Extra Credit - Deployment

(Optional.) Assume you are asked to get your code running in the cloud using AWS. What tools and AWS services would you use to deploy the API, database, and a scheduled version of your data ingestion code? Write up a description of your approach.

Answer:

To get my code running or deploy my Django REST API project in the cloud using AWS, I would need few AWS services and tools such as:

- Amazon Elastic Beanstalk to deploy and scale web applications
- Amazon RDS (Relational Database Service) for database
- Amazon VPC (Virtual Private Cloud) for virtual network
- Amazon Lambda for scheduling data ingestion
- CloudWatch EventBridge for setting up a trigger for Amazon Lambda
- Amazon SES (Simple Email Service) to verify if Lambda is getting triggered periodically

The following description gives an overall idea of how we can deploy our Django weatherApp in AWS cloud.

Virtual network:

To start with, we will make sure that we already have a default VPC or, create a default VPC in case we don't have one. **Amazon VPC or amazon Virtual Private Cloud** enables us to launch aws resources into our defined private network. We are going to deploy our app and create an Amazon RDS database in the default VPC. This virtual network closely resembles a traditional network where we will operate in our own data center. This facilitates the use of scalable infrastructure of AWS.

Security groups:

Next step, we will create two security groups: one for EC2 instances in the Elastic Beanstalk environment and the other for Amazon RDS database. We set up the inbound rules in the security group for EC2 instances for allowing SSH, HTTP and HTTPS access from source Anywhere-IPv4. We also set up the inbound rules for the Amazon RDS database for type PostgreSQL to allow PostgreSQL client access from eb environment.
This configuration now will allow only EC2 instances in the Elastic Beanstalk environment to connect the Amazon RDS database.

Database:

Next, we create a **PostgreSQL database on Amazon RDS** and attach it with the security group we created. In this step, we have to choose a subnet group and create a database with a few specifications, for example, engine type, templates, username, password etc. We will set a database name and it should give us a hostname under the endpoint. We will need this information to pass as environment variables to connect our app to the database.

Deployment:

Now, it's time we can deploy our app. For this, we will use **Amazon Elastic Beanstalk service**. Our django weatherApp is using SQLite3 on development. On production, we can use the PostgreSQL database that we created on Amazon RDS. We just need to configure the production settings with the environment variables: Django_SECRET_KEY, WEBSITE_HOSTNAME, DB_HOST, DB_NAME, DB_USER, DB_PASSWORD.

Since we have used a virtual environment , we will generate a requirements.txt file with 'pip freeze > requirements.txt' for Elastic Beanstalk to determine the packages to be installed. We create a virtual environment for elastic Beanstalk and install all the dependency from requirements.txt.

We will also need to create a directory called .ebextensions where a config file ebapp.config will be written that will attach the security groups for the EC2 instances and set the path of the app's WSGI object. We can now commit the file to the repository.

Next steps are:

- Deploy our site with **EB CLI (Elastic Beanstalk Command Line Interface)**. For this, we need to have awsebcli installed.
- Then immediately after we deploy, we need to edit Django's configuration file to add the domain name that Elastic Beanstalk assigned to our application in Django's ALLOWED_HOSTS. Then redeploy the application. This is **Django's security requirement** that is designed to prevent HTTP header attacks.
- **Initialize EB CLI repository**. This step will ask for some information such as: default location, app name, programming language, platform branch, Set up SSH, keypair etc. With that we now have the hostname of our app, database hostname, database name, database user, database user's password.
- **Setup environment variables for Elastic Beanstalk environment**. We pass the above information as the environment variables to the EB environment.
- **Run database migration command on deployment.** We need to modify the .ebextensions/ebapp.config file with the migration commands.
- **Commit the changes in the repository** with git add and git commit.
- **Redeploy** with 'eb deploy' command.

Testing:

We can now test our REST API using **HTTPie** or **Postman** and play around with retrieving and filtering data.

Connecting database instance with psql:

If we need to do some database management or run SQL queries, we may need to connect the Amazon RDS instance with a database client, for example, psql. **Psql** is a terminal based frontend to PostgreSQL. Psql will enable us to type in queries interactively, communicate with PostgreSQL and see the query results. We can do this connection of the database to psql from EC2 instance with ssh command.

Scheduling data ingestion from code:

We can use **Amazon Lambda** and then create a timed trigger so that Lambda runs at a certain schedule. AWS Lambda is an event-driven, serverless computing platform/service that runs code in response to events and automatically manages the computing resources required by that code.

We will write the Lambda function which will read the data (from a given source) and push/migrate it into the database. Then we can use a trigger using **CloudWatch EventBridge**. EventBridge allows us to set periodic triggers with rate() trigger. For example, if we set the schedule expression as rate(15 minutes), this will cause EventBridge to send a trigger to the Lambda function every 15 minutes to perform the scheduled action, ingest data in this case.

We can verify if Lambda is getting triggered periodically as planned using another service called Amazon SES (Simple Email Service). We can send email using Amazon SES each time the code is being executed. With the above example, an email will be sent every 15 minutes when the code is executed and data is ingested successfully.

Cleaning up:

Cleaning up is necessary to save instance hours and other AWS resources between development sessions. We can do this using the ‘eb terminate’ command. This terminates the Elastic Beanstalk environment , and doesn't delete the application. We can always create more environments with the same configurations running ‘eb create’ command.