS_AGENT_APPLICAT
ation name [gc2-sockshop]
 DEBUG  Full Agent Registration Info Resolver using selfService [false]
 DEBUG  Full Agent Registration Info Resolver using selfService [false]
```

- Finally, you can add notes, a title and an issue tracking URL (e.g. a pointer to Jira) by clicking on the "edit" button.

How To Get The Best Results

Zebrium's ML will start working within a few minutes of logs arriving, detecting RCAs for problems that occur in your environment, and presenting them as "Suggested Alerts" within the Zebrium UI. Signal to noise ratio improves with time, and typically achieves a good level in about 24 hours. If you're not satisfied with the quality of the results, there are a few things you can do:

Ingest Complete Logs That Contain A Real Problem

Sometimes users connect Zebrium to (or upload a set of logs from) a software environment in steady state, where nothing bad happens. In such cases the logs don't actually contain any unusual events or significant errors. Naturally in such cases Zebrium will not find a useful (or perhaps any) root cause report.

Also sometimes users will upload a subset of the logs (in an extreme scenario a single log file), which also degrades Zebrium's ability to create meaningful root cause reports. For good results, please 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 care must be taken to ensure the log collection is complete (anything a human would need for troubleshooting should be included), the files are tagged with correct metadata, and 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 ML 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 (say 1-2 hours after setup), but be prepared for noisier results.

Also, if the same kind of problem keeps occurring within a day, Zebrium may consider it "typical", and therefore 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 don't realize that default settings will only show the first occurrence of the problem (these settings can be changed – in the UI, or under the settings menu).

Review Service Group Setup

Service groups are a way to inform the Zebrium ML about the failure domains within your log streams. Only log streams/files coming from services/containers/hosts that could affect each other should be placed in the same service group. If you see log events in a RCA 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.

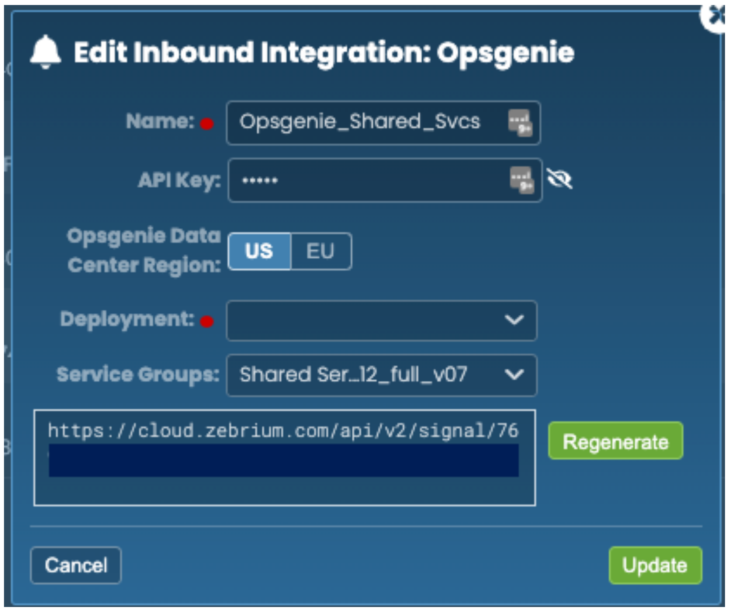
Review RCA Settings

A handful of ML settings are visible in the Root Cause Settings menu. The most common one to consider adjusting is the "Significance" setting. Think of this like a filter level – the higher the significance setting, the more selective the ML will be in alerting. A way to think about significance is a combination of rareness and badness – the higher the significance setting, the more rare and bad (e.g. log severity level) the RCA 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/string that have a special meaning for your software stack (contact us if you need either of these).

There are other settings that may 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

Zebrium's ML creates RCA reports when it identifies clusters of rare events and bad events (i.e. events with higher log severity, like warning, error) that are highly unlikely to occur by random chance. Nevertheless, all such clusters may not be due to high priority (e.g. 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, VictorOps etc. When an incident is created in one of these tools (say due to an alert from some other observability tool for example), the integration signals Zebrium to analyze logs from the same environment, and respond with a root cause report that is automatically appended to the incident (e.g. in the timeline or notes fields). By doing so, Zebrium reports can be matched up with incident priorities that were already assigned based on other rules. More details can be found here and here.



