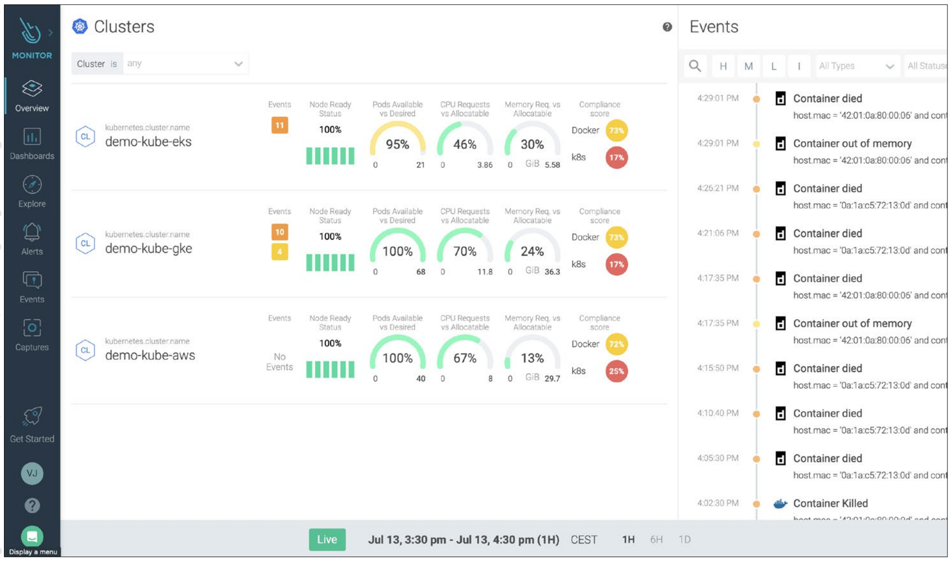
sysdig

Kubernetes Monitoring Checklist

Kubernetes has taken the container ecosystem by storm. It acts as the brain for your distributed container deployment and is designed to manage service-oriented applications using containers distributed across clusters of hosts. Kubernetes provides mechanisms for application deployment, service discovery, scheduling, updating, maintenance, and scaling.

You are adopting a DevOps approach, using Kubernetes and containers to accelerate innovation. While this dramatically simplifies deploying applications in containers — and across clouds — it also adds a new set of complexities for managing, securing and troubleshooting applications. Kubernetes and container monitoring is critical to managing application performance, service uptime and troubleshooting.



kubernetes.cluster.name	Events	Node Ready Status	Pods Available vs Desired	CPU Requests vs Allocatable	Memory Req. vs Allocatable	Compliance score
demo-kube-eks	11	100%	95% (0–21)	46% (0–3.86)	30% (0 GiB 5.58)	Docker 73%, k8s 17%
demo-kube-gke	10, 4	100%	100% (0–68)	70% (0–11.8)	24% (0 GiB 36.3)	Docker 73%, k8s 17%
demo-kube-aws	No Events	100%	100% (0–40)	67% (0–8)	13% (0 GiB 29.7)	Docker 72%, k8s 25%

Jul 13, 3:30 pm - Jul 13, 4:30 pm (1H) CEST

```
4:29:01 PM  Container died
            host.mac = '42:01:0a:80:00:06' and cont
4:29:01 PM  Container out of memory
            host.mac = '42:01:0a:80:00:06' and cont
4:26:21 PM  Container died
            host.mac = '0a:1a:c5:72:13:0d' and cont
4:21:06 PM  Container died
            host.mac = '0a:1a:c5:72:13:0d' and cont
4:17:35 PM  Container died
            host.mac = '42:01:0a:80:00:06' and cont
4:17:35 PM  Container out of memory
            host.mac = '42:01:0a:80:00:06' and cont
4:15:50 PM  Container died
            host.mac = '0a:1a:c5:72:13:0d' and cont
4:10:40 PM  Container died
            host.mac = '0a:1a:c5:72:13:0d' and cont
4:05:30 PM  Container died
            host.mac = '0a:1a:c5:72:13:0d' and cont
4:02:30 PM  Container Killed
```

What makes Kubernetes monitoring so challenging?

