SurveyMonkey®

Stateless Fluentd With Kafka

Steven McDonald
Site Reliability Engineer
2020-08-20
KubeCon + CloudNativeCon Europe

Who am I?

A brief synopsis of Steven McDonald

- Site Reliability Engineer at Usabilla by SurveyMonkey by day.
- Undergrad physics student by night.
- Joined Usabilla in March 2018 to help create and migrate to new cloud-native infrastructure.
- Took on the logging component at the start of 2019.

What does our infrastructure look like?

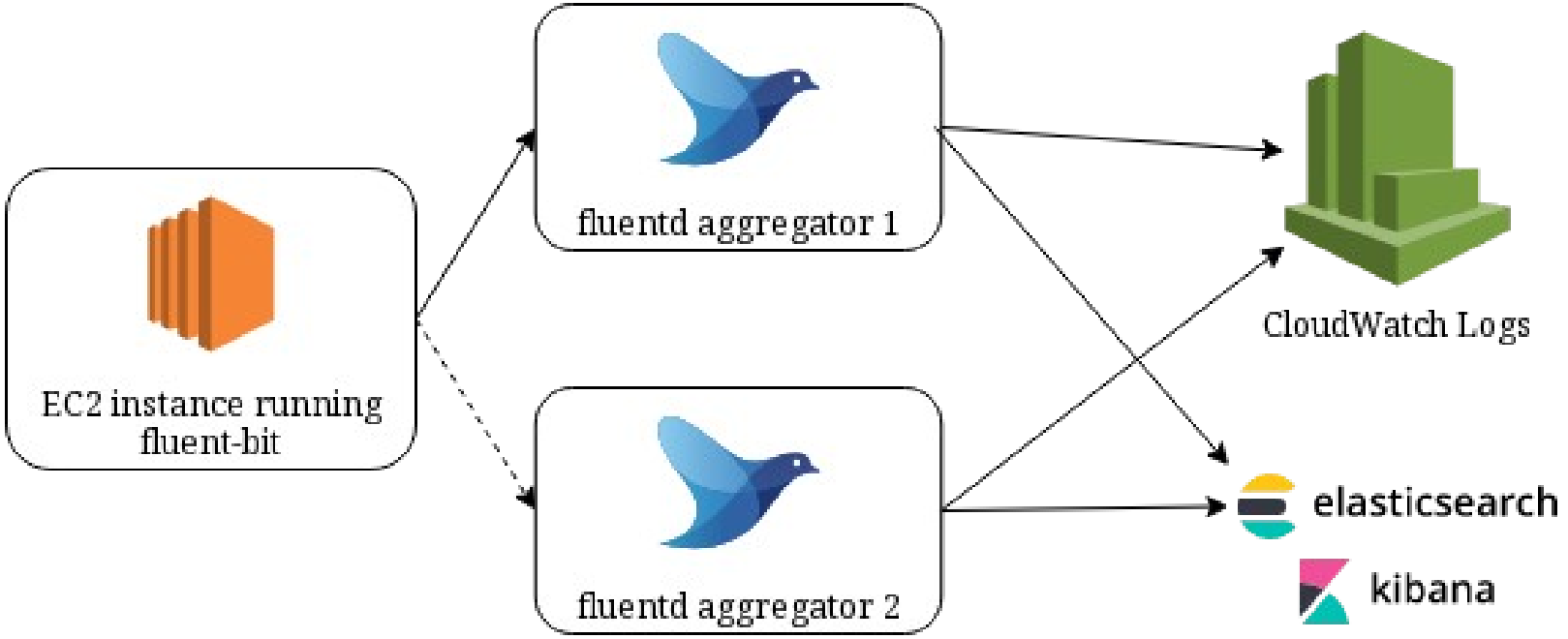
A brief synopsis of Usabilla by SurveyMonkey

- We're running entirely in EC2.
- We have a variety of instances with a variety of services, and we want to aggregate all the logs in one place.
- This includes a combination of cloud-native and legacy infrastructure.

