**Workshop 2**

**To complete Task 1, follow these steps carefully:**

**Step 1: EC2 Instance Security Group Configuration**

1. **Access EC2 Management Console**:
   * Log into your AWS Management Console.
   * Go to **EC2**.
2. **Locate Your EC2 Instance**:
   * Find the EC2 instance you are working with.
   * Copy the **Public IPv4 DNS** of your EC2 instance and paste it into your scratchpad. Example:
   * ec2-xx-xxx-xx-xxx.compute-1.amazonaws.com
3. **Configure Security Group**:
   * In the EC2 Management Console, locate your EC2 instance and check the **Security Group** it is associated with. In this case, it’s named django\_docker\_aws.
   * Click on the **Security Group** name (e.g., django\_docker\_aws) to open the security group settings.
   * Under the **Inbound rules** section, click **Edit inbound rules**.
   * Add a new rule:
     + **Type**: PostgreSQL
     + **Protocol**: TCP
     + **Port Range**: 5432
     + **Source**: Anywhere-IPv4
   * Save the changes.

Once added, you should see a rule with the type PostgreSQL in the inbound rules list.

**Step 2: Amazon Elastic Container Registry (ECR)**

1. **Go to Amazon ECR**:
   * In your AWS Management Console, go to **Elastic Container Registry (ECR)**.
2. **Create a New Repository**:
   * Click **Create repository**.
   * Name the repository workshop2.
3. **View Push Commands**:
   * After the repository is created, locate the **Push Commands** section.
   * Copy the first **macOS/Linux** push command (not Windows) and paste it into your scratchpad. The command will look similar to:
   * aws ecr get-login-password --region region\_name | docker login --username AWS --password-stdin aws\_account\_id.dkr.ecr.region\_name.amazonaws.com
4. **Copy Repository URI**:
   * In the ECR dashboard, find and copy the **Repository URI** for your workshop2 repository. The URI will look like:
   * aws\_account\_id.dkr.ecr.region\_name.amazonaws.com/workshop2

**What to Save in Your Scratchpad:**

1. EC2 Public IPv4 DNS:
2. ec2-xx-xxx-xx-xxx.compute-1.amazonaws.com
3. ECR Push Command (for macOS/Linux):
4. aws ecr get-login-password --region region\_name | docker login --username AWS --password-stdin aws\_account\_id.dkr.ecr.region\_name.amazonaws.com
5. ECR Repository URI:
6. aws\_account\_id.dkr.ecr.region\_name.amazonaws.com/workshop2

By following these steps, you should have all the important details copied into your scratchpad for Task 1.

**Task 2: Amazon Web Services Part 2 - Amazon RDS**

Follow these steps to complete Task 2 and set up an Amazon RDS PostgreSQL instance:

**Step 1: Access RDS Console**

1. **Search for RDS**:
   * In your AWS Console search bar, type rds and select **RDS - Managed Database Relational Service**.
2. **Click Create Database**:
   * In the RDS Console, click the **Create database** button.

**Step 2: Database Configuration**

1. **Database Creation Method**:
   * Leave **Database creation method** set to **Standard create**.
2. **Engine Options**:
   * Select **PostgreSQL** for the database engine.
3. **Version**:
   * Choose **PostgreSQL 12.15-R1** as the version.
4. **Templates**:
   * Select **Free Tier** from the list of available templates to make sure you qualify for the free tier option.

**Step 3: Settings Section**

1. **DB Identifier**:
   * In the **Settings** section, enter nctutorials for the **DB Identifier**.
2. **Master Username**:
   * Leave the **Master username** as the default postgres.
3. **Master Password**:
   * Enter admin123 for the **Master password**, and then confirm the password by entering it again.

**Step 4: Connectivity Section**

1. **VPC Security Group**:
   * Under **Connectivity**, for **Existing VPC security groups**, add the **Security Group** that you defined for your EC2 instance, django\_docker\_aws.
   * Click inside the input field and search for django\_docker\_aws. Select it from the list when it appears.

**Step 5: Additional Configuration**

1. **Database Name**:
   * Scroll down to **Additional configuration**, expand it, and set the **Database name** as nc\_tutorials\_db.

**Step 6: Review and Create**

1. **Review Estimated Monthly Costs**:
   * Ensure that you see and read the **Estimated monthly costs** section.
   * If you don’t see this section, go back to the **Templates** section and confirm that you selected the **Free Tier** template.
2. **Create Database**:
   * Once all fields are filled in, click **Create database**.