Static vs. dynamic:

While legacy tools may have worked for monitoring applications with static attributes, they cannot handle monitoring the dynamic nature of container-based applications. In our latest container usage report, we found that 22% of the containers live for 10 seconds or less while 54% of them live less than five minutes. This creates a high level of churn. Labels like the container id or the pod name change all the time, so we stop seeing old labels while we need to deal with new ones. Once a container dies, everything inside is gone. You cannot SSH or look at logs, and most of the tools you are accustomed to using for troubleshooting are not installed. Containers are great for operations as we can package and isolate applications to consistently deploy them everywhere, but at the same time, this makes them black boxes, which are hard to troubleshoot.

Increased Infrastructure complexity:

Dynamic provisioning via Infrastructure as a Service, automated configuration management tools, and orchestration platforms like Kubernetes enable efficiency but also add to monitoring and troubleshooting complexity. Therefore, there is a need for a new approach to gain the visibility you need to deploy and manage cloud services.

Microservices architecture:

In addition to increased infrastructure complexity, applications are being designed using microservices, where the number of components has increased by an order of magnitude. Each service can be distributed across multiple instances, and containers move across your infrastructure as needed. Monitoring the Kubernetes orchestration state is key to understanding if Kubernetes is keeping all of the service instances up and running.

Component explosion and scale requirements

When we adopt containerized architectures, the number of components and monitoring metrics grows exponentially. Traditional monitoring systems just can't keep up with this explosion. With legacy application architectures, we knew how many instances we had of each service component and where they were located. This is no longer the case in a containerized architecture. Kubernetes adds multidimensional levels like cluster, node, namespace, or service, as well as cardinality. The different aggregations, or perspectives, that need to be monitored can be staggering in volume!

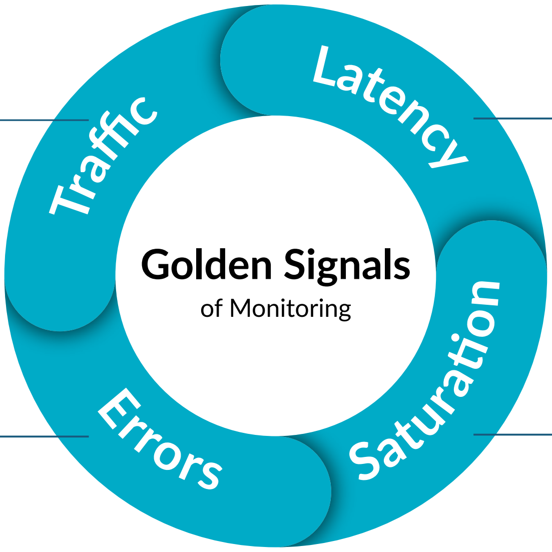
So, how does this affect Kubernetes monitoring methodology and tooling?

We have put together this checklist to address this question and to help you develop an approach for successfully monitoring Kubernetes workloads.

1

Monitoring Kubernetes With Golden Signals

Golden Signals are a standard for Kubernetes application monitoring. These signals help troubleshoot microservices applications with a reduced set of metrics that offer a wide view of a service from a user or consumer perspective.



Traffic

A measure of how much demand is being placed on your system, measured in a high-level system-specific metric.

Latency

The time it takes to service a request. It's important to distinguish between the latency of successful requests and the latency of failed requests.

Errors

The rate of requests that fail, either explicitly (e.g., HTTP 500), implicitly, or by policy.

Saturation

How "full" your service is. A measure of your system fraction, emphasizing the resources that are most constrained.

Benefits:

- Knowing the Golden Signals for Kubernetes monitoring enables you to save time by looking at what really matters and avoiding traps that could mask the real problem.
- Metrics can be aggregated at different levels for Kubernetes entities. Slicing data at the cluster, node, deployment, and pod level helps you locate issues efficiently. Troubleshooting is much faster with the ability to dig in and expose performance problems at various levels of the Kubernetes entities.

