The Definitive Guide to Observability in Kubernetes

OBSERVE

Introduction

If you're a seasoned IT professional — or just someone who paid attention in IT Ops 101 — you may think you know the ins and outs of observability. And you probably do, at least when you are dealing with conventional application environments, like virtual machines.

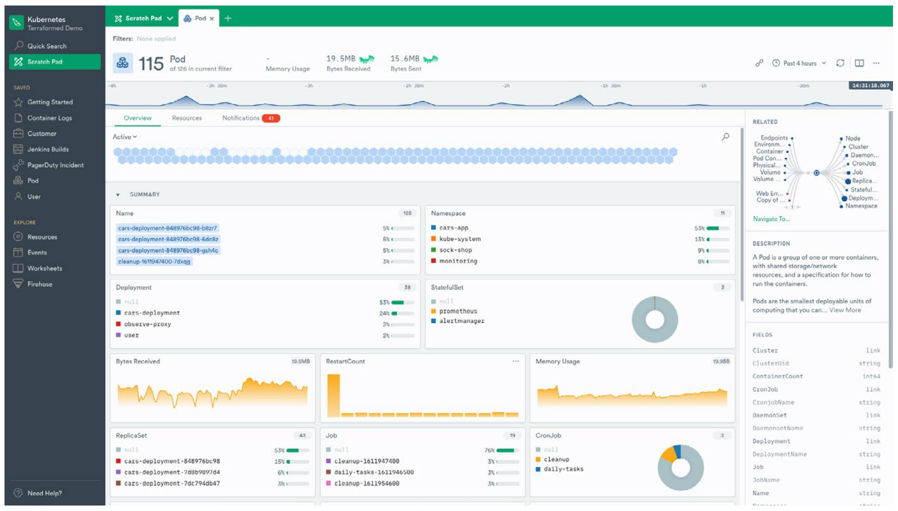
When it comes time to monitor and manage a Kubernetes cluster, though, even a seasoned IT pro can quickly start to hate life. Kubernetes is a unique beast, and achieving observability for K8s is a daunting task even for the best and brightest among us. Although the famous "pillars of observability" — logs, metrics and traces — remain relevant in Kubernetes, simply collecting and analyzing these data points is hardly enough on its own to deliver the observability you need to understand what's really happening in your clusters.

Kubernetes is a unique beast, and achieving observability for K8s is a daunting task even for the best and brightest among us.

That's because a Kubernetes cluster is a complex, multi-layered, ever-changing web of resources and services. Full observability in this context requires not simply collecting logs, metrics and traces from individual applications and services within the cluster, but also relating these various data points together in a way that helps you understand the complex events taking place deep within your clusters. In addition, teams should think beyond just logs, metrics and traces by collecting all of the data available to them, including from sources like the CI/CD pipelines that feed into their K8s clusters and the GitOps workflows that drive them.

When it comes to Kubernetes observability, in other words, correlation of each and every data source available is the real kicker. It's what separates mere monitoring from complete, actionable insight into the monster that we call K8s.

On top of all of this, simply accessing observability data in Kubernetes is a challenge to many admins. Kubernetes doesn't expose logs, traces and metrics in the way that conventional applications and operating systems do. It demands more effort on the part of the IT team to get basic observability information, let alone make sense of that information.



Everything you've always wanted to know about K8s observability

We know there's a steep learning curve to Kubernetes observability, and we want to help you overcome it. We've prepared this eBook to guide Kubernetes admins as they implement observability solutions for their clusters.

The following pages explain everything you've always wanted to know about Kubernetes observability, but were probably afraid to ask (because — let's face it — who these days wants to admit not being a K8s guru?). The eBook walks through the key challenges associated with monitoring and managing Kubernetes clusters, then offers concrete tips for overcoming them.

As we'll see, achieving true observability for Kubernetes requires not just updating your tools and processes, but also rethinking the very meaning of observability — which, in a fast-changing Kubernetes cluster, entails something quite different than it does in a static application environment.

