DATADOG

Optimal Kubernetes Performance: 15 Metrics Every DevOps Team Should Track

Table of contents

1

Introduction

Optimizing Kubernetes performance

Kubernetes is a complex system, so there are hundreds of potential metrics your DevOps team could measure. This guide highlights 15 of the most essential, splitting them into three broad categories: cluster state metrics, resource metrics, and control plane metrics.

If your team stays on top of these crucial metrics, you'll be well on your way to ensuring optimal Kubernetes performance.

2

Where do Kubernetes metrics come from?

Before we dive in, a quick word on sourcing. The Kubernetes ecosystem includes two add-ons for monitoring data from your clusters: **Metrics Server** and **kube-state-metrics**.

- Metrics Server exposes statistics about the resource usage of Kubernetes objects, reporting the demands of individual nodes, pods, and clusters on memory, disk, CPU, and so on.
- kube-state-metrics reports on the broader state of Kubernetes objects, making cluster state information easier to consume with metrics on node status, node capacity, pod status, etc.

Metrics Server and kube-state-metrics can be accessed via the command line or Kubernetes Dashboard, a web-based UI that presents the last few minutes of data from your containerized environment.

While the command line and Kubernetes Dashboard are useful for retrieving snapshots of metrics, effective monitoring requires a more robust solution. At the very least, DevOps teams need longer data retention, lookback times, and greater visibility into Kubernetes infrastructure (like the control plane).

Datadog Kubernetes Monitoring provides total visibility into the health and performance of your containerized environment. From cluster status to low-level resource metrics to distributed traces and container logs, Datadog brings together all the data from your infrastructure, applications, and services in one platform, providing:

- Out-of-the-box integrations with Kubernetes, Docker, containerd, and all your containerized applications, so you can see all your metrics, logs, and traces in one place
- Autodiscovery so you can seamlessly monitor applications in large-scale dynamic environments
- Advanced monitoring features including outlier and anomaly detection, forecasting, and automatic correlation of observability data

Whether you're a mature Kubernetes shop or just beginning your move to containers, you can visualize the health and performance of all your clusters, regardless of what platform they're running on. To ensure your DevOps team has all the Kubernetes data it needs, set up Datadog and you'll have visibility into every part of your containerized environment within minutes.

3

15 key metrics for optimal Kubernetes performance

With a wealth of potential data at your fingertips, what are the most important Kubernetes performance metrics to monitor?

Kubernetes performance metrics can be split into three broad categories: cluster state metrics, resource metrics, and control plane metrics. Let's look at the most important metrics from each category.

Cluster state metrics

Cluster state metrics provide a high-level view of the state of your cluster, including the count, health, and availability of various Kubernetes objects. They can quickly surface issues with nodes or pods, helping DevOps teams ensure application reliability, support troubleshooting, and enable efficient scaling.

While Datadog collects and provides insight into all possible cluster state metrics, here are three of the most important for your DevOps team to actively monitor.

1. Node status

NAME IN KUBE-STATE-METRICS	`kube_node_status_condition`
DATADOG METRIC NAME	`kubernetes.kubelet.check`
DESCRIPTION	`Current health status of a node (kubelet)`

The node status metric provides information about the health and availability of nodes within the cluster, indicating whether the scheduler can place pods on the nodes. It runs checks on the following node conditions, with each check returning `true`, `false`, or unknown:

- `OutOfDisk`
- `Ready` (node is ready to accept pods)
- `MemoryPressure` (node memory is too low)
- `PIDPressure` (too many running processes)
- `DiskPressure` (remaining disk capacity is too low)
- `NetworkUnavailable`

Monitoring this metric is crucial for ensuring that nodes are functioning properly and capable of running workloads efficiently. In particular, the `Ready` and `NetworkUnavailable` checks warn you about nodes that aren't usable, so you should set up an alert here to begin troubleshooting problems as soon as they arise.

2. Desired vs. current pods

NAME IN KUBE-STATE-METRICS	`kube_deployment_spec_replicas` vs. `kube_deployment_status_replicas`
	Or, if using DaemonSets:
	`kube_daemonset_status_desired_number_scheduled` vs. `kube_daemonset_status_current_number_scheduled`

DATADOG METRIC NAME	`Kubernetes_state.deployment.replicas_desired` vs. `kubernetes_state.deployment.replicas`
	Or, if using DaemonSets:
	`kubernetes_state.daemonset.desired` vs. `kubernetes_state.daemonset.scheduled`

DESCRIPTION	Number of pods specified for a Deployment or DaemonSet vs. number of pods currently running in a Deployment or DaemonSet

Kubernetes provides metrics that reflect the number of *desired* pods and the number of *currently running pods*. Typically, these numbers should match, and so it's advisable to set up an alert to flag any differences between them.

