# The Most Graceful Termination™

## How Kubernetes Pods Terminate and What Your Application Can Expect

Slides

Harrison Katz - SRE

ngrok

# Who is this guy?

- Harrison Katz
- He/Him Pronouns
- Automation Enthusiast
- 5+ Years Kubernetes Experience
- Enjoys Reading and Writing Docs
- hkatz@ngrok.com

I read the docs so you don't have to!

Harrison Katz – SRE

ngrok

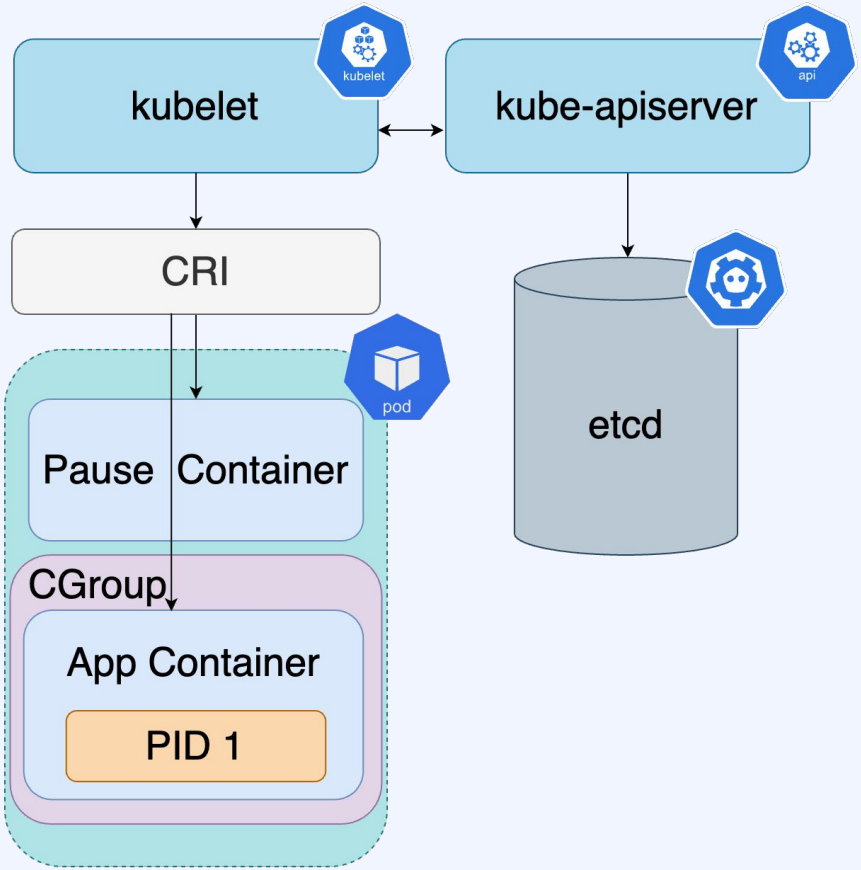
# Overview

- What is a Pod?
- Pod/Container Lifecycles
- How Pods are Deleted
- kubectl delete --flags
- Dynamic Graceful Shutdown

