sumo logic

WHITEPAPER

Navigating Kubernetes monitoring

From its humble open source beginnings to market dominance (or ubiquity)

By: Melissa Sussmann
Lead Technical Advocate,
Sumo Logic

Table of contents

"Kubernetes has grown from a niche technology to something utterly ubiquitous."

Chris Aniszczyk, CTO at the Cloud Native Computing Foundation (CNCF)

Introduction

Modern applications: The genesis of containerization

While you probably have heard about containerized applications and Kubernetes, you may be unfamiliar with their shared foundation: Modern Applications Reference Architecture (MARA).

MARA is a composition of best practices for building scalable, portable, resilient and agile applications. As a result, the infrastructure for these applications has moved away from the monolithic, virtual machine (VM)-hosted systems towards a more distributed, container-based approach. Decomposing monolithic applications into smaller microservices improves the scalability and maintenance of the infrastructure. It also helps teams easily adjust resources and adapt infrastructure to change.

How does containerization support MARA best practices?

The most important aspects of MARA are portability, scalability, resiliency and agility. How does containerization help?

- **Portability**: It should be easy to deploy the application on multiple devices and infrastructures, on public clouds and on-premises. Containerized applications can easily move between environments, such as cloud platforms or on-premises infrastructure. This can make it easier to migrate applications to new environments as IT organizations change or grow.

- **Scalability**: The app can quickly and seamlessly scale up or down to accommodate spikes or reductions in demand anywhere in the world. Containers are more flexible than VMs because they allow you to package an application and its dependencies in a single container image. This makes updating and maintaining applications easier, as you can deploy a new container image rather than updating each VM.

- **Resiliency**: The app can gracefully fail over to newly spun-up clusters or virtual environments in different availability regions, clouds or data centers. Containers enable fragmentation for distributed components of a cloud-native application. If something goes wrong with one pod, it does not need to shut down the entire application.

- **Agility** – Automated continuous integration/continuous delivery (CI/CD) pipelines can update the app quickly; in modern apps, this also implies a higher code velocity and more frequent code pushes. Containers deploy faster than VMs due to better resource utilization. They do not have an operating system to install and configure, which makes it easier to deploy and scale applications quickly and saves on storage needs.

Modern applications — key terms

Now, let's review a few key terms related to modern applications:

- **Microservices:** Modern applications often use a microservices architecture, which means that they are composed of small, independent services that can be developed, deployed and scaled independently.
- **Containerization:** Modern applications come packaged with their dependencies in a single container image, making it easy to deploy them to different environments.
- **Cloud-native infrastructure:** Modern applications are often cloud-native, meaning they take advantage of the scalability and agility of cloud computing platforms.
- **Application programming interface (API):** Modern applications are often API-driven, exposing their functionality through APIs, making it easy for other applications and services to integrate with them. APIs are tools for building software applications and allow different systems to communicate and exchange data. They are provided as web-based interfaces or libraries and enable interoperability that allows developers to build complex applications using other systems' capabilities.

Now that we have established that containers have gained widespread adoption via MARA best practices let's discuss container orchestration options.

Container orchestration tools: Why K8s is king

Many open-source container orchestration tools are available in the market to manage containerized applications. Some of the well-known ones include:

- **Kubernetes:** Kubernetes is an open-source container orchestration platform originally developed by Google and gifted to the Cloud-Native Computing Foundation (CNCF) that is designed to automate the deployment, scaling and management of containerized applications. It is the most popular open-source container orchestration tool with the largest community of supporters.
- **Docker Swarm:** Docker Swarm is a container orchestration platform developed by Docker that makes it easy to manage a cluster of Docker engines and deploy containerized applications to them. It is a good, user-friendly tool for simple deployments. It is also relevant for ITOps organizations already using Docker.
- **OpenShift:** OpenShift is an enterprise-grade, open-source container orchestration platform developed and supported by Red Hat and built on top of Kubernetes. It includes several additional features that make it easy to deploy and manage containerized applications in a variety of environments. It includes several additional features and tools that make it easy to deploy and manage containerized applications in various environments. OpenShift includes key features such as deployment pipelines, integration with Git repositories, and the ability to roll back deployments. However, unlocking all enterprise features comes with a hefty cost.
- **OpenStack:** OpenStack is an open-source cloud computing platform that includes Nova, a component to manage compute resources. Nova can create and manage both VMs and containers, allowing for the deployment and management of containerized applications. While OpenStack is not primarily designed as a native container orchestration platform like Kubernetes, it is a great option for organizations utilizing hybrid on-premises VMs and containers. However, this must be in conjunction with Kubernetes for cloud-native workloads.

Notice a pattern? While there are some great options for container orchestration, Kubernetes holds clear advantages over others. It is the current de-facto open-source container orchestrator. Many commonly used alternate container orchestration tools rely on Kubernetes or need features with it. It is no wonder that Kubernetes has the largest active community out of any open-source project from the CNCF.

Chapter 1

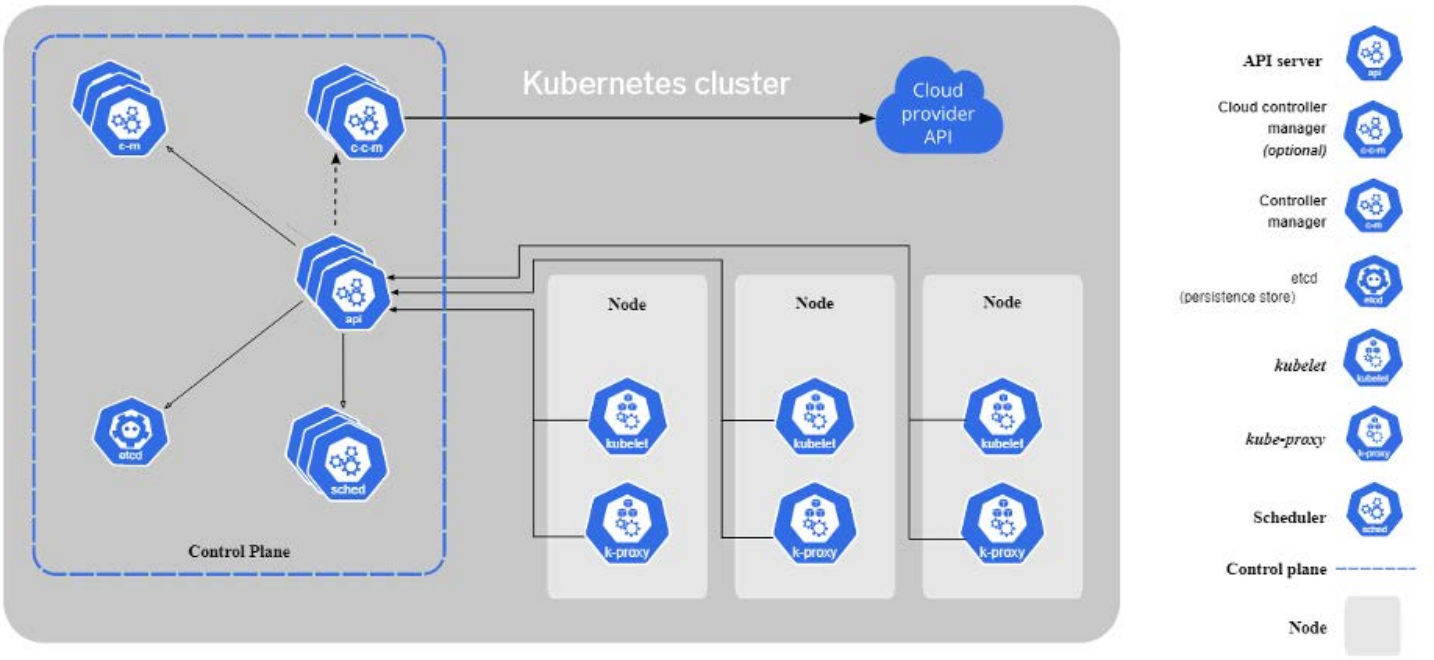
Kubernetes fundamentals

If you're looking for the basics of Kubernetes, including terms and architecture, this chapter will help you recognize the key features and building blocks of Kubernetes. As covered in the introduction, Kubernetes (or K8s) is an open-source orchestration system for automating, deploying, scaling and management of containerized applications.

Kubernetes provides a production-ready cluster consisting of controller and worker nodes for running containerized applications on-demand with the necessary resources and tools. It abstracts away the underlying physical infrastructure and offers services and features for scheduling, resiliency, storage and networking for the containerized application.

Many of the diagrams found in this section can also be found in the Kubernetes docs.

Kubernetes components



Above is a diagram of a Kubernetes cluster and its fundamental components. As you can see, when you deploy Kubernetes, you get a cluster made up of nodes and a control plane. The nodes are like worker machines that run your containerized apps and host the pods that make up the workload. The control plane manages the nodes and pods, makes decisions about the cluster and detects and responds to events in the cluster.

These are other must-know Kubernetes components:

- **Pods are components** of the application workload and are one of the most basic units in a deployment. Pods can host one or more containers. It is the smallest deployable unit in K8s infrastructure.
- **Volumes** are persistent storage locations for pods. Volumes allow data to flow across pods and be stored outside of a given pod to save it in case of a restart.
- **Nodes** are the worker machines that host pods. You can have several pods to one node.
- **Kubelet** is an agent that runs on each node in the cluster and ensures containers are run in a pod.
- **Services** represent a group of pods and their underlying policies. Services load balance traffic between network endpoints.
- **Kube-proxy** is a network proxy that runs on each node in the cluster and maintains network rules that allow communication to the pods from inside or outside the cluster.
- **Container runtime** is responsible for running containers. Kubernetes supports a few options like Docker containerd, and CRI-O.
- **Kube-controller manager** runs controller processes like the node controller that detects and responds when a node goes down.
- **Job controller** creates pods to run one-off tasks.
- **Endpoints controller** connects services and pods.
- **Service account and token controllers** create default accounts and API access tokens.
- **Kube-API server** is the front end for the Kubernetes control plane and exposes the Kubernetes API.
- **ETCD** is the key value store used as Kubernetes' backing store for all cluster data.
- **Kube-scheduler** selects a node for newly created pods.

Kubernetes key features

Kubernetes offers many robust capabilities for deploying and managing containerized applications, including:

- **Service discovery and load balancing** that assigns unique IP addresses and DNS names to pods and uses these to load balance traffic between them.
- **Storage orchestration** that provides automated mounting of various types of storage systems, including local storage, public cloud providers and network storage systems.
- **Automated rollouts and rollbacks** that monitor application health and can rollback in the event of deployment issues.
- **Secret and configuration management** which allows you to deploy and update secrets and application configuration without rebuilding the image.
- **Automatic bin packing** that places containers on nodes based on resource requirements and other constraints to improve utilization.
- **Horizontal scaling** (an application scaling) which allows you to scale your application up or down using a command through the UI or automatically based on CPU usage.
- **Batch execution** that helps you manage batch and continuous integration (CI) workloads.
- **Support for both IPV4 and IPV6 stacks,** which allows you to allocate both types of IP addresses to pods and services.
- **Extensibility** that enables you to add new features to your Kubernetes cluster without modifying the source code.
- **Self-healing capabilities** that provide automatic restart, replacement, and rescheduling of containers if they fail or become unresponsive.

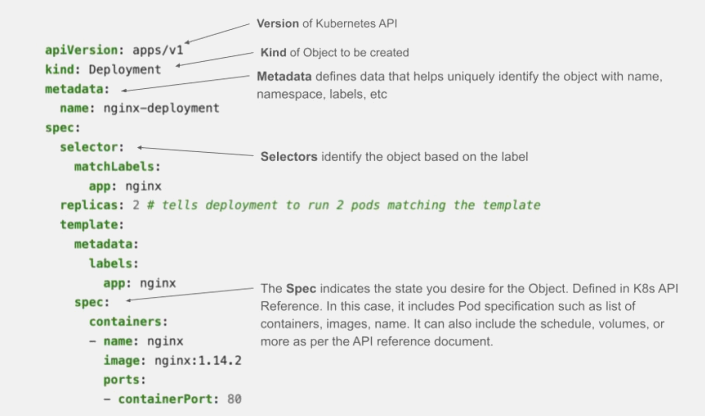
Kubernetes objects

Let's take a closer look at Kubernetes objects — the Kubernetes control plane, cluster components, and external components that enable communication between components. The Kubernetes control plane allows you to query and manipulate the state of API objects.