Edit Inbound Integration: Opsgenie

Name: Opsgenie_Shared_Svcs

API Key: •••••

Opsgenie Data Center Region: [✓ US] EU

Deployment:

Service Groups: Shared Ser_12_full_v07

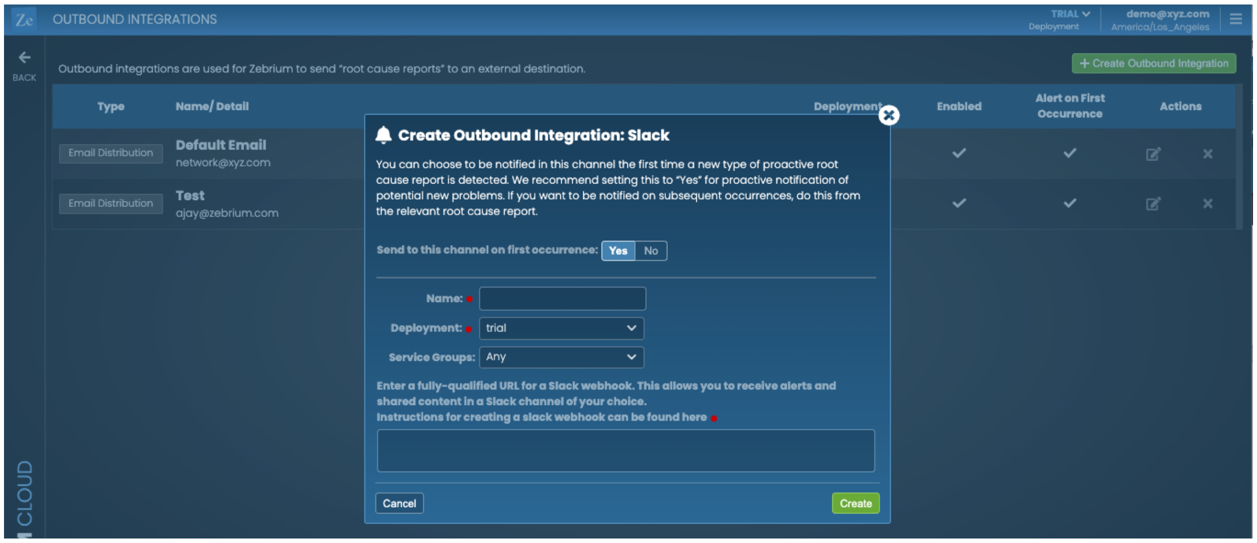
```
https://cloud.zebrium.com/api/v2/signal/76
```
[redacted]

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

Note that Zebrium's ML will continue to proactively detect RCAs (even when there is no signal from a 3rd 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 simplest way is to set up alert channels for every combination of deployments and/or service groups that you would like to *route uniquely*.



Outbound integrations are used for Zebrium to send "root cause reports" to an external destination.

Type	Name/ Detail	Deployment	Enabled	Alert on First Occurrence	Actions
Email Distribution	Default Email network@xyz.com		✓	✓	
Email Distribution	Test ajay@zebrium.com		✓	✓	

Create Outbound Integration: Slack

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

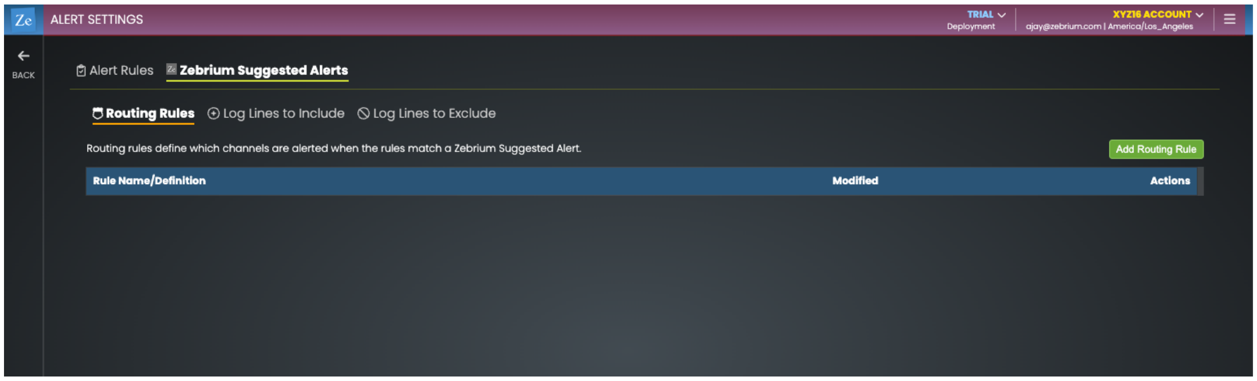
Send to this channel on first occurrence: [✓ Yes] No

Name:
Deployment: trial
Service Groups: Any

Enter a fully-qualified URL for a Slack webhook. This allows you to receive alerts and shared content in a Slack channel of your choice.