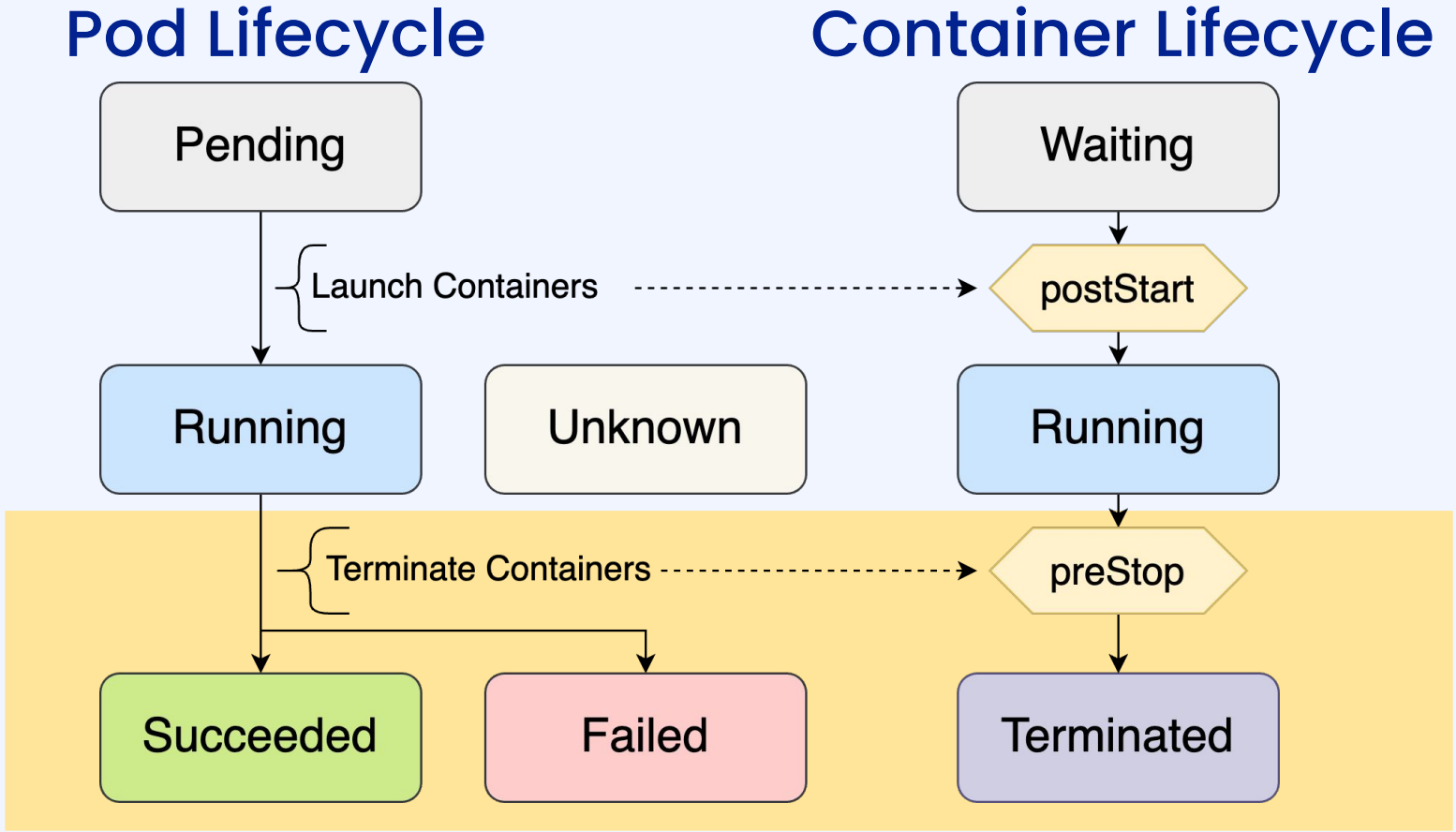
ngrok

# What is a Pod Anyway?

- API representation stored in etcd
- Managed by kubelet through a Container Runtime Interface (CRI)
- Pause container process
- App container process (pid: 1)




