Audit Access Expires May 4, 2020
You lose all access to this course, including your progress, on May 4, 2020.
Upgrade by Apr 13, 2020 to get unlimited access to the course as long as it exists on the site. **Upgrade now**

Exercise 8

In this exercise, you will deploy the application to two Amazon EC2 instances to make the application highly available. You will also create and configure an Application Load Balancer to route the web traffic to the two instances. You will then take one of the web servers down and see how the Application Load Balancer diverts traffic to the web server that is still up, thereby causing minimal outage.

Note: Make sure to sign-in to your AWS account with the AWS IAM user **edXProjectUser** credentials.

To get started, follow the steps below.

1. Start the Amazon RDS database instance.

- In the AWS Console, click **Services**, then click **Relational Database Service** to go to the **Amazon RDS dashboard**.
- In the left navigation pane, click **Instances**. From the list of instances, select **edx-photos-db**.
- At the top, click **Instance actions, and Start**.

2. Create a security group in edx-build-aws-vpc network to allow web traffic to the Amazon EC2 instance acting as the web server.

In this section, you will create a security group for the Amazon EC2 instance in edx-build-aws-vpc. If you are familiar with security groups, you may want to attempt to complete this section before reading the step-by-step instructions.

Security group name: web-server-sg
Region: Oregon (us-west-2)
VPC: edx-build-aws-vpc
Rules: Allow HTTP and SSH

▼ Expand for step-by-step instructions

- In the AWS Console, click **Services**, then click **EC2** to open the **EC2 dashboard**.
- Make sure you are still in the **Oregon** AWS region.
- On the left navigation menu, under Network & Security, click **Security Groups.**

- Click **Create Security Group**.
- For **Security group name**, type **web-server-sg**
- For **Description**, type **Allow SSH and HTTP traffic**
- For **VPC**, select **edx-build-aws-vpc**.
- On the bottom pane, click **Inbound**.
- Click **Add Rule**.
- For **Type**, select **HTTP**.
- For **Source**, type **0.0.0.0/0**
- You will also need to be able to SSH into the instance. To add a rule for SSH traffic, click **Add rule**
- For **Type**, select **SSH**.
- For **Source**, type **0.0.0.0/0**
- Click **Create**.

3. Modify the Amazon RDS database security group to allow traffic from the web server.

In this section, you will modify the security group of the Amazon RDS database instance to allow it to communicate with the Amazon EC2 instance that is acting as the web server.

- In the AWS Console, click **Services**, then click **EC2** to open the **EC2 dashboard**.
- On the left navigation menu, under NETWORK & SECURITY, click **Security Groups**.
- From the list of security groups, select the security group of the Amazon RDS database instance you noted in the previous exercise.

- On the bottom pane, click the **Inbound** tab.
- Click **Edit**.
- Click **Add Rule**.
- For **Type**, select **MYSQL/Aurora**.
- In the **Source** textbox, type **web-server-sg**. Select the security group that is displayed.
- Click **Save**.

4. Create an Amazon S3 bucket for storing the deployment artifacts.

In this section, you will create an Amazon S3 bucket for storing the deployment artifacts. If you are familiar with Amazon S3, you may want to attempt to complete this section before reading the step-by-step instructions.

Region: Oregon (us-west-2)
Bucket name: Type a unique bucket name and make a note of it for later use.

▼ Expand for step-by-step instructions

- In the AWS Console, click **Services**, then click **S3** to open the **S3 dashboard**.
- Click **Create bucket**.
- For **Bucket** name, type a unique bucket name. Write down the name of the bucket for later use.
- For **Region**, make sure you have selected the **Oregon** (us-west-2) region.
- Click **Create**.

5. Update the configuration files.

- The app.ini file contains the settings for the uwsgi server. The uwsgi server hosts the Python Flask application. Just like your AWS Cloud9 IDE, the uwsgi server also needs to be configured with the database information.
- In your AWS Cloud9 environment, open the **exercise-rds/Deploy/app.ini** file. You should see the environment variables DATABASE_HOST, DATABASE_USER, DATABASE_PASSWORD, DATABASE_DB_NAME, FLASK_SECRET, and PHOTOS_BUCKET.
- Fill in the values of the environment variables as per the table shown below.

DATABASE_HOST	Database endpoint you noted in previous exercise
DATABASE_USER	web_user
DATABASE_PASSWORD	Password for the web_user
DATABASE_DB_NAME	Photos
FLASK_SECRET	Type a new secret consisting of random letters and numbers.
PHOTOS_BUCKET	Copy and paste the PHOTOS_BUCKET value from the AWS Cloud9 environment variables list.