A large disparity suggests a problem with your configuration, or bottlenecks with nodes lacking the resource capacity to schedule new pods. In either case, inspecting pod logs can help you find the root cause.

3. Available and unavailable pods

NAME IN KUBE-STATE-METRICS	`kube_deployment_status_replicas_available` vs. `kube_deployment_status_replicas_unavailable`
	Or, if using DaemonSets:
	`kube_daemonset_status_number_available` vs. `kube_daemonset_status_number_unavailable`

DATADOG METRIC NAME	`kubernetes_state.deployment.replicas_available` vs. `kubernetes_state.deployment.replicas_unavailable`
	Or, if using DaemonSets:
	`kubernetes_state.daemonset.ready` and `kubernetes_state.daemonset.desired - kubernetes_state.daemonset.ready`

DESCRIPTION	Number of pods currently available / not available for a Deployment or DaemonSet

If you see spikes in the number of unavailable pods, this is likely to impact application performance and uptime, and may indicate crashes, scheduling problems, or resource shortages — so it's important you monitor this metric closely.

A large number of consistently unavailable pods likely means there's a problem with their configuration, such as with their readiness probes. Checking a pod's logs can provide you with clues as to why it's stuck in a state of unavailability.

Resource metrics

While cluster state metrics report on the state of your Kubernetes objects, resource metrics provide insight into their resource usage. Monitoring memory, CPU, and disk usage across the different layers of your cluster can help you detect and troubleshoot application and system level problems. By understanding how efficiently your environment uses resources, you can inform capacity planning and prevent pod crashes, bottlenecks, and downtime.

Here are seven key resource metrics for your DevOps team to monitor:

4. Memory limits per pod vs. memory utilization per pod

NAME IN KUBE-STATE-METRICS	`kube_pod_container_resource_limits_memory_bytes` vs. N/A
DATADOG METRIC NAME	`kubernetes.memory.limits` vs. `kubernetes.memory.usage`
DESCRIPTION	Total memory limits (bytes) of a pod vs. total memory in use on a node or pod

For each container running on a pod, you can declare a request and a limit for memory usage. It's essential to actively compare the memory limits you've configured to your pods' actual memory usage. If a pod uses more memory than its defined limit, it will be `OOMKilled`, leading to downtime as the container restarts and, if memory accumulates fast enough, perhaps even triggering a CrashLoop state.

At the other end of the scale, setting a pod's limit too high can lead to poor scheduling decisions. For example, a pod with a memory request of 1GiB and a limit of 4GiB can be scheduled on a node with 2GiB of allocatable memory — more than sufficient to meet its request. But if the pod suddenly needs 3GiB of memory, it will be killed even though it's well below its memory limit.

It's thus imperative you monitor memory limits vs. utilization to help get the balance right in your configurations and prevent `OOMKills`.

5. Memory utilization

NAME IN KUBE-STATE-METRICS	N/A
DATADOG METRIC NAME	`kubernetes.memory.usage`
DESCRIPTION	Total memory in use on a node or pod

As well as keeping an eye on how actual memory usage compares to your configured limits, it's important to monitor memory usage generally at the pod and node level. While pods that exceed their memory limits will be `OOMKilled`, those that use significantly more memory than the request you've set are also at risk of termination — even if they stay within their assigned limits.

If a node runs low on available memory, for instance, the kubelet flags it as under memory pressure and begins to reclaim resources, prioritizing the eviction of pods that most exceed their defined memory requests.

Keeping on top of how requests compare to *actual memory usage* is thus crucial for minimizing unintended pod evictions, ensuring the requests and limits specified in your manifests make sense, and informing how and when you need to scale your cluster.

6. Memory requests per node vs. allocatable memory per node

NAME IN KUBE-STATE-METRICS	`kube_pod_container_resource_requests_memory_bytes` vs. `kube_node_status_allocatable_memory_bytes`
DATADOG METRIC NAME	`Kubernetes.memory.requests` vs. `kubernetes_state.node.memory_allocatable`
DESCRIPTION	Total memory requests (bytes) vs. total allocatable memory (bytes) of the node

Allocatable memory reflects the amount of memory on a node that is available for pods. Specifically, it takes the overall capacity and subtracts memory requirements for OS and Kubernetes system processes to ensure they will not fight user pods for resources.