**Step 7: Wait for Database Creation**

* The database creation process may take several minutes. During this time, you can proceed with other tasks or take a break.
* You will see a green banner once the database is successfully created.

**Step 8: Copy Database Endpoint**

1. **View Database Details**:
   * After the database is created, click on the **DB identifier** (nctutorials) to view the database details.
2. **Find Endpoint & Port**:
   * Under the **Connectivity & security** section, look for **Endpoint & port**. This is where you will find the endpoint URL for your PostgreSQL database.
   * If you see a dash (-) instead of an endpoint, it means the database is still being created. Refresh the page until the endpoint appears.
3. **Copy the Endpoint**:
   * Copy the **Endpoint** URL (which will look like nctutorials.xxxxxxxx.us-west-2.rds.amazonaws.com), and paste it into your scratchpad.

**What to Save in Your Scratchpad:**

1. **Database Endpoint** (Example):
2. nctutorials.xxxxxxxx.us-west-2.rds.amazonaws.com

After following these steps, your Amazon RDS PostgreSQL database should be set up and accessible.

### ****Task 3: Local Environment Setup****

To complete Task 3, follow these steps carefully:

### ****Step 1: Download and Unzip the**** workshop2.zip ****File****

1. **Download the workshop2.zip** file provided for this task.
2. **Unzip the file** into the week2/ folder. You should have a folder structure that looks like this:
3. week2/
4. ├── workshop2/
5. ├── app/
6. ├── data/
7. ├── nginx/
8. ├── .gitignore
9. ├── docker-compose.yml
   * If you notice a nested workshop2/ folder inside the outer workshop2/ folder, move the contents from the inner folder to the outer one and delete the inner folder.

### ****Step 2: Open the**** workshop2 ****Folder in VS Code****

1. **Open VS Code**.
2. **Open the workshop2/ folder** you just extracted.
   * In VS Code, the **WORKSHOP2** folder should appear at the top of the Explorer panel.

Your folder structure inside VS Code should look like this:

WORKSHOP2/

├── app/

├── data/

├── nginx/

├── .gitignore

├── docker-compose.yml

### ****Task 4: Add Allowed Hosts and Configure NGINX as a Reverse Proxy Server to Serve Static Assets****

#### **Step 1: Update** settings.py

1. **Open settings.py**:
   * In VS Code, navigate to the app/nc\_tutorials/settings.py file.
2. **Import os**:
   * Below the existing import for decouple, add the following import:
   * import os
3. **Add Allowed Hosts**:
   * Find the section for **ALLOWED\_HOSTS**.
   * Add your **EC2 DNS address** (which you copied in Task 1) to the list of allowed hosts, along with '0.0.0.0', 'localhost', and '127.0.0.1'. Your updated ALLOWED\_HOSTS should look like this:
   * ALLOWED\_HOSTS = ['your-ec2-dns.amazonaws.com', '0.0.0.0', 'localhost', '127.0.0.1']
4. **Configure Static Files**:
   * Scroll down to the section for **Static files** at the bottom of settings.py.
   * Add the following if/else statement to correctly serve static files:
   * if os.environ.get('DJANGO\_DEVELOPMENT') == 'True':
   * STATIC\_URL = '/static/'
   * STATICFILES\_DIRS = [os.path.join(BASE\_DIR, 'static')]
   * else:
   * STATIC\_URL = '/static/'
   * STATIC\_ROOT = os.path.join(BASE\_DIR, 'staticfiles')

#### **Step 2: Update** urls.py

1. **Open urls.py**:
   * Navigate to app/nc\_tutorials/urls.py.
2. **Import Static and Settings**:
   * Add the following imports below the existing imports:
   * from django.conf.urls.static import static
   * from django.conf import settings
3. **Append Static Path**:
   * After the existing URL patterns in urlpatterns, append the following static files configuration:
   * urlpatterns += static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT)

This setup ensures that NGINX can serve static files correctly.