Instructions for creating a slack webhook can be found here

Use Routing Rules To Classify And Route Alerts/RCAs

An even more powerful way to manage and route alerts is to setup "Routing Rules" under the "Alert Settings" menu, and clicking on the "Zebrium Suggested Alerts" tab.

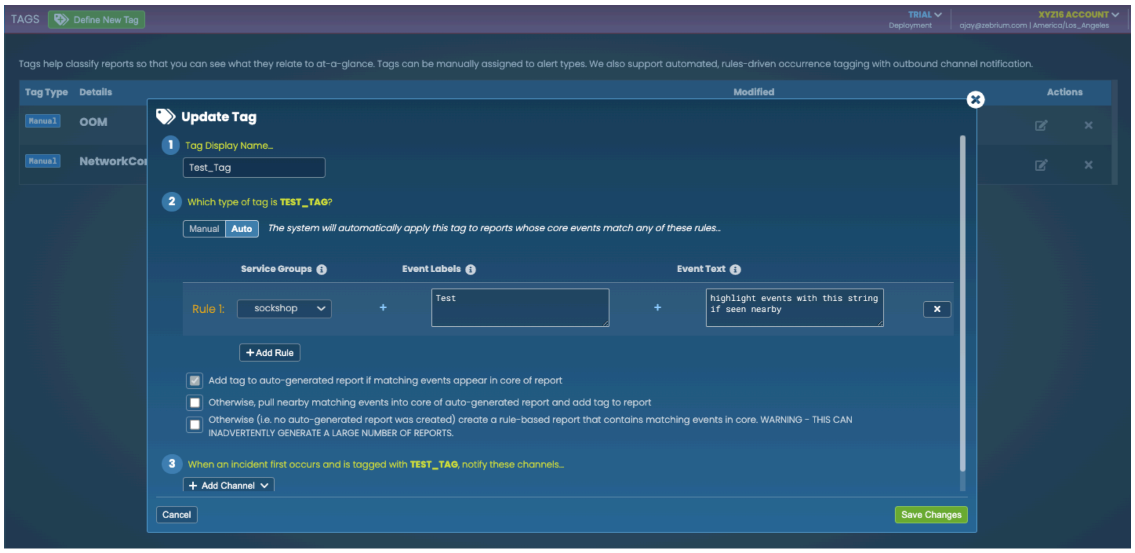


Routing rules define which channels are alerted when the rules match a Zebrium Suggested Alert.

Rule Name/Definition	Modified	Actions

This allows you to set up rules regarding service group, event labels (e.g. k8s app or pod name), as well as string matches in the actual log event. Each routing rule lets you automatically triage Alerts/RCA reports, and send them to the appropriate destination.

For example you may want to create a "Networking" tag for RCAs which involve logs from k8s 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.



Tags help classify reports so that you can see what they relate to at-a-glance. Tags can be manually assigned to alert types. We also support automated, rules-driven occurrence tagging with outbound channel notification.

Tag Type	Details	Modified	Actions
Manual	OOM		
Manual	NetworkCo		

Update Tag

1 Tag Display Name… Test_Tag

2 Which type of tag is **TEST_TAG**?

Manual [✓ Auto] *The system will automatically apply this tag to reports whose core events match any of these rules…*

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

+Add Rule

- [✓] Add tag to auto-generated report if matching events appear in core of report
- [ ] Otherwise, pull nearby matching events into core of auto-generated report and add tag to report
- [ ] Otherwise (i.e. no auto-generated report was created) create a rule-based report that contains matching events in core. WARNING - THIS CAN INADVERTENTLY GENERATE A LARGE NUMBER OF REPORTS.