Logging

The first iteration

- Two fluentd aggregators, with fluent-bit on every host configured to forward local logs to fluentd.
- Both fluentd and fluent-bit were configured for disk-backed buffering for reliability.
- Fluentd then forwarded logs on to CloudWatch and Elasticsearch.



This worked well,
until...

Small misconfiguration leads to cascading failure

- Our Elasticsearch cluster turned out to be too small to handle the logs we were sending to it.

Small misconfiguration leads to cascading failure

- Our Elasticsearch cluster turned out to be too small to handle the logs we were sending to it.
- Due to a misconfiguration of the fluentd Elasticsearch plugin, this caused logs to be duplicated within fluentd hundreds of times over.

Small misconfiguration leads to cascading failure

- Our Elasticsearch cluster turned out to be too small to handle the logs we were sending to it.
- Due to a misconfiguration of the fluentd Elasticsearch plugin, this caused logs to be duplicated within fluentd hundreds of times over.
- This both caused hundreds of duplicate log messages in CloudWatch, and placed so much load on the fluentd aggregators that they were unable to accept new logs.

Small misconfiguration leads to cascading failure

- Our Elasticsearch cluster turned out to be too small to handle the logs we were sending to it.
- Due to a misconfiguration of the fluentd Elasticsearch plugin, this caused logs to be duplicated within fluentd hundreds of times over.
- This both caused hundreds of duplicate log messages in CloudWatch, and placed so much load on the fluentd aggregators that they were unable to accept new logs.
- Which, in turn, triggered a bug in fluent-bit on nodes all over our network that caused it to exhaust its open file limit and crash.

Small misconfiguration leads to cascading failure

- Our Elasticsearch cluster turned out to be too small to handle the logs we were sending to it, causing nodes to go offline.
- Due to a misconfiguration of the fluentd Elasticsearch plugin, this caused logs to be duplicated within fluentd hundreds of times over.
- This both caused hundreds of duplicate log messages in CloudWatch, and placed so much load on the fluentd aggregators that they were unable to accept new logs.
- Which, in turn, triggered a bug in fluent-bit on nodes all over our network that caused it to exhaust its open file limit and crash.
- After being restarted, fluent-bit began spamming fluentd with endless streams of invalid log messages, making the load situation even worse and causing more problems with fluent-bit on other hosts.

Lessons learned

- The root cause, an overload of Elasticsearch, is easily fixed. However, this should not be able to cause all the other problems we had.
- The Elasticsearch fluentd plugin re-emitting log messages that could not be written is documented behaviour, but highlights that it can be difficult to configure fluentd for reliability.
- Fluent-bit crashing highlights that fluentd not working reliably in this configuration can have knock-on effects on hundreds of hosts.
- If it is difficult to configure fluentd for reliability, and the lack of reliability amplifies problems, then we must have a buffer in between fluent-bit and fluentd that makes fluentd failures less critical.

Okay, new plan!

Enter Kafka

- Kafka is a distributed, fault-tolerant message broker.
- Consumers may be stopped entirely if necessary, and pick up processing where they left off, with no interruption to producers.
- An added benefit is that it allows reprocessing of the same messages by independent consumers, which facilitates testing of new consumers on real data.
- We are already using Kafka in our application architecture, so it’s a technology we are already familiar with.
- Given all of the above, it seemed like a natural fit as a logging buffer.

journald

- Journald, a component of systemd, provides a unified logging buffer for aggregating all logs on a given host.
- It follows that any log exporter that provides its *own* buffer is duplicating existing functionality.
- We only need half of what fluent-bit does, namely, something to read the systemd journal and shunt the entire thing off to Kafka.
- This approach is only suitable if using systemd on all hosts, which is the case for us.

So what code do we need to make this happen?