Kubernetes observability 101

Let's begin with an overview of which types of observability data you can collect from a Kubernetes environment and where it all comes from.

Fortunately, the sources of K8s observability are simple enough. The types of data available from Kubernetes are essentially the same as those you'd focus on in any typical application environment:

- **Logs:** Logs are produced by some components of Kubernetes itself. In addition, applications running inside pods within a Kubernetes cluster usually generate log data in the same way that they would when running directly on a server.
- **Metrics:** Kubernetes exposes cluster-level metrics data about resource consumption. You can also collect this type of data from individual nodes.
- **Traces:** You can perform traces in Kubernetes just as you can in any type of environment. Traces allow you to map interactions between different services and resources in order to identify the root cause of a failure or event.

To get this data, you need to look in multiple locations. Some of it, like cluster-level metrics data, can be accessed through APIs provided by Kubernetes itself. Other data, such as application logs, must be collected from inside containers (unless the applications include logic to stream logs directly to an external location, which they probably don't unless you spent a long time refactoring them just for this purpose). The operating system logs stored on individual master and worker nodes are another important source of observability, too.

We could go into more detail here about exactly where to find which types of data within a Kubernetes cluster and how to access it manually on the CLI. But we won't, not only because we want to spare you from death by kubectl commands, but also because ***we believe that the best Kubernetes observability strategy hinges on deploying a solution that can automatically collect and correlate observability data from any and all available sources.*** Admins shouldn't have to set up log streams manually, juggle long kubectl commands or SSH into individual nodes to get observability data.

Observability solutions that automatically collect the data you need mean you can stop focusing on the tedium of Kubernetes logs and metrics collection, and turn your attention to the bigger picture: How to transform the data that Kubernetes gives you into actionable information that helps you optimize the performance of every part of your cluster.

Key observability sources in Kubernetes

Data Type	Data source or location
Cluster-level resource metrics	Metrics API
Cluster-level logs (Kube-apiserver, Kube-scheduler, etc.)	Master node(s) file system
Application logs	Containers (export logs to persistent storage to retain them)
Operating system logs (from master and worker nodes)	Node file system

The Kubernetes observability dilemma

The term observability may have become buzzworthy as of late, but the concepts behind it are hardly new. For years, developers, IT engineers and DevOps teams have been accustomed to integrating logs, metrics and traces in order to understand trends and problems within their applications.