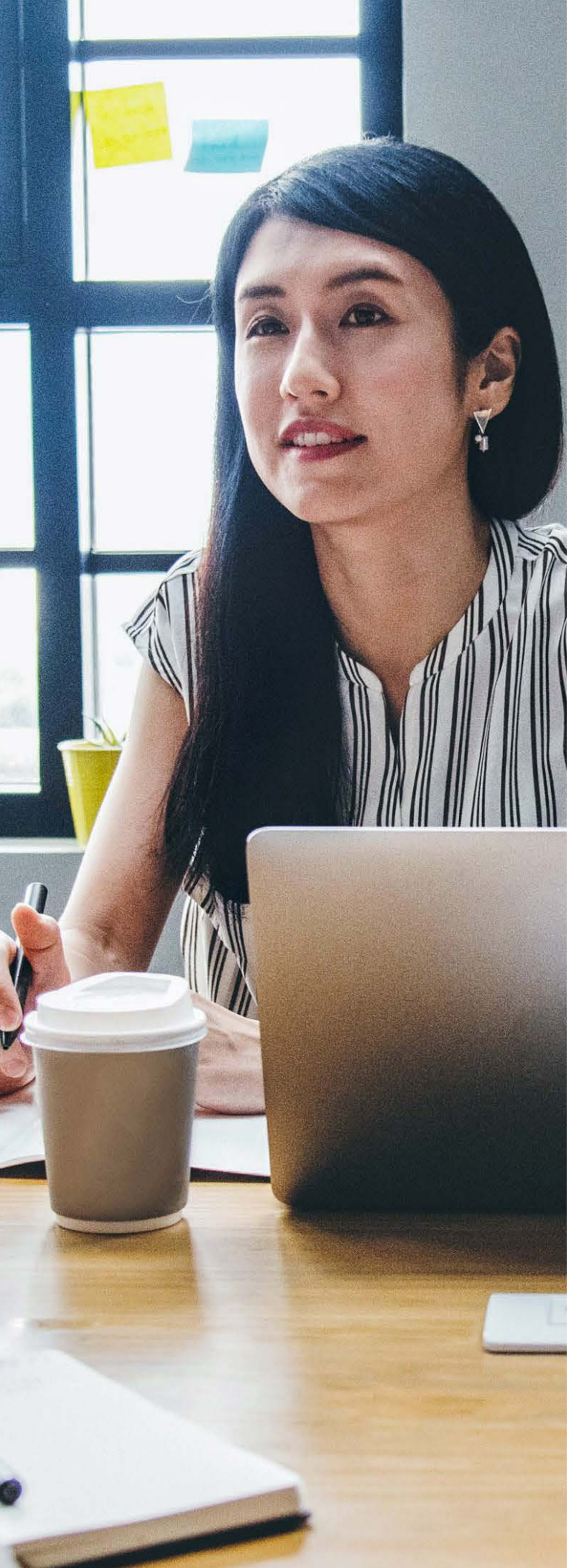
Kubernetes objects are persistent entities in the Kubernetes system that represent the state of the cluster. Some examples include which containerized applications are running on which nodes, policies regarding application behaviors (such as upgrades and restarts), and resources available to the applications (such as memory utilization). An object has a specification which describes the characteristics of the object, such as a YAML file and status which describes the current state of the object to kubectl.

Labels identify objects known as "tags". A selector uses tags to identify an object onto which an action needs to be performed. Let's look at the YAML file example below:

A basic understanding of Kubernetes anatomy and the history of cloud-native infrastructure will help you understand the best practices for monitoring. Now that you also understand how a YAML file sets parameters for an object, you can learn more about monitoring K8s infrastructure as you run deployments.



Additional Resources/Sources:
K8s certification and training
Kubernetes objects



Chapter 2

Monitoring and troubleshooting Kubernetes

Now that you understand the basics of container orchestration and Kubernetes fundamentals let's explore deployments and some of the monitoring challenges that come with distributed tracing, disparate data sources, hosting models and shadow IT.

Introduction to deployments

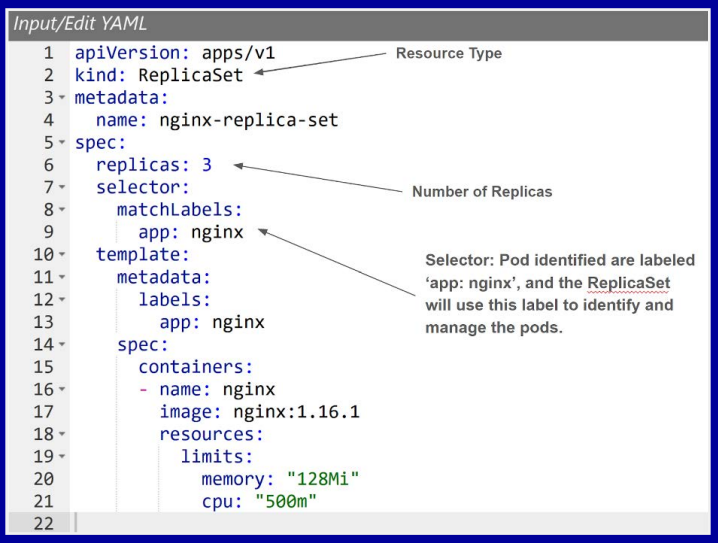
Ten years prior to writing this white paper, DevOps terms such as CI/CD were just barely emerging in the market. They referred to maintaining and updating on-prem or VM-based infrastructure. With the adoption of containers, there is now a strong interest in "continuous deployments" best practices, but what does that mean? Without context, it sounds like jargon.

We have covered pods previously, but we did not delve into ReplicaSets. A ReplicaSet is a controller in Kubernetes that is key to its self-healing capabilities. A ReplicaSet will ensure a specified number of replicas of a pod are running at any given time, and if a pod goes down, it will create a new one to replace the pod. This functionality allows for Kubernetes applications to not only self-heal but scale up and down as needed.

Deployments are higher-level controllers that manage the ReplicaSets via declarative updates. Deployments allow for rolling updates and rollbacks. You can also declare the new state of a pod or clean up older ReplicaSets.

Here is an example of a YAML file that defines a ReplicaSet for an NGINX container. You can get the free code editor I am using.

Deployments are declarative updates for pods and ReplicaSets. As you can see, under **spec**, this sample **ReplicaSet** template ensures that there are always three replicas of the **nginx** container running. The pod specifications are provided as part of the pod definition. This template uses the **matchLabels** field in the selector to identify the pods that belong to the **ReplicaSet**, and the **template** field specifies the pod template that should be used to create new replicas. The pod template includes the labels that are used to identify the pods, and the containers field specifies the container image and resource limits.



Services expose an application running on a set of pods as a network service. It usually defines a set of pods through the selector. Service types include:

- **ClusterIP,** which exposes the service on an IP address reachable from other pods
- **NodePort,** which exposes the service on the node IP using port mapping
- **LoadBalancer,** which exposes the service directly using a cloud provider load balancer. This load balancer can come from AWS or NGINX for example.

Now that we've outlined the fundamentals of Kubernetes deployments and you have a basic understanding of ReplicaSets, let's talk about best practices in monitoring Kubernetes. What do you do when a deployment fails or goes awry? How do you troubleshoot and what are some of the most common challenges you might encounter?

Fundamentals of Kubernetes monitoring

Monitoring Kubernetes and microservices introduce challenges for traditional application performance monitoring (APM) tools due to the ephemerality of nodes, containers and pods in a Kubernetes environment. Container orchestration is already difficult, and errors in the Kubernetes application can make it even harder to manage.

Monitoring various data sources: metrics, logs and traces

Errors can include issues with pods due to high CPU utilization, problems with the Kubernetes operator causing errors on Windows servers or nodes, resource utilization issues, and many other failures related to pods, scheduling or deployments. To effectively troubleshoot these issues, you need a monitoring tool that can help you identify the exact combination of pods and nodes that are causing problems, and then drill down into the associated container logs to find the root cause.

A good Kubernetes monitoring tool should be able to identify infrastructure failures through metrics, logs and trace data and transform that data into a human-readable format like charts and histograms. Some key terms to identify and track include:

- Metrics are numeric samples of data collected over time. They measure infrastructure, such as operating system performance, disk activity, application performance, or custom business and operational data that is coded into your organization's applications. Metrics are an effective tool for monitoring, troubleshooting and identifying the root causes of problems. They can help your organization:
- Gain end-to-end visibility into application performance.
- Track key performance indicators (KPIs) over time.
- Determine if an outage has occurred and restore service.
- Diagnose why an event occurred and and how to prevent it in the future.
- Logs are a record of events that happen in a system or application. Logs in Kubernetes environments are often ephemeral, meaning that they can disappear permanently when containers shut down unless you save them to an external location. It's important to analyze this data in real-time because the state of Kubernetes clusters is constantly changing and data collected at one point may not be relevant even just a few minutes later. Make sure to prioritize real-time analysis to ensure you get the most useful insights from your data. Check out our page on Log Management to learn more about this topic. Read the Real User Monitoring (RUM) section of this ebook to learn more about how to deal with data ephemerality.
- Traces are a sequence of events that occur within a system or application. Tracing is often used in distributed systems to understand the behavior of requests and operations as they flow through different components and services. Read more about traces in our documentation page. Traces are especially useful in Kubernetes environments because they help you understand how different data sources interact and how they impact your applications' performance.

Tools like Sumo Logic allow you to view your Kubernetes environment through various hierarchies using logs, metrics, traces and events. For example, you can use native Kubernetes metadata like namespaces to visualize the performance of all pods.

Kubernetes and Real User Monitoring (RUM)

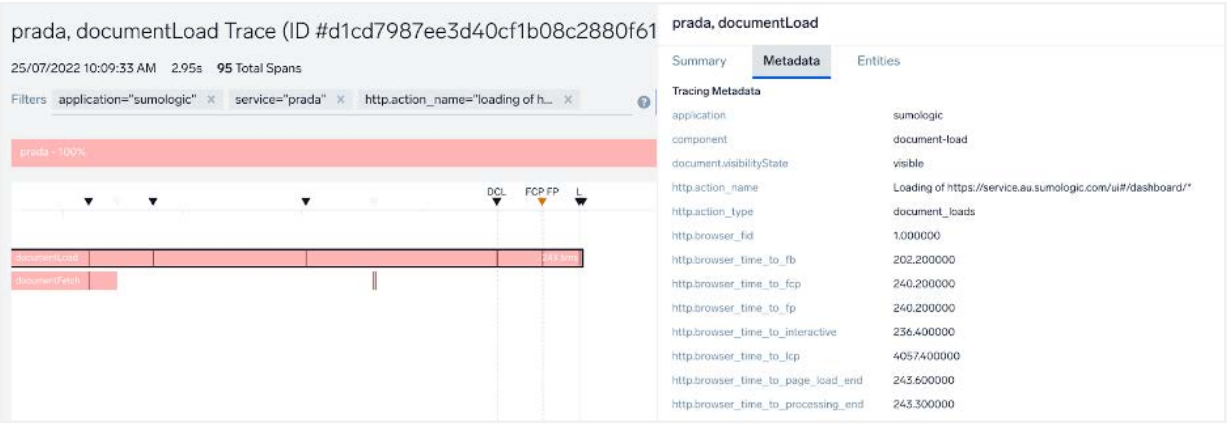
Whether you are an application developer, business analyst, DevOps Engineer, SRE, or product owner, you care a lot about user experience and want to see what is happening in real-time. To connect microservice performance issues and errors with user experience, it is essential to understand end-to-end user transactions, uncover latency issues and see which services are impacted. Distributed transaction tracing provides the telemetry to connect the monitoring of key performance indicators to the real experience of your users. This is where RUM comes into play.