### ****Task 5: Set Up the Django Application for Deployment****

#### **Step 1: Update** docker-compose.yml

1. **Open docker-compose.yml**:
   * In VS Code, navigate to the docker-compose.yml file in the workshop2/ folder.
2. **Update Repository URIs**:
   * Replace YOUR\_REPO\_HERE with your **ECR repository URI** for both the web and nginx image keys:
   * web:
   * image: your-ecr-uri:workshop2\_web
   * ...
   * nginx:
   * image: your-ecr-uri:workshop2\_nginx
   * ...
3. **Set DB Host and Virtual Host**:
   * In the web service’s environment section, replace the placeholders for DB\_HOST and VIRTUAL\_HOST with:
     + DB\_HOST → Your **RDS database endpoint** from Task 2.
     + VIRTUAL\_HOST → Your **EC2 DNS address** from Task 1. Example:
   * environment:
   * DB\_HOST: your-rds-endpoint.amazonaws.com
   * VIRTUAL\_HOST: your-ec2-dns.amazonaws.com

#### **Step 2: Update** .env **File**

1. **Open .env**:
   * In the app/ folder, open the .env file.
2. **Set DB\_HOST**:
   * Replace the placeholder YOUR\_DB\_ENDPOINT\_HERE with your **RDS database endpoint** from Task 2:
   * DB\_HOST=your-rds-endpoint.amazonaws.com

### ****What to Save/Use****:

1. **ECR Repository URI**: Ensure that the URI from Task 1 is correctly used in the docker-compose.yml.
2. **RDS Endpoint**: The RDS endpoint from Task 2 should be added to both docker-compose.yml and .env.

Once these configurations are done, your environment should be ready for deployment with Docker, NGINX, and Django.

### ****Task 6: Create New GitHub Repository****

#### **Step 1: Create a New Repository on GitHub**

1. Go to GitHub and create a new repository named workshop2.
   * The repository URL should follow this pattern:  
     https://github.com/<YOUR GITHUB USERNAME>/workshop2.git
2. **Copy the repository URL** to your scratchpad for use in the next steps.

#### **Step 2: Initialize Git and Push Your Local Files**

1. Open a terminal in the workshop2/ folder on your local machine.
2. Run the following commands to initialize the Git repository and push the files to GitHub:
3. echo "# workshop2" >> README.md # Create a README file
4. git init # Initialize a new git repository
5. git add . # Add all files to staging
6. git commit -m "first commit" # Commit files
7. git branch -M main # Rename the default branch to main
8. Add the remote repository URL to your local repository:
9. git remote add origin <GIT REPO URL> # Replace <GIT REPO URL> with the GitHub URL you copied earlier
10. Finally, push the changes to GitHub:
11. git push -u origin main # Push to the main branch of the remote repository

#### **Confirming the Push**

After running the above commands, confirm that your files have been successfully pushed to the GitHub repository by checking the GitHub page.

### ****Task 7: Clone Repository on EC2 Instance, Build, and Run Docker Application****

#### **Step 1: SSH into Your EC2 Instance**

1. Open an SSH session to your EC2 instance (if not already open) using the following command:
2. ssh -i <your-key.pem> ubuntu@<your-ec2-public-dns>
3. **Stop and Remove Running Containers**:
   * Run the following command to list running containers:
   * docker ps -a
   * Stop and remove any containers that are currently running:
   * docker stop <container-id>
   * docker rm <container-id>
4. **Navigate to Home Directory**: Make sure you are in your user's home folder:
5. cd ~

#### **Step 2: Clone Your GitHub Repository**

1. **Clone the repository** you created earlier:
2. git clone https://github.com/<YOUR GITHUB USERNAME>/workshop2.git
3. **Navigate into the workshop2/ folder**:
4. cd workshop2
5. **Verify Folder Contents**: Run ls to make sure the folder contains the necessary files:
6. ls

The output should look like this:

app/

data/

nginx/

.gitignore

docker-compose.yml

#### **Step 3: Run the Dockerized Application**

1. **Bring the Containers Up in Detached Mode**: Run the following command to start the Docker containers:
2. docker compose up -d
3. **Check for Errors**: The first time you use docker compose up, you may see an error related to missing images. Docker will automatically build the images once it detects that they don't exist. You can monitor the build process, and once it's done, check the status of the containers:
4. docker compose ps

#### **Step 4: Access the Web Application**

1. Open your web browser and enter the following URL, replacing <your EC2 instance DNS> with your EC2 public DNS:
2. http://<your-ec2-instance-dns>:8000
   * Make sure the URL begins with http:// and **not** https://.
3. **Possible Error Messages**:
   * You may see an error message saying "Error establishing a database connection" or a DisallowedHost error. Don't worry—this is expected as migrations have not yet been applied.

