

Debugging pods in Kubernetes

Table of Contents

Prerequisites

What you'll learn

Debug pods in a Kubernetes cluster

Step 1: Check pod status

Step 2: Review pod logs

Step 3: Execute container commands

Step 4: Use ephemeral debug containers

Additional resources

Pods are the smallest deployable units in Kubernetes. Their health is critical to application functionality. Debugging pods helps ensure applications run smoothly, scale seamlessly, and use cluster resources effectively.

Prerequisites

Before starting, verify:

- Access to a Kubernetes cluster with running pods
- Sufficient permissions to execute `kubectl` commands

What you'll learn

This guide helps Kubernetes cluster administrators and developers troubleshoot pod issues. After reading, you'll be able to:

- Inspect pod status in a cluster
- Analyze logs for debugging
- Execute shell commands within containers
- Use ephemeral containers for interactive debugging

Estimated time: 30 minutes

Debug pods in a Kubernetes cluster

The following sections describe common debugging techniques. Use these techniques sequentially or combine them based on your specific scenario.

Step 1: Check pod status

Begin debugging by checking the status of your pods.

To list all pods:

```
$ kubectl get pods
```

SHELL

To list pods in a specific namespace:

```
$ kubectl get pods --namespace=<namespace-name>
```

SHELL

Example:

```
$ kubectl get pods --namespace=default
```

SHELL

Expected output:

NAME	READY	STATUS	RESTARTS	AGE
hello-node-7b87cd5f68-2wp4m	1/1	Running	0	21m
nginx-deployment-66b6c48dd5-8k4h2	0/1	Pending	0	5m
redis-master-58db8984f-xp4c8	0/1	ImagePullBackOff	0	2m

SHELL

TIP

If a pod is stuck in `Pending` status, check cluster resource availability using `kubectl describe node`.

For a graphical interface, use the Kubernetes dashboard:

- 1. Open the Kubernetes dashboard
- 2. Navigate to the Pods section
- 3. Select a pod to view its details

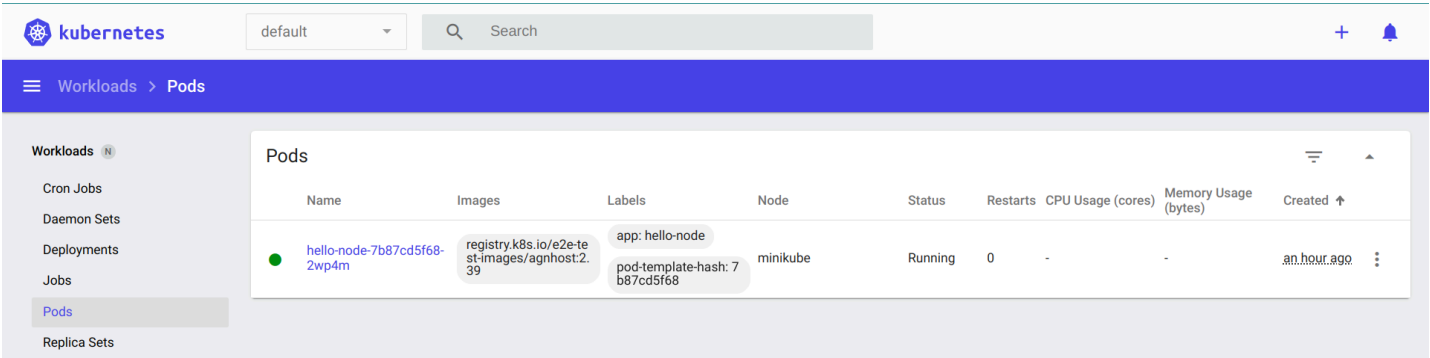


Figure 1. Pod details in the Kubernetes dashboard

Step 2: Review pod logs

Logs help identify what a container is doing or why it failed.

