# Lab:
# Deploying the Case Study App to Amazon EKS

# Introduction

- In the final project, you will:
    - Log into AWS and create access keys for your user
    - Install the **`aws`** and **`eksctl`** CLI tools in Codespaces
    - Configure the **`aws`** CLI for your credentials and region
    - Create an EKS cluster with the **`eksctl`** CLI tool
    - Deploy your events app to the EKS cluster

# Log into AWS and Create Access Keys

- Your instructor will explain how to get logged in to AWS
  - An AWS account, username, and password will be provided to you for use in this lab
- Once logged into AWS:
  - In the search field near the top, type `iam` and click the **IAM service**
  - On the left, click **Users** and then click on your user name
  - Click the **Security credentials** tab and scroll down to **Access keys**
  - Click the **Create access key** button, select **Command line interface (CLI)** use case, check the **Confirmation** checkbox, and click **Next**
  - Provide a description of `class-keys` and click **Create access key**
  - Use the copy buttons to copy both keys and paste them in a local editor - you will need them shortly
    - These keys are sensitive and should be protected

# Install the aws CLI

- From a codespaces terminal, run the following commands to install the aws CLI:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
rm awscliv2.zip
```

- Verify the installation with the following command:

```
aws --version
```

# Install `eksctl` CLI

- The easiest way to create an EKS cluster is to use the `eksctl` CLI tool
- From a codespaces terminal, run the following commands to install the `eksctl` CLI:

```
ARCH=amd64
PLATFORM=$(uname -s)_$ARCH
curl -sLO \
"https://github.com/eksctl-io/eksctl/releases/latest/download/\
eksctl_$PLATFORM.tar.gz"

tar -xzf eksctl_$PLATFORM.tar.gz -C /tmp && rm eksctl_$PLATFORM.tar.gz
sudo install -m 0755 /tmp/eksctl /usr/local/bin && rm /tmp/eksctl
```

- Verify the installation with the following command:

```
eksctl -h
```

# Configure the `aws` CLI

- From a codespaces terminal, run the following command to configure the aws CLI:
  `aws configure`
  - Paste your access key you saved earlier and press **Enter**
  - Paste your secret access key you saved earlier and press **Enter**
  - For region, type: **`us-east-2`**
  - For output format type: **`text`**
- Verify the configuration with the following command:
  `aws ec2 describe-availability-zones`

# Creating an EKS Cluster

- `eksctl` uses the `aws` CLI configuration to communicate to AWS
  - That is why you configured the AWS CLI
- Switch to the EKS folder:

  ```
  cd /workspaces/eventsapp/eks
  ls
  ```

  - This folder contains a file **`cluster.yaml`** which specifies the settings when creating an EKS cluster
- Edit the **`cluster.yaml`** and change **`userX`** in the cluster name to your AWS user number
  - For example: `user1-cluster`, `user2-cluster`, `user3-cluster`, etc,

## Creating an EKS Cluster (continued)

- Before continuing, be sure you changed **`userX`** to your AWS user number in the `cluster.yaml` file
- Run the following command to create the cluster

```
eksctl create cluster -f cluster.yaml
```

- It will take approximately 15-20 minutes to create the cluster
- While you are waiting, feel free to investigate the `cluster.yaml` file
  - Note the `aws-ebs-csi-driver` add on
  - This driver enables your cluster to dynamically provision Amazon EBS volumes whenever a PVC is created
- Wait for the command to complete

# Creating an EKS Cluster (continued)

- The kubectl CLI uses a config file in the `<userhome>/.kube` folder
  - The eksctl command automatically updated that config file with information about the EKS cluster you created
- When the cluster is finished creating, verify the cluster is created and has 2 nodes:

```
kubectl get nodes
```

# Create a Default StorageClass

- By default, the installed CSI driver doesn't automatically set up a **StorageClass**
  - Creating a default StorageClass simplifies the process of creating PVCs so you do not need to specify the storageClassName
  - The `eks` folder contains a `storageclass.yaml` file which creates a storage class with the gp3 Amazon EBS volume type
    - Feel free to investigate the file
- Run the following command to create the StorageClass