3 When an incident first occurs and is tagged with **TEST_TAG**, notify these channels…

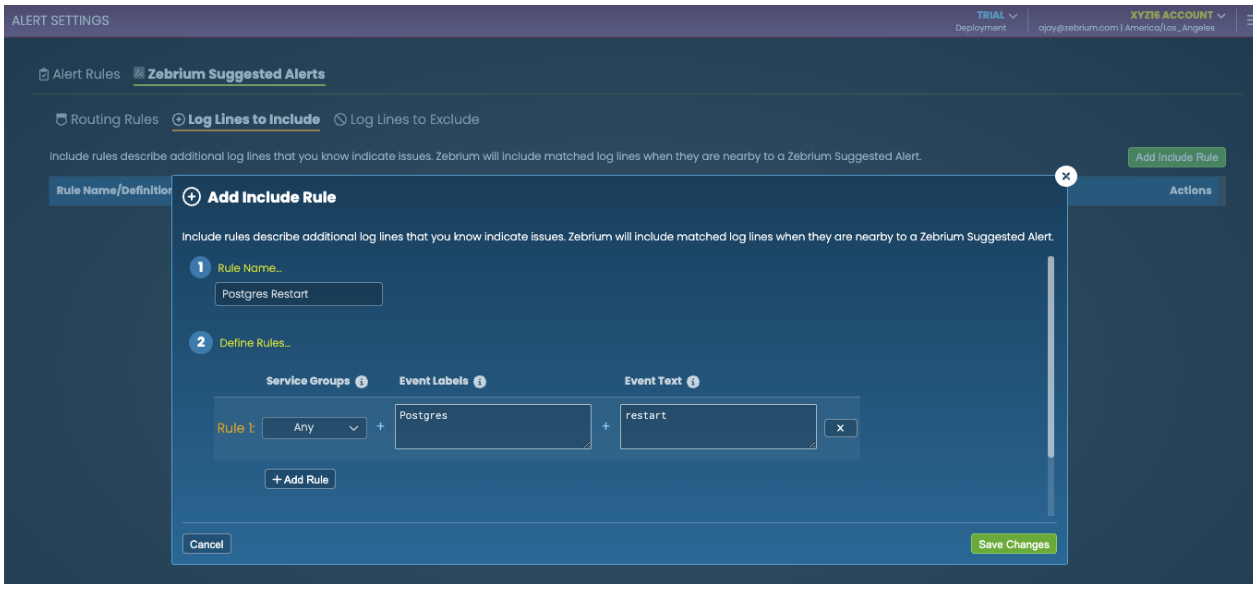
+ Add Channel

Cancel | Save Changes

Ensure Zebrium's ML Highlights Significant Events When They Happen Nearby

Let's say 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/RCA, you may want to ensure that it does get pulled into the core event list of any RCA. If you want this outcome, simply click on the "Log Lines To Include" 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 ML detects unusual events within the vicinity of this restart event.



Include rules describe additional log lines that you know indicate issues. Zebrium will include matched log lines when they are nearby to a Zebrium Suggested Alert.

Add Include Rule

Include rules describe additional log lines that you know indicate issues. Zebrium will include matched log lines when they are nearby to a Zebrium Suggested Alert.

