Launching an EKS Cluster

eks-aws: Description

Description:
Launch an EKS cluster with three worker nodes in the useast-1 region. Then, create
a Deployment, and a Pod running container instances using the public nginx image.
Then, expose your application to the internet by adding a LoadBalancer service.

This repo will describe the use of AWS command line interface and console, eksctl and kubectl to launch and EKS cluster, provision a Kubernetes deploymnet-
and pod running instances of nginx, create a LoadBalancer service to expose your
application over the internet.
My Objective:
---

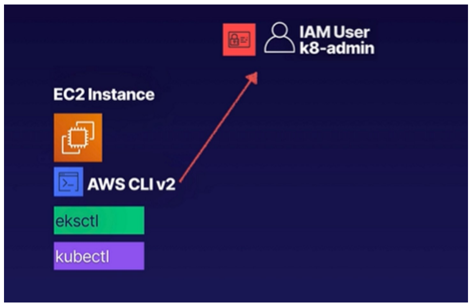
step 1: Create an IAM User with Admin Rights:

1. Create sn IAM user with programmatic access and admintrator privileges.
   IAM User: k8-admin
   Aws console: services -> IAM - Users -> Add User: k8-admin enable programmatic access
   Next -> attach existing policies directly -> (AdministratorAccess) -needed for cloudformation, EC2 instances, VPC, security -> Next -> Next -> Create User

   Access key: AKIAS7DW6XFLRYTY3ABL
   Secret Access Key: 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

2. Take note of the access key and secret access key you'll need them.

step 2: Launch an EC2 instance and configure the command line tools.



1. Create and EC2 instance in us-east-1 region.
   AWS Console: EC2 -> Launch Instance -> Amazon Linux 2 AMI -> T2 Micro -> Auto Assign Public IP set to enable -> Next -> Next -> Review and Launch -> create a new key pair -> my-nvkp - Download -> Launch Instance

2. Upgrade the AWS CLI on your EC2 instance to CLI v2x or later.
   # aws --version
   # curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
   # unzip awscliv2.zip

   - To upgrade

   # sudo ./aws/install --bin-dir /usr/bin --install-dir /usr/bin/aws-cli --update
   # aws --version

3. configure the AWS CLI using the credentials you created.
   # aws configure

   AWS Access Key ID [None]: AKIAS7DW6XFLRYTY3ABL
   AWS Secret Access Key [None]: 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
   Default region name [None]: us-east-1
   Default output format [None]: json

4. Install kubectl on EC2 instan

   - Download kubectl
     # curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.16.8/2020-04-16/bin/linux/amd64/kubectl

   - Apply execute permissions to kubectl
     # chmod +x ./kubectl

   - Copy binary to a directory in my path

     # mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin

- Ensure kubectl is installed:

```
# kubectl version --short --client
```

- Download eksctl:

```
# curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp

#  sudo mv /tmp/eksctl /usr/bin
```

step 3: Provision an EKS cluster

1. Use EKS to provision an EKS cluster with three worker nodes in us-east-1

```
#  eksctl create cluster --name dev --version 1.16 --region us-east-1 --nodegroup-name standard-workers --node-type t3.micro --nodes 3 --nodes-min 1 --nodes-max 4 --managed
```

Note: It will take 10–15 minutes since it's provisioning the control plane and worker nodes, attaching the worker nodes to the control plane, and creating the VPC, security group, and Auto Scaling group.

1. In the AWS Management Console, navigate to CloudFormation and take a look at what’s going on there.
2. Select the eksctl-dev-cluster stack (this is our control plane).
3. Click **Events**, so you can see all the resources that are being created.
4. We should then see another new stack being created — this one is our node group.
5. Once both stacks are complete, navigate to **Elastic Kubernetes Service** > **Clusters**.
6. Click the listed cluster.
7. Click the **Compute** tab, and then click the listed node group. There, we'll see the Kubernetes version, instance type, status, etc.
8. Click **dev** in the breadcrumb navigation link at the top of the screen.
9. Click the **Networking** tab, where we'll see the VPC, subnets, etc.
10. Click the **Logging** tab, where we'll see the control plane logging info.
    - The control plane is abstracted — we can only interact with it using the command line utilities or the console. It’s not an EC2 instance we can log into and start running Linux commands on.
11. Navigate to **EC2** > **Instances**, where you should see the instances have been launched.
12. Close out of the existing CLI window, if you still have it open.
13. Select the original t2.micro instance, and click **Connect** at the top of the window.
14. In the *Connect to your instance* dialog, select **EC2 Instance Connect (browser-based SSH connection)**.
15. Click **Connect**.
16. In the CLI, check the cluster:
    `eksctl get cluster`
17. Enable it to connect to our cluster:
    `aws eks update-kubeconfig --name dev --region us-east-1`
    Create a Deployment on Your EKS Cluster
18. Install Git:
    `sudo yum install -y git`
19. Download the course files:
    `git clone https://github.com/ACloudGuru-Resources/Course_EKS-Basics`
20. Change directory:
    `cd Course_EKS-Basics`
21. Take a look at the deployment file:
    `cat nginx-deployment.yaml`
22. Take a look at the service file:
    `cat nginx-svc.yaml`
23. Create the service:
    `kubectl apply -f ./nginx-svc.yaml`
24. Check its status:
    `kubectl get service`
    Copy the external IP of the load balancer, and paste it into a text file, as we'll need it in a minute.
25. Create the deployment:
    `kubectl apply -f ./nginx-deployment.yaml`
26. Check its status:
    `kubectl get deployment`
27. View the pods:
    `kubectl get pod`
28. View the ReplicaSets:
    `kubectl get rs`
29. View the nodes:
    `kubectl get node`
30. Access the application using the load balancer, replacing <LOAD_BALANCER_EXTERNAL_IP> with the IP you copied earlier:
    `curl "<LOAD_BALANCER_EXTERNAL_IP>"`
    The output should be the HTML for a default Nginx web page.
31. In a new browser tab, navigate to the same IP, where we should then see the same Nginx web page.
    Test the High Availability Features of Your EKS Cluster
32. In the AWS console, on the EC2 instances page, select the three t3.micro instances.
33. Click **Actions** > **Instance State** > **Stop**.
34. In the dialog, click **Yes, Stop**.
35. After a few minutes, we should see EKS launching new instances to keep our service running.

36. In the CLI, check the status of our nodes:
    `kubectl get node`
    All the nodes should be down (i.e., display a NotReady status).
37. Check the pods:
    `kubectl get pod`
    We'll see a few different statuses — Terminating, Running, and Pending — because, as the instances shut down, EKS is trying to restart the pods.
38. Check the nodes again:
    `kubectl get node`
    We should see a new node, which we can identify by its age.
39. Wait a few minutes, and then check the nodes again:
    `kubectl get node`
    We should have one in a Ready state.
40. Check the pods again:
    `kubectl get pod`
    We should see a couple pods are now running as well.
41. Check the service status:
    `kubectl get service`
42. Copy the external IP listed in the output.
43. Access the application using the load balancer, replacing <LOAD_BALANCER_EXTERNAL_IP> with the IP you just copied:
    `curl "<LOAD_BALANCER_EXTERNAL_IP>"`
    We should see the Nginx web page HTML again. (If you don't, wait a few more minutes.)
44. In a new browser tab, navigate to the same IP, where we should again see the Nginx web page.
45. In the CLI, delete everything:
    `eksctl delete cluster dev`