```
kubectl apply -f storageclass.yaml
```

# Install the Database with Helm

- Run the following command to install the database:

```
helm install database-server oci://registry-1.docker.io/bitnamicharts/mariadb
```

- Run the following to view the installation:

  ```
  kubectl get pods
  kubectl get statefulsets
  kubectl get service
  ```

- Wait for the pod to be ready
- These are the exact same commands that were run earlier to install the database on Minikube

# Run the Job to Initialize the Database

- Switch to the folder containing the kubernetes yamls:
  `cd /workspaces/eventsapp/kubernetes-config`
- Run the following command to deploy the job:
  `kubectl apply -f db_init_job.yaml`
- Watch the job until it has completed once:
  `kubectl get jobs -w`
  - Press **CTRL+C** when done
- Get the name of the pod created by the job and view the logs:
  `kubectl get pods`
  `kubectl logs DB-INITIALIZER-POD-NAME-HERE`
  - You should see similar output to shown here:

Again, these are all the exact same commands you ran earlier for Minikube

```
$ kubectl logs db-initializer-7tjtm
1
Connected!
Database created
Tables created
Records added
8:07:22 PM
Exiting
$
```

# Deploy the Events App

- Run the following commands to deploy the application:

```
kubectl apply -f web-service.yaml
kubectl apply -f web-deployment.yaml
kubectl apply -f api-service.yaml
kubectl apply -f api-deployment.yaml
```

Once again, these are all the exact same commands you ran earlier for Minikube. No changes at all.

- Wait for all the pods to be ready:

```
    kubectl get pods -w
```

# View the Running App in EKS

- View the kubernetes services

```
kubectl get svc
```

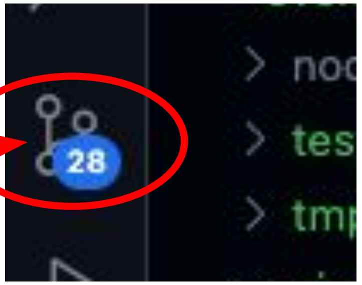
- This time in the `EXTERNAL-IP` column you will see a host name for an AWS Elastic Load Balancer (ELB)
    - When using EKS, load balancer services are implemented as an AWS ELB
- Copy the entire hostname of the ELB and view it in a new browser tab
    - It may take 1-2 minutes for for load balancer to be ready
        - If you do not see the app, wait a minute and reload the page
    - The application should work as before
    - You will only see the original 2 events since you just deployed a brand new database in the new cluster
    - Feel free to add new events

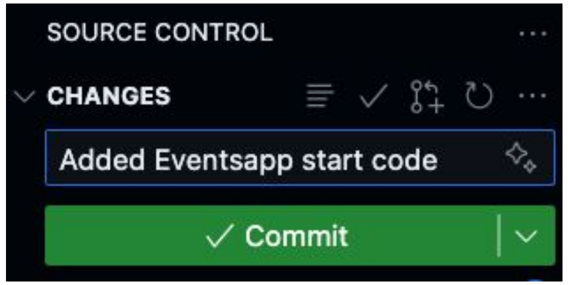
## Delete the EKS Cluster

- Run the following command to delete the EKS cluster
  - Replace X with your user number
  - `eksctl delete cluster userX-cluster`
- It will take a few minutes for the cluster to delete
  - You can just let it run

# Syncing the Changes to Git

- Commit these changes to your Git repository
  - On the left side, click the **Source control** button
  - Most changes should be staged automatically
    - Be sure ALL changes are staged by clicking the **+** button
  - Type a commit message of: **`Added Events app start code`** and click the **Commit** button





Added Eventsapp start code

  - Press the **Sync Changes** button and press **OK** to push the changes
- The code has now been saved to your **eventsapp** Git repository created earlier

## Cleaning Up

- Once completed, have your instructor verify everything is cleaned up

## Success!

- **Congratulations**! You have successfully deployed the Case Study App to EKS