How to fix CrashLoopBackOff Kubernetes error

Kubernetes Troubleshooting

What is the CrashLoopBackOff error

`CrashLoopBackOff` is a common error in Kubernetes, indicating a pod constantly crashing in an endless loop.

You can identify this error by running the `kubectl get pods` command – the pod status will show the error like this:

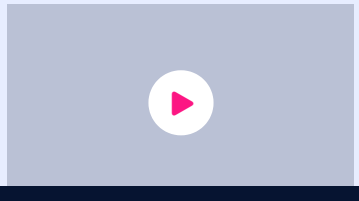
```
NAME        READY   STATUS             RESTARTS   AGE
my-pod-1    0/1     CrashLoopBackOff   2          1m44s
```

We'll provide best practices for diagnosing and resolving simple cases of these errors, but more complex cases will require advanced diagnosis and troubleshooting, which is beyond the scope of this article.

CrashLoopBackOff: Common Causes

The `CrashLoopBackOff` error can be caused by a variety of issues, including:

- **Insufficient resources**—lack of resources prevents the container from loading
- **Locked file**—a file was already locked by another container
- **Locked database**—the database is being used and locked by other pods
- **Failed reference**—reference to scripts or binaries that are not present on the container
- **Setup error**—an issue with the init-container setup in Kubernetes
- **Config loading error**—a server cannot load the configuration file
- **Misconfigurations**—a general file system misconfiguration
- **Connection issues**—DNS or kube-DNS is not able to connect to a third-party service
- **Deploying failed services**—an attempt to deploy services/applications that have already failed (e.g. due to a lack of access to other services)



Diagnosis and Resolution

The best way to identify the root cause of the error is to start going through the list of potential causes and eliminate them one by one, starting with the most common ones first.

Check for "Back Off Restarting Failed Container"

Run `kubectl describe pod [name]` .

If you get a `Liveness probe failed` and `Back-off restarting failed container` messages from the kubelet, as shown below, this indicates the container is not responding and is in the process of restarting.

```
From       Message
-----      -----
kubelet    Liveness probe failed: cat: can't open '/tmp/healthy': No such file o
kubelet    Back-off restarting failed container
```

If you get the `back-off restarting failed container` message this means that you are dealing with a temporary resource overload, as a result of an activity spike. The solution is to adjust `periodSeconds` or `timeoutSeconds` to give the application a longer window of time to respond.

If this was not the issue, proceed to the next step.

Check Logs From Previous Container Instance

If pod details didn't provide any clues, your next step should be to pull information from the previous container instance.

You originally ran `kubectl get pods` to identify the pod that was exhibiting the `CrashLoopBackOff` error. You can run the following command to get the last ten log lines from the pod:

```
kubectl logs --previous --tail 10
```

Search the log for clues showing why the pod is repeatedly crashing. If you cannot resolve the issue, proceed to the next step.

Check Deployment Logs

Run the following command to retrieve the kubectl deployment logs:

```
kubectl logs -f deploy/ -n
```

This may also provide clues about issues at the application level. For example, below you can see a log file that shows **./ibdata1** can't be mounted, likely because it's already in use and locked by a different container.

```
[ERROR] [MY-012574] [InnoDB] Unable to lock ./ibdata1 error:11
[ERROR] [MY-012574] [InnoDB] Unable to lock ./ibdata1 error:11
```

Failing all the above, your next step is to perform advanced debugging on the crashing container.

Advanced Debugging: Bashing Into CrashLoop Container

Step 1: Identify entrypoint and cmd

You will need to identify the **entrypoint** and **cmd** to gain access to the container for debugging. Do the following:

1. Run `docker pull [image-id]` to pull the image.
2. Run `docker inspect [image-id]` and locate the **entrypoint** and **cmd** for the container image, like in the screenshot below:

Step 2: Change entrypoint

Because the container has crashed and cannot start, you'll need to temporarily change the **entrypoint** in the container specification to `tail -f /dev/null` .

Step 3: Install debugging tools

With the **entrypoint** changed, you should be able to use the default command line kubectl to execute into the buggy container.

Make sure that you have debugging tools (e.g. **curl** or **vim**) installed, or add them. In Linux, you can use this command to install the tools you need:

```
sudo apt-get install [name of debugging tool]
```

Step 4: Check for missing packages or dependencies

Check if any packages or dependencies are missing, preventing the application from

Step 5: Check application configuration

Inspect your environment variables and verify if they're correct.

If that isn't the problem, then perhaps your configuration files are missing, causing the application to fail. You can download missing files using **curl**.

If there are any configuration changes required, like the username and password of the database configuration file, then you can use **vim** to resolve that.

