Manage pod stability

Kubernetes clusters manage the allocation of system resources and can evict pods to release system resources. Here are some ways that you can configure the Redis Enterprise node pods to maintain pod stability:

Guaranteed quality of service

A running pod has a quality of service measure assigned to it that is one of three quality of service classes: Guaranteed, Burstable, and Best Effort. You can assure the Guaranteed class is assigned to the Redis Enterprise node pods by following the right guidelines.

To get a Guaranteed quality of service class assigned:

- 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.

If resources limits and requests are not specified in the Redis Enterprise CRD, these requirements are met in the default version created by the operator of your Redis Enterprise cluster CRD. Otherwise, you must set the limits and requests to the same value for memory and CPU in the `redisEnterpriseNodeResources` section of the CRD.

Sidecar containers also impact the quality of service class assignment for the pod.

To check the quality of service class of any running Redis Enterprise node pod, run:

```
kubectl get pod rec-0 --o=jsonpath="{.status.qosClass}"
```

where `rec-0` is the name of one of the pods associated with the Redis Enterprise cluster.

Using priority to protect from preemption

When a Redis Enterprise node pod is scheduled, it can be assigned a priority class with the `priorityClassName` property. This property value is the name of a priority class that must already exist within the cluster.

A sufficiently high priority will prevent other workloads with a lower priority from preempting the scheduling of Redis Enterprise Nodes. Similarly, a high value may also prevent eviction when lower priority workloads are deployed on the same cluster.

The successful use of this strategy involves first creating a priority class with a very large priority value:

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: redis-enterprise-priority
value: 100000000
globalDefault: false
description: "This priority class should be used for Redis Enterprise pods only."
```

Then, you refer to the priority class by name in your Redis Enterprise cluster CRD:
Alternatively, you can also disable preemption entirely.

**Managing eviction thresholds**

Eviction thresholds are typically managed by kubelet arguments. You can set the thresholds:

- On OpenShift - In the [config file](#).
- On GKE - In the [managed settings](#).

We recommend that you:

- Set the [soft eviction threshold](#) to be higher than the [hard eviction threshold](#). The high soft threshold makes the node condition change earlier, and alerts the administrator.
- Set `eviction-max-pod-grace-period` high enough to allow the RS pods to migrate the Redis databases before the pods are force killed.
- Set the `eviction-soft-grace-period` high enough that the administrator (or a k8s auto-scaling mechanism) scales k8s up or out.

**Monitoring for memory and disk usage**

We recommend that you monitor the node for MemoryPressure and DiskPressure. When both of these conditions are true, then an eviction threshold is met and the pod is evicted.

To retrieve the flags, run the command:

```bash
kubectl get nodes -o jsonpath='{range .items[*]}name:{.metadata.name} 
{"\t"}MemoryPressure:{.status.conditions[?(@.type == "MemoryPressure")].status} 
{"\t"}DiskPressure:{.status.conditions[?(@.type == "DiskPressure")].status}{"\n"}{end}'
```

```
name:gke-55d1ac88-213c MemoryPressure:False DiskPressure:False
ame:gke-55d1ac88-vrpp MemoryPressure:False DiskPressure:False
ame:gke-7253cc19-42g0 MemoryPressure:False DiskPressure:False
```

**Updated:** October 6, 2021