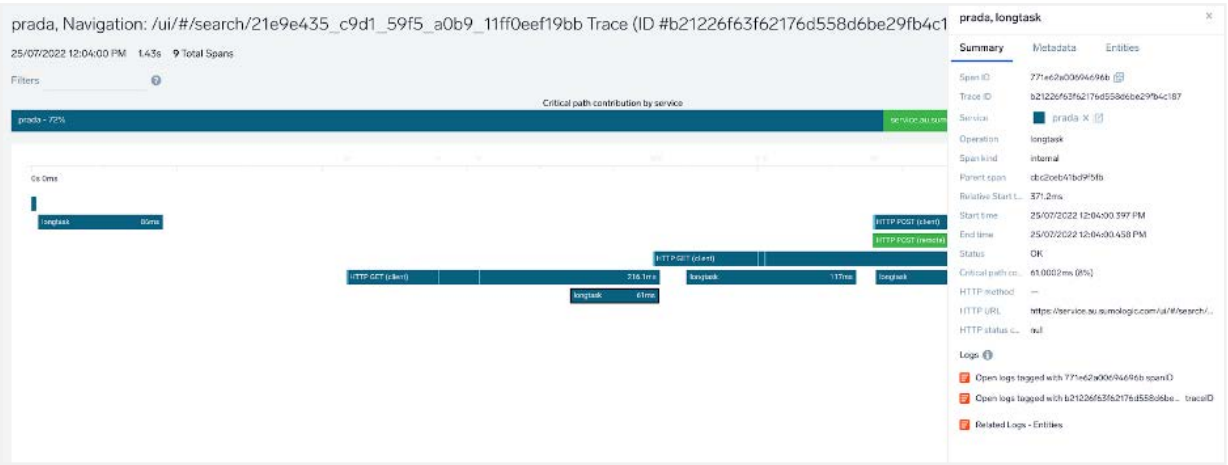
Kubernetes data ephemerality

As discussed in our data sources section, the main challenge of RUM in Kubernetes environments is that data sources can be ephemeral, meaning they are not permanent and may not exist for long. This can be the case with logs for containers, which are deleted when the containers are shut down unless they are saved elsewhere. Additionally, because Kubernetes clusters are constantly changing, data collected through methods like distributed tracing may quickly become outdated and may not provide useful insights if it is not analyzed in real-time.

Sumo Logic tracing solution enables teams to monitor and troubleshoot transaction execution and performance across a Kubernetes-based application. Tracing data is fully integrated with logs, metrics, and metadata in order to provide a seamless end-to-end experience during the process of managing and responding to production incidents, and to reduce downtime by streamlining root cause analysis.

Sumo Logic RUM supports the Open Telemetry (OTel) instrumentation libraries and other open-source components from the Cloud Native Computing Foundation (CNCF) to collect logs, metrics, and distributed tracing data. We will look at OTel in more detail in the coming chapter.





Kubernetes observability best practices

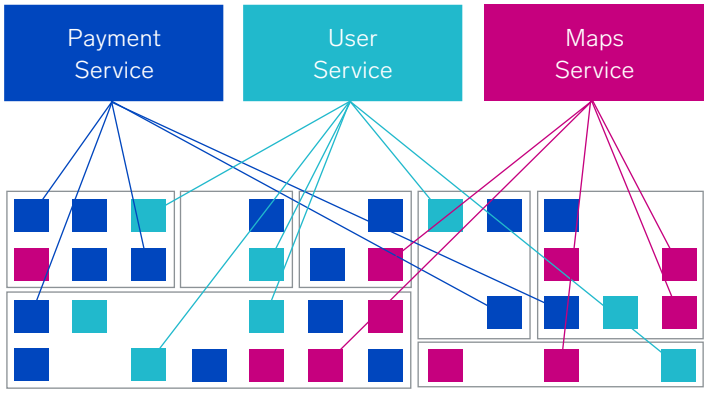
Kubernetes data sources are ephemeral, and that presents challenges, particularly when monitoring metrics. You have to track things like the number of instances in a pod, expected on-progress deployment, health checks, network data, and memory usage on containers…among many others!

Modern Kubernetes discoverability

Legacy monitoring solutions often struggle to effectively monitor microservices due to their server-based approach. This can make it difficult to quickly identify and fix customer and security issues that may be related to problems at the pod, container or node level in the infrastructure.

Sumo Logic has a different approach to Kubernetes monitoring, offering a solution that lets you navigate data from multiple dashboard views customized to fit your needs, whether you want to focus on the service, namespace, cluster, node or container level. This helps you focus on troubleshooting, rather than manually parsing through data without visualization tools.

Infrastructure-centric visibility

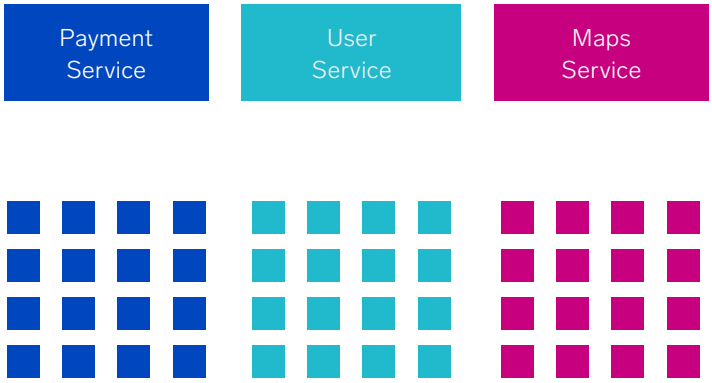


Complex

Slow to find and troubleshoot issues

Disconnected from the customer reality

Service-centric visibility



Simple

Quick to find and troubleshoot issues

Tightly connected to the customer reality

Traditional monitoring tools have a few problems.

1. **They typically look at applications from a hardware or server-centric perspective.** This makes sense for legacy solutions where the underlying infrastructure would often stay the same for months or even years, but this is no longer the case.

2. **Most solutions only provide visibility into a piece of the Kubernetes environment.** Admins are forced to navigate between tools for logs, metrics, events and security threats to build a real-time picture of application health.

3. **The data is fragmented.** A metric might be tagged with the pod and cluster it was collected from, while a log might be labeled using a different naming convention. The metadata enrichment process must be streamlined and centralized to gain consistent tagging, and therefore, correlation. This can make it near impossible to connect the dots between metrics on a node to logs from a pod in that node.

The reality is that pods, nodes, even clusters can all be destroyed and rebuilt with ease. Effectively monitoring what is running in Kubernetes means to monitor at the application level, focusing on the service and deployment abstractions. Understanding what is happening from a service and deployment perspective is critical to understanding the overall health of your application, and by extension, the customer experience. Monitoring solutions should align with the way Kubernetes is organized, as opposed to trying to fit Kubernetes into our legacy modes.

To learn more about this topic, check out this Advanced Metrics Certification Course.

Troubleshooting Kubernetes deployments

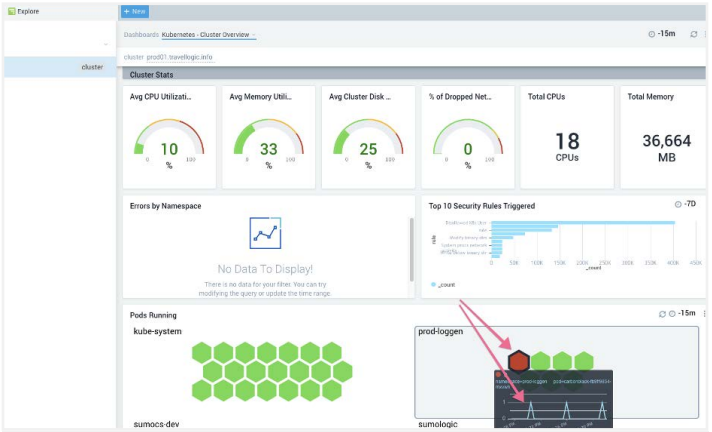
A common issue that happens when running Kubernetes-based applications is a crash loop backoff scenario. When a container in a Kubernetes cluster repeatedly crashes and restarts but cannot run to completion, you get a loop of crashing and restarting, also known as a crash loop. This can happen for a variety of reasons, such as an application error, resource constraints, or issues with the container itself. We have an example like this featured in our Kubernetes metrics certification course. To troubleshoot, you can follow a standard observe, orient, decide, act (OODA) cycle:

OODA is a military term originally coined by Colonel Boyd, a United States Air Force fighter pilot and Pentagon consultant. According to Boyd, the key to victory is the ability to create situations in which one can make appropriate decisions more quickly than one's opponent. While this was originally developed for air-to-air combat, his process easily applies to internet security. This is why OODA loops were eventually adopted as a standard strategy for the center for internet security (CIS). Taking a templated approach when dealing with app deployments frees development teams up to focus on the issues that automation can't address. By building and automating an "OODA" cycle, devops engineers, security analysts, and security engineers are all able to speedily pinpoint issues, determine available options, decide on a remediation strategy, and implement it. This frees up the team to work on more interesting projects and less monotonous tasks. To learn more about OODA cycles and how to use this military strategy for better observability and security, check out this talk.

John R. Boyd – Colonel, United States Air Force

Observe: You can use the Kubernetes dashboard in Sumo Logic to view the status of your pods in real-time and track the number of restarts over time. Start by looking at your cluster view and then drill down to identify the issue.

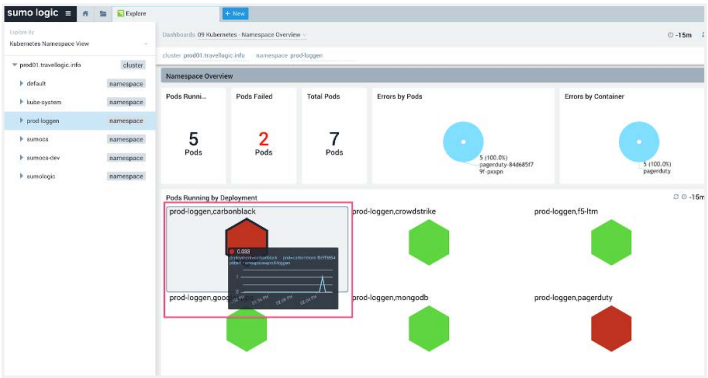
You see the cluster view for an application while running the Sumo Logic observability tool. Immediately, you can see that the prod-loggen namespace is experiencing errors. There are several pods in red in the prod-loggen namespace. Let's focus on the Carbonblack Pod and investigate what's causing the issue by drilling down into that namespace.



Observe

Orient: Get more context on the issue. Analyze the cause of the crashes by looking for clues in the data visualization tool that can help you understand why the container is crashing. This could include resource utilization data, application errors, or issues with the container itself. You may analyze logs from the pod, the host node, or other related pods and services to identify patterns and correlations that may be relevant to the issue.

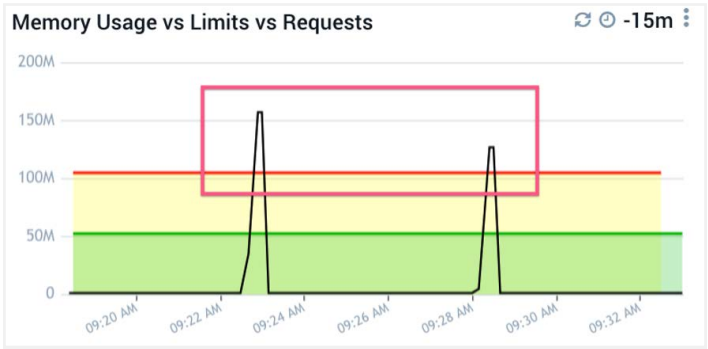
Once you have identified the problematic pod, use Sumo Logic to gather more context about the issue by drilling down into the namespace view. Hovering over the carbonblack pod, you will see your first real clue. The metric bar goes up and down, which means the prod-loggen.carbonblack pod is starting and stopping. Let's continue to investigate!



Orient

Let's say you see a few more clues while orienting yourself with this pod. Perhaps you see that there are way too many restarts, OOM killed metrics pop up, and you investigate to find out why the pod is getting OOMKilled. You look beyond that and find out that the pod is experiencing high memory utilization by hovering over memory usage limits vs. the request panel in Sumo Logic.

Decide: Based on the information you have gathered, decide on a course of action to resolve the issue. This may involve modifying the pod's configuration, scaling up or down the number of pods.



Decide and Act

Act: Congrats! You've made it to the final step in a given OODA cycle. Use Sumo Logic to implement the chosen course of action from the prior step and monitor the status of the pod to ensure that it is running as expected. If the issue persists, continue to iterate through the OODA cycle until the issue is resolved.

In the case of this example in particular, once you have confirmed the reason the container is terminated (due to memory allocation issues), you can adjust the container memory limit in K8s to resolve this issue. You can also learn more about memory resource allocations in this Medium post.

Chapter 3

Why unified collection is important for Kubernetes monitoring

OpenTelemetry (OTel) is an open standard for distributed tracing and metric data collection. At the time of writing this ebook, OpenTelemetry has the second-most active open-source CNCF project behind Kubernetes. This comes as no surprise, given the need in this industry for unified collection and a centralized telemetry pipeline. To effectively monitor a Kubernetes environment, a centralized telemetry pipeline is necessary to collect, analyze and manage all data from IT assets consistently. With that in mind, let's start with why unified collection is important in the field of observability and the fundamentals of telemetry pipelines.

Why unified collection matters

Once a disparate system scales up, standardization around telemetry becomes very important.

