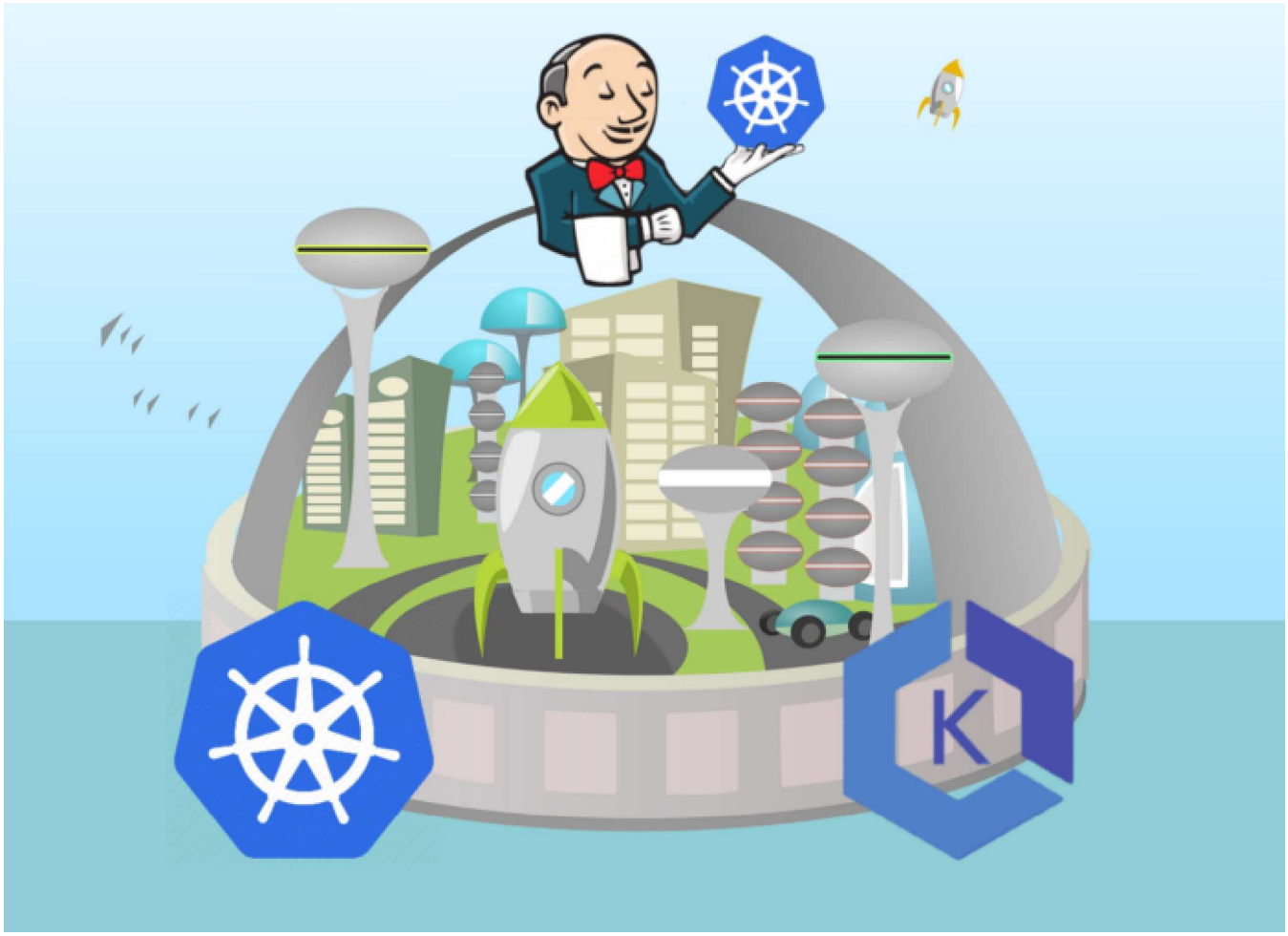
Building the CI/CD of the Future, Adding the Cluster Autoscaler

Igor Zhivilo
Aug 13, 2020 · 5 min read ★



In this tutorial, I will share my experience as a DevOps engineer at Cloudify.co, this is the **third post** of the tutorial in which I will describe how to add Cluster Autoscaler to the EKS cluster we created in the previous post.