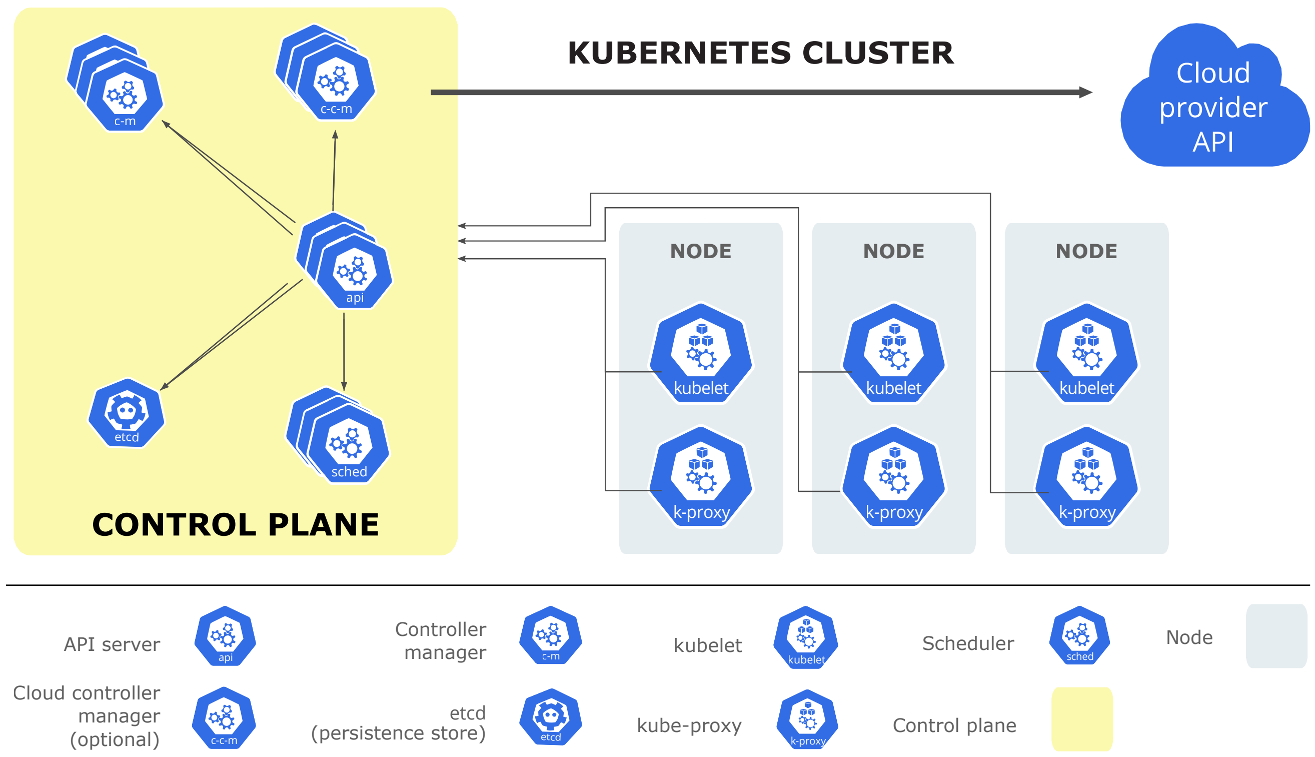
What makes life for K8s admins hard, however, is that simply lifting and shifting traditional observability paradigms into Kubernetes does not work well, for several reasons.

Multiple components

For starters, Kubernetes is not a single entity, but rather a complex collection of distinct services. It includes an API server, a controller, a key-value store and a network proxy, to name just some of the key components.

On top of this, every Kubernetes cluster consists of multiple layers of infrastructure. There are containers, which run in pods, which run on nodes. There are host operating systems running on the nodes. There may or may not be hypervisors in the mix, too, depending on whether you use bare-metal servers or virtual machines to power your nodes. Add in multiple namespaces, storage volumes and network plugins, and the multi-layered infrastructure behind each Kubernetes cluster becomes truly dizzying.

What all of this means is that there is not a single set of logs, metrics or traces to manage in Kubernetes. Each component of the cluster and infrastructure generates its own observability data, which needs to be collected separately ***and then — most importantly — correlated in a way that displays the complete context of each event or change*** to an admin who needs to understand it. Traditional observability solutions that are designed for one application running on one server don't work well in Kubernetes.



Dynamic environments: Pets vs. cattle

Not only are Kubernetes clusters highly complex, but they are also constantly changing. Container instances spin up and down in response to fluctuations in demand. Pods terminate on one node and move to another depending on factors like scheduling priorities and node availability. The mappings between storage volumes and individual containers may change depending on storage requirements. And so on.

Traditional infrastructure is not so dynamic. A virtual machine may boot up or shut down periodically, but this typically doesn't happen frequently. Virtual machines also rarely move from one host server to another. The data inside a virtual disk may change frequently, but the disk's location on the network typically won't. Virtual machines, in other words, are treated like "pets": Each one is a unique entity that is managed with care.