Vendor-neutrality

OpenTelemetry is a standard that offers vendor-agnostic observability for metrics and distributed tracing. Most commercial monitoring and observability solutions will come with some vendor lock-in, as they do not use a single agent for collection. You could become dependent on a vendor's unique data format and APIs for managing, querying, and analyzing data.

Even when using open-source software and agents from Jaeger or Prometheus, the choice to collect data from separate agents could mean years of developing and managing a confusing and ever-growing mess of ad-hoc scripts and custom APIs. Additionally, the vendor's solution may not be easily integrated and correlated with data from other sources. This makes it difficult to find anomalies or detect potential security threats in your application.

Scalability

We talk a lot about scalability in this paper, but it is worth noting that the OpenTelemetry standard helps when gathering logs, metrics, and trace data from multiple disparate applications, microservices, containers, and pods. You can do this via a single agent for collecting and forwarding this data from all layers of your system throughout the telemetry pipeline. You must be able to gain end-to-end observability throughout a large, federated system to troubleshoot easily. Having a holistic view that can easily scale up and down with your system is vital for your productivity and peace of mind.

A brief history of OpenTelemetry

The cloud-native telemetry landscape tracks and gathers data from distributed traces, metrics, and logs. You would typically integrate all of these data sources with manual instrumentation. Industry-wide collaboration had been sorely lacking until recently (Aug 2021), but much needed due to the complexity involved before auto-instrumentation. You would have needed to navigate dozens of custom API libraries and spend hours developing ad-hoc code.

OpenTracing and OpenCensus were the first projects to lead standards unification in the observability space, but they were ultimately two different projects focusing on fundamentally different architectures. This caused a lot of confusion in the community and developers were unable to achieve the common goal of unified standards for data collection.

OpenTelemetry comes from unifying projects OpenCensus and OpenTracing. OTel routes traces, metrics, logs and other application telemetry data to your preferred backend.

A few key differences between these three are highlighted below:

OpenCensus

- metrics and tracing focused
- originated at Google, based on Census concepts
- Omnition started incorporating it into a complete observability solution

OpenTracing

- distributed-tracing focused
- originated at Google, based on Dapper concepts
- CNCF project since 2016

OpenTelemetry

- merge of OpenCensus + OpenTracing
- announced May 2019
- backed by all major vendors
- CNCF project (incubating since Aug 2021)

Luckily, the leadership of the OpenTracing and OpenCensus projects were able to come together and focus on unifying their architectures and communities for a common goal: the transition to OpenTelemetry. Instead of focusing on building more features, engineers from these two communities built software bridges to architect backward compatibility across projects. They prioritized the goal of simplifying the telemetry pipeline collection process and focused on standardizing a single solution for the broader community.

For more information, check out this history on OTel from the CNCF.

What is a telemetry pipeline?

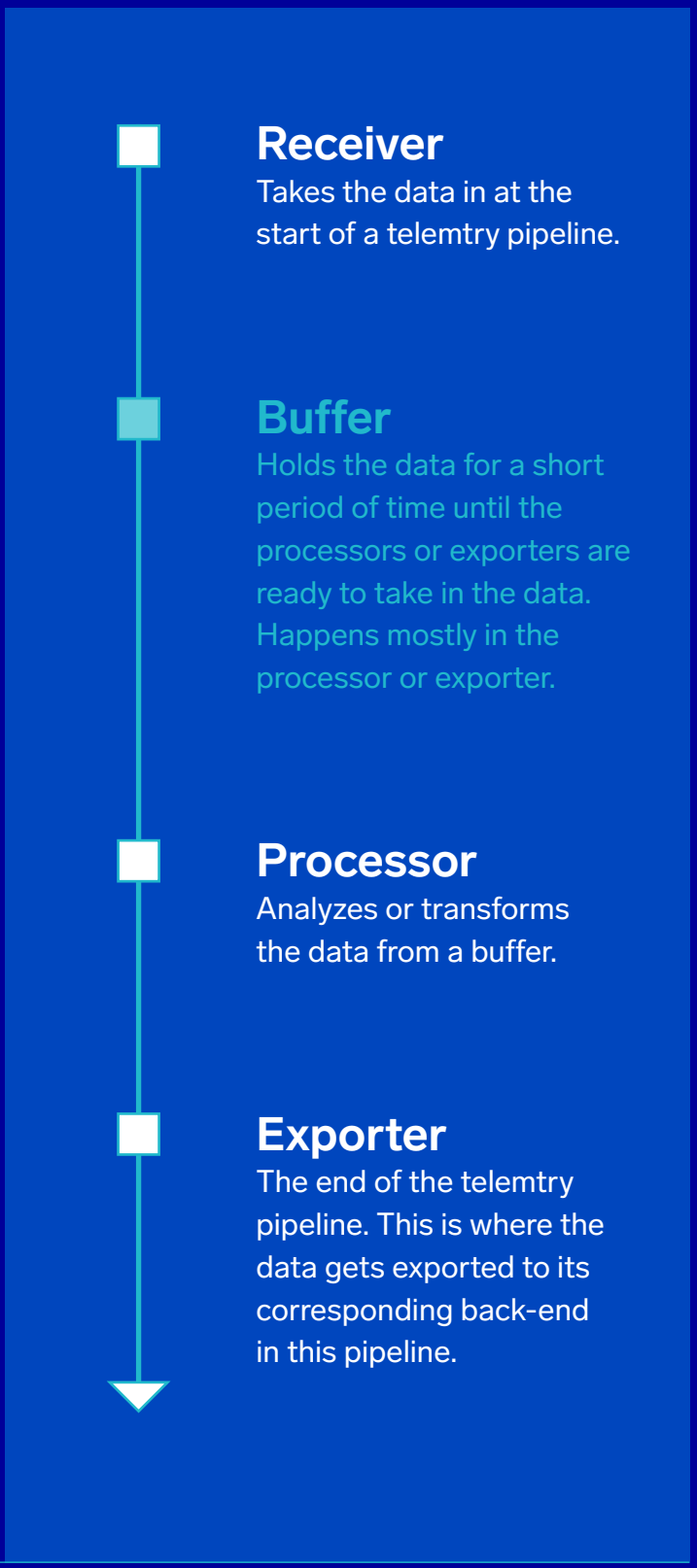
Telemetry pipelines route data (**logs**, **metrics**, and **traces**) from where they are generated to wherever they need to go. This also intersects with how this routing data is expressed via monitoring. Telemetry pipelines not only filter data, but also produce enriched metadata that is available to various backends regarding disparate information such as Kubernetes container information, region information, GeoIP information, and other logging and trace data.

In the image above, you see an example of a simplistic telemetry pipeline and its corresponding components. An extremely basic setup has a receiver, buffer, and exporter. The receiver is how you get data into the collector, at the start of a telemetry pipeline. Our distro supports a wide variety of receivers and data formats.

Typically, a pipeline chains buffers and processors and takes data from a buffer to analyze or transform the data.

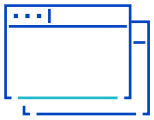
Many developers want to automate compliance checks, for example, on deployments. They use software that takes data from exported logs and goes through various compliance and security checks (like GDPR and HIPAA) via chains of buffers and processors before finally exporting a result. A telemetry pipeline can get extremely complex, but typically keeps the pattern of receiver, buffer, processor and exporter for every type of backend.

Learn more about components (receivers, processors, exporters) included in our OTel Distro. Sumo Logic delivers telemetry pipeline components. The ones with an asterisk are upstream OpenTelemetry components with additional contributions from us.



How unified telemetry works

Telemetry pipelines divide into three main layers: the application layer, the collector layer, and the backend layer. Of course, it would be quite helpful to use a single agent or have a single back end. However, then you'd face issues associated with vendor lock-in on a single agent. Moreover, it is unlikely that you will be sending everything to a single backend if you have a highly federated system or security countermeasures that do not allow you to store all application data in one place.

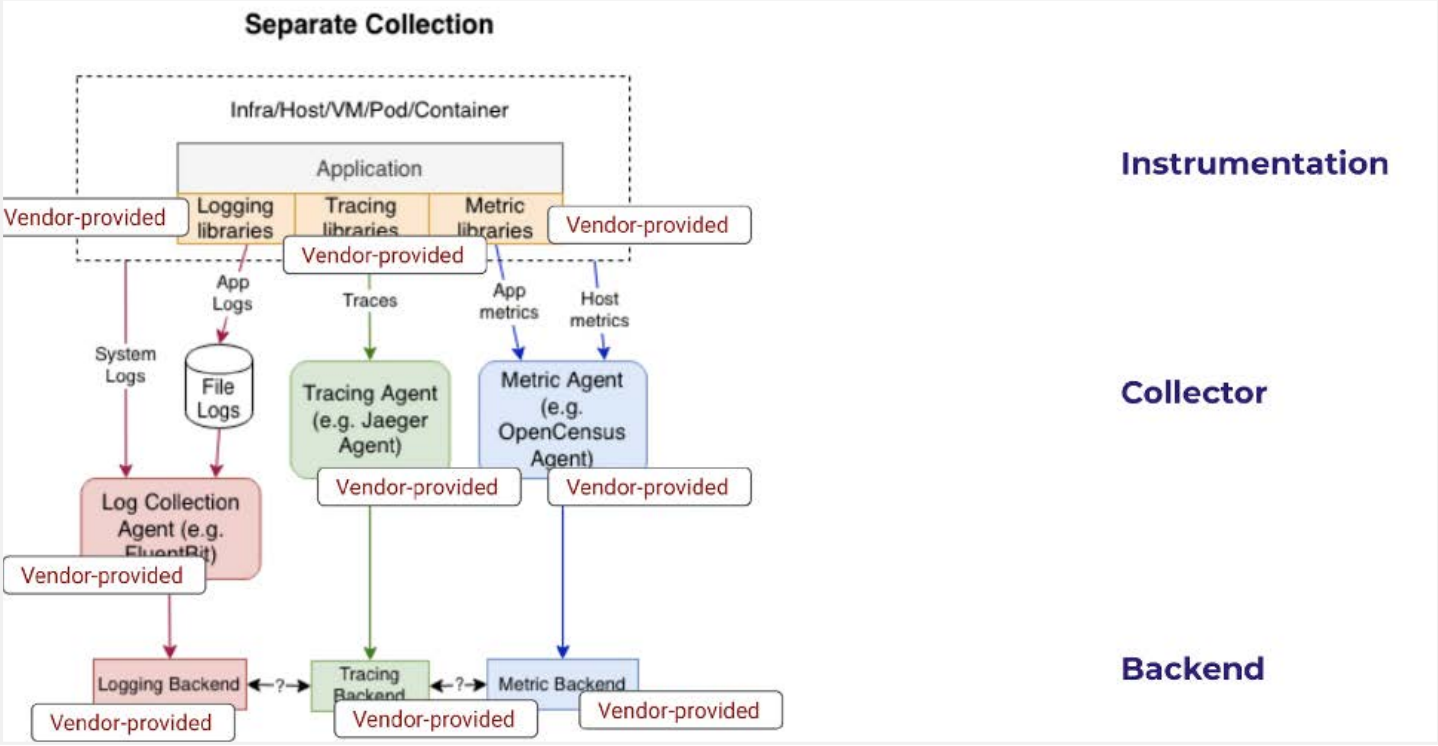


Application layer: As applications run, they can create machine data output in the form of logs, metrics and traces. Developers can turn up (or down) the quantity and granularity of data that they collect at the application level by instrumenting their applications to emit more data. This instrumentation can happen manually (using an SDK), semi-automatically or automatically.

With the solutions available just a couple of years ago, practitioners would need to have a separate library to handle each of the signals from many disparate agents with their own collector (such as Jaegar for traces, Prometheus for metrics, etc). OTel typically aims to provide instrumentation libraries for all majorly adopted languages (like Go or JS). Vendors like Sumo Logic can support data created with vanilla OT libraries or with their own library distributions, like our Sumo Logic OpenTelemetry JS library.

Collector: The collector provides data processing capabilities (to analyze and transform data in the collection pipeline), buffering, and handling connections to the backend. It also enriches the information to include metadata about the origin of data. This enrichment helps operators navigate the signals which now include information about the source - such as the instance ID, Kubernetes pod, custom tags and so on. OTel typically aims to unify collection through a collector (like our OT Collector Distribution) at this level.

Backend: The backend stores the data and provides query and visualization capabilities. This is what operators use to monitor what's going on in their critical application environments in real-time, such as container information, GeoIP, pod health, and other customer data. For Sumo Logic, our backend is our scalable SaaS service. The OpenTelemetry standard is to use the OTLP protocol to transfer telemetry data between libraries, collectors and backends.