Although memory capacity is a static value, monitoring pod memory requests against a node's allocatable memory is important for capacity planning. The resulting metric informs you as to whether node memory is sufficient to meet the needs of current pods, and whether the control plane is able to schedule new ones. If fewer pods are running than desired, for instance, reviewing this metric can be a good first step in your troubleshooting efforts.

7. Disk utilization

NAME IN KUBE-STATE-METRICS	N/A
DATADOG METRIC NAME	`kubernetes.filesystem.usage`
DESCRIPTION	The amount of disk used

Disk space, like memory, is a non-compressible resource in Kubernetes. If a node's root volume is low on disk space, it can trigger scheduling issues. When disk space drops below a threshold, the node is flagged as being under disk pressure, prompting the kubelet to run garbage collection, removing unused images or dead containers. If space remains limited, the kubelet will begin evicting pods — so it's critical you actively monitor disk utilization.

Beyond node-level disk usage, tracking the volume-level disk usage of your pods is important, too. If a volume runs out of space, applications may encounter errors when writing data. It's thus prudent to set alerts for when a volume reaches, say, 80% usage so that you can prevent issues by adding more storage or adjusting requests.

8. CPU requests per node vs. allocatable CPU per node

NAME IN KUBE-STATE-METRICS	`kube_pod_container_resource_requests_cpu_cores` vs. `kube_node_status_allocatable_cpu_cores`
DATADOG METRIC NAME	`kubernetes.cpu.requests` vs. `kubernetes_state.node.cpu_allocatable`
DESCRIPTION	Total CPU requests (cores) of a pod vs. total allocatable CPU (cores) of the node

As with memory, tracking overall CPU requests per node and comparing them to each node's allocatable CPU capacity is invaluable for capacity planning, helping you understand whether your cluster can support more pods.

9. CPU limits per pod vs. CPU utilization per pod

NAME IN KUBE-STATE-METRICS	`kube_pod_container_resource_limits_cpu_cores` vs. N/A
DATADOG METRIC NAME	`kubernetes.cpu.limits` vs. `kubernetes.cpu.usage.total`
DESCRIPTION	The limit of CPU cores set vs. total CPU cores in use

Unlike memory, CPU is compressible, meaning if a pod exceeds the CPU limit you set, the node will throttle its CPU usage without terminating the pod. However, throttling can still cause performance issues. By keeping an eye on these metrics, you can ensure your CPU limits are properly configured to meet the pods' actual needs, and reduce the risk of slowdowns.

10. CPU utilization

NAME IN KUBE-STATE-METRICS	`kube_pod_container_resource_limits_cpu_cores` vs. N/A
DATADOG METRIC NAME	`kubernetes.cpu.limits` vs. `kubernetes.cpu.usage.total`
DESCRIPTION	The limit of CPU cores set vs. total CPU cores in use

As with memory, it's important to monitor CPU utilization generally at both the pod and node level. Nurturing an understanding of CPU usage will help you reduce throttling and ensure optimal cluster performance. Consistently higher than expected CPU usage, meanwhile, might point to problems with the pod for you to identify and address.

Control plane metrics

While cluster state and resource metrics report the general status, performance, and efficiency of your Kubernetes objects, control plane metrics center around cluster management, revealing the health of your cluster's central nervous system.

Are your API server, controller managers, schedulers, and etcd working optimally? The following five control plane metrics will help inform your answer.

Note: *in managed Kubernetes environments, the control plane is managed by the cloud provider, and you may not have access to all the components and metrics listed below. The availability or names of certain metrics may also be different depending on which version of Kubernetes you are using.*

11. Whether the etcd cluster has a leader

NAME IN KUBE-STATE-METRICS	`etcd_server_has_leader`
DESCRIPTION	Indicates whether the member of the cluster has a leader (1 if a leader exists, 0 if not)

For a cluster to function correctly, etcd requires a stable leader. If a majority of nodes don't recognize a leader (i.e., they have the metric value 0), the etcd cluster may become unavailable, which would affect the entire Kubernetes cluster's ability to schedule workloads, store configurations, and maintain its desired state.

You should thus set up an alert so that if `etcd_server_has_leader` is 0 for any etcd node, you can investigate the state of the etcd cluster immediately. This will help identify any problems related to cluster partitioning, slow leader elections, or network issues that prevent nodes from seeing the leader.

