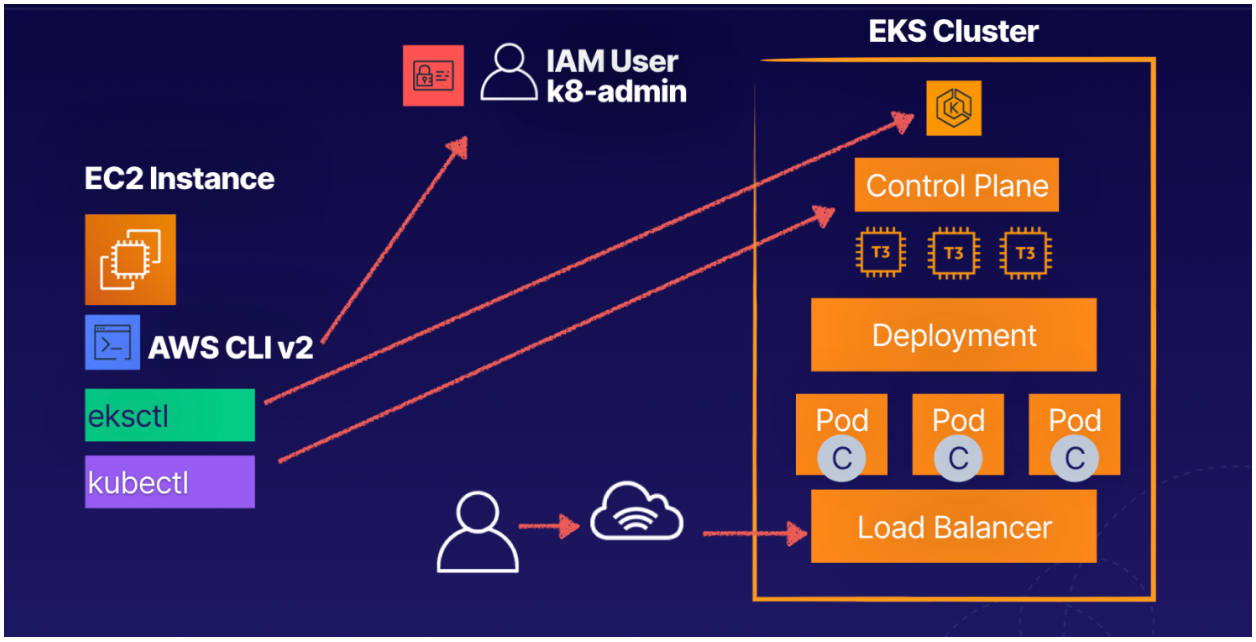
Launching an EKS Cluster

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
```

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- Create an IAM User with Admin Permissions

1. Navigate to **IAM** > **Users**.
2. Click **Add user**.
3. Set the following values:
    - User name: **k8-admin**
    - Access type: **Programmatic access**
4. Click **Next: Permissions**.
5. Select **Attach existing policies directly**.
6. Select **AdministratorAccess**.
7. Click **Next: Tags > Next: Review**.
8. Click **Create user**.
9. Copy the access key ID and secret access key, and paste them into a text file, as we'll need them in the next step.

2- Launch an EC2 Instance and Configure the Command Line Tools

1. Navigate to EC2 > Instances.
2. Click Launch Instance.

3. On the AMI page, select the Amazon Linux 2 AMI.
4. Leave t2.micro selected, and click Next: Configure Instance Details.
5. On the Configure Instance Details page:
    a. Network: Leave default
    b. Subnet: Leave default
    c. Auto-assign Public IP: Enable
6. Click Next: Add Storage > Next: Add Tags > Next: Configure Security Group.
7. Click Review and Launch, and then Launch.
8. In the key pair dialog, select Create a new key pair.
9. Give it a Key pair name of "mynvkp".
10. Click Download Key Pair, and then Launch Instances.
11. Click View Instances, and give it a few minutes to enter the running state.
12. Once the instance is fully created, check the checkbox next to it and click Connect at the top of the window.
13. In the Connect to your instance dialog, select EC2 Instance Connect (browser-based SSH connection).
14. Click Connect.
15. In the command line window, check the AWS CLI version:

    ```
    aws --version
    ```

    It should be an older version.
16. Download v2:

    ```
    curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
    ```
17. Unzip the file:

    ```
    unzip awscliv2.zip
    ```

18. See where the current AWS CLI is installed:

    ```
    which aws
    ```

    It should be /usr/bin/aws.
19. Update it:

    ```
    sudo ./aws/install --bin-dir /usr/bin --install-dir /usr/bin/aws-cli --update
    ```

20. Check the version of AWS CLI:
    ```
    aws --version
    ```
    It should now be updated.
21. Configure the CLI:
    ```
    aws configure
    ```
22. For AWS Access Key ID, paste in the access key ID you copied earlier.
23. For AWS Secret Access Key, paste in the secret access key you copied earlier.
24. For Default region name, enter **us-east-1**.
25. For Default output format, enter **json**.
26. Download kubectl:
```
 curl -o kubectl
https://amazon-eks.s3.us-west-2.amazonaws.com/1.16.8/2020-04-16/bin/linux/amd64/kubectl
```
27. Apply execute permissions to the binary:
    ```
    chmod +x ./kubectl
    ```
28. Copy the binary to a directory in your path:
    ```
    mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin
    ```
29. Ensure kubectl is installed:
    ```
    kubectl version --short --client
    ```
30. Download eksctl:
    ```
    curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
    ```
31. Move the extracted binary to /usr/bin:
    ```
    sudo mv /tmp/eksctl /usr/bin
    ```
32. Get the version of eksctl:
    ```
    eksctl version
    ```
33. See the options with eksctl:
    ```
    eksctl help
    ```

3- Provision an EKS Cluster

1. 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-a911-bf86c4e93373)\" (RequestToken: 18b731b0-92a1-a779-9a69-f61e90b97ee1, HandlerErrorCode: InvalidRequest)"

In this example, the **--zones** parameter was added using the *us-east-1a,us-east-1b,us-east-1c,us-east-1d,us-east-1*f 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
```