The image above represents observability without the OTel standard in place. Here we see a hodgepodge of vendor-provided collection, instrumentation and backends.

There are a lot of moving parts at each layer. Additionally, since there are several instrumentation libraries available through different vendors when organizations want to switch from one to another, it can be problematic. Organizations might have already invested some time in writing integrations using disparate APIs with unique configurations. This would need to be redone when switching between libraries.

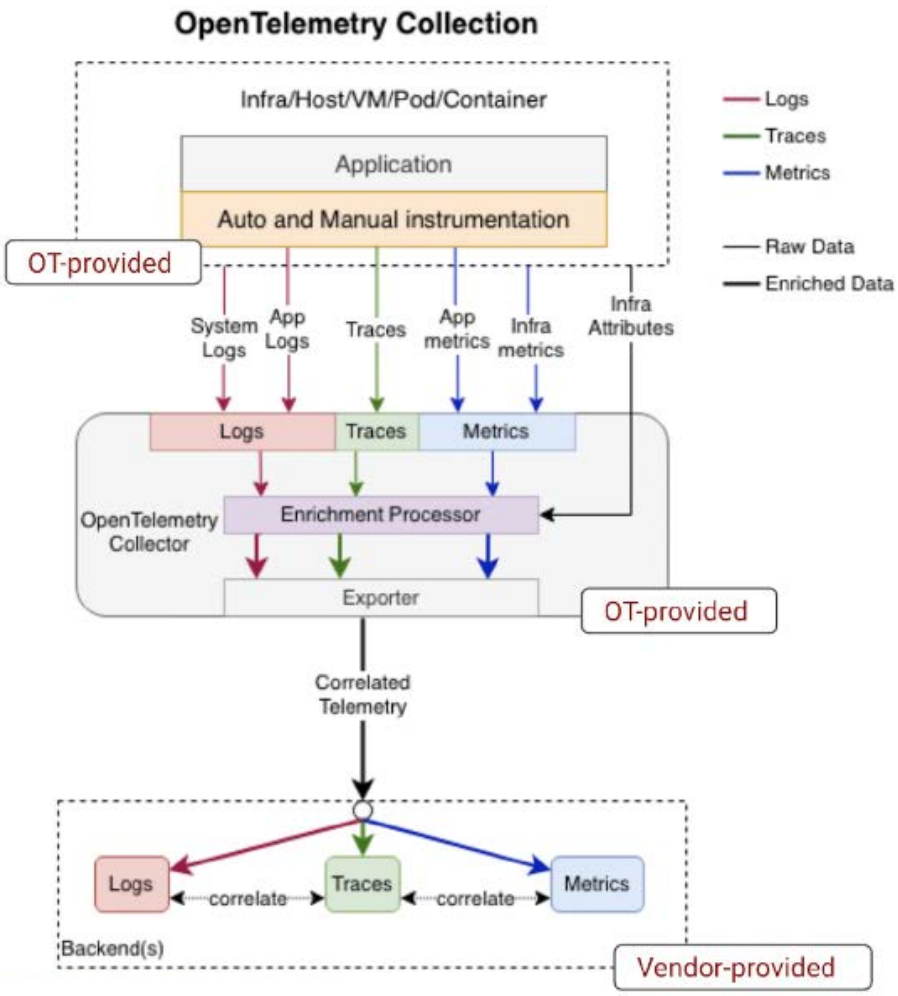
What a headache! Users have options like running a lot more ingress/egress or re-working architecture every time they choose to work with a new library or agent.

Wouldn't it be much easier if there were a standard that vendors adhered to which allows for a more unified collection method?

Data collection with OpenTelemetry

While unified collection methodology is a unicorn and the community is not fully there today, here is a visual representation of a unified collection agent using the OpenTelemetry standard. When we discovered OpenTelemetry at Sumo Logic, we knew we would want to standardize all our collection on it to benefit our customers.

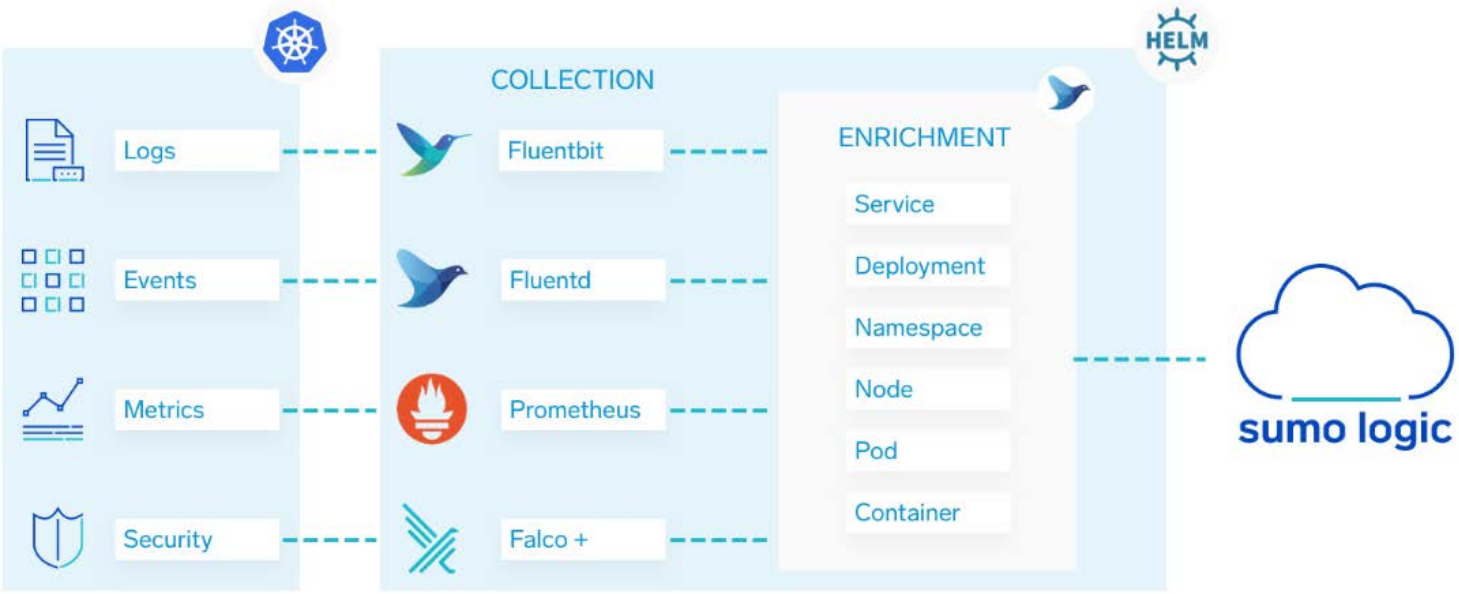
Observability 2.0 uses the OpenTelemetry standard with OTLP, for both instrumentation libraries and collection agents. The prior diagram required a separate library and agent for each of the signals. This is not the case when using OpenTelemetry, where almost everything except the backend is streamlined and provided by the initiative. Here, we see there are common libraries and collectors. The vendors can provide their own distributions, but they all share the same core code.



Kubernetes data ingestion with Sumo Logic

Now that we covered the fundamentals of OpenTelemetry and the importance of unified collection for metrics, events, logs, and traces, we can discuss how this fits in with Sumo Logic. Kubernetes monitoring in particular has many challenges involved with the high scalability of IT infrastructure, ephemeral data and metadata enrichment that is needed to drill down and investigate problems with your cloud-native infrastructure.

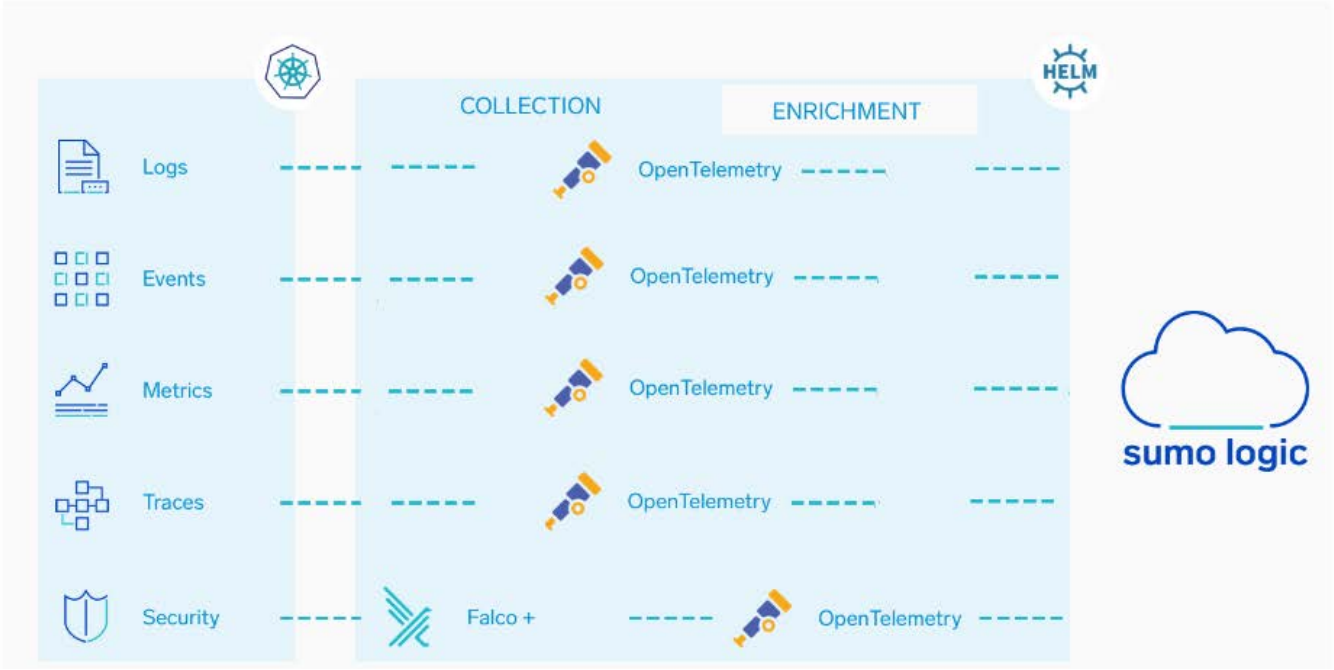
In the past, we talked about data collection and ingestion from the perspective of several different agents. It used to look like this:



Collection

As you can see from the image above, we traditionally collected log, metric, event, and security data via Fluentbit, Fluentd, Prometheus, and Falco — all open source data collectors maintained by the CNCF.

Using open-source tools for collection and data enrichment prior to the adoption of OpenTelemetry was ideal given that these collectors receive a lot of community support and typically have integrations for any type of data you might want to collect. However, it is not as scalable and significantly more time-consuming than using a single agent.



Sumo Logic has worked toward unifying data ingest to all be on our OTel Distro. We invested early on to use OpenTelemetry as part of our data collection and the OpenTelemetry agent replaces Fluentbit and Fluentd for logs and events collection. Eventually, the goal is to also replace Prometheus for metrics collection as well. Overall, this helps DevOps practitioners with ease of use, standardization, and extensibility that helps future-proof the observability of their cloud-native infrastructure.

An incredibly practical advantage of replacing Fluentbit and Fluentd in favor of our OTel Distro has saved time internally. Fluentd is slow to run and we were utilizing hundreds of pipelines that were able to be replaced with dozens of Open Telemetry Collectors (OTCs). Fluentbit is C-based and more time-consuming to code and debug. By using OTC instead of Fluentbit, we were able to use Go libraries instead, which saves hours of time for our engineers for many issues and pull requests.

Metadata enrichment

Metadata enables us to build a hierarchical view of a cluster. By connecting pods to their services or grouping nodes by cluster, it becomes easier to explore the Kubernetes stack. By tapping into the auto-discovery capabilities inherent in Prometheus, we can ensure that the hierarchy visualized in Sumo Logic is accurate and up to date.

Not only do we tag the container, pod, node, and cluster, but we also identify the service and deployment for each entry. K8s is rich with metadata — pods, labels, namespaces, etc. — that are descriptors of these entities and how they relate to one another.

However, by themselves, they don't expose everything. The enrichment pipeline ensures that you know for every log, metric, event, etc., what container it's from, what pod that container is in, what the pod labels are, what node it's on, what namespace it runs in, what cluster it runs in, etc.