#### **Step 5: Apply Django Migrations**

1. Run the following commands inside the Docker container to apply the migrations:
2. docker compose exec web python manage.py makemigrations --noinput
3. docker compose exec web python manage.py migrate --noinput
4. **Expected Output**: The migration commands should execute without errors, and you should see something like this:
5. Applying migrations...
6. Operations to perform:
7. Apply all migrations: admin, auth, contenttypes, sessions, etc.

#### **Step 6: Refresh the Web Page**

1. After applying the migrations, **refresh your web page** in the browser:
2. http://<your-ec2-instance-dns>:8000
3. You should now see the web application running. You should also see static assets like social media icons at the bottom of the page, which are served from the app/static/images/icons/ folder.

Your Dockerized Django application should now be up and running on your EC2 instance.

### ****Task 8: Use Insomnia to POST to the Database****

#### **Step 1: Create a New POST Request in Insomnia**

1. **Open Insomnia** and click on New Request.
2. Name the request (e.g., Create Tutorial).
3. Set the method to POST.
4. **Enter the URL** for your API endpoint. It should look like this, replacing <your-ec2-dns> with your actual EC2 DNS address:
5. http://<your-ec2-dns>:8000/api/tutorials/

#### **Step 2: Add the JSON Body**

1. Click on the Body tab.
2. Choose JSON as the body type.
3. Enter the following JSON body, replacing "by Alice Smith" with your full name:
4. {
5. "title": "Sample Tutorial by Hasan Bhuiyan",
6. "description": "This is a sample tutorial.",
7. "published": true
8. }

#### **Step 3: Send the Request**

1. Click on the Send button in Insomnia.
2. You should see a result like this in the response:
3. {
4. "id": 1,
5. "title": "Sample Tutorial by Hasan Bhuiyan",
6. "description": "This is a sample tutorial.",
7. "published": true,
8. "createdAt": "2025-01-11T00:00:00Z",
9. "updatedAt": "2025-01-11T00:00:00Z"
10. }

#### **Step 4: Screenshot of Insomnia**

Take a screenshot that includes:

* The **POST URL**.
* The **JSON body** you entered.
* The **Preview panel** of the HTTP response showing the successful POST.

### ****Task 9: Push to Amazon ECR****

#### **Step 1: Login to ECR**

1. Open your **SSH session** to your EC2 instance if it's not already open.
2. Run the login command you copied from the ECR repository page in AWS Console. It should look something like this:
3. aws ecr get-login-password --region <region> | docker login --username AWS --password-stdin <aws\_account\_id>.dkr.ecr.<region>.amazonaws.com

Replace <region> and <aws\_account\_id> with your AWS region and account ID.

* + If you see a permissions error, ensure the **django-docker-aws IAM role** is attached to your EC2 instance.

#### **Step 2: Push Docker Images to ECR**

1. Once logged in, run the following command in the workshop2 folder:
2. docker compose push
3. This command will push both the web and nginx images to ECR.

#### **Step 3: Verify in ECR Console**

1. Go to the **ECR Console**.
2. Check your workshop2 repository. You should see the pushed images for both web and nginx.

### ****Task 10: Use Python to Test and Grade the Assignment****

#### **Step 1: Locate the Python Script**

1. Navigate to the 3-DEVOPS folder on your machine.
2. Verify that you have the **autograder.py** script.

#### **Step 2: Run the Autograder Script**

1. Open a new terminal in the 3-DEVOPS folder.
2. Run the script with the following command, replacing host with your unique EC2 DNS name:
3. python autograder.py --workshop=2 --transport=http --host=<your\_unique\_ec2\_dns\_name> --port=8000

#### **Step 3: Submit the Results**

1. The script will generate a results.json file. This file will be used as part of your assignment submission.
2. If any issues are detected, fix them and run the script again until all tests pass.

### ****Final Clean-Up Instructions****

#### **Step 1: Dissociate and Release Elastic IP**

1. Go to the **Elastic IPs** page in the AWS console.
2. Select the Elastic IP and click on **Actions** > **Dissociate**.
3. After dissociating, click on **Actions** again and select **Release** to release the Elastic IP.

#### **Step 2: Terminate EC2 Instance**

1. Go to the **EC2 Console** and select your instance.
2. Click on **Actions** > **Instance State** > **Terminate**.

#### **Step 3: Delete RDS Database**

1. Go to the **RDS Console**.
2. Select the database you created and click on **Actions** > **Delete**.

#### **Step 4: Delete ECR Repository**

1. Go to the **ECR Console**.
2. Select your workshop2 repository and click on **Delete** to remove it.

By following these steps, you'll successfully clean up your resources after completing the workshop assignment.