Container Lifecycle Hooks

This page describes how kubelet managed Containers can use the Container lifecycle hook framework to run code triggered by events during their management lifecycle.

Overview

Analogous to many programming language frameworks that have component lifecycle hooks, such as Angular, Kubernetes provides Containers with lifecycle hooks. The hooks enable Containers to be aware of events in their management lifecycle and run code implemented in a handler when the corresponding lifecycle hook is executed.

Container hooks

There are two hooks that are exposed to Containers:

`PostStart`

This hook is executed immediately after a container is created. However, there is no guarantee that the hook will execute before the container ENTRYPOINT. No parameters are passed to the handler.

`PreStop`

This hook is called immediately before a container is terminated due to an API request or management event such as a liveness/startup probe failure, preemption, resource contention and others. A call to the `PreStop` hook fails if the container is already in a terminated or completed state and the hook must complete before the TERM signal to stop the container can be sent. The Pod's termination grace period countdown begins before the `PreStop` hook is executed, so regardless of the outcome of the handler, the container will eventually terminate within the Pod's termination grace period. No parameters are passed to the handler.

A more detailed description of the termination behavior can be found in Termination of Pods.

Hook handler implementations

Containers can access a hook by implementing and registering a handler for that hook. There are two types of hook handlers that can be implemented for Containers:

- Exec - Executes a specific command, such as `pre-stop.sh`, inside the cgroups and namespaces of the Container. Resources consumed by the command are counted against the Container.
- HTTP - Executes an HTTP request against a specific endpoint on the Container.

Hook handler execution

When a Container lifecycle management hook is called, the Kubernetes management system executes the handler according to the hook action, `httpGet` and `tcpSocket` are executed by the kubelet process, and `exec` is executed in the container.

Hook handler calls are synchronous within the context of the Pod containing the Container. This means that for a `PostStart` hook, the Container ENTRYPOINT and hook fire asynchronously. However, if the hook takes too long to run or hangs, the Container cannot

reach a `running` state.

`PreStop` hooks are not executed asynchronously from the signal to stop the Container; the hook must complete its execution before the TERM signal can be sent. If a `PreStop` hook hangs during execution, the Pod's phase will be `Terminating` and remain there until the Pod is killed after its `terminationGracePeriodSeconds` expires. This grace period applies to the total time it takes for both the `PreStop` hook to execute and for the Container to stop normally. If, for example, `terminationGracePeriodSeconds` is 60, and the hook takes 55 seconds to complete, and the Container takes 10 seconds to stop normally after receiving the signal, then the Container will be killed before it can stop normally, since `terminationGracePeriodSeconds` is less than the total time (55+10) it takes for these two things to happen.

If either a `PostStart` or `PreStop` hook fails, it kills the Container.

Users should make their hook handlers as lightweight as possible. There are cases, however, when long running commands make sense, such as when saving state prior to stopping a Container.

Hook delivery guarantees

Hook delivery is intended to be *at least once*, which means that a hook may be called multiple times for any given event, such as for `PostStart` or `PreStop`. It is up to the hook implementation to handle this correctly.

Generally, only single deliveries are made. If, for example, an HTTP hook receiver is down and is unable to take traffic, there is no attempt to resend. In some rare cases, however, double delivery may occur. For instance, if a kubelet restarts in the middle of sending a hook, the hook might be resent after the kubelet comes back up.

Debugging Hook handlers

The logs for a Hook handler are not exposed in Pod events. If a handler fails for some reason, it broadcasts an event. For `PostStart`, this is the `FailedPostStartHook` event, and for `PreStop`, this is the `FailedPreStopHook` event. To generate a failed `FailedPostStartHook` event yourself, modify the lifecycle-events.yaml file to change the postStart command to "badcommand" and apply it. Here is some example output of the resulting events you see from running `kubectl describe pod lifecycle-demo`:

```
Events:
  Type     Reason               Age              From               Message
  ----     ------               ----             ----               -------
  Normal   Scheduled            7s               default-scheduler  Successfully
  Normal   Pulled               6s               kubelet            Successfully
  Normal   Pulling              4s (x2 over 6s)  kubelet            Pulling image
  Normal   Created              4s (x2 over 5s)  kubelet            Created conta
  Normal   Started              4s (x2 over 5s)  kubelet            Started conta
  Warning  FailedPostStartHook  4s (x2 over 5s)  kubelet            Exec lifecycl
  Normal   Killing              4s (x2 over 5s)  kubelet            FailedPostSta
  Normal   Pulled               4s               kubelet            Successfully
  Warning  BackOff              2s (x2 over 3s)  kubelet            Back-off rest
```

What's next

- Learn more about the Container environment.
- Get hands-on experience attaching handlers to Container lifecycle events.

Feedback

Was this page helpful?

Yes No

Last modified October 24, 2022 at 11:34 AM PST: updating page weights for content/en/docs/concepts/containers (3bb617369e)