Using Kafka to monitor at scale

Kafka’s unique ability to combine high throughput with persistence makes it ideal as the pipeline underlying all of Splunk Infrastructure Monitoring. Throughput is critical to the kind of data your Splunk Infrastructure Monitoring deployment handles: high-volume, high-resolution streaming times series. And persistence allows you to smoothly upgrade components, do performance testing, respond to outages, and fix bugs without losing data. However, in your demanding environment, you have very high performance expectations. Monitoring and alerting on Kafka in production, at scale, can be complicated. You need some recommendations on how to be more effective in your use.

How to use Splunk software for this use case

Learn what metrics matter most

Possibly the most important metrics for large environments are:

- Log flush latency (95th percentile)
- Under-replicated partitions
- Messages in / sec per broker and per topic
- Bytes in / sec per broker (collected as a system metric from each broker using collectd)
- Bytes in / sec per topic
- Bytes / message

Log flush latency and under replicated partitions tend to be the leading indicators regarding what’s going on and help you prepare to investigate a new bug or regression. Log flush latency is important because the longer it takes to flush log to disk, the more the pipeline backs up and the worse our latency and throughput. You should monitor for changes in the 95th percentile. When this number goes up, even 10ms going to 20ms, end-to-end latency balloons and all Splunk Infrastructure Monitoring is affected.

Under-replicated partitions show that replication is not going as fast as configured, which adds latency as consumers don’t get the data they need until messages are replicated. It also suggests that an organization is more vulnerable to losing data if it has a master failure.

Changes in these two metrics generally lead to looking at the other three metrics.

Messages and bytes in tell how well balanced your traffic is. If there is a change in their standard deviations, you know that a broker(s) is overloaded. Using messages in / sec and bytes in / sec, you can calculate bytes / msg. Bigger messages have a higher cost in performance because it takes a proportionally longer time to process those messages and can cause the pipeline to back up. A sudden, positive change in message size could indicate a bubble in the pipeline or a fault in an upstream component. A trend of larger message sizes over time suggests an unintended
architectural change or an undesirable side effect of a change to another service causing it to produce larger messages.

---

**Leverage prebuilt dashboards**

If you use collectd and the [GenericJMX plugin](https://www.apache.org/kafka/docs/genericjmx.html) configured for Kafka, Splunk Infrastructure Monitoring provides built-in dashboards that display the metrics that are most useful when running Kafka in production. Since topics are set by you when you set up Kafka, for per topic metrics, we provide templates where you can insert your topic names.

---

**Focus on leading indicators**

You will likely find notifying on alerts for the two leading indicators—Under Replicated Partitions and Log Flush Latency (95P)—the most useful, and investigation usually leads to something at the broker level.

Any under replicated partitions at all are problematic. For this, you can use a greater-than-zero threshold against the metric exposed from Kafka.

Log flush latency is a little more complicated. Because some topics are more or less latency sensitive, you should set different alert conditions on a per-topic basis. Each broker’s metrics have metadata that you apply (as key value pairs of property:value) to identify the topics impacted. For example, raw customer data being ingested is highly latency sensitive, so it gets a 100ms threshold. But email can wait plenty of time, so the threshold is orders of magnitude higher.

Finally, you should alert on having less active controllers than expected, since this is a clear signal that you have a big problem.

---

**Establish a plan for scaling and capacity**

Adding capacity is easy, but re-balancing topics/partitions across brokers can be quite hard. For smaller or simpler setups, Kafka can generate an assignment plan for you that provides even distribution across brokers. Which is fine if your brokers are homogenous and co-located. This does not work well if your brokers are heterogenous or spread across data centers. You'll need to manage the process manually. Fortunately, Kafka takes care of the actual movement of data, given the partition to broker assignments. And with the expected addition of rack/region awareness, Kafka will soon allow for this kind of spreading of replicas across racks and regions.

This is the tricky part. For example, if you have a lot of traffic on one topic and are adding capacity for it, the topic partitions have to get spread across the new brokers. Although Kafka currently can do quota-based rate limiting for producing and consuming, that’s not applicable to partition movement. Kafka doesn’t have a concept of rate limiting during partition movement. If you try to migrate many partitions, each with a lot of data, it can easily saturate your network. So trying to go as fast as possible can cause migrations to take a very long time and increase the risk of message loss.

Soon Kafka's built-in rate-limiting capability will be extended to cover partition data balancing. In the meantime, to reduce migration time and the risks, you'll need to move one partition at a time, watching the bytes in/out on the source and target brokers, as well as message loss. Those metrics help you control the pace of rebalancing to minimize...
message loss and resource starvation, thus minimizing service impact. The general approach is:

- Baseline the use case to understand performance trade-offs (for example, learn that small changes to Kafka log flush latency have large impacts on end-to-end service latency).
- Find the unit of data that can be operated on without causing performance issues.
- Make sure that it's evenly distributed, even if not random.
- Break up large or long running processes that run across those units into smaller components to prevent overload, bottlenecks, and resource starvation so app level performance is not impacted by service level changes such as scaling and rebalancing.

Next steps

These additional Splunk resources might help you understand and implement these recommendations:

- Splunk Docs: Integration reference: Kafka
- Blog: How we monitor and run Kafka at scale