Machine Learning for k8s Logs and Metrics

AUTOMATING INCIDENT AND ROOT-CAUSE DETECTION

Larry Lancaster
Founder and CTO
Zebrium
Machine data is my life

- NetApp - Engineering Informatics
- EMC / Data Domain - Product Analytics
- Glassbeam - Chief Technology Officer
- Nimble Storage - Chief Data Scientist
- Zebrium - Founder and CTO
20 YEARS AGO

Shrink-Wrap:

1 incident 1 user
1 incident 1 monolith
1 incident 10 logfiles

Log use for root-cause:
index and search
20 YEARS AGO

Shrink-Wrap:

1 incident 1 user
1 incident 1 monolith
1 incident 10 logfiles

Log use for root-cause:
index and search

TODAY

SaaS:

1 incident 100K users
1 incident 100 services
1 incident 1K logstreams

Log use for root-cause:
still index and search(!)
"The proportion of medium performers is up. Some are likely improved low performers, while others may be high performers who dropped as they struggled with increased complexity."

Source: State of DevOps (2019)
Automation can't fix it

"Time to restore service performance stayed the same for both elite and low performers when compared to the previous year."

Source: State of DevOps (2019)
Our vision

Autonomous RCA will save the world from the cost of complexity.
What I want from a tool

Automatically Detect Incidents
Without Setting Up Manual Alert Rules

Automatically Find Root Cause
Without Manually Searching Across GBs of Logs
My requirements

- Arbitrary application
- Arbitrary runtime
- Arbitrary infrastructure
- Arbitrary environment

- Zero required tracing
- Zero required training
- Zero required alert rules

Is it really too much to ask? :}
Why so harsh?

Because complexity also means:

Manual inputs may not scale
Stack assumptions may not hold
Outrageous opinion

Autonomous RCA has to start with logs.
Logs are self-describing

A free-text log tells a story:

[syslog] 2020-12-10 04:17:37 mars systemd[1]: Stopped PostgreSQL RDBMS.
...<191 lines>...
[jira] Caused by: org.postgresql.util.PSQLException: FATAL: terminating connection due to administrator command
People use logs for RCA

...so why aren't they better at helping us monitor?
Log monitoring today

Setup Agents / Exporters / Parsers

Configure Alert Rules for Known Symptoms

Get alerted or otherwise detect incident

Manually Search Logs for Root Cause

Resolve Incident

Tune Alerts & Build Dashboards

SLOW (MTTR)

FRAGILE (FORMATS CHG)

ANNOYING (ALERT FATIGUE)

HUMAN-DRIVEN
What keeps logs "dumb"?

Logs are stuck in "index + search"
Why are logs so hard?

Formats change
Parses are ambiguous
Experts are needed to interpret
Apps are bespoke
The junior SRE problem

"Hey, I hadn't seen that happen before... then everything went sideways!"

--

Figure out when rare stuff and bad stuff are unusually correlated.
Ze: How it works

Complete relational structuring of logs
## Ze: How it works

### ETYPES: axr_statsd_wait_ms_errors

<table>
<thead>
<tr>
<th>pid::int</th>
<th>ts::ttz</th>
<th>sev::str</th>
<th>mod::str</th>
<th>fun::str</th>
<th>wait_ms::int</th>
<th>errors::int</th>
</tr>
</thead>
<tbody>
<tr>
<td>19563</td>
<td>2016-08-09, 00:10:22.797797-07</td>
<td>INFO</td>
<td>regmgr</td>
<td>axr_statsd</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>19563</td>
<td>2016-08-09, 00:15:34.769823-07</td>
<td>INFO</td>
<td>regmgr</td>
<td>axr_statsd</td>
<td>34</td>
<td>1</td>
</tr>
<tr>
<td>19563</td>
<td>2016-08-09, 00:20:33.316922-07</td>
<td>INFO</td>
<td>regmgr</td>
<td>axr_statsd</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
No information included or required about:

- Known prefix formats
- Specific logtype keywords
- Event grammar / syntax

*We embrace free-text logs*
Anomaly detection on relationally structured data
Ze: How it works

