Manage Long-lived and Short-lived Applications by Using the Kubernetes Workload API

Objectives

- Deploy containerized applications as pods that Kubernetes workload resources manage.

Kubernetes Workload Resources

The Kubernetes Workloads API consists of resources that represent various types of cluster tasks, jobs, and workloads. The Workloads API is composed of various Kubernetes APIs and extensions that simplify the deployment and management of applications. The following are Workload API resource type that are most often used:

- Jobs
- CronJobs
- Deployments
- DaemonSets
- StatefulSets

Jobs

A *job* resource represents a one-time task to perform on the cluster. As with most cluster tasks, the jobs are executed via pods. If a job's pod fails, then the cluster retries a number of times that the job specifies. The job does not run again after a specified number of successful completions.

Jobs differ from using the `oc run` command which creates only a pod. The pod is not managed by a higher-level controller. If the node where the pod is running fails, the pod is not rescheduled. The pod is not automatically restarted if its process exits with an error code.

Common uses for jobs include the following tasks:

- Initializing or migrating a database
- Calculating one-off metrics from information within the cluster
- Creating or restoring from a data backup

The following example command creates a job that logs the date and time:

```
[user@host ~]$ oc create job \ 1
date-job \ 2
--image registry.access.redhat.com/ubi9/ubi \ 3
-- /bin/bash -c "date" 4
```

1. Creates a job resource.
2. Specifies a job name of `date-job`.
3. Sets `registry.access.redhat.com/ubi9/ubi` as the container image for the job pods.
4. Specifies the command to run within the pods.

You can also create a job from the web console by clicking the **Workloads** → **Jobs** menu. Click **Create Job** and customize the YAML manifest.

Cron Jobs

A *cron job* resource builds on a regular job resource by allowing you to specify how often the job should run. Cron jobs are useful to create periodic tasks that recur, such as backups or report generation. Cron jobs can also schedule individual tasks for a specific time, such as to schedule a job for a low activity period. Similar to the `crontab` cron table file on a Linux system, the `CronJob` resource uses the `Cron` format to schedule cron jobs. A `CronJob` resource creates a `job` resource based on the configured time zone on the control plane node that runs the cron job controller.

In OpenShift Container Platform 4.18, you can specify a time zone for the cron job schedule to ensure consistent timing regardless of the node's location.

The following example command creates a cron job named `date` that prints the system date and time every minute:

```
[user@host ~]$ oc create cronjob date-cronjob \ 1
--image registry.access.redhat.com/ubi9/ubi \ 2
--schedule "*/1 * * * *" \ 3
-- date 4
```

1. Creates a cron job resource with a name of `date-cronjob`.
2. Sets the `registry.access.redhat.com/ubi9/ubi` as the container image for the job pods.
3. Specifies the schedule for the job in `Cron` format.
4. The command to execute within the pods.

You can also create a cronjob from the web console by clicking the **Workloads** → **CronJobs** menu. Click **Create CronJob** and customize the YAML manifest.

Deployments

A *deployment* creates a replica set to maintain pods. A *replica set* maintains a specified number of replicas of a pod. Replica sets use selectors, such as a label, to identify pods that are part of the set. Pods are created or removed until the replicas reach the number that the deployment specifies. Replica sets are not managed directly. Instead, deployments indirectly manage replica sets.

Unlike a job, a deployment's pods are re-created after a crash or deletion, because deployments use replica sets.