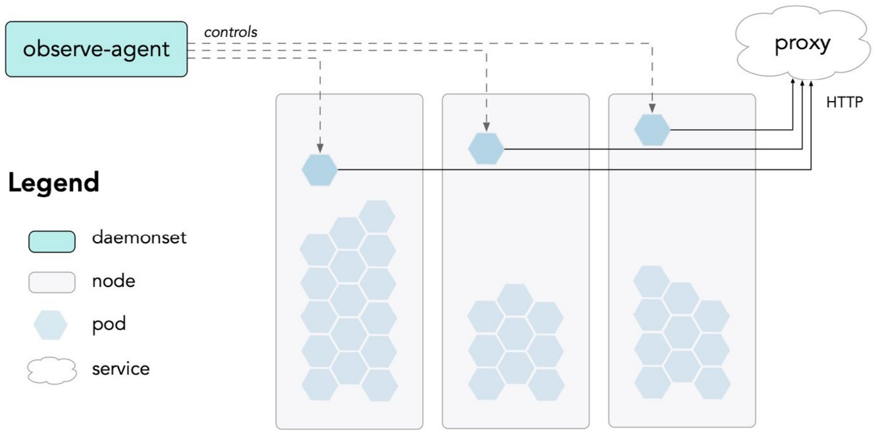
In contrast, Kubernetes treats resources like "cattle." They are immutable resources that the system constantly moves around, terminates and restarts. Because of the ever-changing nature of a Kubernetes cluster, you can never assume that logs, metrics or traces collected at one point in time represent the state of the cluster at another point in time. Nor can you assume that a log stream or metrics stream that you configure initially will continue to provide observability on an ongoing basis.

Instead, you need to manage observability continuously and in true real time — while simultaneously maintaining the ability to reference historical observability data to inform ongoing observability operations, even if the historical data no longer reflects the current state of your cluster or resources.

Rapid application deployment

The applications running inside Kubernetes containers and pods change constantly, too. If you continuously deliver new application versions — which you probably do if you use Kubernetes, because Kubernetes goes hand-in-hand with continuous delivery and DevOps — your application versions may change from one hour to the next. The resources they consume could change, too, as you roll out new features or optimizations.

This wouldn't be a challenge if Kubernetes logs and metrics clearly distinguished between different application deployments and versions. But they don't. Mapping application state to Kubernetes cluster state is an exercise that K8s leaves to the user. To pull it off efficiently, you must track data about application state, then use that data to contextualize the rest of your Kubernetes observability insights.