1 Rule Name… Postgres Restart

2 Define Rules…

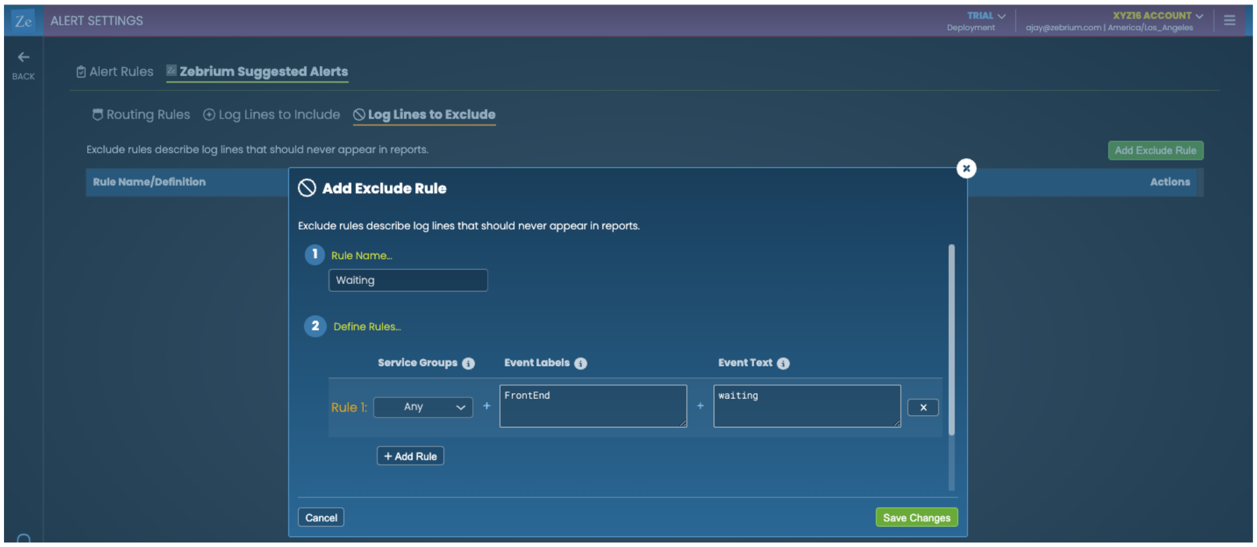
	Service Groups	Event Labels	Event Text
Rule 1:	Any	Postgres	restart

+Add Rule

Cancel | Save Changes

Ensure Zebrium’s ML Ignores Spammy Events When They Happen Nearby

This does the opposite of the previous feature. Let’s say your engineers know that a specific log event is spammy 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 inform the ML engine to exluce these events.



Exclude rules describe log lines that should never appear in reports.

Exclude rules describe log lines that should never appear in reports.

1 Rule Name...
Waiting

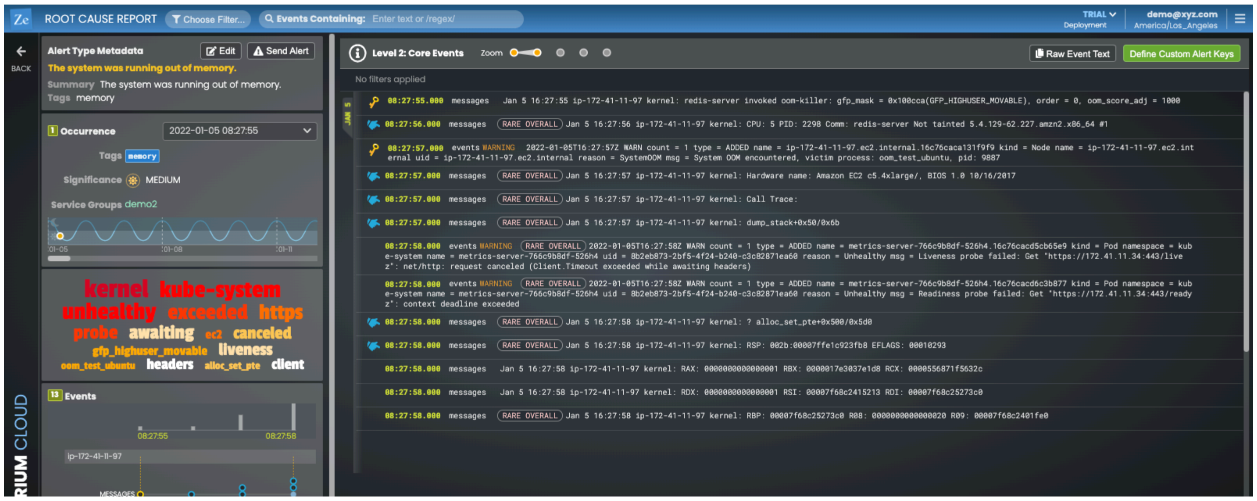
2 Define Rules...

	Service Groups	Event Labels	Event Text
Rule 1:	Any	FrontEnd	waiting

Edit RCA Fingerprint Alert Keys

One of the benefits of Zebrium’s ML is that it will track repeat occurrences of a particular type of issue – and will reuse any user added notes, Jira links, and alert preferences that were set the first time. *Note: if no alert preference is set for a particular issue upon the first occurrence, no alerts will be sent for subsequent (repeat) occurrences of the same type of issue.*

A unique issue is identified by log events that are considered fingerprints of that type of issue RCA. The fingerprint events are annotated with the key icons. Zebrium’s ML will auto-identify 2 such events based on rareness and severity. However if you think different events should define the “fingerprint” of an RCA, you can edit these using the “Define Custom Alert Keys” button in the details view (top right corner). In the current software version you can choose 1 key, or 2.


