

Launching an EKS Cluster

Introduction

Elastic Kubernetes Service (EKS) is a fully managed Kubernetes service from AWS. In this lab, you will work with the AWS command line interface and console, using command line utilities like `eksctl` and `kubectl` to launch an EKS cluster, provision a Kubernetes deployment and pod running instances of `nginx`, and create a LoadBalancer service to expose your application over the internet.

Note that `us-east-1` can experience capacity issues in certain Availability Zones. Since the AZ numbering (lettering) system differs between AWS accounts we cannot exclude that AZ from the lab steps. If you *do* experience an *UnsupportedAvailabilityZoneException* error regarding capacity in a particular zone, you can add the `--zones` switch to `eksctl create cluster` and specify three AZs which do not include the under-capacity zone. For example,

```
eksctl create cluster --name dev --region us-east-1 --zones=us-east-1a,us-east-1b,us-east-1d --nodegroup-name standard-workers --node-type t3.medium --nodes 3 --nodes-min 1 --nodes-max 4 --managed
```

Solution

Log in to the live AWS environment using the credentials provided. Make sure you're in the N. Virginia (`us-east-1`) region throughout the lab.

1. Navigate to **IAM > Users**.
2. Click **Add users**.
3. In the **User name** field, enter `k8-admin`.
4. Click **Next**.
5. Select **Attach policies directly**.
6. Select **AdministratorAccess**.
7. Click **Next**.
8. Click **Create user**.
9. Select the newly created user **k8-admin**.
10. Select the **Security credentials** tab.
11. Scroll down to **Access keys** and select **Create access key**.
12. Select **Command Line Interface (CLI)** and checkmark the acknowledgment at the bottom of the page.
13. Click **Next**.
14. Click **Create access key**.

15. Either copy both the access key and the secret access key and paste them into a local text file, or click **Download .csv file**. We will use the credentials when setting up the AWS CLI.
16. Click **Done**.
17. Navigate to **EC2 > Instances**.
18. Click **Launch Instance**.
19. On the AML page, select the Amazon Linux 2 AML.
20. Leave *t2.micro* selected, and click **Next: Configure Instance Details**.
21. On the *Configure Instance Details* page:
 - *Network*: Leave default
 - *Subnet*: Leave default
 - *Auto-assign Public IP*: **Enable**
22. Click **Next: Add Storage > Next: Add Tags > Next: Configure Security Group**.
23. Click **Review and Launch**, and then **Launch**.
24. In the key pair dialog, select **Create a new key pair**.
25. Give it a *Key pair name* of "mynvkp".
26. Click **Download Key Pair**, and then **Launch Instances**.
27. Click **View Instances**, and give it a few minutes to enter the *running* state.
28. Once the instance is fully created, check the checkbox next to it and click **Connect** at the top of the window.
29. In the *Connect to your instance* dialog, select **EC2 Instance Connect (browser-based SSH connection)**.
30. Click **Connect**.
31. In the command line window, check the AWS CLI version:
 - a. `aws --version`
32. It should be an older version.
33. Download v2:
34. `curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"`
35. Unzip the file:
36. `unzip awscliv2.zip`
37. See where the current AWS CLI is installed:
38. `which aws`
39. It should be `/usr/bin/aws`.
40. Update it:
41. `sudo ./aws/install --bin-dir /usr/bin --install-dir /usr/bin/aws-cli --update`
42. Check the version of AWS CLI:
43. `aws --version`
44. It should now be updated.
45. Configure the CLI:
46. `aws configure`
47. For AWS Access Key ID, paste in the access key ID you copied earlier.
48. For AWS Secret Access Key, paste in the secret access key you copied earlier.

49. For Default region name, enter us-east-1.
50. For Default output format, enter json.
51. Download kubectl:
52. `curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.16.8/2020-04-16/bin/linux/amd64/kubectl`
53. Apply execute permissions to the binary:
54. `chmod +x ./kubectl`
55. Copy the binary to a directory in your path:
56. `mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin`
57. Ensure kubectl is installed:
58. `kubectl version --short --client`
59. Download eksctl:
60. `curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp`
61. Move the extracted binary to /usr/bin:
62. `sudo mv /tmp/eksctl /usr/bin`
63. Get the version of eksctl:
64. `eksctl version`
65. See the options with eksctl:
66. `eksctl help`

Provision an EKS Cluster

67. Provision an EKS cluster with three worker nodes in us-east-1:

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

If your EKS resources can't be deployed due to AWS capacity issues, delete your eksctl-dev-cluster CloudFormation stack and retry the command using the --zones parameter and suggested availability zones from the CREATE_FAILED message:

```
AWS::EKS::Cluster/ControlPlane: CREATE_FAILED – "Resource handler returned message:
\'Cannot create cluster \'dev\' because us-east-1e, the targeted availability zone, does not
currently have sufficient capacity to support the cluster. Retry and choose from these
availability zones: us-east-1a, us-east-1b, us-east-1c, us-east-1d, us-east-1f (Service: Eks,
Status Code: 400, Request ID: 21e7e4aa-17a5-4c79-ag11-bf86c4e93373)\'' (RequestToken:
18b731b0-g2a1-a779-9a69-f61e90b97ee1, HandlerErrorCode: InvalidRequest)"
```