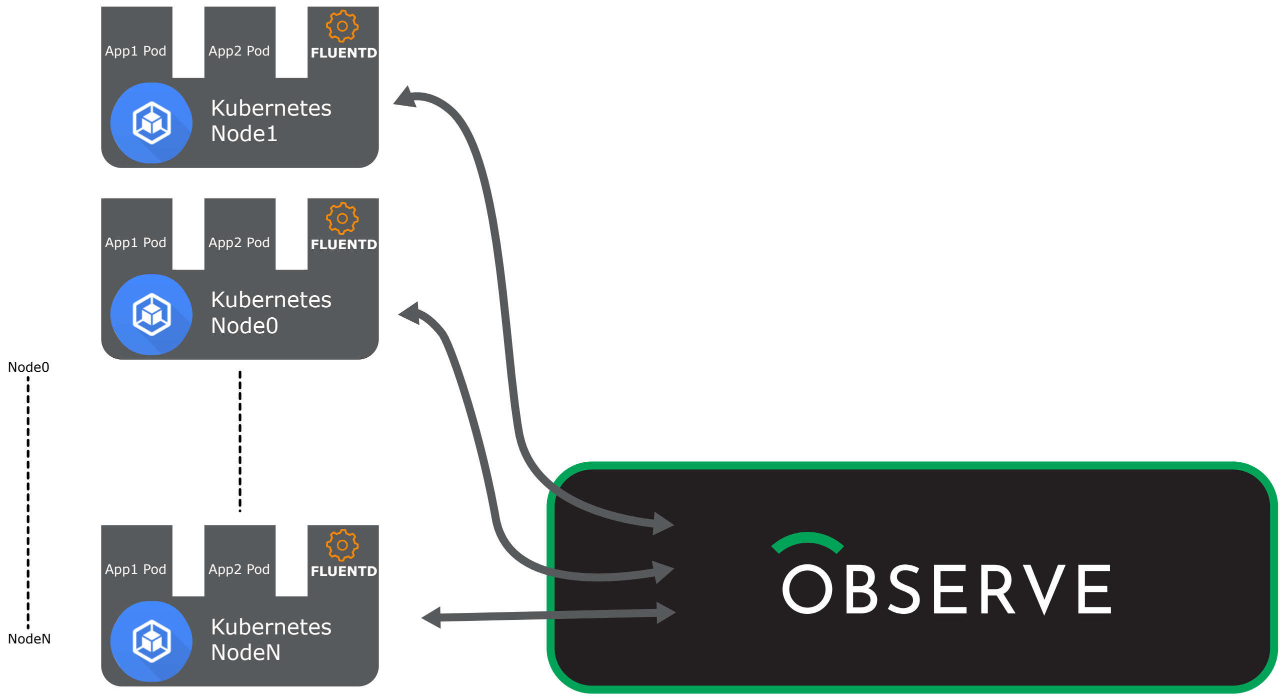


Abstract data sources

A final key observability challenge in Kubernetes (which we hinted at above) is that Kubernetes doesn't expose observability data in a straightforward way. Kubernetes itself does not generate any master log file that you can simply tail in order to keep track of the cluster. Users need to ingest log data from multiple sources in order to obtain full visibility into logs. Kubernetes does offer a metrics API, but collecting metrics from it is an exercise that Kubernetes leaves to the user, too: There is no built-in Kubernetes tooling for metrics streaming.

Likewise, at the application level, logs are not aggregated within a central location by default. Applications running inside containers and pods instead write log data to their various internal environments. You need to export the data to a centralized persistent location if you want to use it for observability purposes. And because a container could live for only a matter of seconds, you need to export the data continuously and in real time if you want full visibility. Even if you check container logs as frequently as every minute, you may miss data sources that don't persist in their original location for that long.

Kubernetes itself does not generate any master log file that you can simply tail in order to keep track of the cluster. Users need to ingest log data from multiple sources in order to obtain full visibility into logs.



The 4 sins of Kubernetes observability

How does a Kubernetes admin work through the observability roadblocks and challenges described above? We'll explain later in this eBook. But first, let's take a look at how not to address these issues. It's easy to fall into various traps — you might even call them sins — in your quest to make Kubernetes observability work. We want to be sure you avoid the common pitfalls and stick to the straight and narrow way on your journey toward K8s observability salvation.

1. Don't just aggregate logs

It can be tempting to attempt to solve Kubernetes observability challenges by collecting all of the log data you can — from your master nodes, worker nodes, containers and the underlying physical infrastructure — and then aggregating all of that data in the mistaken belief that analyzing it will give you the holistic visibility you need.

The problem with this approach is that every component in your cluster logs different types of information at different rates. As a result, if you look at aggregated log data from a specific point in time — say, the moment that a pod crashed — you are unlikely to gain the complete context you need to understand what happened. The events that caused the pod to crash may have occurred in different components at an earlier time, but you probably won't see that by looking just at aggregated log data based on a single event.

2. Don't focus on metrics alone

Collecting metrics data from the Kubernetes metrics API is another tempting way to attempt to gain across-the-board visibility into your cluster. After all, the metrics API covers the entire cluster, and it exposes critical data like CPU and memory usage.

Those are useful sources of visibility, and they should be part of any Kubernetes observability strategy. On their own, however, they are hardly enough to understand the state of your cluster. Focusing just on cluster-level metrics would be like trying to monitor a virtual machine based solely on the CPU and memory metrics of the physical server that hosts it: It would give you some clue as to what is happening inside the virtual machine, but not the level of detail necessary to gain true observability.

Instead, you need context — which depends on the correlation of data of multiple types from across your cluster — to understand what is happening.

You need context — which depends on the correlation of data of multiple types from across your cluster — to understand what is happening.

3. Don’t focus just on applications

On the opposite end of the spectrum, you might decide to ignore cluster-level metrics and focus just on the logs, traces and metrics you can get from applications running in Kubernetes. That data is straightforward to collect if you use a so-called sidecar container to stream application data to an external monitoring tool.