Building the CI/CD of the Future published posts:

- Introduction

- Creating the VPC for EKS cluster
- Creating the EKS cluster
- Adding the Cluster Autoscaler
- Add Ingress Nginx and Cert-Manager
- Install and configure Jenkins

Let's start.

What is Cluster Autoscaler?

> *Cluster Autoscaler is a tool that automatically adjusts the size of the Kubernetes cluster when one of the following conditions is true:*
>
> *there are pods that failed to run in the cluster due to insufficient resources,*
>
> *there are nodes in the cluster that have been underutilized for an extended period of time and their pods can be placed on other existing nodes.*

https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler

Cluster Autoscaler adjusts the number of nodes for the 'ng-spot' node group according to load on the cluster by changing the desired capacity of an AWS Autoscaling Group. To do that Cluster Autoscaler must have certain permissions/policies for this node group, so let's add them.

IAM Policy for Cluster Autoscaler

The Cluster Autoscaler requires the following IAM permissions to change the desired capacity of the autoscaling group.

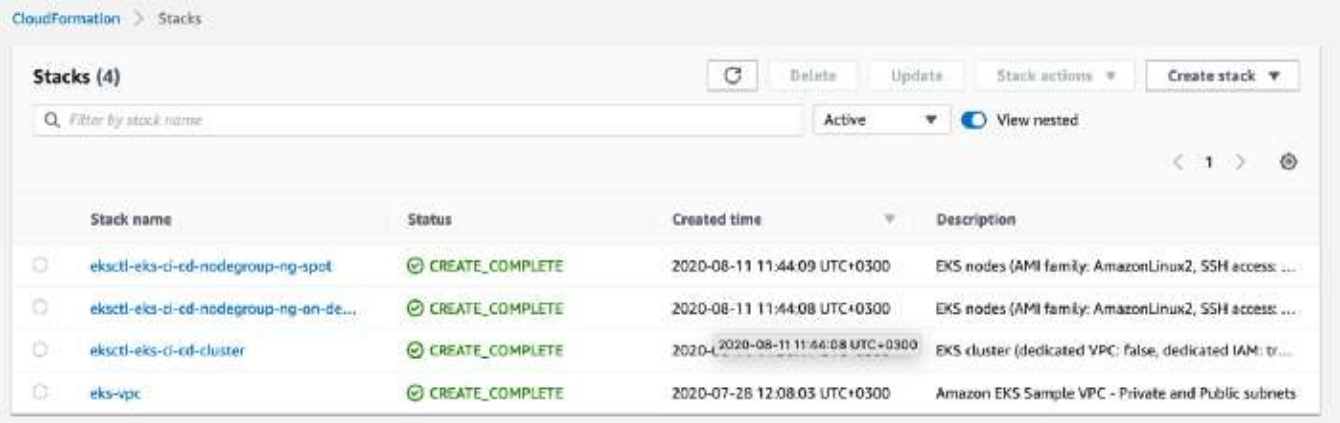
```
{
 "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
              "autoscaling:DescribeAutoScalingGroups",
              "autoscaling:DescribeAutoScalingInstances",
              "autoscaling:DescribeTags",
              "autoscaling:DescribeLaunchConfigurations",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

