this pattern, which allow separation of initializing activities from the main application duties.

Solution

Init Containers in Kubernetes are part of the Pod definition, and they separate all containers in a Pod into two groups: init containers and application containers. All init containers are executed in a sequence, one by one, and all of them have to terminate successfully before the application containers are started up. In that sense, init containers are like constructor instructions in a Java class that help object initialization. Application containers, on the other hand, run in parallel, and the startup order is arbitrary. The execution flow is demonstrated in Figure 14-1.

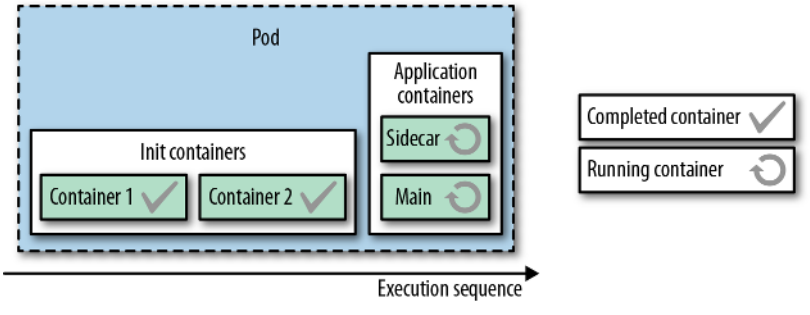

Figure 14-1. Init and application containers in a Pod

Typically, init containers are expected to be small, run quickly, and complete successfully, except when an init container is used to delay the start of a Pod while waiting for a dependency, in which case it may not terminate until the dependency is satisfied. If an init container fails, the whole Pod is restarted (unless it is marked with `RestartNever`), causing all init containers to run again. Thus, to prevent any side effects, making init containers idempotent is a good practice.

On one hand, init containers have all of the same capabilities as application containers: all of the containers are part of the same Pod, so they share resource limits, volumes, and security settings, and end up placed on the same node. On the other hand, they have slightly different health-checking and resource-handling semantics. There is no readiness check for init containers, as all init containers must terminate successfully before the Pod startup processes can continue with application containers.