It’s only by contextualizing application data with cluster data, and vice versa, that you can begin to understand what is actually happening at all layers of your environment.

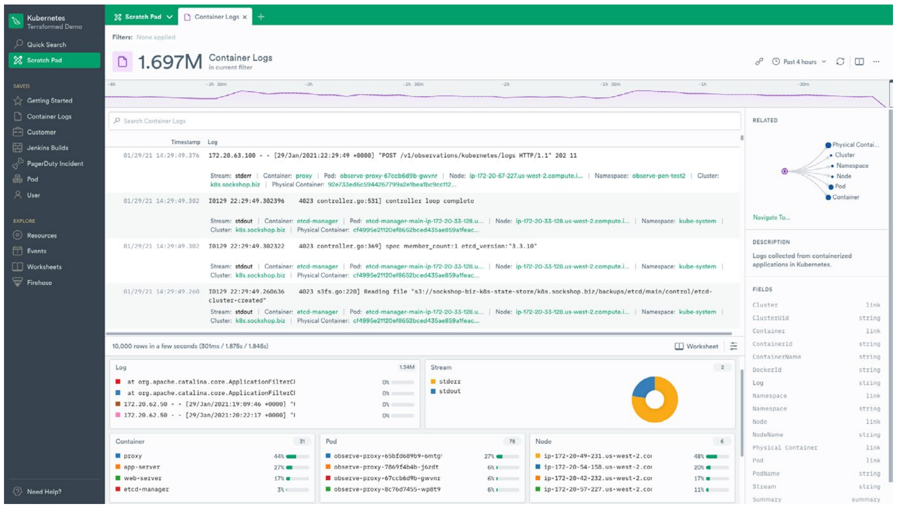
The fallacy in this approach is obvious enough: If you look only at application-level observability data, you can’t know how changes in the cluster — such as the failure of a node or the exhaustion of storage volume capacity — impacts your applications. It’s only by contextualizing application data with cluster data, and vice versa, that you can begin to understand what is actually happening at all layers of your environment.

You shouldn’t stop with those data sources, by the way. Complete observability means bringing data that is external to your cluster and applications — things like CI/CD pipeline metrics — into the picture, too.

4. Don’t rely on your managed Kubernetes service

If you run Kubernetes on a managed platform, such as Amazon Elastic Kubernetes Service or Azure Kubernetes Service, you may believe that you don’t need a sophisticated observability strategy at all because your Kubernetes service will send you alerts when something goes wrong. After all, the vendor probably promises that its managed K8s platform is pain-free, so you don’t need to worry about observability for it, right?

Not quite. The reality is that, although managed Kubernetes services typically offer basic alerting and monitoring functionality as built-in platform features, they focus mainly on notifying users about critical disruptions, not overall performance management. If you want a more nuanced level of observability, such as understanding how a new application deployment performs relative to a previous version, you’ll need to collect, correlate and analyze the necessary data yourself.