```
                "autoscaling:TerminateInstanceInAutoScalingGroup",
                "ec2:DescribeLaunchTemplateVersions"
            ],
            "Resource": "*"
        }
    ]
}
```

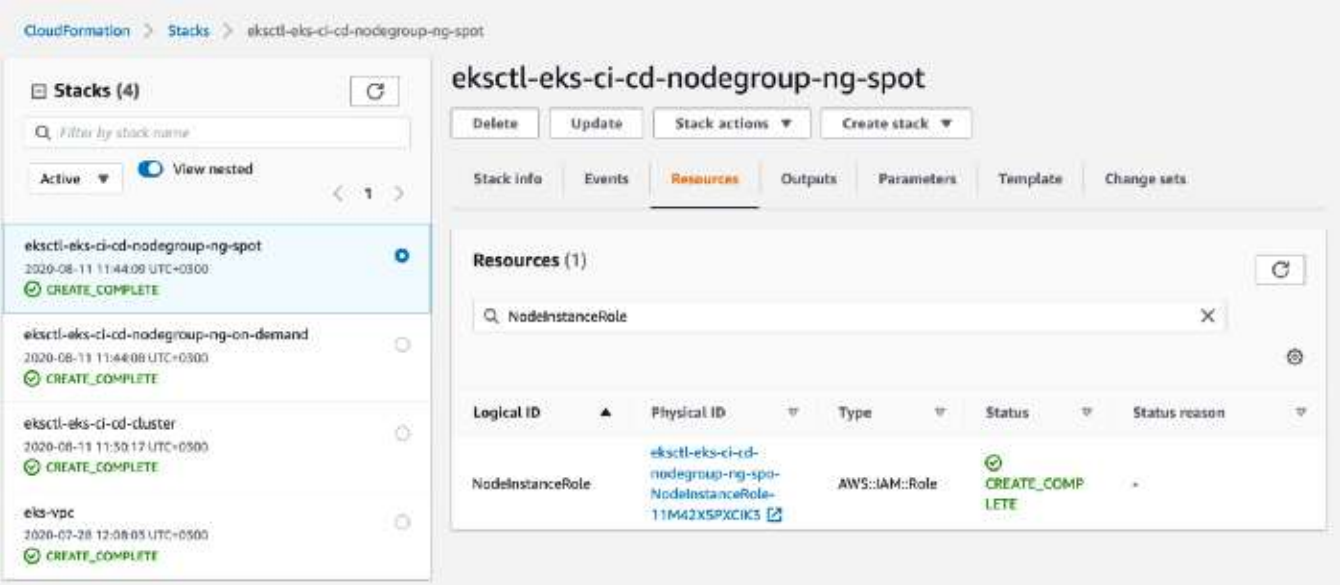
Save it as ‘eks-ca-asg-policy.json’

Let’s apply the policy above, but first, we need to find ‘ng-spot’ node group id.

In AWS account go to Services -> CloudFormation -> Stacks



Go to ‘eksctl-eks-ci-cd-nodegroup-ng-spot’ stack -> Resources



The Physical ID of ‘NodeInstanceRole’ is what we need: ‘eksctl-eks-ci-cd-nodegroup-ng-spo-NodeInstanceRole-11M42X5PXCIK3’

Applying the policy

```
$ aws iam put-role-policy --role-name eksctl-eks-ci-cd-nodegroup-ng-spo-NodeInstanceRole-11M42X5PXCIK3 --policy-name ASG-Policy-For-Worker --policy-document file://./eks-ca-asg-policy.json
```

Checking the policy applied successfully