ngrok

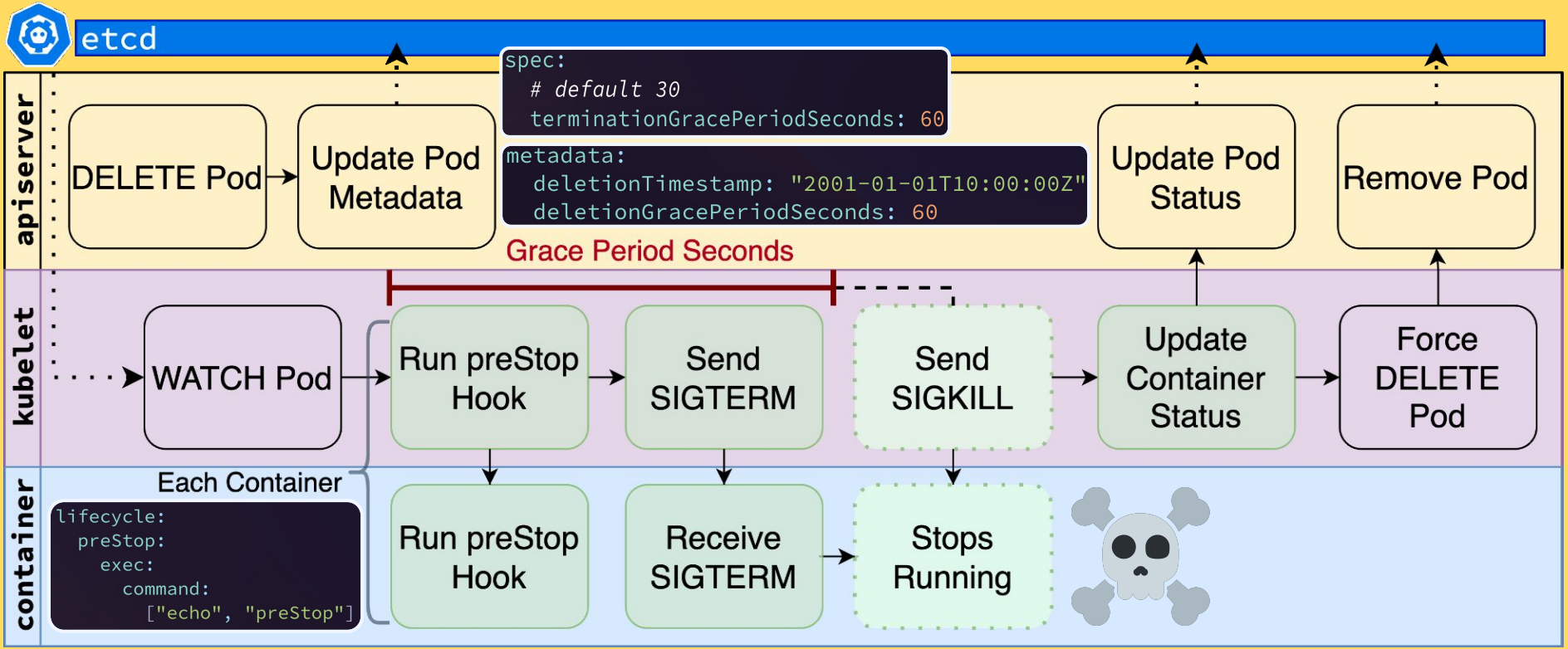


# When are Pods Terminated?

- Node Failure (abruptly)
- Liveness probe failure (container restart)
- Eviction (resource contention)
- API Request (graceful) 


**Pods are mortal: they live and *must* die.**

# How Pods are Deleted



# kubectl delete --flags

`--grace-period` ⇒ `.deletionGracePeriodSeconds`

- `-1` ⇒ “use the Pod’s configured value”
- `0` ⇒ “immediately kill the Pod”
- `1` ⇒ “shortest possible graceful delete”
- `N+1` ⇒ “use this exact grace period”

`--force` ⇒ skip termination process &&
remove Pod from etcd immediately

**Caveats**:

- “Terminating” is a fake status
- Only first `--grace-period` is respected

“Terminating”

```
metadata:
  deletionTimestamp: "2001-01-01T10:00:00Z"
  deletionGracePeriodSeconds: 60
```

ngrok

# Graceful Termination Full Example

```yaml
kind: Pod
metadata:
  name: graceful-terminator
  # set by apiserver => "Terminating"
  deletionTimestamp: "2001-01-01T10:00:00Z"
  deletionGracePeriodSeconds: 60
spec:
  terminationGracePeriodSeconds: 60 # default 30
  containers:
  - name: graceful-terminator
    image: hjkatz/graceful-terminator # pid 1 will receive SIGTERM
    lifecycle:
      preStop:
        exec:
          # uses grace period before SIGTERM -> container pid 1
          command: ["echo", "Received preStop hook"]
```

ngrok

# Dynamic Graceful Shutdown

0. Expose envars **`$POD_NAME`** and **`$POD_NAMESPACE`** via the downward API
1. **`SIGTERM`** → poll for **`.m.deletionTimestamp & .m.deletionGracePeriodSeconds`**
2. **`deadline = deletionTimestamp + deletionGracePeriodSeconds - overhead`**
3. Enter State: **`isTerminating() -> true`**
4. **`os.Exit(0)`** before deadline expires (**`SIGKILL`**)

# Example: Server Status

```
func main() {
    // Example function to run forever,
    // print the global state,
    // and move between states
    for range time.Tick(1 * time.Second) {
        printGlobalState()

        switch _global.State {
        case StateStartup:
            // use go routine to showcase states
            _startupOnce.Do(func() { go startup() })
        case StateRunning:
            // nothing to do
            // pretend to serve web requests or a heartbeat or something
        case StateShutdown:
            // use go routine to showcase states
            _shutdownOnce.Do(func() { go shutdown() })
        default:
            panic("unknown state: " + _global.State)
        }
    }
}
```

# Example: Shutdown Handler

```
log(_c.shutdown, "Shutting down...")

deadline := calculateShutdownDeadline()
log(_c.warn, "Server must shutdown before deadline: ", deadline)

ctx, cancel := context.WithDeadline(context.Background(), deadline)
go drainConnections(ctx)
go stopWebserver(ctx)

// wait for deadline
<-ctx.Done()

// ensure everything is stopped
cancel()

// wait a moment for go routines to finish any cleanup
time.Sleep(200 * time.Millisecond)

log(_c.success, "Exiting gracefully")
os.Exit(0)
```