Filters: None applied

1.697M Container Logs in current filter

```
Timestamp               Log
01/29/21 14:29:49.376   172.20.63.100 - - [29/Jan/2021:22:29:49 +0000] "POST /v1/observations/kubernetes/logs HTTP/1.1" 202 11
                        Stream: stderr | Container: proxy | Pod: observe-proxy-67ccb6d9b-gwvnr | Node: ip-172-20-57-227.us-west-2.compute.i... | Namespace: observe-pen-test2 | Cluster:
                        k8s.sockshop.biz | Physical Container: 92e733ed6c5944267799a2e1bea1bc9cc112...
01/29/21 14:29:49.302   I0129 22:29:49.302396    4023 controller.go:531] controller loop complete
                        Stream: stdout | Container: etcd-manager | Pod: etcd-manager-main-ip-172-20-33-128.u... | Node: ip-172-20-33-128.us-west-2.compute.i... | Namespace: kube-system |
                        Cluster: k8s.sockshop.biz | Physical Container: cf4995e21120ef8652bced435ae859a1feac...
01/29/21 14:29:49.302   I0129 22:29:49.302322    4023 controller.go:369] spec member_count:1 etcd_version:"3.3.10"
                        Stream: stdout | Container: etcd-manager | Pod: etcd-manager-main-ip-172-20-33-128.u... | Node: ip-172-20-33-128.us-west-2.compute.i... | Namespace: kube-system |
                        Cluster: k8s.sockshop.biz | Physical Container: cf4995e21120ef8652bced435ae859a1feac...
01/29/21 14:29:49.260   I0129 22:29:49.260636    4023 s3fs.go:220] Reading file "s3://sockshop-biz-k8s-state-store/k8s.sockshop.biz/backups/etcd/main/control/etcd-
                        cluster-created"
                        Stream: stdout | Container: etcd-manager | Pod: etcd-manager-main-ip-172-20-33-128.u... | Node: ip-172-20-33-128.us-west-2.compute.i... | Namespace: kube-system |
                        Cluster: k8s.sockshop.biz | Physical Container: cf4995e21120ef8652bced435ae859a1feac...
```

10,000 rows in a few seconds (301ms / 1.875s / 1.845s)

Log	1.34M
🟥 at org.apache.catalina.core.ApplicationFilterCl	0%
🟦 at org.apache.catalina.core.ApplicationFilterCl	0%
🟫 172.20.62.50 - - [29/Jan/2021:19:09:46 +0000] "I	0%
🟪 172.20.62.50 - - [29/Jan/2021:20:22:17 +0000] "I	0%

Stream	2
🟨 stderr	
🟦 stdout	

Container	31
🟦 proxy	44%
🟧 app-server	27%
🟩 web-server	17%
🟥 etcd-manager	3%

Pod	78
🟦 observe-proxy-65bfd689b9-6mtg:	27%
🟧 observe-proxy-7869f4b4b-j6zdt	6%
🟥 observe-proxy-67ccb6d9b-gwvnr	6%
🟩 observe-proxy-8c76d7455-wp8t9	6%

Node	6
🟨 ip-172-20-49-231.us-west-2.coi	48%
🟦 ip-172-20-54-158.us-west-2.coi	20%
🟧 ip-172-20-42-232.us-west-2.coi	17%
🟩 ip-172-20-57-227.us-west-2.coi	11%

DESCRIPTION

Logs collected from containerized applications in Kubernetes.

FIELDS	
Cluster	link
ClusterUid	string
Container	link
ContainerId	string
ContainerName	string
DockerId	string
Log	string
Namespace	link
Namespace	string
Node	link
NodeName	string
Physical Container	link
Pod	link
PodName	string
Stream	string
Summary	summary

Kubernetes observability nirvana

The Kubernetes observability sins described above reflect efforts to lift-and-shift conventional observability strategies to fit Kubernetes. That doesn't work. To be truly effective, Kubernetes observability requires a different approach.

As we've already noted, logs, metrics and traces remain the foundation of Kubernetes observability, just as they are in any type of environment. But understanding the what, when and why of a Kubernetes cluster means going further than simply collecting logs, metrics and traces.

Context and data correlation

Above all, understanding what is happening in a Kubernetes cluster requires the ability to contextualize every individual event based on what is happening in the rest of the cluster at the time the event occurs — as well as what was happening leading up to the event.

For instance, if a pod is terminated on one worker node and restarted on another, you need to know what was happening concurrently on the worker nodes, the master nodes, your Kubernetes services and so on in order to gain a full picture of why the change happened and what its implications could be. If cluster CPU usage spikes, you must be able to determine the state of each container, pod and node during the spike in order to identify the source of the event.

Collecting and analyzing logs, metrics and traces from individual parts of your cluster isn't enough. You need to be able to aggregate and relate observability data from multiple sources to gain a holistic understanding of events.

To put this another way: Collecting and analyzing logs, metrics and traces from individual parts of your cluster isn't enough. You need to be able to aggregate and relate observability data from multiple sources to gain a holistic understanding of events.

