CS443
Cloud Computing
Homework 5 Report

Onurcan Ataç
22002194

Date: 15.05.2024

Introduction

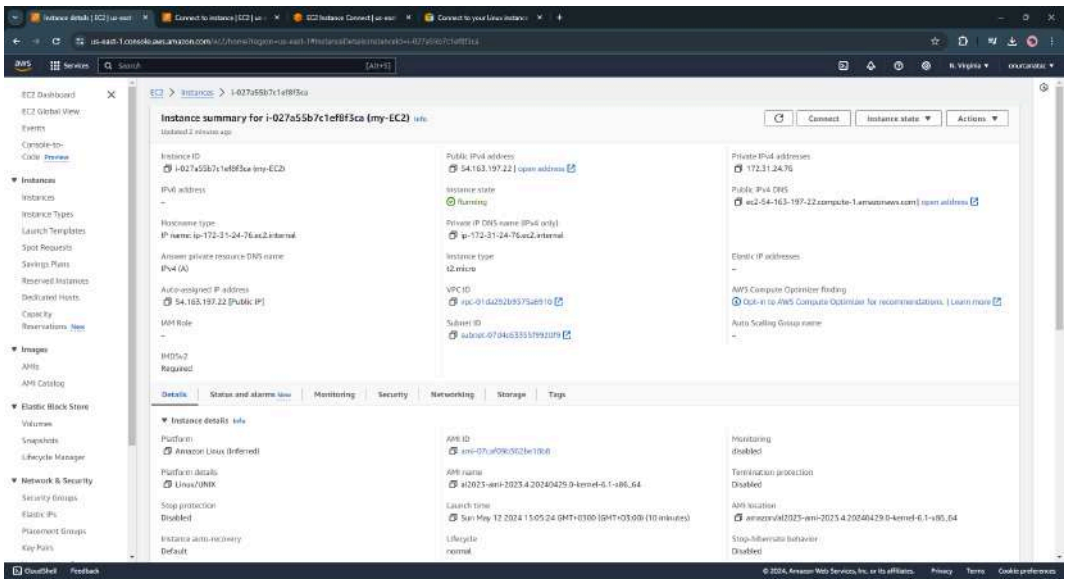
I have answered the questions and completed the demonstrations that are asked in the homework document in this report. I added the necessary screenshots and images of the work I have done to their related sections. I have completed the requirements using a free tier account in my name, “Onurcan Ataç”, and my name can be seen on the top right of most of the screenshots that I have provided.

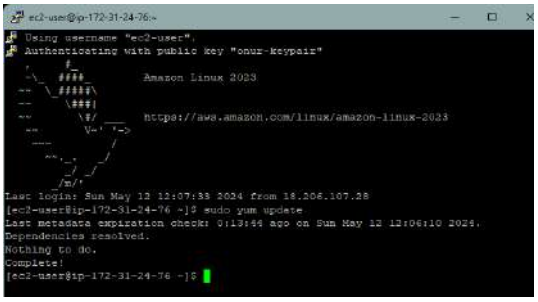
1. EC2

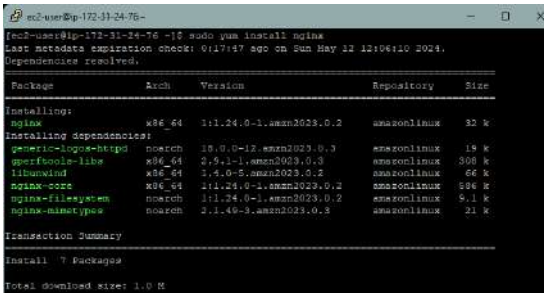
Launching an EC2 instance is an intuitive process. After finding EC2 in the AWS Console, the launch instance button has to be selected in order to see the specifications page. On this page, a name, an AMI, an instance type, and a keypair have to be selected for the EC2. Also, there are additional network options and configurations. The different AMI templates contain different prepared OS, application servers, and applications. There are quick-start templates provided by AWS, that include Ubuntu, macOS, and Windows. Instance types specify the specifications of the EC2: its GPU, memory, and generation. Storage also can be selected as different amounts of SSD storage.

EC2 offers several price models, On-Demand, Reserved, and Spot Instance options. For the On-Demand option, the customer pays for the compute capacity used per time. It is therefore easy to compute the payment. It is advised for applications that have unpredictable spiky workloads, or applications that are being tested on EC2. In the Reserved option, the customer commits to a consistent amount of usage but pays less than the On-Demand option. It is advised for steady applications and committed customers. Amazon Spot Instances are very cheap compared to the On-Demand option, they take advantage of unused EC2 capacity and are not guaranteed to work all the time. Therefore it is advised for fault-tolerant and flexible applications.

