Autoscaling in Kubernetes Summary

Liam Nguyen, Neha Bhoi

Abstract

Container is a standalone, self-contained units that package software and its dependencies together. As the cloud industry adopts container technologies for internal usage and as commercial offering, a large-scale system of containers open many challenges to manage such system. Kubernetes, introduced in 2015, is a container orchestration platform from Google that promises to run distributed system resiliently and effectively. One of its core responsibility is to utilize and balance resources among containers and cluster of containers by auto-scaling against certain metrics. In this paper, we will discuss how Kubernetes gathers metrics and perform its auto-scaling actions. In addition, we also explore proposed improvements in scaling techniques to handle the sharing of resources among containers more fair, robust and efficiently.

Prior to the introduction of containerization, virtualization technology dominates the application world. However, Virtual Machine (VM) take up a lot of expensive resources such as CPU and RAM. As an alternative, containers are light-weight and fast to store an application or application components with all the library dependencies, the binaries. Kubernetes, a container orchestration platform, provides functionalities such as load balancing, deployment and scaling of wide range of workloads. Most importantly, it can automatically restart the failed containers and reschedule them even when the hosts die.

Kubernetes provides 3 mechanisms to support auto-scaling:

1. **Horizontal pod autoscaler (HPA)** changes the shape of your Kubernetes workload by automatically increasing or decreasing the number of Pods in response to the workload's CPU or memory consumption, or in response to custom metrics reported from within Kubernetes or external metrics from sources outside of your cluster.

2. **Vertical Pods Autoscaler (VPA)** aims to allocate more (or less) CPU or memory to existing pods. VPA works for both stateful or stateless pods but generally, it is built for stateful services. When VPA kicks in, it requires the pods to be restarted to change the allocated cpu and memory.

3. **Cluster Autoscaler (CA)** scales the cluster nodes based on pending pods. If needed, CA scales the size of the cluster to fit the needs. In case of expansion, when the node is granted by the cloud provider, the node is joined to the cluster and serve pods. The whole process of scaling up is about 30 seconds. On the other hands, CA has a cool-down of 10 minutes before scales down when nodes are not needed.

Though all these mechanisms are there but still we can improve memory and CPU usage of worker node by applying machine learning approach. In feature we are going to talk about how these approaches.