Collection setup

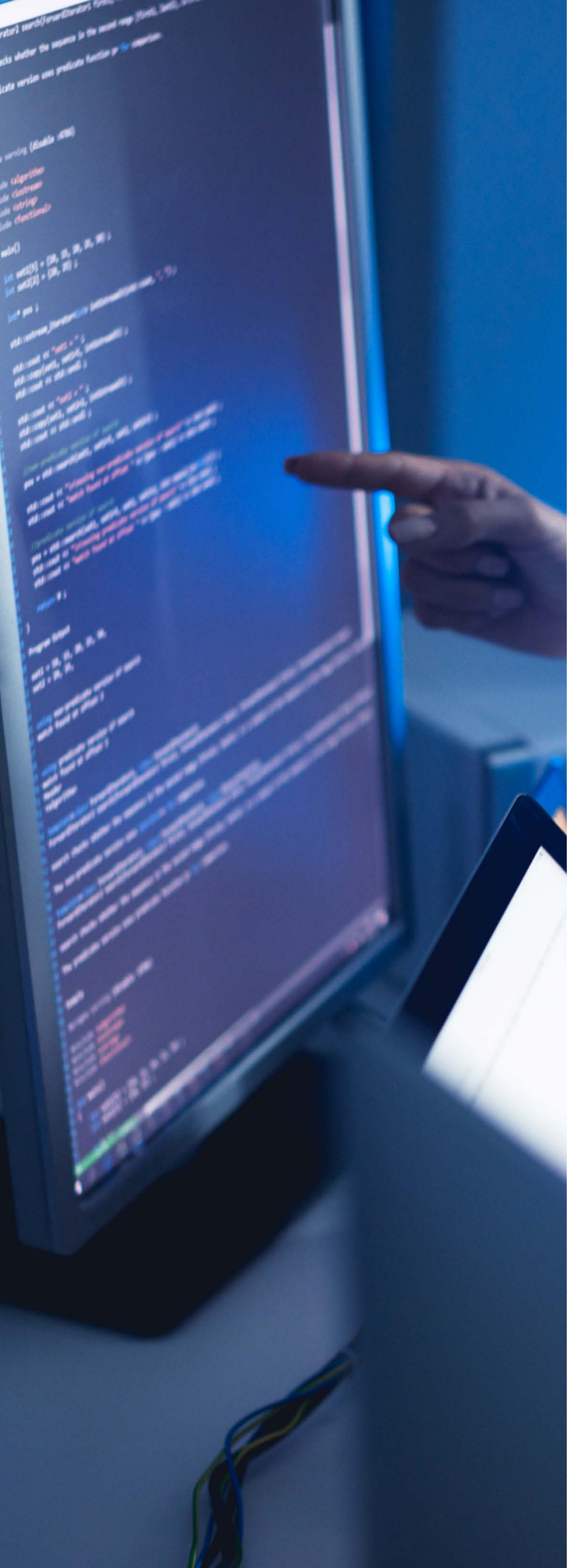
All of this is set up with a single Helm chart command. It's as easy as "helm install sumologic". You give us an access key and ID that we will use to create the collector and sources, it will configure the sources of the data collection (including 3rd party agents like Fluentbit, Fluentd, and Prometheus) and deploy them. Then, the data comes over to Sumo Logic.

Most open source data collectors today support OpenTelemetry. OpenTelemetry lets teams focus their energies on analyzing data rather than figuring out how to collect it, no matter how many data sources they have or how disparate those sources are in form and format.

To illustrate how easy it is to get started with OpenTelemetry-based Kubernetes observability, let's look at what it takes to deploy Sumo Logic's OpenTelemetry data collector, which uses OTel to pull data from K8s and other environments. In a Kubernetes environment, you can deploy the Sumo Logic OpenTelemetry collector as a Helm chart using a helm command:

```
helm upgrade --install collection sumologic/sumologic \
  --namespace sumologic \
  --create-namespace \
  --set sumologic.accessId=<SUMO_ACCESS_ID> \
  --set sumologic.accessKey=<SUMO_ACCESS_KEY> \
  --set sumologic.clusterName="<MY_CLUSTER_NAME>" \
  --set sumologic.traces.enabled=true
```

Once the collector is installed, you can begin moving data into Sumo Logic to build a unified observability pipeline for your Kubernetes cluster. For full details and configuration options, check out this step-by-step explanation for running the common Open Telemetry demo for Kubernetes or Docker data.



Chapter 4

Common trends in observability today

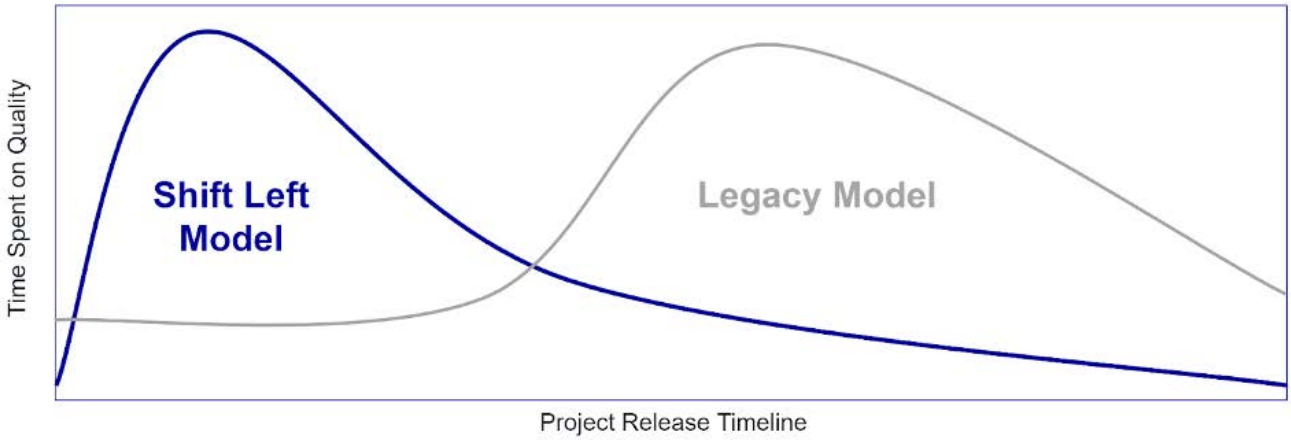
Kubernetes monitoring is a practice that is ever-changing and growing with new trends emerging every few months. Let's focus on three common observability trends that have not only stuck around, but grown since the writing of our last K8s ebook in 2019.

How to "shift left"

You've probably heard the term "shift left" in the context of security best practices, but it is highly relevant in the world of containerization and Kubernetes as well. The practice of shifting left has the benefit of smoother containerized app deployments and better end-user experience.

Shift left is a popular best practice for catching and addressing issues early on in the software development and deployment process, instead of waiting until issues multiply later on, closer to or even after deployments. This is especially pertinent in the world of security but can be applied to DevOps to help with faster deployments, higher quality code and more cost-efficiency. The key is getting developers involved early on in deployment cycles and making a concerted effort as an organization to ensure things run smoothly, rather than placing the onus all on quality assurance teams, security analysts, or ITOps.

Practitioners	Shift-left the activities for security, devops and business intelligence to the earliest stages of release life cycles.
Managers and Team Leads	Encourage a culture of accountability and trust on the practitioner level.



This image above illustrates the relationship between the time invested in quality (on the y-axis) and the project life cycle (on the x-axis). It suggests that investing in quality early on in the project lifecycle results in fewer security risks and quality issues in the long run. This strategy is most effective when there is a strong sense of trust and accountability between developers, security analysts and operators. Developers can take the initiative by adopting a shift left mindset, where quality is prioritized from the beginning of the project.

The practice of shifting left is especially important in the case of container orchestration, because you are coding, scaling and managing an application as one cohesive team. Breaking down organizational silos allows you to pay more attention to the quality of your applications. It is especially pertinent to make sure applications are built, tested and deployed in a consistent and reliable manner by operationalizing your efforts as one team.

Instead of having a small team fight a losing battle against bad actors, the shift left culture means that organizational leaders are ultimately accountable for security and improved customer experience. As subject matter experts in deployment pipelines, putting trust in development teams is essential to getting more out of deployments. If you are a developer and you are reading this to learn more about securing your infrastructure,you can work to get more involved earlier in production cycles.

In summary, some benefits of shift left include:

- **Improved customer experience** in your application due to fewer quality issues. By shifting left, developers test software thoroughly early in the process, thereby minimizing bugs. Additionally, this allows developers to get earlier user feedback and make changes to keep customers happy.
- **Faster time to market** by reducing the risk of delays with a more polished end product.
- **More security** by breaking down organizational silos so that developers mitigate threats and bad actors earlier on in the app deployment process.

"Shadow IT changes" in Kubernetes-based infrastructure

One of the most important Kubernetes monitoring best practices is to document any and all changes in your IT infrastructure. We have covered the issues that come with Kubernetes data ephemerality many times in this paper, but we have yet to cover issues that result from people making unauthorized or "shadow" changes in IT infrastructure.

These changes might not be wrong per se, but the lack of recordkeeping means that if something does go wrong down the line in deployment the team will have no understanding why. On top of this, data ephemerality means that records of the prior state of deployment might be unavailable.

Identifying and remediating systems that have drifted from their desired, compliant state can be a significant issue in a K8s cluster, as it is running a distributed system and it is necessary to keep all the nodes in the cluster in sync. Additionally, ambiguous security policies can limit the ability to use and maintain the infrastructure. Moreover, the evanescent nature of Kubernetes data sources can add complexity and delay in identifying issues and eventual deployments.

Kubernetes offers several features and tools to help you manage and automate containerized infrastructure like declarative configuration and self-healing capabilities. However, you'll also need a good understanding of the platform to use these features effectively. Plus you'll need to keep the cluster updated and running smoothly. Wouldn't it be great if you did not have to manually parse through esoteric YAML files and logs to identify issues or document change management?

IT professionals need to ensure compliance with various regulations and standards, which can be a challenge when using k8s, as it is a relatively new technology and the regulations are still evolving. To address this, you can leverage various tools and solutions that are available to help you ensure compliance, such as Kubernetes-native compliance-as-a-service and compliance-as-code offerings. Continuous integration solutions (from companies like Ansible, Puppet, or Chef) often use compliance solutions to enact changes as needed, but they still need to detect issues that live as ephemeral data flows through Kubernetes-based infrastructure.

Sumo Logic integrates with well-known DevOps toolchains, to ingest data and turn it into insights around release and operational performance. This data is also used to uncover unknown shadow changes or anomalous behavior in the underlying applications or systems.

- **Schema-on-demand:** Sumo Logic does not require log definitions to be pre-defined, allowing us to absorb changes to log signatures while still extracting maximum value from each message
- **Release baseline:** Using patented LogCompare and TimeCompare analytics, Sumo Logic identifies performance or log definition anomalies introduced by new product release
- **Ephemeral instances:** Sumo Logic can automatically monitor and manage ephemeral instances (cloud, K8s, serverless) and alert operators as patterns change

For more information, check out our audit and compliance page.

You might also find our compliance monitoring page useful.

AIOps and Kubernetes monitoring

Artificial Intelligence for IT Operations (AIOps) has become an increasingly popular term used in the world of monitoring and observability, particularly for Kubernetes infrastructure. Given the popular interest in Artificial intelligence (AI), it is no surprise that this topic has made its way into the ITOps space.

At Sumo Logic, we define AIOps as "the use of artificial intelligence, machine learning, and pattern recognition to perform and automate tasks normally executed by IT operations". Often, labeling a piece of intellectual property as "AIOps" obfuscates the complexity of the underlying functionality of that system, but I like to think of this as several inter-woven mathematical models.

Machine learning techniques are simply using mathematical models that help analyze massive amounts of data generated by Kubernetes clusters, such as metrics, events, logs and traces. DevOps practitioners and security analysts then use these models to identify and predict potential issues with their Kubernetes infrastructure and potentially automate resolution from known playbooks. Example use cases include providing enriched data, monitoring and troubleshooting pod issues, optimizing and tracking resource utilization and analyzing or providing insights into system alerts.

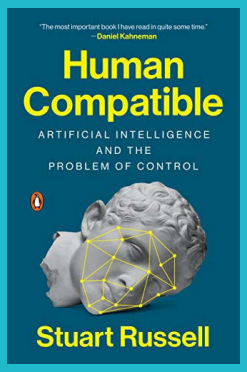
As you can see, AIOps would be highly relevant in the world of Kubernetes monitoring, as a simple alert will not generally get you very far. Logs alone offer a flat interface that is great when analyzing system history, but what would you do in the case of thousands of pods or hundreds of clusters? This data needs to be parsed and analyzed.

