QoS classes

QoS class is used by the Kubernetes scheduler to make decisions about scheduling pods onto nodes.

When Kubernetes creates a Pod, it assigns one of these QoS classes to the Pod:

* BestEffort
* Burstable
* Guaranteed

Let’s understand each of the QoS one by one.

**BestEffort**:

For a Pod to be given a QoS class of BestEffort, the Containers in the Pod must not have any memory or CPU limits or requests.

To see the code snippet, check BestEffort.yml

In case of BestEffort type of QoS – since we have not declared any resource, Kubernetes has no idea where to put the pod. It can put it in any Node. Some node may be very busy some might be idle, but since we have not told our cluster how much room our deployment needs, it is forced to guess.

Kubernetes will do its best effort to fit it into any node, but you’ve not given it much to work with.

Why this is not recommended?

Consider the following scenario. You’ve got a cluster with a single node, and that node is very busy. It is already running lots of applications. Each pod has declared their resources, then comes your pod without declaring any resources. Since there is only single node available, Kubernetes will place your pod into that busy node.   
Imagine your new pod started consuming a lot of resources at higher rate. Kubernetes kills pod if it uses resources beyond the limit, but we don’t have any limit as we have not declared it. In this scenario, Kubernetes has no idea whether this is a bad pod or a good pod. So, it puts its best effort to put it in by adjusting as much as it can.

It won’t just starve your other pods, but unless you’ve got some specific monitoring on your nodes, it is going to escalate and consume the CPU that would otherwise be allocated for important things, like the kubelet.

And finally, your node goes dark and, on some platforms, it can be more than 15 minutes before your cluster self-heals.

When it is recommended

To be frank, it is NOT RECOMMENDED at all. But if your application is very tiny and you are aware of it that it won’t consume much resources moving forward. In that scenarios as well, it is always recommended to allocate a very small set of resources instead of just not declaring it.

**Burstable**:

A Pod is given a QoS class of Burstable if:

* At least one Container in the Pod has a memory or CPU request.
* The Pod does not meet the criteria for QoS class Guaranteed. (discussed below)

This is far better than your BestEffort pod. This happens when you declare the base request for resources, but your limit is higher than your request.

To see the code snippet, check Burstable.yml

And why our limit is slightly higher than the request? – Because we know at some given point our application will consume more request and we are very much aware of it.

**What is the problem in Burstable Pods then?** – Consider a scenario, we have multiple Burstable pods in a node, and imagine the situation if all the pods request for more than their limit AT THE SAME TIME. This is commonly referred to as the “**noisy neighbor**” problem. Unfortunate this will lead to an outage and even the starvation of CPU on a node.

**How do we fix this problem?** – Its more of a better judgment. Some pods sit idle for a long time and, perhaps for an hour, they get busy. It doesn’t make sense to hold onto resources for the other 23 hours. So, you make the choice. Risk vs. reward.

The Burstable QoS is an excellent, k8s-native method for cost optimization in your cluster and it isn’t as black and white as the BestEffort quality of service.

If the pod self-heals quickly and you can tolerate short outages, Burstable might be a good method for you to save some money.

**Guaranteed**:

For a Pod to be given a QoS class of Guaranteed:

* Every Container in the Pod must have a memory limit and a memory request, and they must be the same.
* Every Container in the Pod must have a CPU limit and a CPU request, and they must be the same.

“Everything is Guaranteed from now on”. The most recommended one, but it is expensive, but have good stability. A Guaranteed QoS is achieved when your pod request and limit is exactly the same. This removes the possibility of scaling out into more CPU, but it reserves the exact amount that your containers are going to need.

To see the code snippet, check Guaranteed.yml

If you are clear already, how much resources your pod will utilize, this is by far the best option to go with. Also, remember we discussed about noisy neighbor problem in Burstable pod, it eliminates that problem.

Problem: More cost. More the resource pod reserves, more the node you need. If you have requested for resources but your Pod is actually not using that much resource, it’s a waste of money.

So, it’s your wise decision to go with Guaranteed QoS or Burstable QoS or why not both.

Sources:

1. <https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/>
2. <https://www.replex.io/blog/everything-you-need-to-know-about-kubernetes-quality-of-service-qos-classes#:~:text=Quality%20of%20Service%20(QoS)%20class,about%20scheduling%20pods%20onto%20nodes>
3. <https://medium.com/better-programming/the-kubernetes-quality-of-service-conundrum-eebbbb5f89cf>