Workloads

Understand Pods, the smallest deployable compute object in Kubernetes, and the higher-level abstractions that help you to run them.

A workload is an application running on Kubernetes. Whether your workload is a single component or several that work together, on Kubernetes you run it inside a set of *pods*. In Kubernetes, a Pod represents a set of running containers on your cluster.

Kubernetes pods have a defined lifecycle. For example, once a pod is running in your cluster then a critical fault on the node where that pod is running means that all the pods on that node fail. Kubernetes treats that level of failure as final: you would need to create a new Pod to recover, even if the node later becomes healthy.

However, to make life considerably easier, you don't need to manage each Pod directly. Instead, you can use *workload resources* that manage a set of pods on your behalf. These resources configure controllers that make sure the right number of the right kind of pod are running, to match the state you specified.

Kubernetes provides several built-in workload resources:

- Deployment and ReplicaSet (replacing the legacy resource ReplicationController). Deployment is a good fit for managing a stateless application workload on your cluster, where any Pod in the Deployment is interchangeable and can be replaced if needed.
- StatefulSet lets you run one or more related Pods that do track state somehow. For example, if your workload records data persistently, you can run a StatefulSet that matches each Pod with a PersistentVolume. Your code, running in the Pods for that StatefulSet, can replicate data to other Pods in the same StatefulSet to improve overall resilience.
- DaemonSet defines Pods that provide facilities that are local to nodes. Every time you add a node to your cluster that matches the specification in a DaemonSet, the control plane schedules a Pod for that DaemonSet onto the new node. Each pod in a DaemonSet performs a job similar to a system daemon on a classic Unix / POSIX server. A DaemonSet might be fundamental to the operation of your cluster, such as a plugin to run cluster networking, it might help you to manage the node, or it could provide optional behavior that enhances the container platform you are running.
- Job and CronJob provide different ways to define tasks that run to completion and then stop. You can use a Job to define a task that runs to completion, just once. You can use a CronJob to run the same Job multiple times according a schedule.

In the wider Kubernetes ecosystem, you can find third-party workload resources that provide additional behaviors. Using a custom resource definition, you can add in a third-party workload resource if you want a specific behavior that's not part of Kubernetes' core. For example, if you wanted to run a group of Pods for your application but stop work unless *all* the Pods are available (perhaps for some high-throughput distributed task), then you can implement or install an extension that does provide that feature.

Workload placement

FEATURE STATE: Kubernetes v1.35 [alpha](disabled by default)

While standard workload resources (like Deployments and Jobs) manage the lifecycle of Pods, you may have complex scheduling requirements where groups of Pods must be treated as a single unit.

The Workload API allows you to define a group of Pods and apply advanced scheduling policies to them, such as gang scheduling. This is particularly useful for batch processing and machine learning workloads where "all-or-nothing" placement is required.

What's next

As well as reading about each API kind for workload management, you can read how to do specific tasks:

- Run a stateless application using a Deployment
- Run a stateful application either as a single instance or as a replicated set
- Run automated tasks with a CronJob

To learn about Kubernetes' mechanisms for separating code from configuration, visit Configuration.

There are two supporting concepts that provide backgrounds about how Kubernetes manages pods for applications:

- Garbage collection tidies up objects from your cluster after their *owning resource* has been removed.
- The *time-to-live after finished* controller removes Jobs once a defined time has passed since they completed.

Once your application is running, you might want to make it available on the internet as a Service or, for web application only, using an Ingress.

Feedback

Was this page helpful?

Yes No

Last modified November 18, 2025 at 11:47 AM PST: KEP-4671 Add docs for Workload API and Gang scheduling (fda060d1fe)