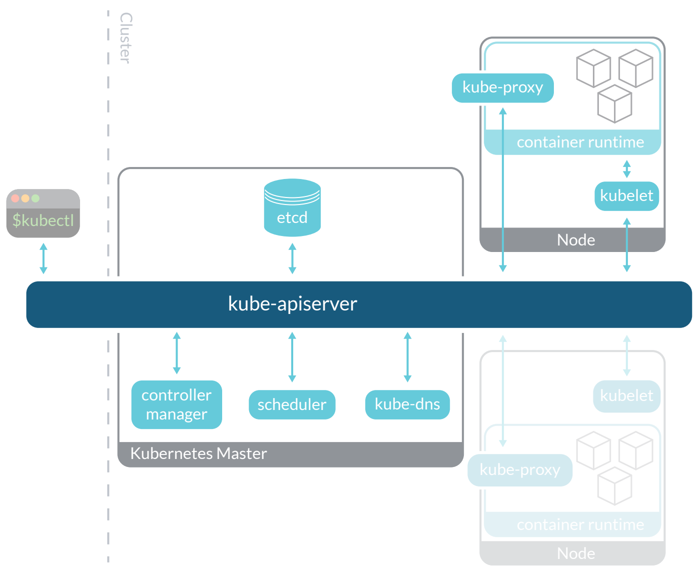
2

KUBERNETES MONITORING CHECKLIST

Monitoring Kubernetes Control Plane

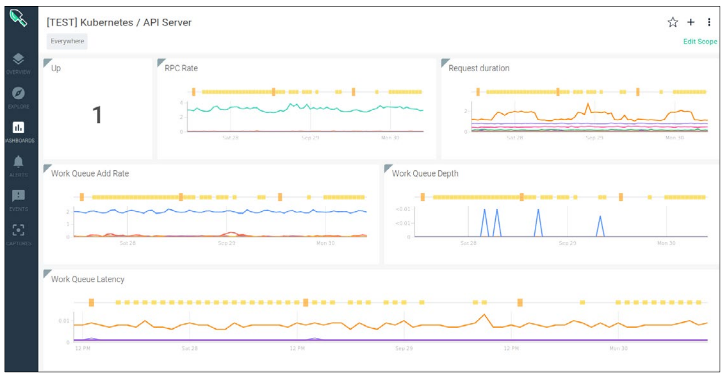
The Kubernetes control plane is the brain of your Kubernetes cluster. It manages all of your cluster resources, can schedule new pods, and can read all of the secrets stored in the cluster. The main components of the Kubernetes control plane are:

- API Server
- Kubelet
- Controller manager
- etcd
- Kube-proxy
- Kube-dns



Benefits of monitoring Kubernetes control plane:

The control plane controls your cluster; monitoring it, especially the kube-apiserver, will let you detect a latency, errors, and validate the service performance. Monitoring the Kubernetes API server provides visibility into all of the communication between the cluster components.

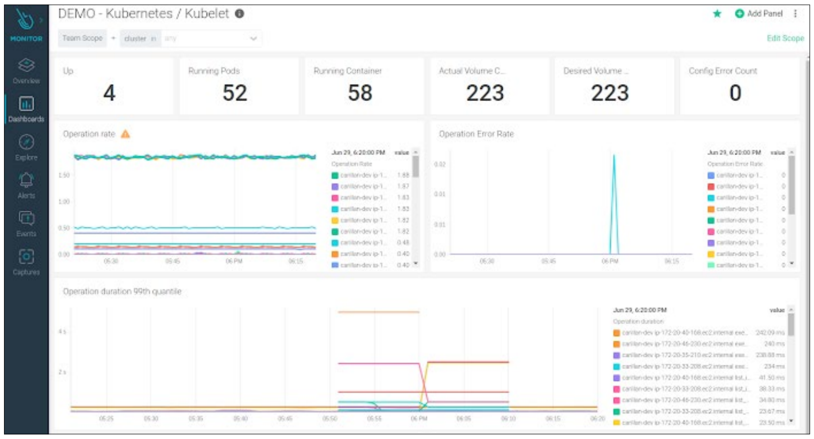


Monitoring API server use cases:

- ***You detect latency increase in the requests to the API.*** This is typically a sign of overload in the API server. Your cluster is most likely loaded and the API server may need to be scaled out.
- ***You detect an increase in the depth and latency of the work queue.*** You are having issues scheduling actions. You should check that the scheduler is working. Perhaps one node is having issues and you may want to replace it.