Perapera

A simple log exporter

- We looked into the existing fluent-bit Kafka plugin, but found it did not meet our standards for reliability, as it removes log messages from its own buffer without awaiting a write acknowledgement from the Kafka broker.
- All of the problems we had encountered with fluent-bit were related to how it does buffering. But, as previously mentioned, we don't need it to do buffering.
- We developed a simple tool called perapera (Japanese for “fluent”), which just reads from the journal and writes everything in it to Kafka using MessagePack (the same serialisation format used by fluentd's native forwarding protocol).
- If perapera crashes or is killed, it will continue reading the journal from the last message that was acknowledged by the Kafka broker.

Making fluentd stateless

- Fluentd pushes the choice of whether to use an internal buffer to the output plugin author.
- Most output plugins are written to support only buffered mode, with good reason—this makes the most sense when fluentd is functioning as a central aggregator.
- In order to run fluentd statelessly with Kafka as the central logging buffer, we would need to reimplement every output plugin as unbuffered.
- Or, we could be smarter...

Making fluentd stateless

… with fluent-plugin-unbufferize!

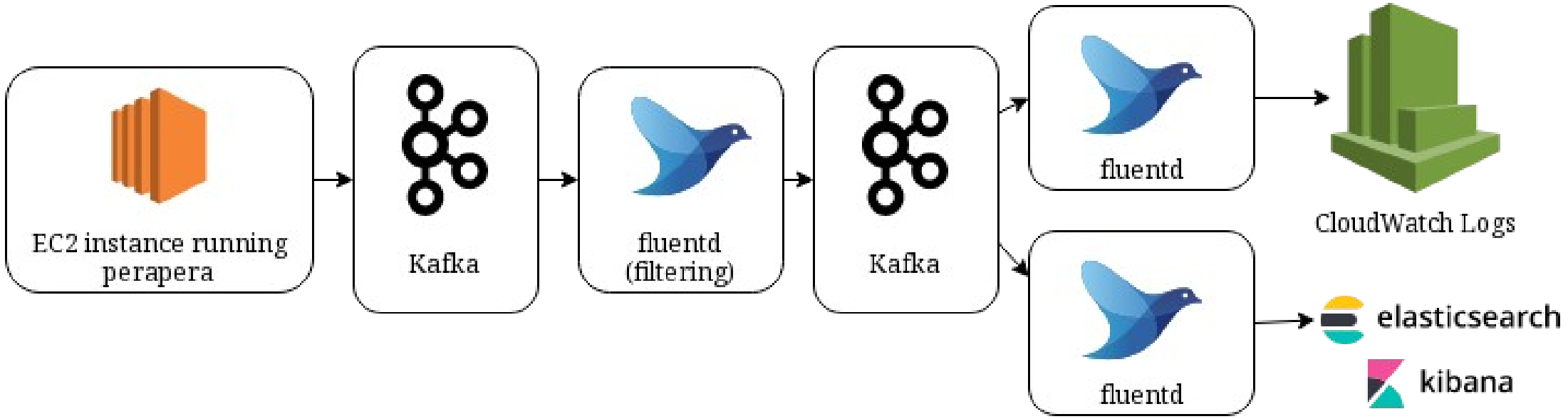
- We implemented a fluentd plugin which is a complement to the existing “bufferize” output plugin.
- This wraps any other output plugin and fakes enough of the buffer plugin API to fool the output plugin into thinking it has a real buffer.
- The unbufferize plugin also performs chunking of each stream of logs it receives, to permit interpolation of chunk keys in strings to work as it would with a real buffer.
- Naturally, this only works well when you can batch reads from the input plugin, which is perfect for our use case reading from Kafka.

Stateless fluentd with Kafka

Putting it all together

- The in_kafka plugin for fluentd will not commit a read until the "emit" call, used to submit a log event for processing by fluentd, returns.
- Ordinarily, this call returns when the logs have been committed to the output plugin’s buffer.
- With fluent-plugin-unbufferize, this call instead returns when the logs have been written to their final destination.
- If an exception is raised writing logs to CloudWatch, or to Elasticsearch, or to another Kafka topic, then this is propagated back to the input plugin and the Kafka read is not committed, so it will be retried.

Our new stateless architecture



The results

How well does it work in practice?