AIOps could be relevant to automated anomaly detection or to perform root cause analysis to quickly pin down the underlying issues in Kubernetes-based infrastructure. Wouldn't it be wonderful to have a monitoring tool that could go so far as to make recommendations for resource utilization or automatically spin up new Kubernetes instances and adjust limits?

The technology for AIOps is constantly growing, but it is important to keep a close eye on what can be automated while not losing track of the inner workings of how the overall system runs. IT practitioners and developers who want to understand everything about how their system operates speak the language of bots, daemons, subroutines, events and triggers.

One caveat I wanted to mention with AIOps is the idea that there is a "soul" inside of the machine that is somehow so advanced that there can no longer be human-in-the-loop interactions with DevOps practitioners who understand the fundamental infrastructure. Technical chauvinism is behind the idea that AIOps is somehow a superior magical black box that lives outside of the ability for humans to understand it.

Stuart Russell touches on this idea in *Human Compatible - AI and the problem of control*. In this book, he explores whether the computer system is a tool of humans or if humans have become the tools of the computer system, supplying the information and fixing bugs when necessary but no longer understanding how the whole system works.



Disastrous outcomes happen when we do not properly understand computer systems or their automation processes. Russell references a computer glitch on April 3, 2018, that caused 15,000 flights in Europe to be delayed or canceled and one in 2010, where a trading algorithm caused the infamous "flash crash" on the NYSE, wiping out one trillion dollars and shutting down the exchange.

Unfortunately, what happened is still not well understood or discussed in detail, but it was likely a failure of the AIOps pipeline of logic. It is important to retain sufficient understanding of the technological systems we use in order to retain our autonomy. AI has incredible enhancement capabilities, but DevOps engineers, site reliability managers, and security analysts are among the most important workers needed to keep systems going and pass the information along as humans progress into the future.

AI-based parsing utilities can help quickly identify threats (particularly in a highly federated Kubernetes-based system), making incident response times much shorter. Monitoring cloud-native systems present a variety of challenges, including a growing number of attack vectors, huge amounts of data to parse, and challenges with Kubernetes data ephemerality. The response to these issues is to have capabilities such as real user monitoring, anomaly and root cause detection, and monitoring analytics that provide insights into your Kubernetes-based system. With that in mind, let's review some key capabilities for AIOps software:

- **Monitoring.** This is Sumo Logic's bread and butter. Monitoring is used to gather data from disparate sources (such as a variety of pods, namespaces, and clusters) to look for patterns for analysis and action. At Sumo Logic, we use metrics, events, logs and trace analytics to process and analyze massive amounts of data from multiple sources.

- **Correlation.** Another term that is used to describe this concept is called "grouping". This is when patterns can be found in system data. For example, anomaly detection can be used in security posture management to find various attack vectors, and even use predictive analytics to anticipate known threats (such as the crash loop backoff example from an earlier chapter).

  Sumo Logic correlates data such as the time of an incident, affected entities, and connections in infrastructure topology to deliver machine intelligence reports. The reports can help discern the probable cause of an incident, via anomaly detection, log summarization, clustering and dimensional analytics. Correlation helps reduce the noise from a variety of logs coming from disparate sources.

- **Analysis.** This is the ability to determine problems and root causes. This is where we not only correlate but parse the data to produce a human-readable output (such as data visualization). Users could read metrics and trace data in charts, graphs, and histograms to understand infrastructure topology in a visually based format. The alternative to analysis is a flatter interface, where all you see is a spreadsheet of confusing data. Machine parsing allows for better data interpretation and analysis.

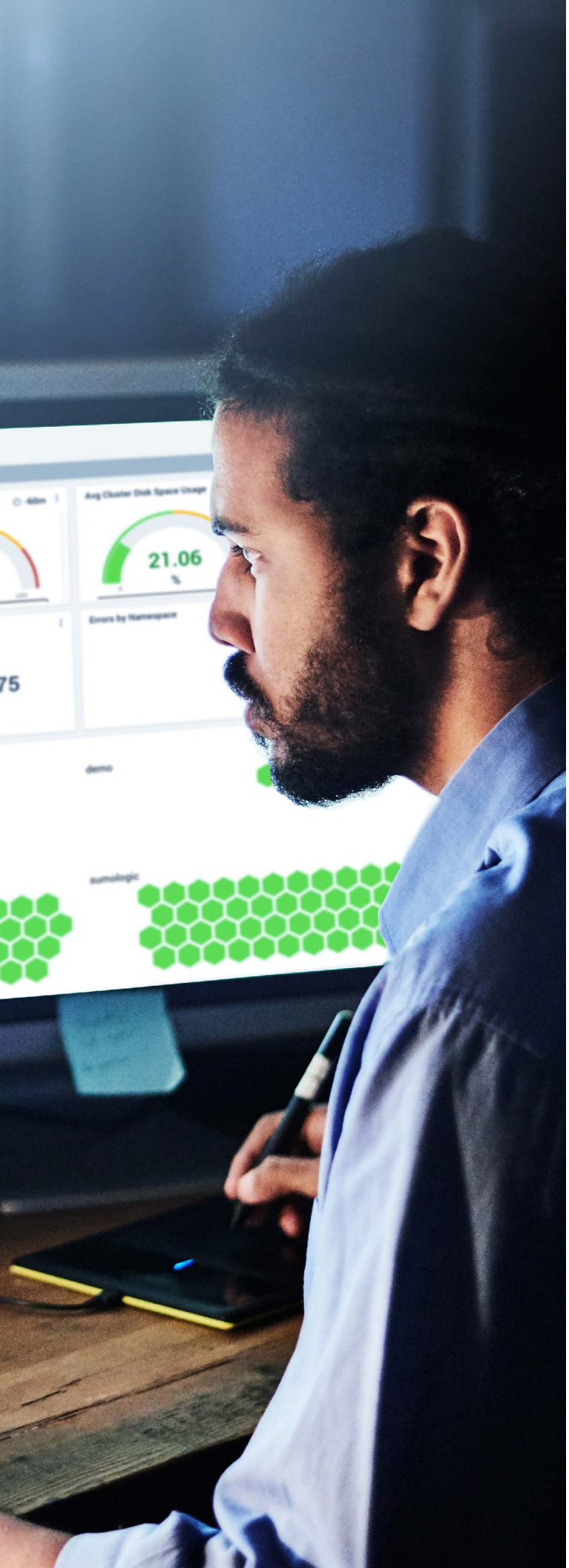
With the help of AIOps, IT teams can better collaborate and continue to break down the silos within organizations. Business process automation with human-in-the-loop response allows for faster intervention when there are incidents. In ITOps, the software can come with playbooks and links to dashboards that allow for better monitoring and communication channels that reduce incident response times.

AIOps software provides enriched machine data that can inform on what caused the incident, thereby allowing for more in-depth monitoring. Ultimately, this helps reduce not only the mean time to respond but mean time to remediation with human-in-the-loop protocols powered by the information from the machine.

Monitors and known playbooks can allow for auto-resolution features. Links to dashboards can also help with quicker diagnosis, but this is further helped by automatic notification via organizational human-readable channels (like email, Slack, Jira, GitHub). Furthermore, known issues and common alerts can be resolved via automation for transient issues (like spinning up a new k8s instance when resources are scarce)

The future of AIOps is nigh, as monitoring tools are now natively acquiring data and analyzing it. This emerging technology will only become more sophisticated as these same tools start enabling more advanced auto-resolution features. According to this public report from OMDIA in 2022, AIOps is "a response from domain expert systems to the rise in cloud-native applications and the use of cloud computing has added a layer of complexity to the role of IT operations".

AIOps functionality is most advanced in the case of performance monitoring, and has the most opportunity for growth in maturity in the coming years for security operations.



Chapter 5

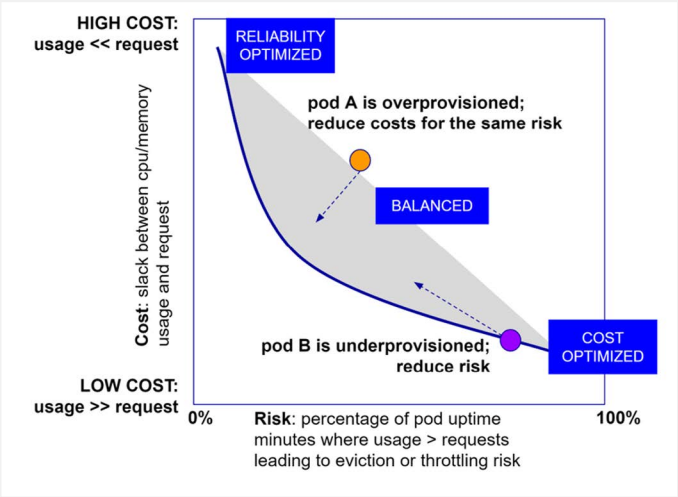
Monitoring Kubernetes deployments with Sumo Logic

Let's look at how Sumo Logic helps with DevSecOps. Get insight into common exploits in video games and how to fix them. Rather than simply regurgitating jargon, these attack vector examples serve to contextualize DevSecOps use cases in the world of IT observability.

Read on to learn more about login hacking, DDOS attacks and usage of containers for securing and scaling games.

Global Intelligence Service for Kubernetes

Global Intelligence Service (GIS) apps continuously profile containers running in Kubernetes deployments. In Kubernetes, a developer has to set CPU/memory requests and limits for their containers. If you configure these too low, your containers can be evicted by the K8s scheduler when usage for CPU or memory exceeds requests. If you set them too high like you see on the left side of the graph, you incur avoidable compute costs.



How widespread is this problem? About 40% of containers are under-provisioned among the containers we have profiled. Another 40% are overprovisioned. Many customers are using 220 distinct containers—each of which requires four parameters to be set. Trial and error is a tedious way to set these parameters as many customers are currently doing.

As demonstrated by the image, there is a tradeoff between cost and risk. If you go cheap and set low requests and limits, you incur a greater risk of OOM, throttling errors and resulting outages.. If you set them too high, you reduce risk but incur costs.

Our Global Intelligence Service (GIS) helps you find the right setting by comparing actual CPU/memory usage over the last eight days to recommend requests and limits. This helps you minimize risk of incidents due to under-provisioning CPU and memory while ensuring that you do not overreach on costs.

Striking the right balance between cost and security risk can be achieved by utilizing observability and monitoring tools like Sumo Logic. There are a wealth of tools available to help provide container level Kubernetes security visibility. These tools can provide insight into behavioral activity to detect anomalous activity and security events. Additionally, it is essential to look into compliance and audit events.

To learn more about available integrations, check out our GIS docs page.

DevSecOps in video games

Games are generally not fun if your data isn't secure or if hackers ruin the experience in-game. Additionally, anti-cheat software often isn't enough to secure user data or stop a growing number of black hats from trying new exploits.

As IT organizations build their response protocols, they greatly depend on active monitoring. It is a balancing act between game developers meeting performance optimization requirements and security analysts managing countermeasures.

In many cases, these teams can even be at odds. If they could encrypt all game-related network transmissions, they would. However, highly federated and disparate cloud-native systems are often too scaled up to do this without slowing games down. DevSecOps teams will need to rely on live metrics and logs, while optimizing for performance.

MMORPG hacking exploits: black hat players crashing servers

A common exploit for MMORPGs like World of Warcraft or Destiny is to send an attack packet continuously, many times per second, to "cheese" the server into allowing for the hacker to easily defeat dungeons and bosses. As part of this same exploit, hackers do not want to be interrupted by other players or have anyone see what they are doing.

Black hat players crash the zone leading up to the boss or dungeon so that they can defeat the zone they are in without being interrupted. The hacker takes another character and warps that one to another invalid location (like a div by zero location) and, if the server does not account for it, it starts throwing errors and lagging. The zone eventually crashes.

A version of this exploit is demonstrated in Mythic Quest, a popular TV show about the lives of game developers. Essentially, it would be possible for the hackers to run multiple attack packets through a distributed denial of service (DDOS) attack.

How to secure against DDOS attacks

There are many ways to monitor, catch and mitigate DDOS attacks. If the makers of Mythic Quest had thought to use the following methods, their servers might not have crashed.

Game developers can monitor network traffic and see if their server is getting bombarded by the same packet from one client repeatedly. If they detect that this is happening, the system can flag this behavior.

The server should see a bunch of errors and the CPU utilization go up to 100 percent (when normally it should be running much lower). By monitoring the network traffic prior to the hacker killing the raid boss, the security analysts should be able to detect an exploit of this kind. They can set an alert to go off when there is a significant network traffic spike from a particular client to the server.