Note: Make sure to delete any white space inserted while copy/pasting.

- Save the app.ini file.
- In your AWS Cloud9 environment, open the **exercise-rds/Deploy/userdata.txt** file.
- On the second line of the file, replace **YOUR-BUCKET-NAME** with the name of the Amazon S3 bucket you created earlier.
  Important: Make sure to replace with the name of the *deployment* bucket created in this exercise, rather than the photos bucket.
- Save the userdata.txt file.

6. Create the deployment .zip package with updated settings.

- In your AWS Cloud9 terminal, change the working directory to **exercise-rds** by using the command below.

  ```
  cd exercise-rds
  ```

  Note: The exercise-rds folder should be in your AWS Cloud9 environment from the previous exercise.

- Create a deployment package by zipping up the modified configuration files and the Python Flask app. Type the command below:

  ```
  zip -r ~/deploy-app.zip Deploy/ FlaskApp/
  ```

  This creates a .zip file of the deployment package with updated configuration settings along with the Python Flask application.

7. Copy the deployment package to the Amazon S3 bucket.

- Using the AWS CLI command below, copy the zipped deployment package to the Amazon S3 bucket you created earlier. In your AWS Cloud9 terminal, type the command shown below.
  Important: Make sure to replace with the name of the *deployment* bucket created in this exercise, rather than the photos bucket.

  ```
  aws s3 cp ~/deploy-app.zip s3://YOUR_BUCKET_NAME/
  ```

  This command copies the deployment package to the specified Amazon S3 bucket.

8. Create an AWS IAM role to allow the user to download the deployment package.

In this section, you will create an AWS IAM role to authenticate the EC2 instance to download the deployment artifacts stored in the Amazon S3 bucket. You will also authenticate the instance to communicate with Amazon Rekognition. If you are familiar with AWS IAM roles, you may want to attempt to complete this section before reading the step-by-step instructions.

Trusted entity: Amazon EC2
Use case: Allows Amazon EC2 instances to call AWS services on your behalf
Permissions: AmazonS3FullAccess, AmazonRekognitionReadOnlyAccess
Role Name: ec2-webserver-role

▼ Expand for step-by-step instructions.

- In the AWS Console, click **Services**, then click **IAM** to open the **IAM dashboard**.
- On the left-side navigation menu, click **Roles**.
- Click **Create role**.
- For **Select type of trusted entity**, scroll down and select **EC2**.
- For **Select your use case**, select the first option: **Allows EC2 instances to call AWS services on your behalf.**
- Click **Next: Permissions**.
- In the search text box, type **S3**
- Select **AmazonS3FullAccess**.
- In the search text box, type **Rekognition**
- Select **AmazonRekognitionReadOnlyAccess**.
- Click **Next: Review**.
- For **Role name**, type **ec2-webserver-role**
- Click **Create role**.

9. Provision Amazon EC2 instances and deploy the application via user data

In this section, you will **create two Amazon EC2 instances** which will act as your web servers. You will install your application on these instances via user data script. Open the userdata.txt file in your AWS Cloud9 environment and explore its contents. You will notice that the script installs Python 3 and dev tools related to Python 3, the nginx web server, and the uwsgi application server. The uwsgi server hosts the Python Flask application. The script then creates a directory for the photos application and downloads the deployment package from the

Amazon S3 bucket. It then installs the requirements as listed in the requirements.txt file. The script then starts the nginx and the uwsgi servers.

Using the properties listed below, create the **first Amazon EC2 instance** and deploy the application via user data. You may want to attempt to complete this section before reading the step-by-step instructions.

Region: Oregon (us-west-2)
Amazon Machine Image (AMI): Amazon Linux AMI *(Do not use the Amazon Linux 2 AMI)*
Instance Type: t2.micro
Network: edx-build-aws-vpc
Subnet: edx-subnet-public-a in us-west-2a availability zone
IAM Role: ec2-webserver-role
User data script: Copy and paste the contents of the userdata.txt file from your AWS Cloud9 environment.
Tag: WebServer1
Security Group: web-server-sg
Key Pair: Use the key pair created in the third exercise.

▼ Expand for step-by-step instructions.

- In the AWS Console, click **Services**, then click **EC2** to open the **EC2 dashboard**.
- Make sure you are in the **Oregon** region.
- From the Amazon EC2 dashboard, click **Launch Instance**.
- On the **Choose an Amazon Machine Image (AMI)** page, select the **Amazon Linux AMI**. This AMI is free-tier eligible.
  Note: Do not select the Amazon Linux 2 AMI option.
- On the **Choose an Instance Type** page, select **t2.micro**.