NOTE

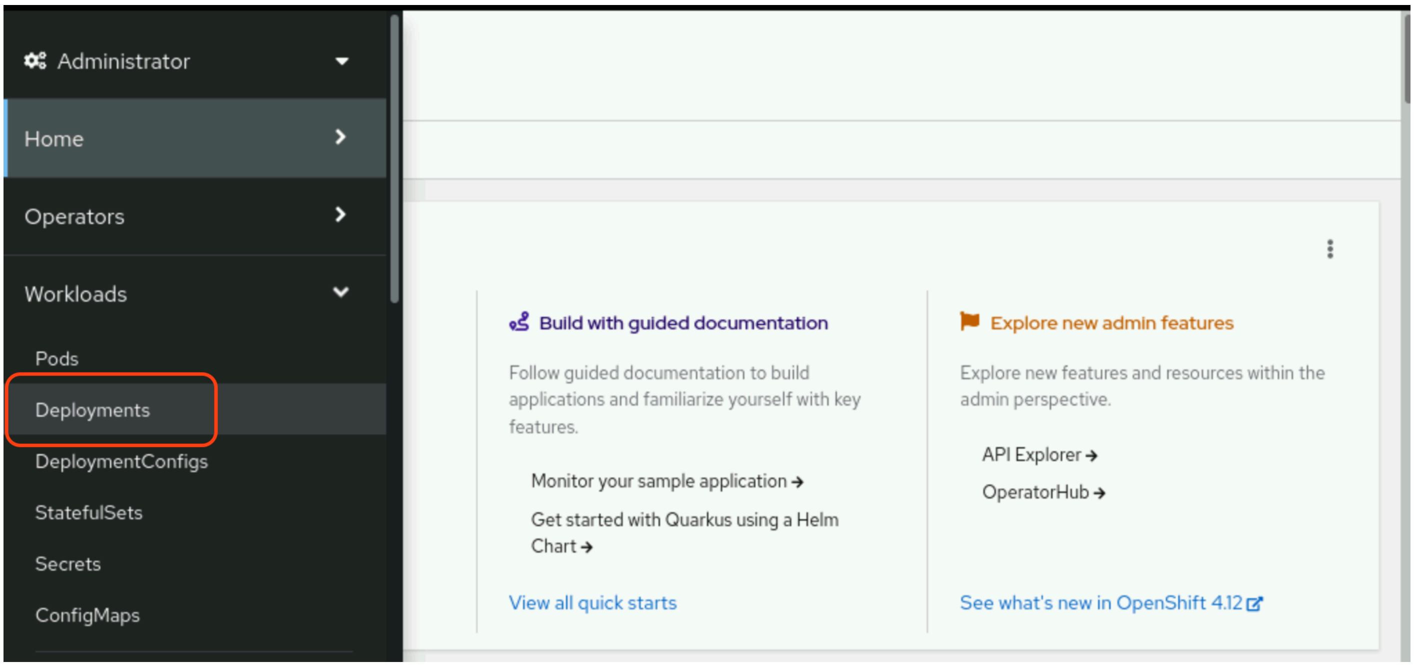
OpenShift previously provided *deployment configs* as an extension to deployments, that offers additional features like automatic triggers for builds and image changes. However, *deployment configs* are deprecated in OpenShift 4.18 and later, and new workloads should use standard Kubernetes *deployments* for better alignment with upstream Kubernetes and future compatibility.

The following example command creates a deployment:

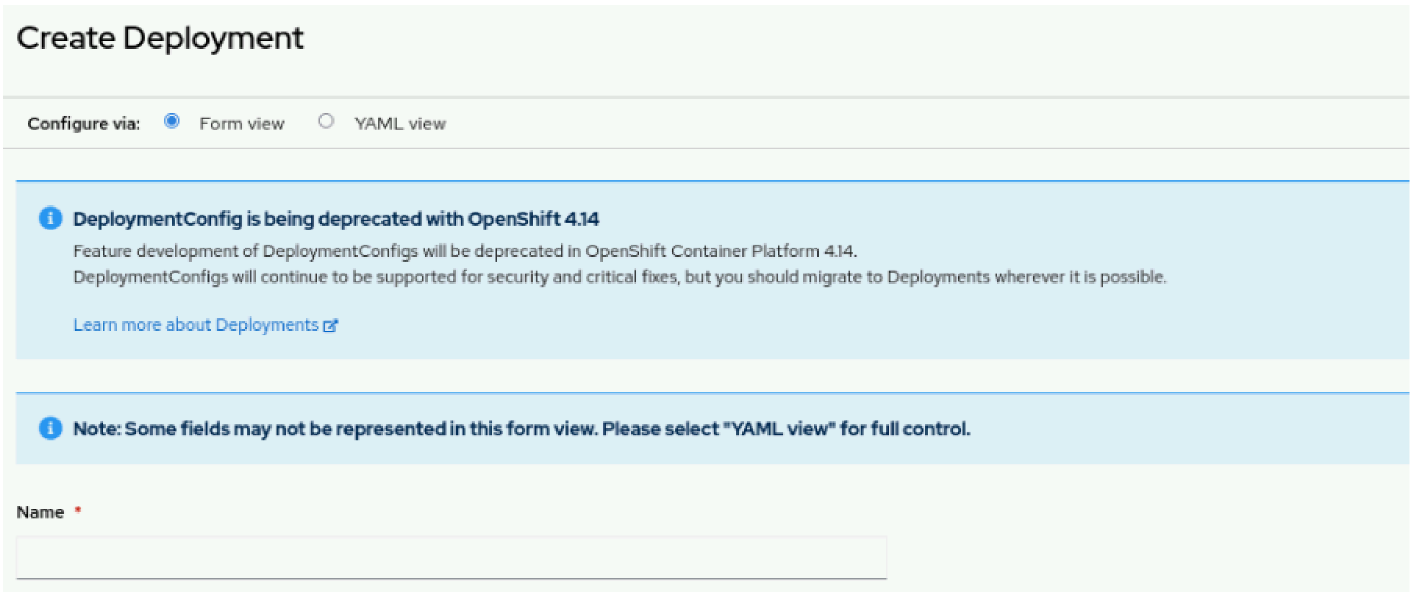
```
[user@host ~]$ oc create deployment \ 1
my-deployment \ 2
--image registry.access.redhat.com/ubi9/ubi \ 3
--replicas 3 4
```