Kube-controller-manager is responsible for having the correct number of elements in all of the deployments, daemonsets, persistent volume claims, and many other Kubernetes elements. An issue in the kube-controller manager can compromise scalability and resilience of the applications running in the cluster. Monitoring the kube-controller manager can prevent complications that would be hard to detect otherwise.

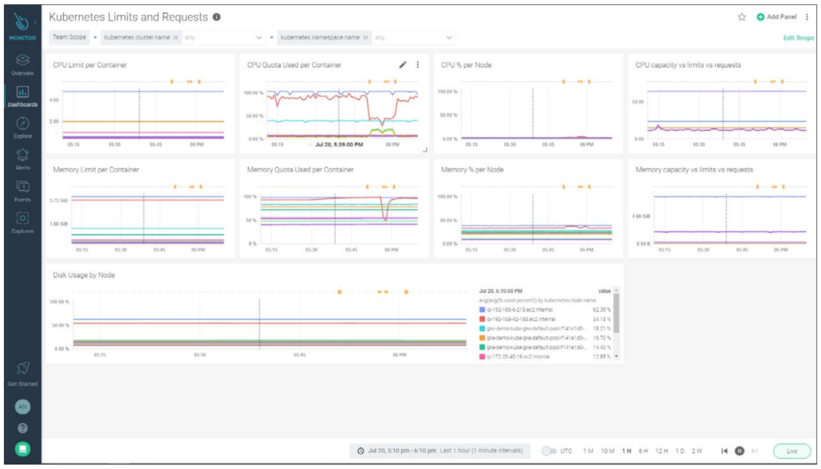
Monitoring **Kubelet** is essential as all of the communication with the container runtime is done through Kubelet. It's the connection between Kubernetes and the OS. Some issues in your Kubernetes cluster that appear to be random can be explained by API server or Kubelet problems.



Monitoring Kubelet use cases:

- **Pods are not starting.** This is typically a sign of Kubelet having problems connecting to the container runtime running below. Check the **pod start rate and duration** metrics to see if there is latency creating the containers or if they are, in fact, starting.
- **A node doesn't seem to be scheduling new pods.** Check the Kubelet job number. There's a chance that Kubelet has died in a node and is unable to schedule pods.
- **Kubernetes seems to be slow performing operations.** Check all of the golden signals in Kubelet metrics. It may have issues with storage, latency, communicating with the container runtime engine, or load issues.

While Kubernetes applications are not associated with a single node, availability of nodes translates to available cluster capacity. It's important for nodes to perform well enough to not create a performance issue, and that we have enough nodes to run our workloads. If a node fails, the workloads running there are automatically migrated to a different node. As long as there are spare resources to run everything, and if the system is well designed, there will likely be minimal interruption.


MONITOR
Overview
Dashboards
Explore
Alerts
Events
Captures
Get Started
Kubernetes Limits and Requests
Add Panel
Team Scope
kubernetes.cluster.name is any
kubernetes.namespace.name in any
Edit Scope
CPU Limit per Container
CPU Quota Used per Container
Jul 20, 5:39:00 PM
CPU % per Node
CPU capacity vs limits vs requests
Memory Limit per Container
Memory Quota Used per Container
Memory % per Node
Memory capacity vs limits vs requests
Disk Usage by Node
Jul 20, 6:10:00 PM
value
avg(avg(fs.used.percent)) by kubernetes.node.name
🟦 ip-192-168-6-218.ec2.internal 62.35 %
🟥 ip-192-168-42-183.ec2.internal 54.13 %
🟦 gke-demo-kube-gke-default-pool-f141e1d0-... 18.21 %
🟧 gke-demo-kube-gke-default-pool-f141e1d0-... 16.70 %
🟩 gke-demo-kube-gke-default-pool-f141e1d0-... 14.45 %
🟪 ip-172-20-40-16.ec2.internal 12.85 %
Jul 20, 5:10 pm - 6:10 pm Last 1 hour (1 minute intervals)
UTC
1 M
10 M
1 H
6 H
12 H
1 D
2 W
Live

Monitoring Kubernetes Cluster and nodes use cases:

- **Host is down.** If a host is down or unreachable, you might want to receive a notification with a suitable wait time to avoid noisy alerts.
- **Do I have enough Kubernetes nodes in the cluster?** A node failure isn't a problem in Kubernetes since the scheduler will spawn the containers from the pods in the failed node into other available nodes. But what if you are running out of nodes, or the resource requirements for the deployed applications overbook existing nodes? And what if you are hitting a quota limit?

