Exploring Quality of Service (QoS) Contracts

In this lesson, we will explore the Quality of Service contracts and types of QoS classes.

WE'LL COVER THE FOLLOWING

Understanding the Scheduling Process

When we send a request to Kubernetes API to create a Pod (directly or through one of the Controllers), it initiates the scheduling process. What happens next or, to be more precise, where it will decide to run a Pod, depends hugely on the resources we defined for the containers that form the Pod. In a nutshell, Kubernetes will decide to deploy a Pod, whenever it is possible, inside one of the nodes that has enough available memory.

Memory Requests and Limits

When memory requests are defined, Pods will get the memory they requested. If the memory usage of one of the containers exceeds the requested amount, or if some other Pod needs that memory, the Pod hosting it might be killed. Please note that we wrote that a Pod *might* be killed. Whether that will happen depends on the requests from other Pods and the available memory in the cluster. On the other hand, containers that exceed their memory limits are always killed (unless it was a temporary situation).

CPU Requests and Limits

CPU requests and limits work a bit differently. Containers that exceed specified CPU resources are not killed. Instead, they are throttled.

The Quality of Service

Now that we shed a bit of light around Kubernetes' killing activities, we should note that (almost) nothing happens randomly. When there aren't enough resources to serve the needs of all the Pods, Kubernetes will destroy one or more containers. The decision which one it will be is anything but random. Who will be the unlucky one depends on the assigned Quality of Service (QoS). Those with the lowest priority are killed first.

Since this might be the first time you heard about QoS, we'll spend some time explaining what it is and how it works.

Pods are the smallest units in Kubernetes. Since almost everything ends up as a Pod (one way or another), it is no wonder that Kubernetes promises specific guarantees to all the Pods running inside the cluster. Whenever we send a request to the API to create or update a Pod, it gets assigned one of the Quality of Service (QoS) classes. They are used to make decisions such as where to schedule a Pod or whether to evict it.

We do not specify QoS directly. Instead, they are assigned based on the decisions we make with resource requests and limits.

Exploring the Types of QoS Classes

At the moment, three QoS classes are available. Each Pod can have the *Guaranteed*, the *Burstable*, or the *BestEffort* QoS.

Guaranteed QoS

Guaranteed QoS is assigned only to Pods which have set both CPU requests and limits, and memory requests and limits for all of their containers. The Pods we created with the last definition match that criteria.

However, there's one more necessary condition that must be met. The requests and limits values must be the same per container. There is one more catch. When a container specifies only limits, requests are automatically set to the same values. In other words, containers without requests will have the same values. In other words, containers without requests will have Guaranteed QoS if their limits are defined.

We can summarize criteria for Guaranteed QoS as follows.

- Both memory and CPU limits must be set.
- Memory and CPU requests must be set to the same values as the limits, or they can be left empty, in which case they default to the limits (we'll explore them soon).

Pods with Guaranteed QoS assigned are the top priority and will never be killed unless they exceed their limits or are unhealthy. They are the *last to go* when things go wrong. As long as their resource usage is within limits, Kubernetes will always choose to kill Pods with other QoS assignments when resource usage is over the capacity.

Let's move to the next QoS.

Burstable QoS

Burstable QoS is assigned to Pods that do not meet the criteria for Guaranteed QoS but have at least one container with memory or CPU requests defined.

Pods with the Burstable QoS are guaranteed minimal (requested) memory usage. They might be able to use more resources if they are available. If the system is under pressure and needs more available memory, containers belonging to the Pods with the Burstable QoS are more likely to be killed than those with Guaranteed QoS when there are no Pods with the BestEffort QoS. You can consider the Pods with this QoS as *medium priority*.

Finally, we reach the last QoS class.

BestEffort QoS

BestEffort QoS is given to the Pods that do not qualify as Guaranteed or Burstable. They are Pods that consist of containers that have none of the resources defined. Containers in Pods qualified as BestEffort can use any available memory they need.

When in need of more resources, Kubernetes will start killing containers residing in the Pods with BestEffort QoS. They are the *lowest priority*, and they are the first to disappear when more memory is needed.

In the next lesson, we will get practical and see QoS in action.