Can We Scale up Too Much or De-scale to Zero Nodes?

In this lesson, we will see how to limit our nodes from either scaling too high or descaling too low.

WE'LL COVER THE FOLLOWING

Scale or descale without threshold

If we let `Cluster Autoscaler` do its “magic” without defining any thresholds, our cluster or our wallet might be at risk.

We might, for example, misconfigure `HPA` and end up scaling Deployments or StatefulSets to a huge number of replicas. As a result, `Cluster Autoscaler` might add too many nodes to the cluster. As a result, we could end up paying for hundreds of nodes, even though we need much less. Luckily, AWS, Azure, and GCP limit how many nodes we can have so we cannot scale to infinity. Nevertheless, we should not allow `Cluster Autoscaler` to go over some limits.

Similarly, there is a danger that `Cluster Autoscaler` will scale down to too few nodes. Having zero nodes is almost impossible since that would mean that we have no Pods in the cluster. Still, we should maintain a healthy minimum of nodes, even if that means sometimes being underutilized.

Minimum number of nodes

A reasonable minimum of nodes is **three**. That way, we have a worker node in each zone (datacenter) of the region. As you already know, Kubernetes requires three zones with master nodes to maintain quorum. In some cases, especially on-prem, we might have only one geographically collocated

datacenter with low latency. In that case, one zone (datacenter) is better than none. But, in the case of Cloud providers, three zones are the recommended distribution, and having a minimum of one worker node in each makes sense. That is especially true if we use block storage.

By its nature, block storage (e.g., EBS in AWS, Persistent Disk in GCP, and Block Blob in Azure) cannot move from one zone to another. That means that we have to have a worker node in each zone so that there is (most likely) always a place for it in the same zone as the storage. Of course, we might not use block storage in which case this argument is unfounded.

Maximum number of nodes

How about the maximum number of worker nodes? Well, that differs from one use case to another. You do not have to stick with the same maximum for all eternity. It can change over time.

As a rule of thumb, I'd recommend having a maximum double from the actual number of nodes. However, don't take that rule seriously. It truly depends on the size of your cluster. If you have only three worker nodes, your maximum size might be nine (three times bigger). On the other hand, if you have hundreds or even thousands of nodes, it wouldn't make sense to double that number as the maximum. That would be too much. Just make sure that the maximum number of nodes reflects the potential increase in demand.

In any case, I'm sure that you'll figure out what should be your minimum and your maximum number of worker nodes. If you make a mistake, you can correct it later. What matters more is how to define those thresholds.

Defining thresholds

Luckily, setting up min and max values is easy in EKS, GKE, and AKS. For **EKS**, if you're using `eksctl` to create the cluster, all we have to do is add `--nodes-min` and `--nodes-max` arguments to the `eksctl create cluster` command. **GKE** follows a similar logic with `--min-nodes` and `--max-nodes` arguments of the `gcloud container clusters create` command. If one of the two is your preference, you already used those arguments if you followed the Gists. Even if you forget to specify them, you can always modify Autoscaling Groups (AWS) or Instance Groups (GCP) since that's where the limits are actually applied.

Azure takes a bit different approach. We define its limits directly in the `cluster-autoscaler` Deployment, and we can change them just by applying a new definition.

Q

Kubernetes recommends how many zones with master nodes to maintain quorum?

COMPLETED 0%

1 of 1

In the next lesson, we will compare the differences between `Cluster Autoscaler` in GKE, EKS, and AKS.