Becoming more effective monitoring Kubernetes at scale

In your organization, you use Kubernetes to expose monitoring and observability data in standardized ways. Yet you have some challenges with scale, churn, and correlation that arise from running applications on Kubernetes.

• Your environment has hundreds of microservices.
• You have many containers running as part of a Kubernetes Pod, and multiple Pods running on each node in your cluster.
• You want to understand metrics in the context of specific categories, or on a per-object basis (for example, requests for each service or latency per customer).
• Kubernetes has additional orchestration components that need to be observed.

You need some recommendations on how to work more effectively in your Kubernetes environment.

Data required

Kubernetes data

How to use Splunk software for this use case

Avoid fragmentation

You could – if you can afford to do so – overprovision capacity or pay for the ability to keep recent data in memory. But while spending more on infrastructure might solve for hotspots in your monitoring system, it still doesn’t address fragmentation issues. One solution is to use an aggregator of aggregators like Thanos, which aims to provide a global query view of existing Prometheus deployments in your environment. A single load-balanced scalable store avoids the problem of fragmentation and hotspots entirely. Keeping monitoring data in a single place makes it easy to search for the right metric, while also ensuring more consistent performance and resource utilization.

Improve slow queries through pre-aggregation

As your environment grows, so do your queries – long or broad queries (looking far back in time, or across a large group of components) will inevitably become slower, and in some cases fail. Slower query performance prevents you from easily seeing historical trends and alerting on regular patterns, or using long-term data for capacity planning. Even worse, problem detection and root-cause analysis during outages takes longer because ad-hoc queries may not run fast, prolonging incidents.

Pre-aggregation can make queries more efficient. You could pre-aggregate data (computing values like the sum,
average, minimum, maximum of a given metric) across a particular grouping, such as application, cluster, or another hierarchy of your choosing. For example, you might compute the average of a certain metric across 1000 containers and store that as a time series for more efficient querying. You can also pre-aggregate across time rather than across services, meaning that for every data stream, you compute mathematical rollups that summarize data across predetermined time windows. For example, you could calculate the average CPU utilization of a host over 1-second, 1-minute, 5-minute, and 1-hour intervals.

Manage churn and metadata explosions and accumulation stemming from churn

Lots of common operational tasks and features of today’s environments cause churn, including:

- CI/CD and immutable infrastructure: pushing new microservices weekly, daily, or multiple times per day causes all of your application metrics to report with a new version each time.
- Auto-scaling: Most cloud services are elastic and offer options for auto-scaling. Kubernetes also has horizontal pod auto-scaling based on a metric of your choosing.
- Ephemerality: Short-lived spot instances and instance retirement in public clouds cause infrastructure churn.

Metadata explosions can happen as a result of component churn, and they are challenging to process quickly. How do you efficiently process a system-wide change, such as during blue-green deployments where you stand up a replica of your entire production environment to receive traffic? What about large-scale metadata tagging? Say for example you wanted to calculate the hourly cost of running your microservice on AWS EC2 instances. To properly work out the cost of each instance, you need to know the AWS instance type associated with each host, so you add an "instance_type" property to all of the hosts in your environment. The number of datapoints or events that your monitoring system is processing remains the same, but the amount of metadata has increased.

Churn also leads to metadata accumulation—you have 365 times more metadata per year if you push changes to your entire system daily—and is especially painful with containers. A 1-year chart has to stitch together 365 different segments together, severely limiting the usability of your system, and can slowly kill your metrics storage backend.

One good approach to managing these problems is to put the metadata into its own independently scaled storage backend. Since this backend only has to scale with metadata churn (when you push a new version of a service), and not with the volume of datapoints being received (the steady state when a microservice is running), it has a simpler problem to deal with, and this can enable you to support high-churn environments extremely well.

You can also build awareness of timing into your metrics pipeline that dynamically addresses the tradeoff between latency and accuracy when it comes to pre-aggregating data. Splunk Infrastructure Monitoring produces timely yet accurate results by individually analyzing and modeling the delay behavior and timing of each data stream. Based on this model, when an aggregation needs to be performed across a set of time series, Splunk Infrastructure Monitoring calculates the minimum amount of time to wait before computing an accurate aggregate value. This is a key component of a good solution’s streaming analytics engine, and reduces alert noise generated by false alarms.

Develop a holistic view of your data sets

Monitoring and observability means gathering and acting on data emitted by many different layers of the stack. It’s
challenging to make sense of this data, correlating behavior across data sources, and keeping all of this actionable.

- Look across data sets. Careful data modeling (for example, using tags and dimensions to group and filter metrics) can make correlation easier. Adopting schemas with 'join' dimensions ("instance_id", "container_id", and "app_id" on metrics) and importing additional metadata (such as from the underlying cloud provider) also helps.
- Looking across data types. The ability to correlate across these metrics, traces, and log tools during an incident is critical – imagine being forced to manually copy context from your metrics solution to search through your logs. This is even worse when naming conventions differ across each of these data types (e.g. using 'host' in your metrics system, 'node' in logs, and 'instance' in traces). Standardizing schemas across types of telemetry data makes correlation easier. Ideally, you’ll also want to construct point-and-click integrations between tools (or look to solutions that have them) to enable preservation of context, because being able to do things like go from an alert to the corresponding time slice in your logging solution is incredibly powerful.

**Next steps**

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

- .Conf Talk: [Real-world strategies for Kubernetes & Docker integrations with Splunk](#)
- Blog: [Strategies for monitoring Docker and Kubernetes environments](#)
- Splunk Docs: [Kubernetes cluster](#)
- Tech Talk: [Splunk Operator for Kubernetes: Effortlessly deploy, scale, and manage Splunk](#)
- Tech Talk: [Maximizing the performance of your Kubernetes deployment](#)

---

The information provided in Splunk Lantern is intended for informational and educational purposes only. All information is provided in good faith, however, Splunk disclaims any and all representations and warranties, express and implied, regarding the information provided, including without limitation any warranties and representations regarding the completeness, adequacy or accuracy of the information. You agree to take full responsibility for the results arising from the use of the information provided.