ROOT CAUSE REPORT
Choose Filter...
Events Containing: Enter text or /regex/
TRIAL Deployment
demo@xyz.com America/Los_Angeles
BACK
Edit
Send Alert
Level 2: Core Events
Zoom
Raw Event Text
Define Custom Alert Keys
No filters applied
kernel kube-system unhealthy exceeded https probe awaiting ec2 canceled gfp_highuser_movable liveness oom_test_ubuntu headers alloc_set_pte client
Events
ip-172-41-11-97
MESSAGES

Alert Type Metadata

The system was running out of memory.

Summary The system was running out of memory.

Tags memory

1 Occurrence 2022-01-05 08:27:55

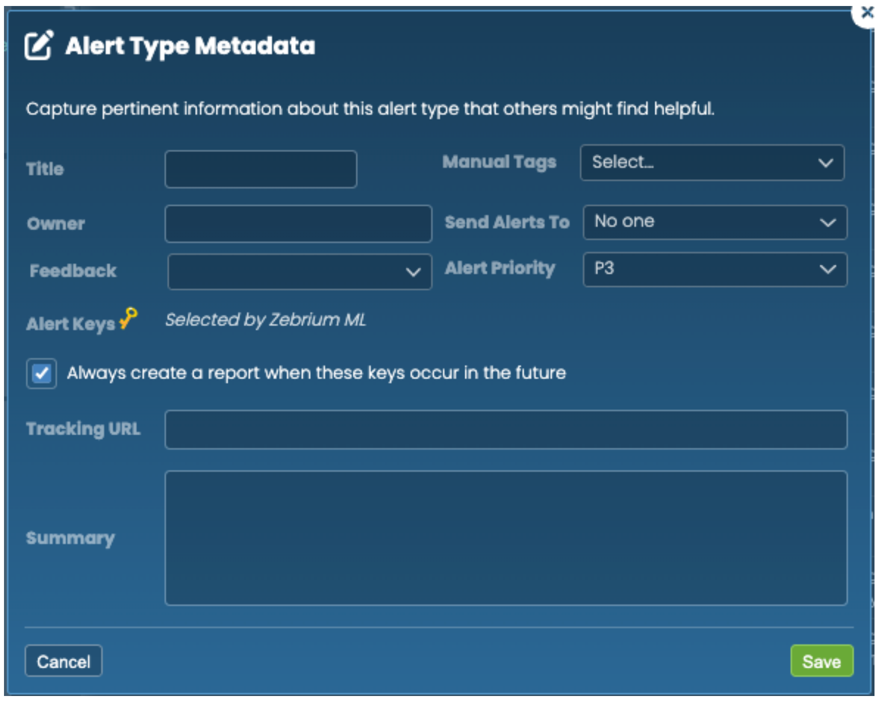
Tags memory

Significance MEDIUM

Service Groups demo2

```
08:27:55.000 messages   Jan 5 16:27:55 ip-172-41-11-97 kernel: redis-server invoked oom-killer: gfp_mask = 0x100cca(GFP_HIGHUSER_MOVABLE), order = 0, oom_score_adj = 1000
08:27:56.000 messages  RARE OVERALL Jan 5 16:27:56 ip-172-41-11-97 kernel: CPU: 5 PID: 2298 Comm: redis-server Not tainted 5.4.129-62.227.amzn2.x86_64 #1
08:27:57.000 events WARNING  2022-01-05T16:27:57Z WARN count = 1 type = ADDED name = ip-172-41-11-97.ec2.internal.16c76caca131f9f9 kind = Node name = ip-172-41-11-97.ec2.int
ernal uid = ip-172-41-11-97.ec2.internal reason = SystemOOM msg = System OOM encountered, victim process: oom_test_ubuntu, pid: 9887
08:27:57.000 messages  RARE OVERALL Jan 5 16:27:57 ip-172-41-11-97 kernel: Hardware name: Amazon EC2 c5.4xlarge/, BIOS 1.0 10/16/2017
08:27:57.000 messages  RARE OVERALL Jan 5 16:27:57 ip-172-41-11-97 kernel: Call Trace:
08:27:57.000 messages  RARE OVERALL Jan 5 16:27:57 ip-172-41-11-97 kernel: dump_stack+0x50/0x6b
08:27:58.000 events WARNING  RARE OVERALL 2022-01-05T16:27:58Z WARN count = 1 type = ADDED name = metrics-server-766c9b8df-526h4.16c76cacd5cb65e9 kind = Pod namespace = kub
