Set up Cluster Autoscaler

This lesson will focus on how to set up Cluster Autoscaler in GKE, EKS and AKS.

WE'LL COVER THE FOLLOWING

Installation of `Cluster Autoscaler`

We might need to install `Cluster Autoscaler` before we start using it. I said that we *might*, instead of saying that we *have to* because some Kubernetes flavors do come with `Cluster Autoscaler` baked in, while others don't. We'll go through each of the "big three" managed Kubernetes clusters. You might choose to explore all three of them, or to jump to the one you prefer. As a learning experience, I believe that it is beneficial to experience running Kubernetes in all three providers. Nevertheless, that might not be your view and you might prefer using only one. The choice is yours.

Setting up `Cluster Autoscaler` in GKE

This will be the shortest section ever written. There's nothing to do in GKE if you specified the `--enable-autoscaling` argument when creating the cluster. It already comes with `Cluster Autoscaler` pre-configured and ready.

Setting up `Cluster Autoscaler` in EKS

Unlike GKE, EKS does not come with `Cluster Autoscaler`. We'll have to configure it ourselves. We'll need to add a few tags to the Autoscaling Group

configure it ourselves. We'll need to add a few tags to the Autoscaling Group

dedicated to worker nodes, to put additional permissions to the Role we're using, and to install `Cluster Autoscaler`.

Let's get going.

```
export NAME=devops25
```

We stored the name of the cluster in the environment variable `NAME`.

Adding permissions

We'll add a few additional permissions to the role created through eksctl. We do not know the name of the role, but we do know the pattern used to create it. Therefore, we'll retrieve the name of the role, before we add a new policy to it.

```
IAM_ROLE=$(aws iam list-roles \
    | jq -r ".Roles[] \
    | select(.RoleName \
    | startswith(\"eksctl-$NAME-nodegroup\")) \
    .RoleName")

echo $IAM_ROLE
```

The **output** of the latter command should be similar to the one that follows.

```
eksctl-devops25-nodegroup-0-NodeInstanceRole-UU6CKXYESUES
```

We listed all the roles, and we used `jq` to filter the output so that only the one with the name that starts with `eksctl-$NAME-nodegroup-ng-143b-NodeInstanceRole` is returned. Once we filtered the roles, we retrieved the `RoleName` and stored it in the environment variable `IAM_ROLE`.

Adding `JSON`

Next, we need `JSON` that describes the new policy. I already prepared one, so let's take a quick look at it.

```
cat scaling/eks-autoscaling-policy.json
```

The **output** is as follows.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeAutoScalingInstances",
        "autoscaling:DescribeLaunchConfigurations",
        "autoscaling:DescribeTags",
        "autoscaling:SetDesiredCapacity",
        "autoscaling:TerminateInstanceInAutoScalingGroup"
      ],
      "Resource": "*"
    }
  ]
}
```

If you're familiar with AWS (I hope you are), that policy should be straightforward. It allows a few additional actions related to `autoscaling`.

Finally, we can `put` the new policy to the role.

```
aws iam put-role-policy \
    --role-name $IAM_ROLE \
    --policy-name $NAME-AutoScaling \
    --policy-document file://scaling/eks-autoscaling-policy.json
```

Now that we added the required tags to the Autoscaling Group and that we created the additional permissions that will allow Kubernetes to interact with the group, we can install **Cluster Autoscaler Helm Chart**.

```
helm install aws-cluster-autoscaler \
    stable/cluster-autoscaler \
    --namespace kube-system \
    --set autoDiscovery.clusterName=$NAME \
    --set awsRegion=$AWS_DEFAULT_REGION \
    --set sslCertPath=/etc/kubernetes/pki/ca.crt \
    --set rbac.create=true

kubectl -n kube-system \
    rollout status \
```

```
    deployment aws-cluster-autoscaler
```

Once the Deployment is rolled out, the autoscaler should be fully operational.

Setting up `Cluster Autoscaler` in AKS

At the time of this writing, `Cluster Autoscaler` does not work in AKS. At least, not always. It is still in the beta stage, and I cannot recommend it just yet. Hopefully, it will be fully operational and stable soon. When that happens, I will update this chapter with AKS-specific instructions. If you feel adventurous or you are committed to Azure, please follow the instructions from the Cluster Autoscaler on Azure Kubernetes Service (AKS) - Preview article. If it works, you should be able to follow the rest of this chapter.

Q **Cluster Autoscaler** comes pre-configured and ready in ____?

COMPLETED 0%

1 of 1

In the next lesson, we will see how to scale up the cluster.