**Introduction to Replication Controller (rc)**

* Replication controller makes sure that a specified number of PODs are always running on a cluster. i.e., the application running in the cluster will not face downtime as there are N number of application replicas created by Replication Controller and running in the cluster, which isn’t possible in standalone Pod.
* Replication Controller will make sure onlyspecified number of Pods are running in the cluster and the Pods will be added or removed, if mismatched. Based on the replica count given, the Kube controller manager will verify the replica count that should be running in the cluster and add or remove Pods, accordingly.
* The Api version used for creating Replication Controller is v1
* In RC the workflow is simple, the Pods are created by RC, not by us. The Pods created by us are basically standalone pods. That is why deploying application on K8s with minimum workload resource replication controller is recommended. Because if the replica count is given 2 and due to an issue one of the Pods get deleted, immediately a pod is created as replica count is 2 which is the desired state. This desired check is done by Kube control manager.
* Basically, RC uses Pod template to create Pods based on the replica count mentioned in the specifications. By default, the replica count is 1. The Pod template has all the information given to create a pod like no.of containers to create, name of the container, image to use, volume type to use etc.
* The scaleout and scalein of the Pods is done using a command – **Kubectl scale –replica=<count> rc/<rcname>**. Scalein is nothing but decreasing the Pod count that should run in a cluster and scaleout is increasing the Pod count.
* The restart policy in the Pod template specification should be always, which is default.
* The selector labels(.spec.selector) mentioned in the RC specification should be equal to the Pod selector labels mentioned in the Pod template metadata labels(.spec.template.metadata.labels). While the Pods are being created these labels get assigned to the Pod through which the RC can identify the Pods and even control them. Only if the labels are matched, the RC can control the Pods if not the RC can’t control the Pods created.
* If the selector labels of the RC are not defined, then the Pod template metadata labels (.spec.template.metadata.labels) get assigned to selector labels of the RC.
* To delete the Pod managed by RC, the RC has to be delete first not the Pod because if the Pod is delete first, it is immediately recreated due to replica concept.
* Deleting an RC without deleting the Pods managed by the same RC is possible by using a command – **kubectl delete –cascade=orphan** **rc/<rc\_name>.**
* The labels given to the Pods in the template can also be used to isolate the pods from the RC i.e., if the labels are removed in the pod template, then the RC loses control on the Pods. And also if a new standalone Pod with same label as RC is created then this pod is also controlled by the RC.