1. Creates a deployment resource.
2. Specifies `my-deployment` as the deployment name.
3. Sets `registry.access.redhat.com/ubi9/ubi` as the container image for the pods.
4. Sets the deployment to maintain three instances of the pod.

You can also create a deployment from the web console by clicking the **Deployments** tab in the **Workloads** menu.


Administrator
Home
Operators
Workloads
Pods
Deployments
DeploymentConfigs
StatefulSets
Secrets
ConfigMaps
Build with guided documentation
Follow guided documentation to build applications and familiarize yourself with key features.
Monitor your sample application →
Get started with Quarkus using a Helm Chart →
View all quick starts
Explore new admin features
Explore new features and resources within the admin perspective.
API Explorer →
OperatorHub →
See what's new in OpenShift 4.12

Click **Create Deployment** and customize the form or the YAML manifest.


Create Deployment

Configure via: ● Form view ○ YAML view

DeploymentConfig is being deprecated with OpenShift 4.14
Feature development of DeploymentConfigs will be deprecated in OpenShift Container Platform 4.14.
DeploymentConfigs will continue to be supported for security and critical fixes, but you should migrate to Deployments wherever it is possible.

Learn more about Deployments

Note: Some fields may not be represented in this form view. Please select "YAML view" for full control.

Name *

Pods in a replica set are identical and match the pod template in the replica set definition. If the number of replicas is not met, then a new pod is created from the template. For example, if a pod crashes or is otherwise deleted, then a new pod is created to replace it.

Labels are a type of resource metadata that are represented as string key-value pairs. A label indicates a common trait for resources with that label. For example, you might attach a `layer=frontend` label to resources that relate to an application's UI components.

Many `oc` commands accept a label to filter affected resources. For example, the following command deletes all pods with the `environment=testing` label:

```
[user@host ~]$ oc delete pod -l environment=testing
```

By liberally applying labels to resources, you can cross-reference resources and craft precise selectors. A *selector* is a query object that describes the attributes of matching resources.

Certain resources use selectors to find other resources. In the following YAML excerpt, an example replica set uses a selector to match its pods.

```
apiVersion: apps/v1
kind: ReplicaSet
...output omitted...
spec:
  replicas: 1
  selector:
    matchLabels:
      app: httpd
      pod-template-hash: 7c84fbdb57
...output omitted...
```

Daemonsets

A *daemon set* ensures that all, or some, nodes run a copy of a pod. As nodes are added to the cluster, pods are added to them. As nodes are removed from the cluster, those pods are garbage collected. Delete the daemon set to clean up the pods it created.

Common uses for daemon sets include running storage daemons, log collectors, or monitoring agents on every node.

The following example command creates a daemon set:

```
[user@host ~]$ oc create daemonset my-daemonset \ 1
--image registry.access.redhat.com/ubi9/ubi 2
```

1 Creates a daemon set resource.

2 Sets `registry.access.redhat.com/ubi9/ubi` as the container image for the pods.

You can also create a daemon set from the web console by clicking the **Workloads** → **DaemonSets** menu. Click **Create DaemonSet** and customize the YAML manifest.

Stateful Sets

Similar to deployments, *stateful sets* manage a set of pods based on a container specification. However, each pod that a stateful set creates is unique. Pod uniqueness is useful when, for example, a pod needs a unique network identifier or persistent storage.

As their name implies, stateful sets are for pods that require a state within the cluster. Deployments are used for stateless pods.

Stateful sets are covered further in a later chapter.

REFERENCES

OpenShift Workloads

Kubernetes Workloads

Kubernetes Labels and Selectors