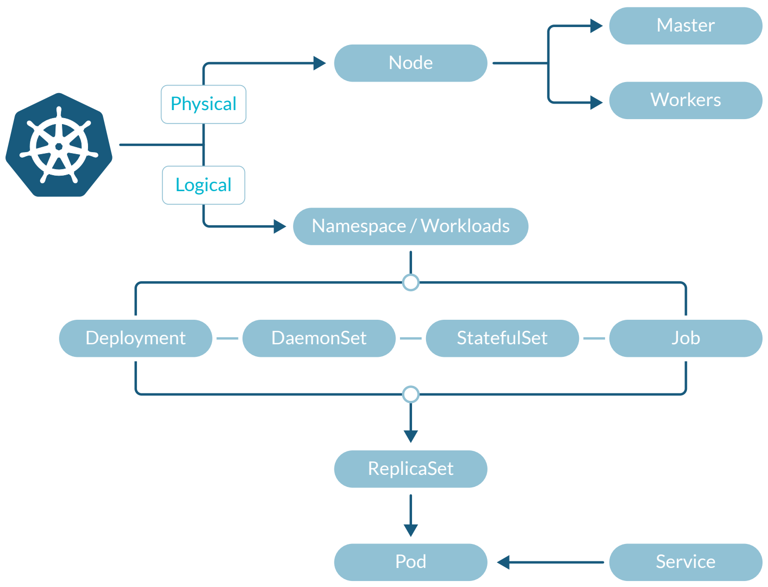
3

Monitoring Kubernetes Workloads

Applications in Kubernetes are structured into several hierarchical layers that define how the services are organized outside of the Kubernetes world.

Importance of Hierarchy in Monitoring Kubernetes

In a Kubernetes environment, a pod is the fundamental unit. Pods and other Kubernetes elements are strongly coupled and it's important to keep that correlation when interpreting the data. A pod can die and respawn, therefore the state of an individual pod is not as important as aggregate information of all the pods in a deployment. Essential parameters like availability of the service, possible points of failure, and availability of resources are important for monitoring Kubernetes health.



Monitoring Kubernetes application use cases:

- **Do I have enough pods/containers running for each application?** There are multiple reasons why the number of running containers can change. That includes rescheduling containers in a different host because a node failed, or because there aren't enough resources and the pod was evicted from a rolling deployment of a new version, and more.
- **Do I have any pod/containers for a given application?** Get an alert if there aren't any containers running for a given application.
- **Is there any pod/container in a restart loop?** When deploying a new version that's broken, if there aren't enough resources available or some requirements/ dependencies aren't in place, you might end up with a container or pod continuously restarting in a loop. That's called CrashLoopBackOff. When this happens, pods never get into ready status and, therefore, are counted as unavailable and not as running.

Understanding Kubernetes limits and requests

Applications were typically designed to run stand-alone in a machine and use all of the resources at hand. The new Kubernetes landscape requires sharing the same space and resources with others, and that makes setting limits and quotas a hard requirement.

Namespace quotas

Kubernetes allows administrators to set quotas, in namespaces, as hard limits for resource usage. This has an additional effect; if you set a CPU request quota in a namespace, then all pods need to set a CPU request in their definition, otherwise they will not be scheduled. The user is responsible for enforcement of these limits since there is no automatic mechanism to tell Kubernetes how much overcommit to allow.

Therefore it is important to:

- Set requests and limits in your workloads.
- Setting a namespace quota will enforce all of the workloads in the namespace to have a request and limit in every container.