After creating my EC2 instance from this page and waiting for its initialization, I used Putty in order to connect to my EC2 instance. I then performed an operating system update, and I installed Nginx in order to make sure applications could be installed as intended.







2. Databases

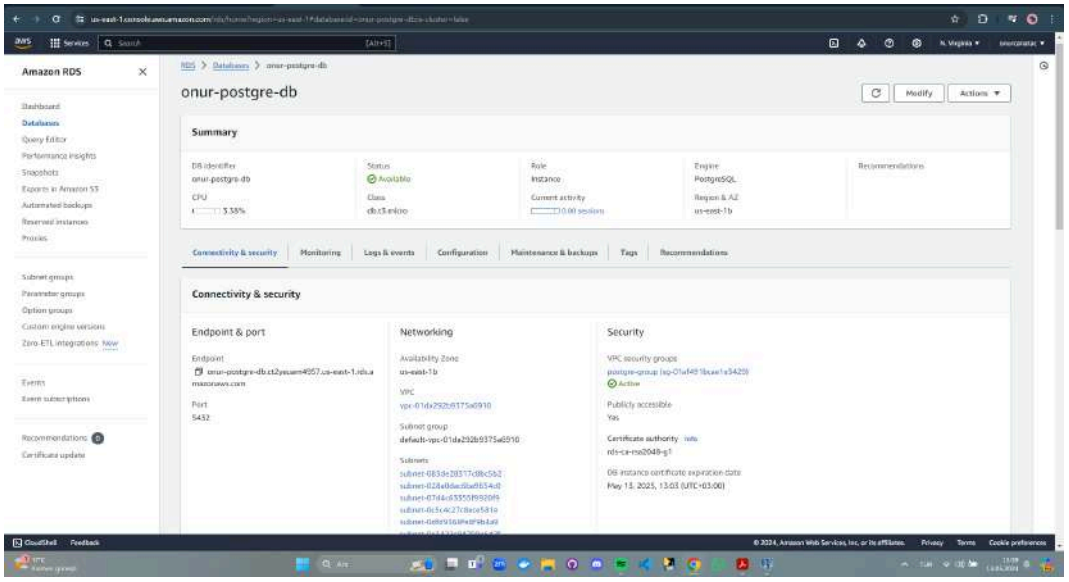
The most popular databases that AWS provides are Amazon RDS, Aurora, and DynamoDB. RDS and Aurora are databases that are specialized for storing relational data. RDS enables the customer to create database services using various popular database engines such as PostgreSQL, MySQL, Oracle, etc. It provides ease of use since Amazon maintains the database infrastructure and provides scalability whenever needed. It is generally advised for applications with sustained throughput, with no big spikes, such as a website that gets a similar number of views every day.

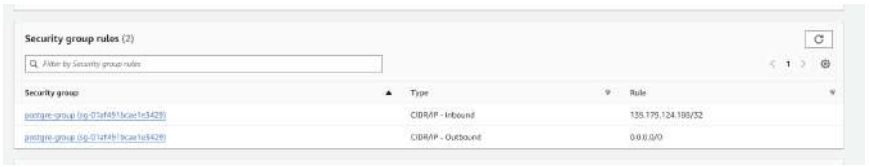
On the other hand, Aurora is another relational database. It is compatible with MySQL and PostgreSQL databases (not many options like RDS). However, Aurora offers higher availability and better durability compared to RDS. It also provides better throughput performance than traditional relational databases. Aurora also has more convenient autoscaling options. Aurora has a Serverless feature that enables it to run Aurora without having to guess how many compute nodes are needed. Therefore Aurora is advised for applications that contain big spikes in their workloads, such as website that set collective deadlines that a lot of people enter at the same time.

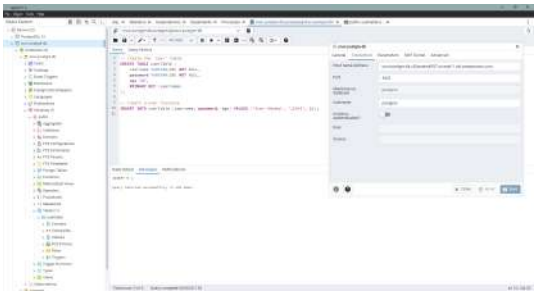
DynamoDB is a NoSQL database option. It is mainly used for key-value data or document storage. It is highly scalable and really usable for real-time analytics. For example, it may be used for a gaming website that provides analytics to the gamer.