1. ML-driven Parsing & Event Categorization
2. Anomaly Detection on Event Types
3. Pattern Recognition on Correlated Anomalies
4. Root Cause Identification
5. Optional User Feedback

Raw Log Stream
Ze: How it works

No information included or required about:

• Connectors, knowledge bases
• Specific application behaviors
• Specific semantic keywords

Works great on bespoke app or stack
Other ML attempts

Use deep learning
Use one algorithm
Work in batch
Use a Swiss army knife

Structure First - Inline
Respect Pareto - multi-stage
AD/RCA gets better w / complexity!!!
GPT3/NLP requires concise RCA
A picture is worth...
Autonomous Monitoring

Probable Root Cause

- 2020-01-27 28:36:54.829034 Step: Get a list of all pods from given namespace
- 2020-01-27 28:36:54.694865 Step: Initialize deletion list
- 2020-01-27 28:36:54.773292 Step: Select a random pod to kill
- 2020-01-27 28:36:54.851502 Step: Construct the deletion list with single random pod
- 2020-01-27 28:36:55.482794 Step: Kill application pod

Symptoms

- 2020-01-27 28:37:81.787 WARN [carts,b3376e3ccb8689494,68b1368168f2bf5a,true] 6 --- [p-nio-00-exec-5] org.mongdb.driver.connection [ConnectionId(localValue=6, serverValue=2)] to carts-db:27017. All connections to carts-db:27017 will be closed.
- 2020-01-27 28:37:86.56128 Step: Wait for the interval timer
- 2018-1-27728:37:59.936697 WARN 1206 docker_container go:216] Cannot create symbolic link because container log file doesn’t exist!
- 2018-1-27728:37:59.816555 ERROR 1206 remote_runtime.go:213] StartContainer "754e38b49282e5592228t27851669a99d15617e6544d8549bfb1c7c64841af1f7" from runtime service failed: rpc error: code = Unknown desc = failed to start container "754e38b49282e5592228t27851669a99d15617e6544d8549bfb1c7c64841af1f7" from runtime service failed: rpc error: code = Unknown desc = failed to start container "754e38b49282e5592228t27851669a99d15617e6544d8549bfb1c7c64841af1f7" from runtime service failed: rpc error: code = Unknown desc = failed to start container "754e38b49282e5592228t27851669a99d15617e6544d8549bfb1c7c64841af1f7" from runtime service failed: rpc error: code = Unknown desc = failed to start container "754e38b49282e5592228t27851669a99d15617e6544d8549bfb1c7c64841af1f7" from runtime
org.postgresql.util.PSQLException: FATAL: terminating connection due to administrator command
The root cause of the problem is that oom-killer was invoked because of a large number of allocations. The kernel's OOM killer is triggered when the system is out of memory and needs to free some memory. Since this action can kill processes, it is protected by a flag (oom_adj) which can be set or cleared by user space applications. When this flag is set, the kernel will kill processes for which there are no more than one page left in their memory cgroups (cgroups are used to control resource usage on a per-process basis). By default, Linux uses an algorithm called "RSS" (Resident Set Size) to decide whether or not to trigger the OOM killer. This algorithm calculates how much physical memory each process has reserved and compares it with its current virtual size. If there's enough memory available, then RSS will not touch any process even if they have been consuming too many resources for too long; but if there isn't enough memory available, then RSS will trigger the OOM killer and start killing processes until there's enough physical space again.

HOSTS: mars, LOG TYPES: kern, syslog, atlassianconfluence
"Zebrium detected every failure and root-cause, with no rules or training."

"I didn’t build any rules for this. Zebrium just picked up the issue."

"Zebrium dropped root-cause time from 3 hours to 15 minutes."
Join us on this journey!

URL: zebrium.com/how-to-try
email: larry@zebrium.com
twitter: stochastimus@twitter.com

Gartner COOL VENDOR 2020
Best Log Platform for Kubernetes 2020

DZone

Forbes 2020
Top 25 Enterprise Software Startups to Watch in 2020