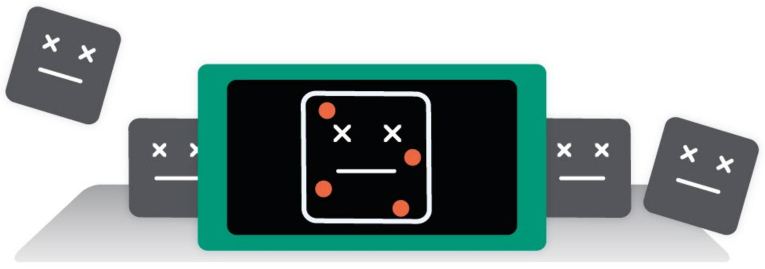
Kubernetes OOM problems

When a node is low on memory, Kubernetes eviction policy steps in and fails/stops pods. If they are managed by a ReplicaSet, these pods are scheduled in a different node. It's important to monitor closely to resolve issues like:

- OOM kill due to container limit reached
- Kubernetes OOM kill due to limit overcommit
- CPU throttling due to CPU limit

In summary, these are some important considerations for monitoring Kubernetes workloads:

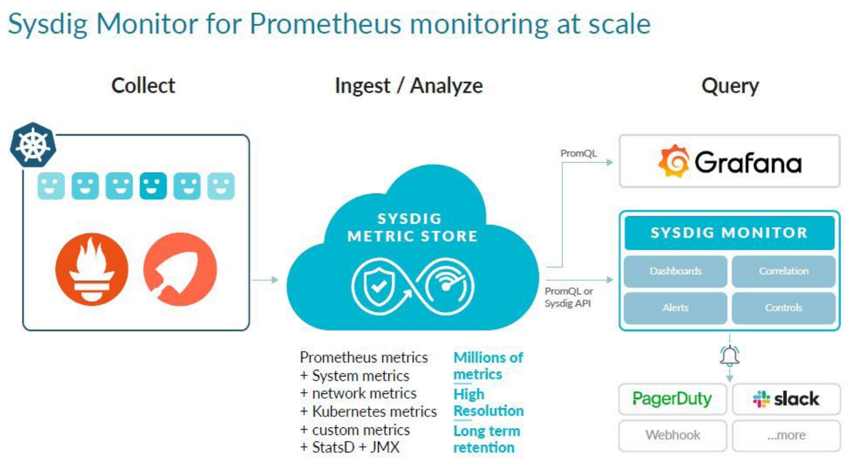
1. Monitor resource usage in your workloads - this will allow you to discover different issues that can affect the health of the applications running in the cluster.
2. Understanding that your resource usage can compromise your application and affect other applications in the cluster is the crucial first step. You have to properly configure your quotas.
3. Monitoring the resources and how they are related to the limits and requests will help you set reasonable values and avoid Kubernetes OOM kills. This will result in a better performance of all the applications in the cluster, as well as a fair sharing of resources.
4. Your Kubernetes alerting strategy can't just focus on the infrastructure layer. It needs to understand the entire stack, from the hosts and Kubernetes nodes at the bottom, up to the top where the application workloads run.
5. Being able to leverage Kubernetes and cloud providers metadata to aggregate and segment metrics and alerts will be a requirement for effective alerting across all layers.



4

Monitoring Kubernetes with Prometheus

Prometheus is the de facto approach for monitoring Kubernetes. While it's really easy to start monitoring Kubernetes with Prometheus, DevOps teams quickly realize that Prometheus has roadblocks like scale, RBAC, and teams support for compliance.



Sysdig Monitor and Sysdig backend are able to store and query Prometheus native metrics and labels. Additionally, users can use Sysdig in the same way that they use Prometheus. You can leverage Prometheus Query Language queries for dashboards and alerts, or the Prometheus API for scripted queries, as part of a DevOps workflow. This way, your investments in the Prometheus ecosystem can be preserved and extended while improving scale and security. Users can take advantage of the troubleshooting, correlation and support that we provide, as part of Sysdig Monitor, to go beyond a basic Prometheus monitoring solution.

For specific guidance on how to start monitoring your Kubernetes cluster, and gain deep visibility and trouble-shooting efficiency, download our Kubernetes Monitoring guide. Additionally, start your free trial of Sysdig Monitor to start delivering on your cloud service availability and performance goals.