68. In this example, the --zones parameter was added using the us-east-1a,us-east-1b,us-east-1c,us-east-1d,us-east-1f AZs from the message above:

```
eksctl create cluster --name dev --region us-east-1 --zones us-east-1a,us-east-1b,us-east-1c,us-east-1d,us-east-1f --nodegroup-name standard-workers --node-type t3.medium --nodes 3 --nodes-min 1 --nodes-max 4 --managed
```

69. 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.
70. In the AWS Management Console, navigate to CloudFormation and take a look at what's going on there.
71. Select the eksctl-dev-cluster stack (this is our control plane).
72. Click **Events**, so you can see all the resources that are being created.
73. We should then see another new stack being created — this one is our node group.
74. Once both stacks are complete, navigate to **Elastic Kubernetes Service > Clusters**.
75. Click the listed cluster.
76. If you see a Your current user or role does not have access to Kubernetes objects on this EKS cluster message just ignore it, as it won't impact the next steps of the activity.
77. Click the **Compute** tab (under **Configuration**), and then click the listed node group. There, we'll see the Kubernetes version, instance type, status, etc.
78. Click **dev** in the breadcrumb navigation link at the top of the screen.
79. Click the **Networking** tab (under **Configuration**), where we'll see the VPC, subnets, etc.
80. Click the **Logging** tab (under **Configuration**), 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.
81. Navigate to **EC2 > Instances**, where you should see the instances have been launched.
82. Close out of the existing CLI window, if you still have it open.
83. Select the original t2.micro instance, and click **Connect** at the top of the window.
84. In the *Connect to your instance* dialog, select **EC2 Instance Connect (browser-based SSH connection)**.
85. Click **Connect**.
86. In the CLI, check the cluster:
87. `eksctl get cluster`
88. Enable it to connect to our cluster:
89. `aws eks update-kubeconfig --name dev --region us-east-1`

Create a Deployment on Your EKS Cluster

90. Install Git:
91. `sudo yum install -y git`
92. use the course files:

nginx-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    env: dev
spec:
  replicas: 3
  selector:
    matchLabels:
      env: dev
  template:
    metadata:
      labels:
        env: dev
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

nginx-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
  labels:
    app: nginx
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: nginx
```

93. Take a look at the deployment file:
94. cat nginx-deployment.yaml
95. Take a look at the service file:
96. cat nginx-svc.yaml
97. Create the service:
98. kubectl apply -f ./nginx-svc.yaml

99. Check its status:
100. `kubectl get service`
101. Copy the external DNS hostname of the load balancer, and paste it into a text file, as we'll need it in a minute.
102. Create the deployment:
103. `kubectl apply -f ./nginx-deployment.yaml`
104. Check its status:
105. `kubectl get deployment`
106. View the pods:
107. `kubectl get pod`
108. View the ReplicaSets:
109. `kubectl get rs`
110. View the nodes:
111. `kubectl get node`
112. Access the application using the load balancer, replacing `<LOAD_BALANCER_DNS_HOSTNAME>` with the IP you copied earlier (it might take a couple of minutes to update):
113. `curl "<LOAD_BALANCER_DNS_HOSTNAME>"`
114. The output should be the HTML for a default Nginx web page.
115. 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

116. In the AWS console, on the EC2 instances page, select the worker node instances.
117. Click **Actions > Instance State > Stop**.
118. In the dialog, click **Yes, Stop**.
119. After a few minutes, we should see EKS launching new instances to keep our service running.
120. In the CLI, check the status of our nodes:
121. `kubectl get node`
122. All the nodes should be down (i.e., display a NotReady status).
123. Check the pods:
124. `kubectl get pod`
125. We'll see a few different statuses — Terminating, Running, and Pending — because, as the instances shut down, EKS is trying to restart the pods.
126. Check the nodes again:
127. `kubectl get node`
128. We should see a new node, which we can identify by its age.
129. Wait a few minutes, and then check the nodes again:

130. `kubectl get node`
131. We should have one in a Ready state.
132. Check the pods again:
133. `kubectl get pod`
134. We should see a couple pods are now running as well.
135. Check the service status:
136. `kubectl get service`
137. Copy the external DNS Hostname listed in the output.
138. Access the application using the load balancer, replacing
 <LOAD_BALANCER_DNS_HOSTNAME> with the DNS Hostname you just copied:
139. `curl "<LOAD_BALANCER_EXTERNAL_IP>"`
140. We should see the Nginx web page HTML again. (If you don't, wait a few
 more minutes.)
141. In a new browser tab, navigate to the same IP, where we should again see
 the Nginx web page.
142. In the CLI, delete everything:
143. `eksctl delete cluster dev`