# Example: Calculate Deadline

```
// choose some buffer time between min(5% of gracePeriod, 10s)
// this time represents however long we think it may maximally take
// our program to reach this point since receiving SIGTERM
//
// e.g. our best guess is we have ~95% of the gracePeriodSeconds left
fivePercentSeconds := int(gracePeriod.Seconds() * 0.05)
floorSeconds := 10
bufferSeconds := min(fivePercentSeconds, floorSeconds)
if time.Until(deletionTime).Seconds() < float64(bufferSeconds) {
    log(_c.warn, "WARN: buffer time longer than remaining gracePeriod, defaulting to 0")
    bufferSeconds = 0
}
log(_c.shutdown, "calculated buffer of ", bufferSeconds, "s")

// calculate deadline
deadline := deletionTime.Add(-(time.Duration(bufferSeconds) * time.Second))
return deadline
```

# Example: Drain Active Connections

```
deadline, _ := ctx.Deadline()
timeRemaining := time.Until(deadline)

// fake numbers
activeConnections := 1000
batchSize := int(
    float64(activeConnections) /
    math.Max(timeRemaining.Seconds()-1, 1.0)
)
maxBatchSize := 150 // per second

if batchSize > maxBatchSize {
    log(_c.warn, "Calculated batchSize of ",
    batchSize = maxBatchSize
}

log(_c.shutdown, "Draining ", activeConnecti
```

```
go func() {
    <-ctx.Done()
    if activeConnections > 0 {
        log(_c.important, "Unable to safely
    }
}()

for activeConnections > 0 {
    activeConnections -= batchSize
    // real connections wouldn't be negative
    if activeConnections < 0 {
        activeConnections = 0
    }

    if activeConnections <= 0 {
        break
    }

    log(_c.shutdown, "Draining ", batchSize,

    time.Sleep(1 * time.Second)
}
```

# Example: Logs (running)

```
[1s] State: startup
[1s] Starting up...
[1s] Running inside kubernetes, setting up k8s client...
[1s] Setting state from 'startup' -> 'running'
[1s] Setting up shutdown signal handler...
[1s] Serving requests on :8080
[2s] State: running
[3s] State: running
[4s] State: running
[5s] State: running
```

ngrok

# Example: Logs (SIGTERM handler)

```
[10s] State: running
[10s] Received signal: terminated
[10s] Setting state from 'running' -> 'shutdown'
[11s] State: shutdown
[11s] Shutting down...
[11s] Running inside kubernetes, polling Pod metadata to calculate deadline
[11s] Pod is expected to be deleted in 1m0s at 2024-03-17 18:10:18 +0000 UTC
[11s] calculated buffer of 3s
[11s] Server must shutdown before deadline: 2024-03-17 18:10:15 +0000 UTC
[11s] Draining 1000 connections in batches of 18 over 55s
[11s] Draining 18 connections, 982 left
[12s] State: shutdown
[12s] Draining 18 connections, 964 left
[13s] State: shutdown
```

# Example: Logs (shutdown)

```
[14s] Draining 144 connections, 136 left
[15s] State: shutdown
[15s] Successfully drained all connections
[16s] State: shutdown
[17s] State: shutdown
[17s] Exiting gracefully
```

ngrok

# Demo 

ngrok

# Things to Remember

- “Terminating” is a fake status controlled by **`.metadata.deletionTimestamp`**
- **`--grace-period`** *overrides* **`.spec.terminationGracePeriodSeconds`**
- **`--grace-period=1`** is the fastest way to gracefully shutdown a Pod
- **`preStop`** hooks run *before* the **`SIGTERM`** signal is sent to PID 1
- Please stop serving requests while your Pod is terminating
- **`os.Exit(0)`** means success!
- Complete this entire process before kubelet sends a **`SIGKILL`**
- Tip your waiters 💸 💸 💸

# Resources and References

- Slides and Demo: github.com/hjkatz/kubernetes-graceful-termination
- Me: hkatz@ngrok.com

---

- Kubernetes Pod Lifecycle
  kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle
- Graceful exit mechanism for Pods in Kubernetes
  www.sobyte.net/post/2022-06/k8s-pod-graceful
- PR: The second time a Pod is deleted the grace period does not take effect
  github.com/kubernetes/kubernetes/pull/113883
- PR Comment: Table of grace period override behaviours
  github.com/kubernetes/kubernetes/issues/115819
- How do you gracefully shut down Pods in kubernetes
  tnext.io/how-do-you-gracefully-shut-down-pods-in-kubernetes-fb19f617cd67
- Graceful shutdown in kubernetes is not always trivial
  blog.palark.com/graceful-shutdown-in-kubernetes-is-not-always-trivial/

# Questions
# Comments
# Concerns?

Slides, Demo Resources

ngrok