At-Most-One Guarantee

Uniqueness is among the fundamental attributes of stateful application instances, and Kubernetes guarantees that by making sure no two Pods of a StatefulSet have the same identity or are bound to the same PV. In contrast, ReplicaSet offers the *At-Least-X-Guarantee* for its instances. For example, a ReplicaSet with two replicas tries to keep at least two instances up and running at all times. Even if occasionally there is a chance for that number to go higher, the controller's priority is not to let the number of Pods go below the specified number. It is possible to have more than the specified number of replicas running when a Pod is being replaced by a new one, and the old Pod is still not fully terminated. Or, it can go higher if a Kubernetes node is unreachable with `NotReady` state but still has running Pods. In this scenario, the ReplicaSet's controller would start new Pods on healthy nodes, which could lead to more running Pods than desired. That is all acceptable within the semantics of At-Least-X.

A StatefulSet controller, on the other hand, makes every possible check to ensure no duplicate Pods—hence the *At-Most-One Guarantee*. It does not start a Pod again unless the old instance is confirmed to be shut down completely. When a node fails, it schedules new Pods on a different node unless Kubernetes can confirm that the Pods (and maybe the whole node) are shut down. The At-Most-One semantics of StatefulSets dictates these rules.

It is still possible to break these guarantees and end up with duplicate Pods in a StatefulSet, but this requires active human intervention. For example, deleting an unreachable node resource object from the API Server while the physical node is still running would break this guarantee. Such an action should be performed only when the node is confirmed to be dead or powered down, and no Pod processes are running on it. Or, for example, forcefully deleting a Pod with `kubectl delete pods _<pod>_ --grace-period=0 --force`, which does not wait for a confirmation from the Kubelet that the Pod is terminated. This action immediately clears the Pod from the API Server and causes the StatefulSet controller to start a replacement Pod that could lead to duplicates.

We discuss other approaches to achieving singletons in more depth in Chapter 10, *Singleton Service*.

Discussion

In this chapter, we saw some of the standard requirements and challenges in managing distributed stateful applications on a cloud-native platform. We discovered that handling a single-instance stateful application is relatively easy, but handling distributed state is a multidimensional challenge. While we typically associate the notion of "state" with "storage," here we have seen multiple facets of state and how it requires different guarantees from different stateful applications. In this space, StatefulSets is