EKS Cluster Autoscaling

* What is Cluster Autoscaler
* When does Cluster Autoscaler change the size of a cluster
* How does Horizontal Pod Autoscaler work with Cluster Autoscaler?
* How does scale-up work?
* How does scale-down work?
* Cluster Autoscaler on AWS
* Auto-Discovery Setup

**What is Cluster Autoscaler?**

Cluster Autoscaler is a standalone program that adjusts the size of a Kubernetes cluster to meet the current needs.

**When does Cluster Autoscaler change the size of a cluster?**

Cluster Autoscaler increases the size of the cluster when:

* there are pods that failed to schedule on any of the current nodes due to insufficient resources.
* adding a node similar to the nodes currently present in the cluster would help.

Cluster Autoscaler decreases the size of the cluster when some nodes are consistently unneeded for a significant amount of time. A node is unneeded when it has low utilization and all of its important pods can be moved elsewhere.

### How does Horizontal Pod Autoscaler work with Cluster Autoscaler?

Horizontal Pod Autoscaler changes the deployment's or replicaset's number of replicas based on the current CPU load. If the load increases HPA will create new replicas for which there may or may not be enough space in the cluster. If there are not enough resources then CA will try to bring up some nodes so that the HPA-created pods have a place to run. If the load decreases, HPA will stop some of the replicas. As a result, some nodes may start to be underutilized or completely empty and then CA will delete such unneeded nodes.

**How does scale-up work?**

Scale-up creates a watch on the API server looking for all pods. It checks for any unschedulable pods every 10 seconds (configurable by --scan-interval flag). A pod is unschedulable when the Kubernetes scheduler is unable to find a node that can accommodate the pod. For example, a pod can request more CPU that is available on any of the cluster nodes. Unschedulable pods are recognized by their PodCondition. Whenever a Kubernetes scheduler fails to find a place to run a pod, it sets "schedulable" PodCondition to false and reason to "unschedulable". If there are any items in the unschedulable pods list, Cluster Autoscaler tries to find a new place to run them.

It is assumed that the underlying cluster is run on top of some kind of node groups. Inside a node group, all machines have identical capacity and have the same set of assigned labels. Thus, increasing a size of a node group will create a new machine that will be similar to these already in the cluster - they will just not have any user-created pods running (but will have all pods run from the node manifest and daemon sets.)

It may take some time before the created nodes appear in Kubernetes. It almost entirely depends on the cloud provider and the speed of node provisioning. Cluster Autoscaler expects requested nodes to appear within 15 minutes (configured by --max-node-provision-time flag.)

**How does scale-down work?**

Every 10 seconds (configurable by --scan-interval flag), if no scale-up is needed, Cluster Autoscaler checks which nodes are unneeded. A node is considered for removal when **all** below conditions hold:

* The sum of cpu and memory requests of all pods running on this node is smaller than 50% of the node's allocatable. Utilization threshold can be configured using --scale-down-utilization-threshold flag.
* All pods running on the node (except these that run on all nodes by default, like manifest-run pods or pods created by daemonsets) can be moved to other nodes. While checking this condition, the new locations of all movable pods are memorized. With that, Cluster Autoscaler knows where each pod can be moved, and which nodes depend on which other nodes in terms of pod migration. Of course, it may happen that eventually the scheduler will place the pods somewhere else.
* It doesn't have scale-down disabled annotation

If a node is unneeded for more than 10 minutes, it will be deleted. Cluster Autoscaler deletes one non-empty node at a time to reduce the risk of creating new unschedulable pods. The next node may possibly be deleted just after the first one, if it was also unneeded for more than 10 min and didn't rely on the same nodes in simulation (see below example scenario), but not together. Empty nodes, on the other hand, can be deleted in bulk, up to 10 nodes at a time (configurable by --max-empty-bulk-delete flag.)

What happens when a non-empty node is deleted? As mentioned above, all pods should be migrated elsewhere. Cluster Autoscaler does this by evicting them and tainting the node, so they aren't scheduled there again.

Example scenario:

Nodes A, B, C, X, Y. A, B, C are below utilization threshold. In simulation, pods from A fit on X, pods from B fit on X, and pods from C fit on Y.

Node A was deleted. OK, but what about B and C, which were also eligible for deletion? Well, it depends.

Pods from B may no longer fit on X after pods from A were moved there. Cluster Autoscaler has to find place for them somewhere else, and it is not sure that if A had been deleted much earlier than B, there would always have been a place for them. So the condition of having been unneeded for 10 min may not be true for B anymore.

But for node C, it's still true as long as nothing happened to Y. So C can be deleted immediately after A, but B may not.

Cluster Autoscaler does all of this accounting based on the simulations and memorized new pod location. They may not always be precise (pods can be scheduled elsewhere in the end), but it seems to be a good heuristic so far.

# Cluster Autoscaler on AWS

The cluster autoscaler on AWS scales worker nodes within any specified autoscaling group. It will run as a Deployment in your cluster.

## Permissions

The worker running the cluster autoscaler will need access to certain resources and actions.

A minimum IAM policy would look like:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"autoscaling:DescribeAutoScalingGroups",

"autoscaling:DescribeAutoScalingInstances",

"autoscaling:SetDesiredCapacity",

"autoscaling:TerminateInstanceInAutoScalingGroup"

],

"Resource": "\*"

}

]

}

If you'd like to auto-discover node groups by specifying the --node-group-auto-discovery flag, a DescribeTagspermission is also required:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"autoscaling:DescribeAutoScalingGroups",

"autoscaling:DescribeAutoScalingInstances",

"autoscaling:DescribeTags",

"autoscaling:SetDesiredCapacity",

"autoscaling:TerminateInstanceInAutoScalingGroup"

],

"Resource": "\*"

}

]

}

**Auto-Discovery Setup**

To run a cluster-autoscaler which auto-discovers ASGs with nodes use the --node-group-auto-discovery flag. For example, --node-group-auto-discovery=asg:tag=k8s.io/cluster-autoscaler/enabled,k8s.io/cluster-autoscaler/<YOUR CLUSTER NAME> will find the ASGs where those tag keys *exist*. It does not matter what value the tags have.

Note that:

* It is recommended to use a second tag like k8s.io/cluster-autoscaler/<YOUR CLUSTER NAME> when k8s.io/cluster-autoscaler/enabled is used across many clusters to prevent ASGs from different clusters recognized as the node groups
* There are no --nodes flags passed to cluster-autoscaler because the node groups are automatically discovered by tags
* No min/max values are provided when using Auto-Discovery, cluster-autoscaler will respect the current min and max values of the ASG being targeted, and it will adjust only the "desired" value.