```
$ aws iam get-role-policy --role-name eksctl-eks-ci-cd-nodegroup-ng-spo-NodeInstanceRole-11M42X5PXCIK3 --policy-name ASG-Policy-For-Worker
```

Please apply this policy for the ‘ng-static’ group also (although it not managed by CA), otherwise, you will see a lot of errors in Cluster Autoscaler’s logs or CA pod will have an error status.

Deploy Cluster Autoscaler

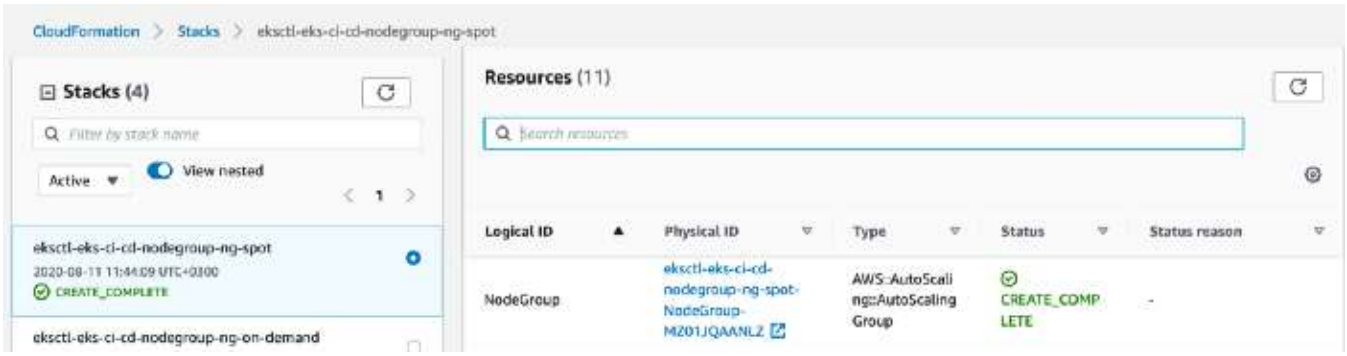
To deploy Cluster Autoscaler I will use cluster-autoscaler-one-asg.yaml from here: https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/aws/examples/cluster-autoscaler-one-asg.yaml

It’s the deployment of Cluster Autoscaler for one Autoscaling Group Only, exactly what we need in our case.

You can see other examples here

We need to modify this file first before apply it.

Look for **command** section, — nodes line must be changed to the real id of your 'ng-spot' node group.




```
command:
  - ./cluster-autoscaler
  - --v=4
  - --stderrthreshold=info
  - --cloud-provider=aws
  - --skip-nodes-with-local-storage=false
  - --nodes=0:10:eksctl-eks-ci-cd-nodegroup-ng-spot-NodeGroup-
MZ01JQAANLZ
```

Deploy the Cluster Autoscaler

```
kubectl apply -f cluster-autoscaler-one-asg.yaml
```

```
$ kubectl apply -f cluster-autoscaler-one-asg.yaml
serviceaccount/cluster-autoscaler created
clusterrole.rbac.authorization.k8s.io/cluster-autoscaler created
role.rbac.authorization.k8s.io/cluster-autoscaler created
clusterrolebinding.rbac.authorization.k8s.io/cluster-autoscaler created
rolebinding.rbac.authorization.k8s.io/cluster-autoscaler created
deployment.apps/cluster-autoscaler created
```

Troubleshooting logs of Cluster Autoscaler

```
kubectl get pods -n kube-system
```

```
$ kubectl get pods -n kube-system
NAME                                  READY   STATUS    RESTARTS   AGE
```

```
NAME                                  READY   STATUS    RESTARTS   AGE
aws-node-5qrbs                        1/1     Running   0          4m8s
cluster-autoscaler-688896dfd5-4n5lg   1/1     Running   1          32s
coredns-55c5fcd78f-997hb              1/1     Running   0          126m
coredns-55c5fcd78f-g8zqv              1/1     Running   0          126m
kube-proxy-8vjcs                      1/1     Running   0          4m8s
```

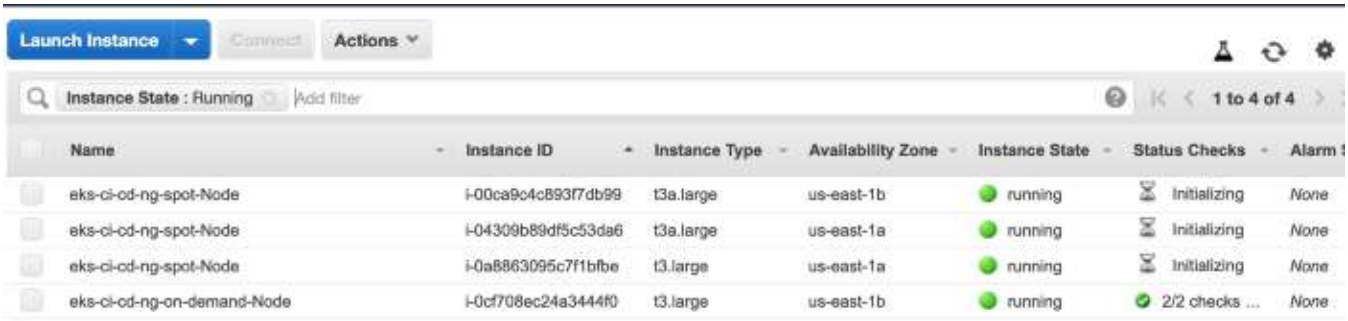
```
kubectl logs cluster-autoscaler-688896dfd5-4n5lg -n kube-system
```

Testing Scaling

To test scaling we will deploy 100 Nginx pods