- Click **Next: Configure Instance Details**.
- For **Network**, select **edx-build-aws-vpc**.
- For **Subnet**, select **edx-subnet-public-a** in the **us-west-2a** Availability Zone.
- For **IAM role**, select **ec2-webserver-role**.
- Leave the default settings, scroll down to the **Advanced Details** section, and expand it.
- Copy and paste the contents of the userdata.txt file from your AWS Cloud9 environment in the text area.
- Click **Next: Add Storage**. Skip through this page and click **Next: Add Tags**.
- Click **Add Tag**.
- In the Key textbox, type **Name**
- In the Value textbox, type **WebServer1**
- Click **Next: Configure Security Group**. Select the **Select an existing security group** option.
- From the list of security groups, select **web-server-sg**.
- Click **Review and Launch**.
- On the Review Instance Launch page, review the details and click **Launch**.
- When prompted for a key pair, select **Choose an existing key pair**, and then choose the key pair you created in the third exercise.
- Select the acknowledgement check box, and then click **Launch Instances**.
- Click **View Instances** to return to the Instances page.
- On the Instances page, you can view the status of the launch. It can

take a few minutes for the instance to be ready so that you can connect to it. Make sure that your instance has passed its status checks. You can view this information in the Status Checks column.

- Once the instance is ready, select the instance and write down the **IPv4 Public IP** found in the **Descriptions** tab at the bottom.

Using the properties listed below, create the **second Amazon EC2 instance** and deploy the application via user data. You may refer to the step-by-step instructions provided above.
Important: Note the changes in the subnet and the tag name for the second Amazon EC2 instance.

Region: Oregon (us-west-2)
Amazon Machine Image (AMI): Amazon Linux AMI *(Do not use the Amazon Linux 2 AMI)*
Instance Type: t2.micro
Network: edx-build-aws-vpc
Subnet: edx-subnet-public-b in us-west-2b availability zone
IAM Role: ec2-webserver-role
User data script: Copy and paste the contents of the userdata.txt file from your AWS Cloud9 environment.
Tag: WebServer2
Security Group: web-server-sg
Key Pair: Use the key pair created in the third exercise.

Note: It takes a few minutes for the status checks to pass. Wait until the status check changes from **Initializing** to **2/2 checks passed**. Once the instance is ready, select the instance and write down the **IPv4 Public IP** found in the **Descriptions** tab at the bottom.

10. Test the deployed application on the Amazon EC2 instances.

- Copy the IP addresses of the Amazon EC2 instances you created in the previous section.
- Go to your browser, paste the IP addresses, and click ENTER. The application should be deployed on the instances.
- If you do not see the application deployed on the instances, you may check the instance log by selecting the instance and then clicking **Actions** -> **Instance Settings** -> **Get System Log** in the Amazon EC2 dashboard. Note that logs can take up to five minutes to appear. You may also double check the web-server-sg and Amazon RDS security groups to make sure they are set up as per the instructions provided in this exercise.
- Some additional notes for debugging:
  - You may check the **cloud-init-output.log** log file for additional information. Refer to the instructions in the third exercise to view the log file.
  - You may view the uwsgi log file on the instance by running the following command on your web server instance: **sudo cat /var/log/uwsgi.log**

  Note: The resources created in this exercise are not needed in any of the subsequent exercises. Feel free to continue working on the next exercises (even if this exercise could not complete) and come back to this another time.

11. Create and configure the Application Load Balancer.

- In the AWS Console, click **Services**, then click **EC2** to open the **Amazon EC2 dashboard**.
- On the left navigation pane, under **LOAD BALANCING**, click **Load**

Balancers.

- Click **Create Load Balancer**.
- Click **Application Load Balancer**, and then click **Create**.
- For **Name**, type **photos-alb**
- Leave the default settings for **Scheme** and **Listeners**.
- Under **Availability Zones**, for **VPC**, select **edx-build-aws-vpc**.
- For **Availability Zones,** select the two Availability Zones shown in the list.
- You will be prompted to select a subnet in each Availability Zone. Select the **public subnet** in each availability zone by clicking the row of the public subnet.
- Click **Next: Configure Security Settings**. Skip this section and click **Next: Configure Security Groups.**
- For **Assign a security group,** select **Select an existing security group.**
- From the list of security groups shown below, select **web-server-sg**. You may need to un-select the default selection.
- Click **Next: Configure Routing**.
- For **Target group**, keep the default selection, **New target group**.
- For **Name**, type **webserver-target**.
- Leave the rest of the default settings as they are and click **Next: Register Targets**.
- For **Instances**, select the two Amazon EC2 instances that you created earlier: **WebServer1** and **WebServer2**.
- Click **Add to registered**.
- Click **Next: Review**.