e-system name = metrics-server-766c9b8df-526h4 uid = 8b2eb873-2bf5-4f24-b240-c3c82871ea60 reason = Unhealthy msg = Liveness probe failed: Get "https://172.41.11.34:443/live
z": net/http: request canceled (Client.Timeout exceeded while awaiting headers)
08:27:58.000 events WARNING  RARE OVERALL 2022-01-05T16:27:58Z WARN count = 1 type = ADDED name = metrics-server-766c9b8df-526h4.16c76cacd6c3b877 kind = Pod namespace = kub
e-system name = metrics-server-766c9b8df-526h4 uid = 8b2eb873-2bf5-4f24-b240-c3c82871ea60 reason = Unhealthy msg = Readiness probe failed: Get "https://172.41.11.34:443/ready
z": context deadline exceeded
08:27:58.000 messages  RARE OVERALL Jan 5 16:27:58 ip-172-41-11-97 kernel: ? alloc_set_pte+0x500/0x5d0
08:27:58.000 messages  RARE OVERALL Jan 5 16:27:58 ip-172-41-11-97 kernel: RSP: 002b:00007ffe1c923fb8 EFLAGS: 00010293
08:27:58.000 messages   Jan 5 16:27:58 ip-172-41-11-97 kernel: RAX: 0000000000000001 RBX: 0000017e3037e1d8 RCX: 0000556871f5632c
08:27:58.000 messages   Jan 5 16:27:58 ip-172-41-11-97 kernel: RDX: 0000000000000001 RSI: 00007f68c2415213 RDI: 00007f68c25273c0
08:27:58.000 messages  RARE OVERALL Jan 5 16:27:58 ip-172-41-11-97 kernel: RBP: 00007f68c25273c0 R08: 0000000000000020 R09: 00007f68c2401fe0
```

Further, if you want to catch this exact issue ***every*** time it occurs (not just the times the events are anomalous and unusually clustered), make sure to check this option by clicking on the "Edit" menu (top left corner of the report details page). This option is useful for issues that happen very frequently (many times a day) to make sure the ML does not ignore them because they are so common.



Alert Type Metadata

Capture pertinent information about this alert type that others might find helpful.

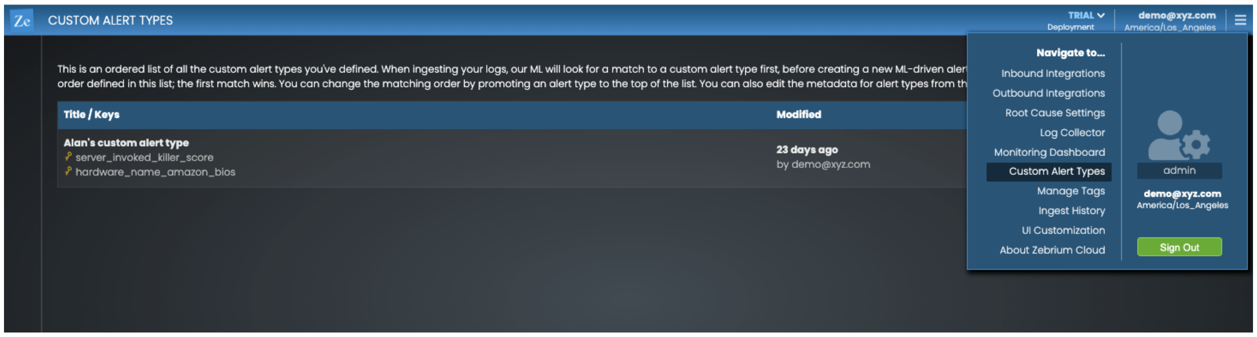
Title		Manual Tags	Select...
