Introduction

This lesson focuses on the introduction of this chapter.

WE'LL COVER THE FOLLOWING

- Kubernetes system
- Dynamic nature of Kubernetes

Change is the essential process of all existence.

- Spock

Kubernetes system

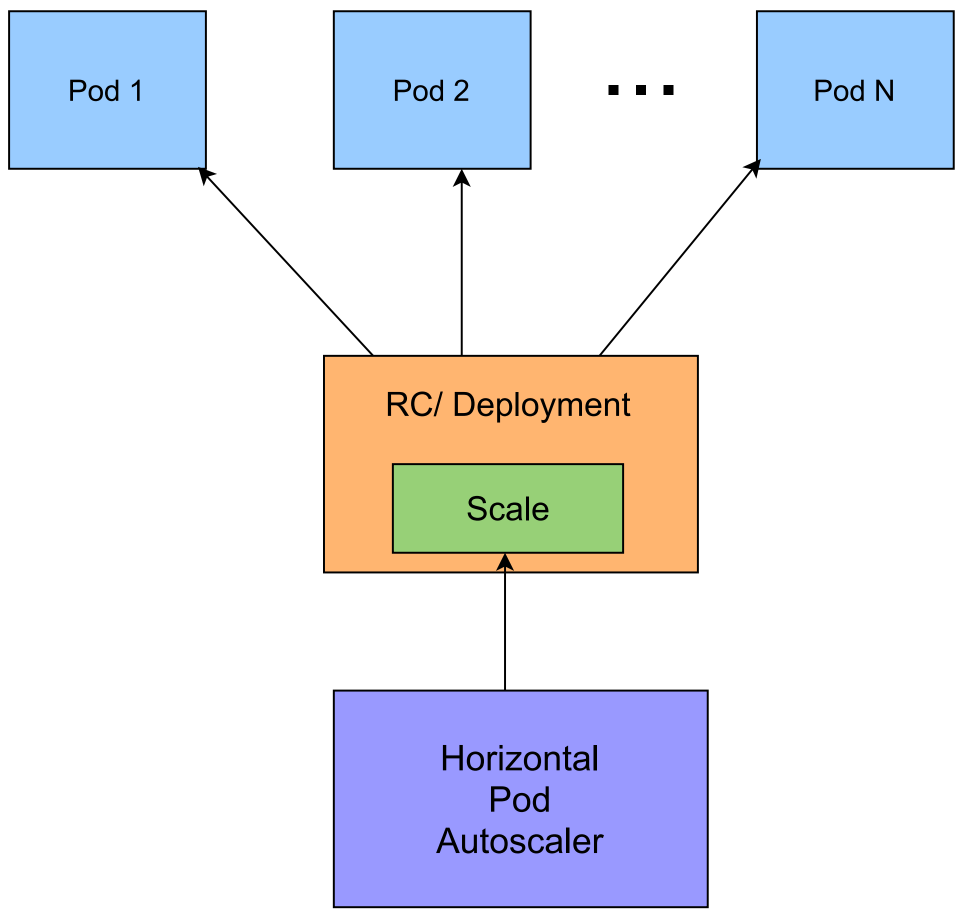
By now, you probably understood that one of the critical aspects of a system based on Kubernetes is a high level of dynamism. Almost nothing is static. We define Deployments or StatefulSets, and Kubernetes distributes the Pods across the cluster. In most cases, those Pods are rarely sitting in one place for a long time. Rolling updates result in Pods being re-created and potentially moved to other nodes. Failure of any kind provokes the rescheduling of the affected resources. Many other events cause the Pods to move around. A Kubernetes cluster is like a beehive. It's full of life, and it's always in motion.

Dynamic nature of Kubernetes

The dynamic nature of a Kubernetes cluster is not only due to our (human) actions or rescheduling caused by failures. Autoscaling is to be blamed as well. We should fully embrace Kubernetes' dynamic nature and move towards autonomous and self-sufficient clusters capable of serving the needs of our applications without (much) human involvement. To accomplish that, we need to provide sufficient information that will allow Kubernetes' to scale the applications as well as the nodes that constitute the cluster. In this chapter, we'll focus on the former case. We'll explore commonly used and basic ways to auto-scale Pods based on memory and CPU consumption. We’ll accomplish that using `HorizontalPodAutoscaler`.

> `HorizontalPodAutoscaler's` only function is to automatically scale the number of Pods in a Deployment, a StatefulSet, or a few other types of resources. It accomplishes that by observing CPU and memory consumption of the Pods and acting when they reach pre-defined thresholds.

`HorizontalPodAutoscaler` is implemented as a Kubernetes API resource and a controller. The resource determines the behavior of the controller. The controller periodically adjusts the number of replicas in a StatefulSet or a Deployment to match the observed average CPU utilization to the target specified by a user.



How HorizontalPodAutoScaler Works?

We'll see `HorizontalPodAutoscaler` in action soon and comment on its specific features through practical examples. But, before we get there, we need a Kubernetes cluster as well as a source of metrics.

In the next lesson, we will create a cluster and get on with the process of Autoscaling Deployments and StatefulSets.