```
$ kubectl run test-ca-scaling --replicas=100 --image=nginx --port=80
$ kubectl get nodes
```

```
$ kubectl get nodes
NAME                                STATUS     ROLES    AGE     VERSION
ip-192-168-141-196.ec2.internal     NotReady   <none>   13s     v1.16.13-eks-2ba888
ip-192-168-191-113.ec2.internal     NotReady   <none>   16s     v1.16.13-eks-2ba888
ip-192-168-212-234.ec2.internal     Ready      <none>   3h15m   v1.16.13-eks-2ba888
ip-192-168-245-7.ec2.internal       NotReady   <none>   13s     v1.16.13-eks-2ba888
```



In a couple of minutes, you will see new spot instances provisioned.

Let's delete those pods now

```
kubectl delete deployment test-ca-scaling
```

It will take something like 10 minutes for Cluster Autoscaler to terminate unneeded nodes.

Run Windows workflows on EKS

To use windows workflows in our cluster we can add an additional node group to eks-cluster config:

```
- name: ng-spot-windows
  amiFamily: WindowsServer2019FullContainer
  desiredCapacity: 0
  minSize: 0
  maxSize: 10
  privateNetworking: true
  instancesDistribution:
    instanceTypes: ["t2.large", "t3.large", "m3.large"]
    onDemandBaseCapacity: 0
    onDemandPercentageAboveBaseCapacity: 0
    spotInstancePools: 3
  tags:
    k8s.io/cluster-autoscaler/node-template/label/instance-type:
spot-windows
  availabilityZones: ["eu-west-1a", "eu-west-1b", "eu-west-1c"]
  labels:
    instance-type: spot-windows
  iam:
    withAddonPolicies:
      autoScaler: true
```

This node group also will run on spot instances and will use EC2 instances with AMI: WindowsServer2019, you can run pods(containers) with windows based images only on windows worker nodes.

You need to be careful and use the node selector to properly put your pods on correct instances.

If you want to understand how to run properly windows workflows on your EKS cluster, please read my article about it: https://levelup.gitconnected.com/running-workflows-on-windows-with-jenkins-pipeline-and-kubernetes-52752a89a0e7

In the next post, I will explain how to install and configure ‘Ingress Nginx’ and ‘certificate manager’ to your cluster.

Conclusion

In the third post of **Building the CI/CD of the Future** tutorial, I explained what’s Cluster Autoscaler and why it’s so powerful. Showed how to apply policies to give permission to CA scale up/down and then we deployed demo app to test CA at work. In the end, I added explanation of how to run windows workflows on your EKS cluster.

Thank you for reading, I hope you enjoyed, see you in next post.

If you want to be notified when the next post of this tutorial is published, please follow me here on medium and on Twitter (@warolv).

My personal blog in which I will duplicate this tutorial: http://igorzhivilo.com, I will save all configuration created in this tutorial in my Github (warolv).

References

https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler

https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/aws/examples/cluster-autoscaler-one-asg.yaml

https://levelup.gitconnected.com/running-workflows-on-windows-with-jenkins-pipeline-and-kubernetes-52752a89a0e7