Owner		Send Alerts To	No one
Feedback		Alert Priority	P3
Alert Keys	Selected by Zebrium ML		

[✓] Always create a report when these keys occur in the future

Tracking URL

Summary

If you do end up customizing the fingerprints for some reports (by choosing different key events than chosen by the ML), you can also view the list of all your custom keys under the settings menu (see below).



This is an ordered list of all the custom alert types you've defined. When ingesting your logs, our ML will look for a match to a custom alert type first, before creating a new ML-driven aler
order defined in this list; the first match wins. You can change the matching order by promoting an alert type to the top of the list. You can also edit the metadata for alert types from th

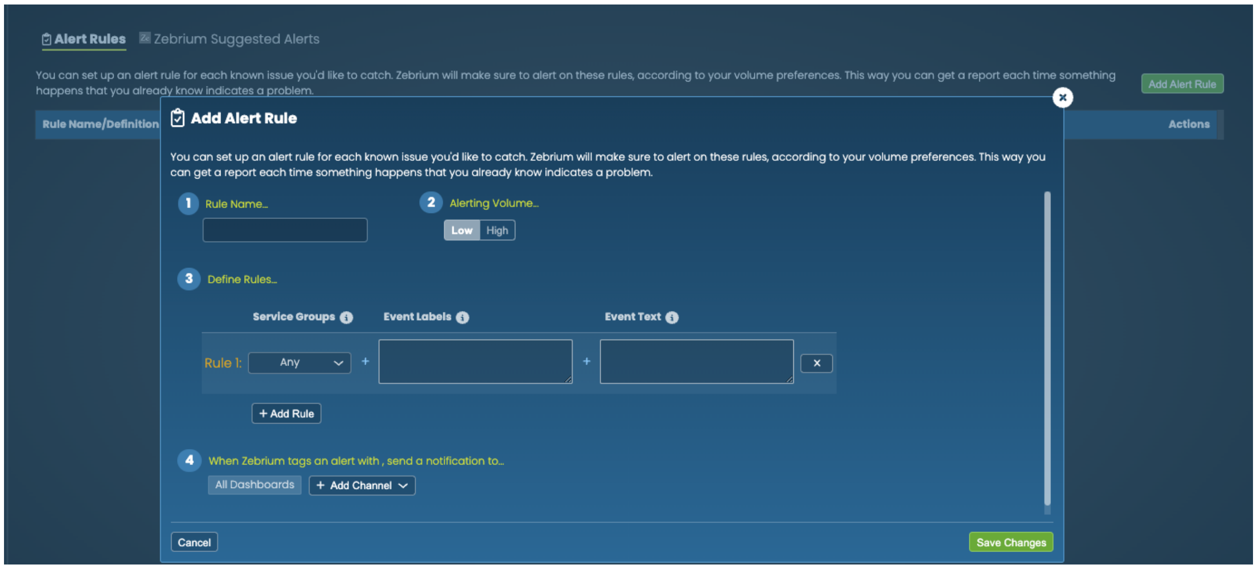
Title / Keys	Modified
Alan's custom alert type server_invoked_killer_score hardware_name_amazon_bios	**23 days ago** by demo@xyz.com

Manually Defined Alerts

Let's say the example of the log event from #2 is so critical that you want to be notified EVERY TIME the matching rule is seen in the logs. If this particular event is a common occurrence, the ML might start to ignore it because it does not seem unusual.

To ensure you always catch it, use the "Alert Rule" option, under the "Alert Settings" menu. Note that this is similar to a simple alert rule in any log manager – it does not rely on ML.

Warning: Also note that if you are not careful with the definition of your rule, you may inadvertently generate a LOT of alerts. To mitigate this possibility, use the "alerting volume" option (#2 in the picture below). The "Low" volume option will only send a maximum of one alert per day for this rule, while the "High" volume option will alert up to once per hour.



You can set up an alert rule for each known issue you'd like to catch. Zebrium will make sure to alert on these rules, according to your volume preferences. This way you can get a report each time something happens that you already know indicates a problem.

You can set up an alert rule for each known issue you'd like to catch. Zebrium will make sure to alert on these rules, according to your volume preferences. This way you can get a report each time something happens that you already know indicates a problem.