- On the Review page, click **Create**.
- To return to the Load Balancer page, click **Close**.

  It should take a few minutes for the load balancer to become active and for the instance heath checks to pass. You will notice that initially, the state of the load balancer is displayed as **Provisioning**. After a few minutes, the state changes to **Active**.
  Note: You can ignore the warning you see besides the name of your load balancer. This is occurring because of the restricted permissions of the edxProjectUser.

- To check if the instance health checks have passed, click the **Target Groups** on the left-side navigation menu under **LOAD BALANCING**.
- Select the **webserver-target** target group and click **Targets** tab on the bottom pane.
- Under **Registered targets**, you should see information about the health checks for the instances.
- To test whether the load balancer is routing traffic to the target group, click **Load Balancers** under **LOAD BALANCING** and select **photos-alb**.
- In the **Description** tab at the bottom, copy the **DNS name** of the load balancer, paste it in a browser, and click ENTER.

  You should see the application load. The web traffic to the application is now being routed via the load balancer.

12. Take one web server down and test for the availability of the application.

- In the AWS Console, go to the Amazon EC2 dashboard and click **Instances** on the left navigation menu.

- Select the instance with name **WebServer2**. This is the instance you created and deployed the application on by using user data in a previous section.
- Write down the public IP address of the instance and connect to the instance via SSH. You may refer to the detailed step-by-step instructions for connecting to the instance in the third exercise.
- To take WebServer2 down, type the commands below in your instance terminal window.

  ```
  sudo rm –rf /photos/
  sudo restart uwsgi
  ```

  WebServer2 is now down and unhealthy.

- To check whether WebServer2 is down, copy the public IP address of WebServer2, paste it in a browser, and click ENTER. You should see an nginx error page.
- To check whether the target group under the load balancer has detected this failure, on the left-side navigation menu under **LOAD BALANCING**, click **Target Groups** .
- Select the **webserver-target** target group and click **Targets** tab at the bottom.
- Notice that the instance **WebServer2** is now showing as **unhealthy**. You may need to refresh the page few times for the health check to reflect the change.
- To check if the load balancer is still routing traffic to the one healthy instance, Webserver1, in the target group, paste the load balancer DNS name in a browser and click ENTER.

  You should see that the application is still served via the healthy instance in the target group, thereby causing minimal outage.

13. Terminate the Amazon EC2 instances.

To keep your AWS bill to a minimum, terminate the Amazon EC2 instances **WebServer1** and **WebServer2** by selecting the instances in the Amazon EC2 dashboard and clicking **Actions -> Instance State -> Terminate**.

14. Delete the load balancer.

To keep your AWS bill to a minimum, delete the load balancer you created in this exercise. Here's how to delete the load balancer: open the Amazon EC2 dashboard, and under **LOAD BALANCING**, select **photos-alb** and then click **Actions -> Delete**.

15. Stop the Amazon RDS database instance.

In order to keep your AWS account bill to a minimum, consider stopping the Amazon RDS instance and then starting it again when needed. Follow the steps below to stop the Amazon RDS database instance.

- In the AWS Console, go to the RDS dashboard.
- In the left-side navigation pane, click **Instances**. From the list of instances, select **edx-photos-db**.
- At the top, click **Instance actions**, and then **Stop**. You will get a prompt. Click **Yes, stop now**.

Optional Challenge

AWS Elastic Beanstalk isn't covered in this course. This will be covered in the next course: AWS Developer: Deploying on AWS. A lot of the deployment we have done in this exercise could be automated using

Elastic Beanstalk. For more information, see Deploying a Flask Application to AWS Elastic Beanstalk.

Try to create an Elastic Beanstalk single instance environment for your application. For more information, see Single-Instance Environment.

Your Elastic Beanstalk environment will need to be configured with environment variables. For more information, see Configuring Environment Properties.

Hint: The instance created by Elastic Beanstalk will not be able to communicate with the Amazon RDS database. You will need to update security groups to allow the database traffic.

Hint: The .zip file you just created is very similar to the .zip file you would use with Elastic Beanstalk. The contents of an Elastic Beanstalk would follow this pattern:

```
application.py
config.py
database.py
util.py
requirements.txt
templates/main.html
static/...
static/fonts/...
static/js/...
static/css/...
```

Learn About Verified Certificates