To view logs from all containers in a pod:

```
$ kubectl logs <pod-name> --all-containers=true
```

SHELL

To view logs from a specific container:

```
$ kubectl logs <pod-name> -c <container-name>
```

SHELL

NOTE

For pods with a single container, omit the container name.

Example:

```
$ kubectl logs hello-node-7b87cd5f68-2wp4m
```

SHELL

Expected output:

```
I0715 06:51:04.198447      1 log.go:195] Started HTTP server on port 8080
I0715 06:51:04.198572      1 log.go:195] Started UDP server on port 8081
```

SHELL

TIP

For pods in `CrashLoopBackOff` status, check logs with the `--previous` flag to see the last container's logs before it crashed:

```
$ kubectl logs <pod-name> --previous
```

SHELL

Step 3: Execute container commands

Inspect container state by running shell commands directly inside it.

Syntax:

```
$ kubectl exec <pod-name> -c <container-name> -- <command>
```

SHELL

NOTE

If not specified, commands run in the first container of the pod.

Examples:

Check container environment variables

```
$ kubectl exec nginx-deployment-66b6c48dd5-8k4h2 -- env
```

SHELL

Expected output:

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=nginx-deployment-66b6c48dd5-8k4h2
NGINX_VERSION=1.21.1
NJS_VERSION=0.6.1
PKG_RELEASE=1~buster
HOME=/root
```

SHELL

Verify network connectivity

```
$ kubectl exec nginx-deployment-66b6c48dd5-8k4h2 -- curl -I localhost:80
```

SHELL

Expected output:

```
HTTP/1.1 200 OK
Server: nginx/1.21.1
Date: Tue, 14 Jan 2025 10:15:23 GMT
Content-Type: text/html
Content-Length: 612
Connection: keep-alive
```

SHELL

Check running processes

```
$ kubectl exec nginx-deployment-66b6c48dd5-8k4h2 -- ps aux
```

SHELL

Expected output:

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.1	10640	5548	?	Ss	10:00	0:00	nginx: master process
nginx	31	0.0	0.1	11088	5164	?	S	10:00	0:00	nginx: worker process

SHELL

TIP

For containers that crash immediately, create a copy of the pod with a sleep command:

```
$ kubectl debug <pod-name> --copy-to=<pod-name>-debug --container=<container-name> -- sleep 1d
```

SHELL

Step 4: Use ephemeral debug containers

Ephemeral containers let you attach debugging tools to running pods without modifying the original containers.

To create an ephemeral debug container:

```
$ kubectl debug <pod-name> -it --image=<debug-image>
```

SHELL

Examples:

Debug networking issues using netshoot

```
$ kubectl debug nginx-deployment-66b6c48dd5-8k4h2 -it --image=nicolaka/netshoot
```

SHELL

Expected output:

```
Defaulting debug container name to debugger-nx8j2.
If you don't see a command prompt, try pressing enter.
~ # dig kubernetes.default.svc.cluster.local
~ # curl -v telnet://nginx-service:80
~ # tcpdump -i any port 80
```

SHELL

Analyze memory usage with tools

```
$ kubectl debug redis-master-58db8984f-xp4c8 -it --image=ubuntu
```

SHELL

Expected output:

Defaulting debug container name to debugger-7xj4d.
If you don't see a **command** prompt, try pressing enter.
root@redis-master-58db8984f-xp4c8:/# apt-get update
root@redis-master-58db8984f-xp4c8:/# apt-get install -y procps
root@redis-master-58db8984f-xp4c8:/# top
...Memory usage details...

TIP

For pods with `ImagePullBackOff` status, verify image name and registry credentials. Check image pull secrets using:

```
$ kubectl get pod <pod-name> -o=jsonpath='{.spec.imagePullSecrets[0].name}'
```

SHELL

Additional resources

For more information about debugging pods, see:

- [Debug running pods](https://kubernetes.io/docs/tasks/debug/debug-application/debug-running-pod/) (https://kubernetes.io/docs/tasks/debug/debug-application/debug-running-pod/) in the Kubernetes documentation