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.

2. In the AWS Management Console, navigate to CloudFormation and take a look at what’s going on there.

3. Select the **eksctl-dev-cluster** stack (this is our control plane).
4. Click Events, so you can see all the resources that are being created.
5. We should then see another new stack being created — this one is our node group.
6. Once both stacks are complete, navigate to Elastic Kubernetes Service > Clusters.
7. Click the listed cluster.
8. 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.

9. Click the Compute tab (under Configuration), and then click the listed node group. There, we'll see the Kubernetes version, instance type, status, etc.
10. Click dev in the breadcrumb navigation link at the top of the screen.
11. Click the Networking tab (under Configuration), where we'll see the VPC, subnets, etc.
12. 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.

13. Navigate to EC2 > Instances, where you should see the instances have been launched.
14. Close out of the existing CLI window, if you still have it open.

15. Select the original **t2.micro** instance, and click Connect at the top of the window.
16. In the Connect to your instance dialog, select EC2 Instance Connect (browser-based SSH connection).
17. Click Connect.
18. In the CLI, check the cluster:

```
eksctl get cluster
```

19. Enable it to connect to our cluster:

```
aws eks update-kubeconfig --name dev --region us-east-1
```

4- Create a Deployment on Your EKS Cluster

1. Install Git:
   ```
   sudo yum install -y git
   ```
2. Download the course files:
   ```
   git clone https://github.com/ACloudGuru-Resources/Course_EKS-Basics
   ```
3. Change directory:
   ```
   cd Course_EKS-Basics
   ```
4. Take a look at the deployment file:
   ```
   cat nginx-deployment.yaml
   ```
5. Take a look at the service file:
   ```
   cat nginx-svc.yaml
   ```
6. Create the service:
   ```
   kubectl apply -f ./nginx-svc.yaml
   ```
7. Check its status:
   ```
   kubectl get service
   ```
   Copy the external DNS hostname of the load balancer, and paste it into a text file, as we'll need it in a minute.

8. Create the deployment:
   ```
   kubectl apply -f ./nginx-deployment.yaml
   ```
9. Check its status:
   ```
   kubectl get deployment
   ```
10. View the pods:
   ```
   kubectl get pod
   ```
11. View the ReplicaSets:
   ```
   kubectl get rs
   ```
12. View the nodes:
   ```
   kubectl get node
   ```
13. 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):
   ```
   curl "<LOAD_BALANCER_DNS_HOSTNAME>"
   ```

The output should be the HTML for a default Nginx web page.

14. In a new browser tab, navigate to the same IP, where we should then see the same Nginx web page.

5- Test the High Availability Features of Your EKS Cluster

1. In the AWS console, on the EC2 instances page, select the three **t3.medium** instances.
2. Click Actions > Instance State > **Stop**.
3. In the dialog, click Yes, Stop.
4. After a few minutes, we should see EKS launching new instances to keep our service running.
5. In the CLI, check the status of our nodes:
   ```
   kubectl get node
   ```

   All the nodes should be down (i.e., display a *NotReady* status).

6. 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.

7. Check the nodes again:
   `kubectl get node`
   We should see a new node, which we can identify by its age.

8. Wait a few minutes, and then check the nodes again:
   `kubectl get node`
   We should have one in a Ready state.

9. Check the pods again:
   `kubectl get pod`
   We should see a couple pods are now running as well.

10. Check the service status:
    `kubectl get service`

11. Copy the external DNS Hostname listed in the output.

12. Access the application using the load balancer, replacing <LOAD_BALANCER_DNS_HOSTNAME> with the DNS Hostname 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.)

13. In a new browser tab, navigate to the same IP, where we should again see the Nginx web page.

14. In the CLI, delete everything:
    `eksctl delete cluster dev`