- It works a lot better than the previous approach, with lower overall latency as we are now processing log batches as they come in rather than waiting for fluentd to flush its buffer.
- The pain point now is the design of the ruby-kafka library, which cannot handle large batches of logs without long session timeouts.
- The fact that session timeouts were the problem was difficult to debug, as the default logging configuration is not very helpful for tracking down Kafka issues.
- fluentd still uses a lot of memory.

What did we need to fix?

In the course of this work, we submitted 9 patches to open-source projects. Some highlights include:

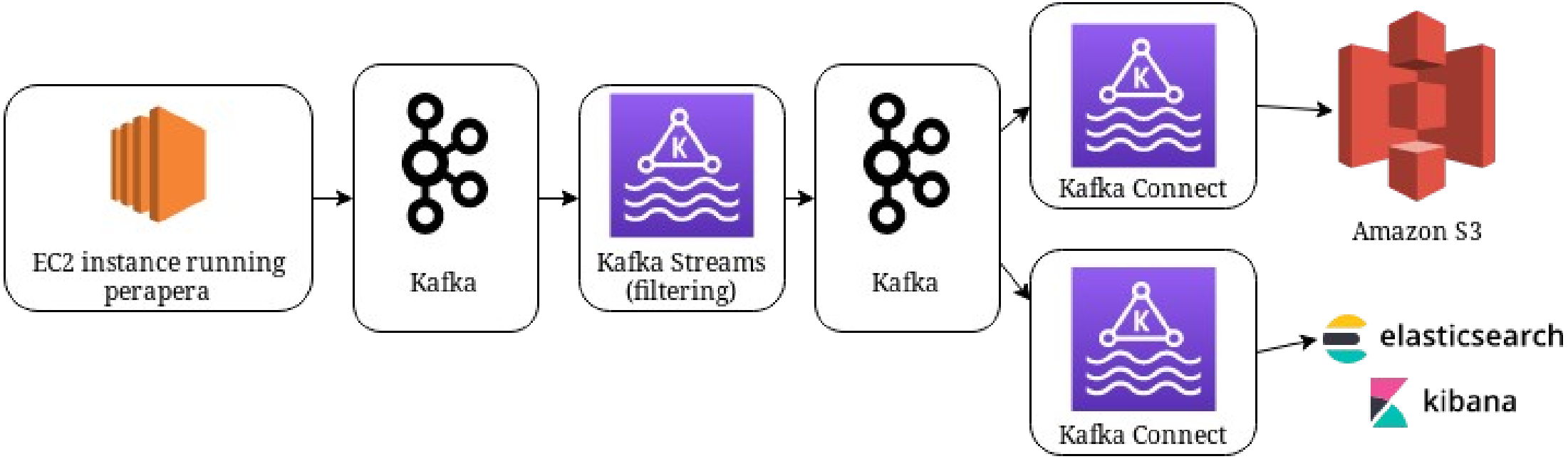
- Metrics in fluent-plugin-prometheus to report the oldest and newest timekeys currently in the buffer (before we stopped using fluentd's own buffering).
- Fixing of timestamp handling in ruby-kafka, and making use of these timestamps in fluent-plugin-kafka.
- Implementing correct consumer group rebalancing in ruby-kafka.

The bad news...

The third iteration

- We finally decided (after this talk was submitted) to drop fluentd for processing log messages from Kafka.
- We found it was less work to implement our own logging processor using Kafka Streams than to rework the ruby-kafka library to handle large batches better.
- The approach we've taken is still potentially useful for anyone else who wants to try a deployment like this.
- But it’s not all bad news: we are still using fluentd (in unbuffered mode) to write our Kubernetes container logs to the journal (so that they can be picked up by perapera).

The third iteration



The future

- In addition to the open-source contributions made during this work, we may be open-sourcing some of the new software we've written, if it is not too specific to our setup.
- For any updates, follow our GitHub org at https://github.com/GetFeedback, or feel free to reach out to me at smcdonald@surveymonkey.com.

Questions?

smcdonald@surveymonkey.com