How Kubernetes "self-heals"

Let's first have an overview over the pieces of 'self-healing' mechanism by K8s.

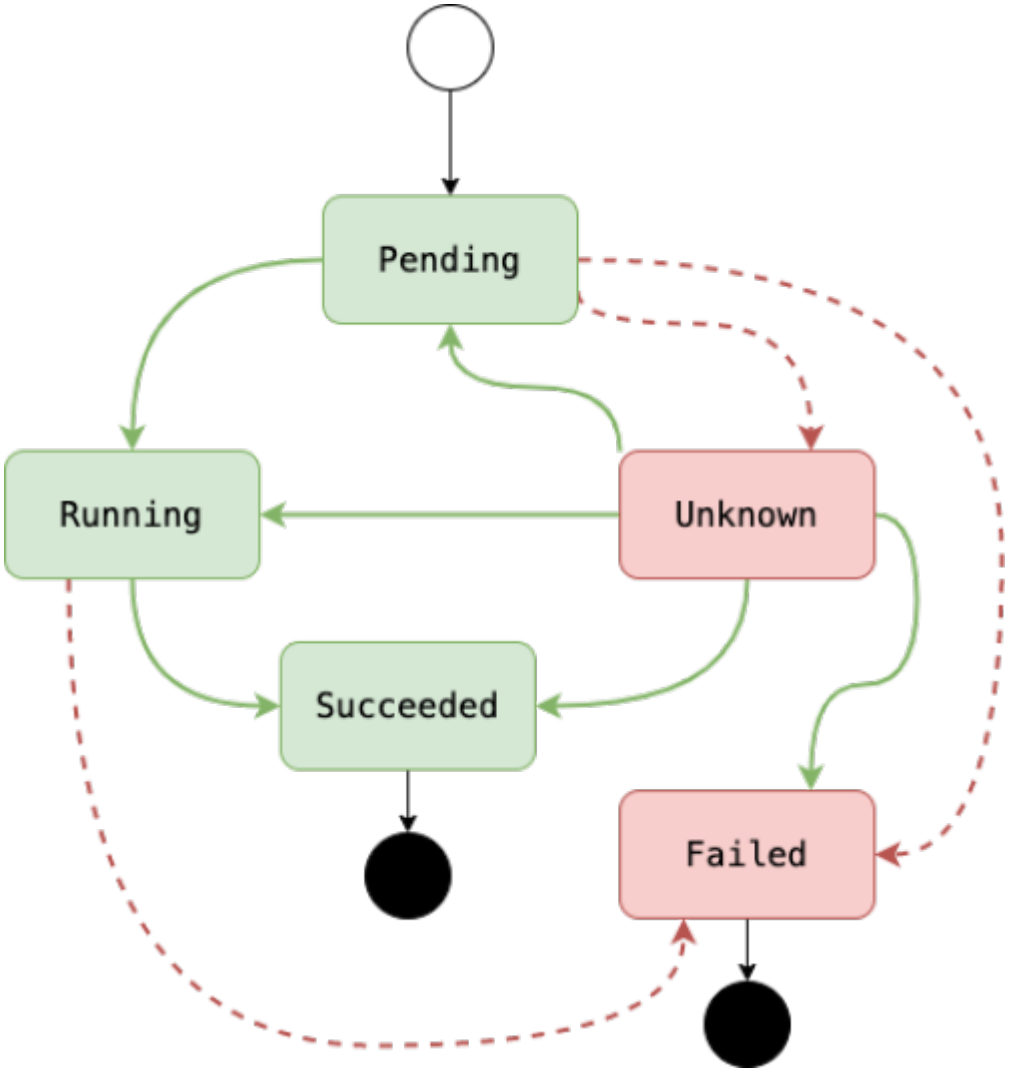
1. **Controllers** (Deployment → ReplicaSet → Pods): A Deployment declares the desired number of pod replicas. The `ReplicaSet` owned by that Deployment ensures that number is kept. If a pod is deleted or a pod's container exits, the `ReplicaSet` (and controller manager) will create a replacement pod.
2. restartPolicy: Container-level restarts are controlled by the Pod's `restartPolicy` (for Deployments it's Always by default). If the container process exits, **kubelet** will restart it according to that policy.
3. Health checks
    - **Liveness probe**: if it fails, kubelet will kill & restart the container (useful for recovering stuck processes).
    - **Readiness probe**: controls whether a pod is considered „ready" and included in Services' endpoints. During shutdown you should fail readiness, so the pod is removed from Service endpoints before termination.
4. **Pod lifecycle**: When a pod is terminated (e.g., kubectl delete pod, node shutdown, or rolling update), Kubernetes sends `SIGTERM` to processes (pid 1) in the container, waits `terminationGracePeriodSeconds` (default 30s) for graceful exit, then `SIGKILL`.

Pod Lifecycle

Lifecycle and hooks that are offered for Pods (and Containers) are points where your code can take actions. Kubernetes offers a number of places where you can provide explicit feedback to the system to have it operate as you'd like.

The lifecycle of a Pod is an aggregate of several components, as a Pod has a number of moving parts that can be in a variety of states as it operates. The representation of the lifecycle is how Kubernetes manages running your code for you, in conjunction with the control and feedback loops that work from the various controllers.

The states of a Pod's lifecycle are:



- **Pending**: The Pod has been created through the API, and is in the process of being scheduled, loaded, and run on one of the Nodes
- **Running**: The Pod is fully operational and the software is running within the cluster
- **Succeeded (or) Failed**: The Pod has finished operation (normally or crashed)
- **Unknown**: It occurs fairly rare, and it's typically seen when there's a problem internal to Kubernetes. In an example, K8s doesn't know the current state of containers, or is unable to communicate with its underlying systems to determine that state

Probes

The two probes enabled in Kubernetes are the **liveness probe** and **readiness Probe**. They are complimentary, but different in intent and usage, and can be defined for each container within a Pod. In both cases, they provide a means for your code to influence how Kubernetes manages the containers.

The most basic probe is the **Liveness probe**. It provides a command or URL that Kubernetes can use to determine whether a Pod is still operational. If the call succeeds, Kubernetes will assume the container is healthy. If it fails to respond, then the Pod can be handled as the `restartPolicy` is defined. The result is binary: either the probe succeeds, and Kubernetes believes your Pod is running, or it fails, so Kubernetes believes your Pod is no longer functional.

In the latter case, it will check with the defined `RestartPolicy` to choose what to do.

The default value for `restartPolicy` is Always, meaning if a container within the Pod fails, Kubernetes will always attempt...

Based on „Kubernetes for Developers by Joseph Heck“