Over time, security analysts and DevOps practitioners can build a library of expectations, and when behavior deviates from that, it can give an alert. Sumo helps you quickly hone in on this suspicious behavior using the Geo Lookup operator to confirm and identify IP address's locations on a visual map.

An even easier solution for developers is simply to check if an in-game location is valid before sending a user to that location. This is a great example of why it is important for developers to consider security when coding applications by shifting left. There is nothing IT can do other than forcefully move a character to a valid location automatically or reboot the server. When setting an alert, they can at least see when things are going wrong.

Proper load balancing to prevent server crashes

To bring this topic back to the ambitious Mythic Quest game, they were trying to create persistent changes in-game while running a server on-premises. Any game that implements persistent changes across an in-game map (like with MMORPGs) runs into several performance bottlenecks that can lead to the server crashing without the right load-balancing solution. This is why that would best be achieved with modern applications and containers that are managed with Kubernetes.

What more sophisticated games can do to address the need for scaling up infrastructure is to run a server object container streaming, where each region in game is a different container. Each region can even be made up of many of them.

For those who want persistent changes, but not all players need to be able to interact in-game: If a given container can support N number of players, at the N+1 player, there will be a server load balancer that spawns a new server object container from that location in the game. That N+1 player will get to experience any persistent environmental changes. This makes it much more difficult for a black hat player to crash your server and each zone in the game with a DDOS attack.

If you want persistent changes and want all players to interact: IT will need to change the bounds of the server object container to host a smaller area of the game map. Ideally, the area is so small that the zones in game feel seamless to the player experience and take up less space in game than just a couple of players at a given time.

If too many people are packed up in a small area, then users will have to trigger a load balancer that moves them to another server object container. The bottleneck to this solution is that this needs a lot of hardware (via AWS) for scalability. The game could scale up and down dynamically without servers crashing. This is called server meshing and you can learn more about it in CitizenCon 2951: Server Meshing & The State Of Persistence. The topic of server meshing in gaming is so common that it was also briefly touched upon in the Mythic Quest show.

As you can tell, a robust Kubernetes monitoring solution would be essential in security, scalability, and development for modern video games. Learn more about how to load balance for video games in this Geeks and Gurus video.

Chapter 6

Conclusion

- **Six reasons to choose Sumo Logic for Kubernetes monitoring and observability**
- **Resources you can use to learn more about k8s and get certified with the help of Sumo Logic**

Six reasons to choose Sumo Logic for Kubernetes monitoring and observability

In order to fully take advantage of the benefits of Kubernetes, you need to create a comprehensive monitoring system that can collect, aggregate and analyze all relevant data from all data sources.

Sumo Logic is uniquely positioned to provide end-to-end visibility across logs, metrics, and security posture, enabling real-time insights for Kubernetes environments. By leveraging the Kubernetes ecosystem, we can strike a balance between simplicity and flexibility.

Six reasons to choose Sumo Logic for Kubernetes monitoring and observability

Unified visibility

Sumo Logic combines metrics, logs, events, and security to create a real-time view of the performance, uptime and security of a Kubernetes platform.

Application-centric visibility

Sumo Logic lets admins monitor and troubleshoot their environments using the mental model of Kubernetes and that of their custom application rather than being forced through the lens of a server-based approach. View a Kubernetes environment through its different hierarchies: node, deployment, service and namespace.

CNCF standards-based

Sumo Logic's solution leverages the de facto standards endorsed by the CNCF. Sumo Logic's solution also utilizes the extensive ecosystem of integrations already created and maintained for monitoring Kubernetes..

Centralized metadata enrichment

Sumo Logic centralizes metadata enrichment, enabling consistent tagging across logs, metrics, events and security data. Consistent tagging enables admins to correlate critical metrics data to Kubernetes event data to log data about their application.

Dynamic out-of-the-box dashboards

Sumo Logic auto-discovers the state of Kubernetes environments and provides admins with dynamic dashboards that update automatically based on incoming data. As the Kubernetes environment changes and new services, pods and nodes are added, Sumo Logic's dashboards will adjust in real-time without additional configuration.

End-to-end security visibility

Sumo Logic provides out-of-the-box security visibility in the context of the Kubernetes mental model. Because Sumo Logic is also a security platform, Kubernetes security data can be incorporated into comprehensive security dashboards for security visibility across network, application and Kubernetes cluster.

Resources you can use to learn more about k8s and get certified with the help of Sumo Logic

Sumo Logic provides great resources for building your Kubernetes monitoring career. Kubernetes has been at the forefront of the cloud-native movement as the premier container orchestration tool. Join us for free, live, instructor-led classes that walk you through Kubernetes monitoring best practices. Classes are available for every global region that work with your individual schedule. They go through hands-on examples and end with certificates you can benefit from in your DevOps career. Check out our training page to sign up for classes.

Some notable certificates include:

1. **Observability Fundamentals:** Learn about Sumo Logic's three pillars of the observability solution including Metrics, Tracing, and Logs and become conversant with the use of the tools that will help you identify the root cause of an outage and trace the incidents to troubleshoot an issue.

2. **Kubernetes Fundamentals:** Learn how to install and use the Kubernetes app and associated CI/CD and Security apps. Maintain deployments and learn troubleshooting with the Explore tab interface. By the end, you will also learn how to create custom monitors using our dashboards.

3. **Advanced Metrics with Kubernetes Certification:** Swiftly navigate through Kubernetes cluster namespaces, services, nodes, and deployments and master monitoring and troubleshooting Kubernetes from alerts and dashboards to customized templates to address key use cases.

4. **Observability Administration:** Deploy the AWS Observability solution using Cloudformation Template, gathering metrics from a Kubernetes cluster, and establishing trace points to gather trace data using Open Telemetry.

Sumo Logic is a sponsor of The Linux Foundation, which also provides training and certifications for learning about Kubernetes. Check out the following courses from the Cloud Native Computing Foundation here; many of which are free.

Appendix: detailed Kubernetes metrics

While this paper highlighted the most essential Kubernetes metrics to monitor, there are dozens of additional metrics that you may want to collect and analyze to gain even more insight and context into the state of your cluster as a whole, as well as individual components within it.

Common Kubernetes metrics

The following metrics types apply to all or several components of Kubernetes. Some of these metrics are derived from the runtime for Golang, the language in which Kubernetes is written. Others are etcd metrics that provide insight into how cluster components interact.

Metric	Components	Description
go_gc_duration_seconds	All	A summary of the GC invocation durations.
go_threads	All	Number of OS threads created.
go_goroutines	All	Number of goroutines that currently exist.
etcd_helper_cache_hit_count	API Server, Controller Manager	Counter of etcd helper cache hits.
etcd_helper_cache_miss_count	API Server, Controller Manager	Counter of etcd helper cache miss.
etcd_request_cache_add_latencies_summary	API Server, Controller Manager	Latency in microseconds of adding an object to etcd cache.
etcd_request_cache_get_latencies_summary	API Server, Controller Manager	Latency in microseconds of getting an object from etcd cache.
etcd_request_latencies_summary	API Server, Controller Manager	Etcd request latency summary in microseconds for each operation and object type.

API server

The API Server provides the front-end for the Kubernetes cluster and is the central point that all components interact. The following table presents the top metrics you need to have clear visibility into the state of the API Server.

Metric	Description
apiserver_request_count	Count of apiserver requests broken out for each verb, API resource, client, and HTTP response contentType and code.
apiserver_request_latencies	Response latency distribution in microseconds for each verb, resource and subresource.

Etcd metrics

Etcd is the backend for Kubernetes. It is a consistent and highly-available key-value store where all Kubernetes cluster data resides. All the data representing the state of the Kubernetes cluster resides in etcd. The following are some of the top metrics to watch in etcd.

Metric	Description
etcd_server_has_leader	1 if a leader exists, 0 if not.
etcd_server_leader_changes_seen_total	Number of leader changes.
etcd_server_proposals_applied_total	Number of proposals that have been applied.
etcd_server_proposals_committed_total	Number of proposals that have been committed.

(continued)

Metric	Description
etcd_server_proposals_pending	Number of proposals that are pending.
etcd_server_proposals_failed_total	Number of proposals that have failed.
etcd_debugging_mvcc_db_total_size_in_bytes	Actual size of database usage after a history compaction.
etcd_disk_backend_commit_duration_seconds	Latency distributions of commit called by the backend.
etcd_disk_wal_fsync_duration_seconds	Latency distributions of fsync calle by wal.
etcd_network_client_grpc_received_bytes_total	Total number of bytes received by gRPC clients.
etcd_network_client_grpc_sent_bytes_total	Total number of bytes sent by gRPC clients.
grpc_server_started_total	Total number of gRPC's started on the server.
grpc_server_handled_total	Total number of gRPC's handled on the server.

Scheduler metrics

Scheduler watches the Kubernetes API for newly created pods and determines which node should run those pods. It makes this decision based on the data it has available, including the collective resource availability as well as the resource requirements of the pod. Monitoring scheduling latency ensures you have visibility into any delays the scheduler is facing.

Metric	Description
scheduler_e2e_scheduling_latency_microseconds	The end-to-end scheduling latency, which is the sum of the scheduling algorithm latency and the binding latency.

Controller manager metrics

Controller manager is a daemon that embeds all the various control loops that run to ensure that the desired state of your cluster is met. It watches the API server and takes action depending on the current state versus the desired state. It's important to keep an eye on the requests it is making to your cloud provider to ensure that the controller manager can successfully orchestrate.

Metric	Description
cloudprovider_*_api_request_duration_seconds	The latency of the cloud provider API call.
cloudprovider_*_api_request_errors	Cloud provider API request errors.

Kube-state metrics

Kube-state-metrics is a Kubernetes add-on that provides insights into the state of Kubernetes. It watches the Kubernetes API and generates various metrics, so you know what is currently running. Metrics are generated for just about every Kubernetes resource, including pods, deployments, daemonsets, and nodes. Numerous metrics are available, capturing various information. Below are some of the key metrics.

Metric	Description
kube_pod_status_phase	The current phase of the pod.
kube_pod_container_resource_limits_cpu_cores	Limit on CPU cores that can be used by the container.
kube_pod_container_resource_limits_memory_bytes	Limit on the amount of memory that can be used by the container.
kube_pod_container_resource_requests_cpu_cores	The number of requested cores by a container.
kube_pod_container_resource_requests_memory_bytes	The number of requested memory bytes by a container.
kube_pod_container_status_ready	Will be 1 if the container is ready, and 0 if it is in a not ready state.
kube_pod_container_status_restarts_total	Total number of restarts of the container.
kube_pod_container_status_terminated_reason	The reason that the container is in a terminated state.
kube_pod_container_status_waiting	The reason that the container is in a waiting state.
kube_daemonset_status_desired_number_scheduled	The number of nodes that should be running the pod.
kube_daemonset_status_number_unavailable	The number of nodes that should be running the pod, but are not able to.
kube_deployment_spec_replicas	The number of desired pod replicas for the Deployment.
kube_deployment_status_replicas_unavailable	The number of unavailable replicas per Deployment.

(continued)

Metric	Description
kube_node_spec_unschedulable	Whether a node can schedule new pods or not.
kube_node_status_capacity_cpu_cores	The total CPU resources available on the node.
kube_node_status_capacity_memory_bytes	The total memory resources available on the node
kube_node_status_capacity_pods	The number of pods the node can schedule.
kube_node_status_condition	The current status of the node.

Node metrics

Nodes are the servers on which Kubernetes workloads run. You can track standard infrastructure metrics, such as CPU and memory utilization, for each node to ensure that your cluster has sufficient resources available for your workloads to scale up if necessary. In addition, node metrics monitoring alerts you to situations where your nodes are provisioned with significantly more resources than necessary, which could lead to wasted cost.

You can gain additional insight into the health of nodes using Kubelet metrics. Kubelet is the agent that ensures that the control plane can always communicate with the node that Kubelet is running on. In addition to the common GoLang runtime metrics, Kubelet exposes some internals about its actions that are good to track.

Metric	Description
kubelet_running_container_count	The number of containers that are currently running.
kubelet_runtime_operations	The cumulative number of runtime operations available by the different operation types.
kubelet_runtime_operations_latency_microseconds	The latency of each operation by type in microseconds.

sumo logic

Learn More

Toll-Free: 1.855.LOG.SUMO | Int'l: 1.650.810.8700

sumologic.com

Contributors:

Zoe Hawkins

Megan Bouhamama

855 Main Street, Redwood City, CA 94603