12. Number of leader transitions within a cluster

NAME IN KUBE-STATE-METRICS	`etcd_server_leader_changes_seen_total`
DESCRIPTION	Counter of leader changes seen by the cluster member

The `etcd_server_leader_changes_seen_total` metric tracks the number of leader transitions within the cluster. Sudden or frequent leader changes, though not necessarily damaging on their own, can alert you to issues with connectivity or resource limitations in the etcd cluster, so it's worth keeping an eye on the count. If you notice a gradual increase, it might indicate resource exhaustion, prompting you to scale or optimize your etcd infrastructure.

13. Number and duration of requests to the API server for each resource

NAME IN KUBE-STATE-METRICS	`apiserver_request_latencies_count` and `apiserver_request_latencies_sum`
DESCRIPTION	Count of requests to the API server for a specific resource and verb, and the sum of request duration to the API server for a specific resource and verb (in microseconds)

By monitoring the number and latency of specific kinds of requests to the API server, you can see if the cluster is falling behind in executing any user-initiated commands to create, delete, or query resources. You can also track these metrics over time and divide their deltas to compute a real-time average, which will give you a benchmark to troubleshoot from, allowing you to quickly identify if the API server is being overwhelmed with requests.

14. Controller manager latency metrics

NAME IN KUBE-STATE-METRICS	`workqueue_queue_duration_seconds` and `workqueue_work_duration_seconds`
DESCRIPTION	Total number of seconds that items spent waiting in a specific work queue, and the total number of seconds spent processing items in a specific work queue

These latency metrics provide important insight into the performance of the controller manager, which queues up each actionable item (such as the replication of a pod) before it's carried out — so your DevOps team should monitor them closely. If you see a latency increase in the automated actions of your controllers, you can look at the logs for the controller manager to gather more details about the cause.

15. Number and latency of the Kubernetes scheduler's attempts to schedule pods on nodes

NAME IN KUBE-STATE-METRICS	`scheduler_schedule_attempts_total` and `scheduler_e2e_scheduling_duration_seconds`
DESCRIPTION	Count of attempts to schedule a pod, broken out by result, and the total elapsed latency in scheduling workload pods on worker nodes

Finally: it's important to track the performance of the Kubernetes scheduler. By monitoring its attempts to schedule pods on nodes, as well as the end-to-end latency of carrying out those attempts, you can identify any problems with matching pods to worker nodes.

An increase in unschedulable pods indicates that your cluster may lack the resources needed to launch new pods, whereas an attempt that results in an error status reflects an internal issue with the scheduler itself. If you notice a discrepancy between the number of desired and current pods, you can dig into these latency metrics to see if a scheduling issue is behind the lag.

A word on Kubernetes events

Collecting events from Kubernetes and from the container engine (such as Docker) allows you to see how pod creation, destruction, starting, or stopping affects the performance of your infrastructure — and vice versa. While Docker events trace container lifecycles, Kubernetes events report on pod lifecycles and deployments. Tracking Kubernetes Pending pods and pod failures, for example, can point you to misconfigured launch manifests or issues of resource saturation on your nodes.

So, for easier troubleshooting and investigation, you should aim to correlate Kubernetes events with the metrics outlined in this guide.

4

Beyond the key metrics: all your Kubernetes data in one place

By staying on top of the fifteen metrics we've looked at, you'll be well on your way to optimizing the performance of your containerized environment. But Datadog provides even more Kubernetes monitoring functionality beyond the scope of this guide — including logs, distributed traces, process monitoring, network performance monitoring, resource optimization capabilities, and the collection of custom metrics in Kubernetes.

If you're interested in securing cutting-edge Kubernetes observability for your organization, consider setting up Datadog today. You'll enjoy unprecedented visibility into every part of your containerized environment within minutes.

Finally, for further tips and assistance, be sure to check out our Kubernetes metrics cheat-sheet, our detailed Kubernetes documentation, and request a demo to see our Kubernetes monitoring in action.

About Datadog

Datadog is the observability and security platform for cloud applications. Our SaaS platform enables real-time, end-to-end visibility across infrastructure, applications, logs, front-end performance metrics, and more. Datadog is used by organizations of all sizes and across a wide range of industries to modernize and automate cloud operations, improve collaboration across teams, deliver software faster and more securely, track critical business metrics, and ensure seamless digital experiences.

For more information, visit datadoghq.com