If the issue was not with missing files or configuration, you'll need to look for some of the less generic reasons for the issue. Below are a few examples of what these may look like.

Examples of CrashLoopBackOff Edge Cases

Example 1: Issue With Third-Party Services (DNS Error)

Sometimes, the `CrashLoopBackOff` error is caused by an issue with one of the third-party services. If this is the case, upon starting the pod you'll see the message:

```
send request failure caused by: Post
```

Check the syslog and other container logs to see if this was caused by any of the issues we mentioned as causes of CrashLoopBackoff (e.g., locked or missing files). If not, then the problem could be with one of the third-party services.

To verify this, you'll need to use a debugging container. A debug container works as a shell that can be used to login into the failing container. This works because both containers share a similar environment, so their behaviors are the same. Here is a link to one such shell you can use: ubuntu-network-troubleshooting.

Using the shell, log into your failing container and begin debugging as you normally would. Start with checking `kube-dns` configurations, since a lot of third-party issues, start with incorrect DNS settings.

Example 2: Container Failure due to Port Conflict

Let's take another example in which the container failed due to a port conflict. To identify the issue you can pull the failed container by running `docker logs [container id]` .

Doing this will let you identify the conflicting service. Using `netstat` , look for the corresponding container for that service and kill it with the `kill` command. Delete the `kube-controller-manager pod` and restart.

5 Tips for Preventing the CrashLoopBackOff Error

Here is a list of best practices you can employ to prevent the `CrashLoopBackOff` error from occurring.

1. Configure and Recheck Your Files

A misconfigured or missing configuration file can cause the `CrashLoopBackOff` error, preventing the container from starting correctly. Before deployment, make sure all files are in place and configured correctly.

In most cases, files are stored in `/var/lib/docker` . You can use commands like `ls` and `find` to verify if the target file exists. You can also use `cat` and `less` to investigate files and make sure that there are no misconfiguration issues.

into the container and manually reach the endpoints using `curl` to check.

3. Check Your Environment Variables

Incorrect environment variables are a common cause of the `CrashLoopBackOff` error. A common occurrence is when containers require Java to run, but their environment variables are not set properly. Use env to inspect the environment variables and make sure they're correct.

4. Check Kube-DNS

The application may be trying to connect to an external service, but the `kube-dns` service is not running. Here, you just need to restart the `kube-dns` service so the container can connect to the external service.

5. Check File Locks

As mentioned before, file locks are a common reason for the `CrashLoopBackOff` error. Ensure you inspect all ports and containers to see that none are being occupied by the wrong service. If they are, kill the service occupying the required port.

Solving Kubernetes Errors Once and for All with Komodor

The troubleshooting process in Kubernetes is complex and, without the right tools, can be stressful, ineffective and time-consuming. Some best practices can help minimize the chances of things breaking down, but eventually something will go wrong – simply because it can.

This is the reason why we created Komodor, a tool that helps dev and ops teams stop wasting their precious time looking for needles in (hay)stacks every time things go wrong.

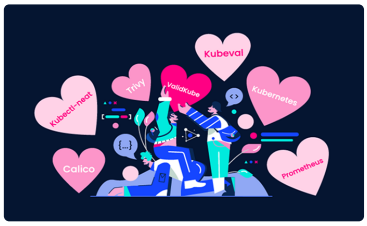
Acting as a single source of truth (SSOT) for all of your k8s troubleshooting needs, Komodor offers:

- **Change intelligence:** Every issue is a result of a change. Within seconds we can help you understand exactly who did what and when.
- **In-depth visibility:** A complete activity timeline, showing all code and config changes, deployments, alerts, code diffs, pod logs and etc. All within one pane of glass with easy drill-down options.
- **Insights into service dependencies:** An easy way to understand cross-service changes and visualize their ripple effects across your entire system.
- **Seamless notifications:** Direct integration with your existing communication channels (e.g., Slack) so you'll have all the information you need, when you need it.

If you are interested in checking out Komodor, use this link to sign up for a Free Trial.

Related Articles

Latest Blogs



Just Launched ValidKube. Here Are 7 Other K8s Open Source Projects We Love!

We just launched our first open source project, ValidKube! Read all about it, as well as our 7 favorite K8s OS tools that we love and highly recommend....

Komodor Kubernetes Tools

How One Company Accidently Autoscaled to 200 Nodes and Crashed The App

This is the first part of an ongoing series detailing some of the most horrific K8s incident stories we've heard from our customers....

DevOps Kubernetes Troubleshooting

The Top 5 Kubernetes Configuration Mistakes—And How to Avoid Them

This post outlines the top five Kubernetes configuration mistakes, as well as best practices to prevent these misconfigurations in future....

DevOps Kubernetes Troubleshooting