Historical observability

An event that impacts one part of your Kubernetes environment at one moment in time could be caused by something that happened on a different component at an earlier point in time. Understanding the relationship between the events requires the ability to reconstruct what happened in the past and map historical developments to the current state.

For example, imagine a pod that begins consuming more resources than expected. You likely won't notice the change until the pod's resource consumption surpasses a certain threshold, even if the root cause of the behavior lies in an historical event, such as the deployment of a new application version. It's only by reconstructing the historical timeline that produced the current state of the pod that you'll know why the change in behavior occurred.

To gain the deepest level of historical observability in Kubernetes, it's helpful to track changes over time in the form of a changelog. By recording each change in state for each resource and service in your environment, you can construct a continuous changelog that allows you not only to understand historical events, but also to trace how historical changes impact the current state of your cluster.

Overall environment state

To be sure, you'll sometimes need to drill down into individual components of your Kubernetes environment — certain pods, nodes or services — to understand their behavior in isolation.

But your overall focus should be on the state of your cluster as a whole. Don't try to monitor individual components in isolation in the hope that, collectively, they'll provide across-the-board observability. Instead, focus on the state of your cluster as a whole by default, and drill down into specific components when necessary.

Achieving Kubernetes observability with Observe Inc.

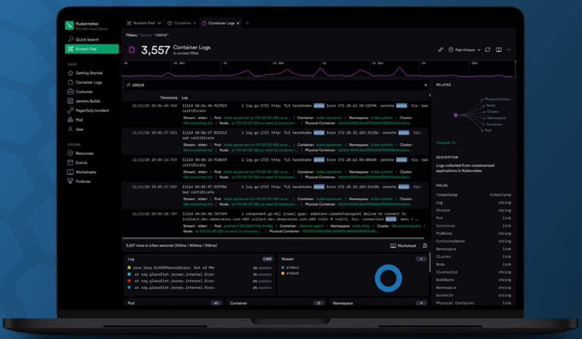
There is no shortage of observability solutions for Kubernetes. Some of them are pretty good at certain aspects of Kubernetes management. Most, however, are monitoring or logging tools that attempt to extend conventional observability features into a Kubernetes cluster — which, as you know if you've read this far, works about as well as eating soup with chopsticks.

Observe Inc. takes a different approach. Observe was designed from the start to thrive in conjunction with cloud-native technologies like Kubernetes. Rather than assuming that Kubernetes environments can be observed in the same way as static environments, or focusing only on one layer of observability (such as cluster-level metrics or application metrics), Observe takes a dynamic, holistic approach to Kubernetes observability.

Deploying Observe in Kubernetes is as simple as running a single kubectl command (after all, as we said above, we don't think having to run kubectl commands all day is good for one's health). From there, Observe does the dirty work. It automatically collects logs, metrics and traces from all layers and components of your cluster, then uses that data to track the overall state of your environment on a continuous basis. You don't need to worry about deploying multiple agents or aggregating different logs manually.

Observe gives admins the power to drill down into a single component if they want, but it also performs holistic data correlation to expose the state of the environment as a whole. You can, for example, track which pods were using which storage volumes at which times, or how container image versions changed within a given pod over time, without having to write complex queries that stitch different logs together.

And because Observe maintains a complete changelog of all events in your cluster, you can easily reconstruct the state of your environment from any point in time. There's no need to wade through historical log data manually in order to understand the past. Observe keeps track of the past for you, while at the same time providing real-time visibility into the present.



Observe isn't just for Kubernetes. It's designed for any type of environment, and it can ingest data from any type of source, ranging from traditional infrastructure to CI/CD pipelines, Git workflows, session data and even ticketing systems. But for teams that are grappling to conquer the unique challenges of Kubernetes observability, Observe delivers holistic, fully contextualized insights in a way that other platforms simply can't.

See for yourself by requesting product access:

https://www.observeinc.com

OBSERVE