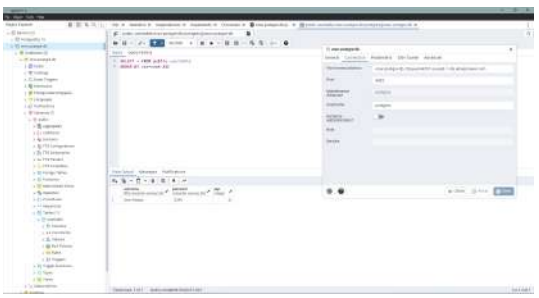
Creating an Amazon RDS instance starts with the engine selection. AWS provides many options such as Aurora, MySQL, MariaDB, PostgreSQL, and Oracle. Instance classes refer to the configurations of the computing skills of the instance such as memory, CPU power, and network performance. I/O optimized storage configurations are also available for specific scenario databases. After choosing the convenient ones from these options and reviewing backup and monitoring options, the user can create the database.

For my demonstration, I created a PostgreSQL database. I used the Pgadmin software to connect to the database endpoint. In the process, I had to create a new security group. I then connected to my database and I added a userTable and a user instance to my database by an SQL query successfully.









3. Security

AWS employs the shared responsibility model as a vendor with its customers. The model indicates that in order to maintain security, both the vendor and customer have responsibilities. The responsibility distribution changes for each service, for example, it is different for EC2 and RDS. In RDS, the vendor manages the platform and the application while it is the customer's responsibility in EC2. The customer's responsibility increases with the customizability of the provided service.

To protect AWS resources, security measures have to be implemented. Some important ones I would implement are access control by utilizing AWS IAM and Security Groups, which would enable me to control inbound and outbound traffic. Network security and subnet configuration is also crucial. I would also use a firewall (AWS Shield) to prevent DDOS attacks and add one more layer of protection.

In order to use IAM, the customer first has to navigate to the IAM page from the AWS Console. From there, users can be created from the User tab, and their permissions can be specified by attaching policies to those users. Policies can also be attached collectively by assigning the users in groups. Non-AWS entities can be represented by roles. They also can be given permissions.

AWS Security Groups come in very handy for controlling inbound and outbound traffic for EC2 instances. They can be used to control incoming requests to EC2 instances by defining constraints based on the IP addresses, port numbers, and protocols. Outbound traffic can also be constrained based on the same criteria by creating necessary rules. Overall, security Groups help in controlling and regulating the communication between EC2s and other destinations.

4. Containers

Containerization is in essence a deployment process that bundles the application's code with the required files and libraries that it needs to run on any infrastructure. Building applications in containers is extremely popular and convenient in our day since it is required for the Microservices Architecture and easy to employ DevOps practices on. Containers are also convenient for easy horizontal scaling, and resource efficiency since they do not need separate operating systems, unlike VMs.

In order to deploy a Docker container to AWS, the Docker image first have to be registered to ECR. After creating a repository in the ECR, the Docker image has to be pushed to the repository. After that, a cluster has to be configured and EC2 instances have to be created to be used as a server for the container from ECS. The networking also has to be configured from the same place. Later, a task has to be created. Inside the task, containers are specified with their respective Docker images that were registered in ECR. Finally, the task can run according to its configurations. That is how a Docker container is deployed to AWS.

ECS and EKS are two containerization services that are provided by AWS. ECS is a fully managed container orchestration service. It uses Docker containers while being able to manage, run, and scale containers. It has a relatively simpler interface and seamlessly

integrates with other AWS services. EKS is a Kubernetes service inside AWS. Kubernetes is already used by most companies and allows high functionality and flexibility in the containerization process.

For the customers that already have a Kubernetes architecture, or for the customers that require extensive control of their containers EKS appears to be better. However, easy-to-use but still functional ECS is better for applications that will work with less complexity.

5. **Monitoring**

AWS offers many monitoring and logging services such as CloudWatch and CloudTrail. CloudWatch takes and tracks metrics from running AWS services. It can show graphs of the collected metrics and also provides the option to create custom metrics for the customer. It can also be used to set alarms for preset metric values. However, the paid parts of the service are known to be expensive and premium. CloudTrail monitors API activity on the user's AWS account and keeps track of the actions taken.

CloudWatch alarms for RDS or EC2 instances can easily be set as well. After creating those instances, the user has to come to the CloudWatch page from the AWS Console. After selecting a metric from the Metrics tab, or creating one and then selecting that metric, the create alarm button becomes clickable. After clicking on the button, the user can set alarm specifications such as the checking period for the pattern that triggers the alarm, the threshold, and whether the value should be greater or lower than the threshold for the alarm to be active. The email accounts that are subject to the alarm can also be selected. Review and submit the options, and the alarm is created.

CloudWatch is very useful for incident response since it keeps track of metrics, and can send notifications or alarms when a preset emergency issue occurs. It can also take direct preset actions. These are extremely valuable, especially at night time when the customer is not aware.

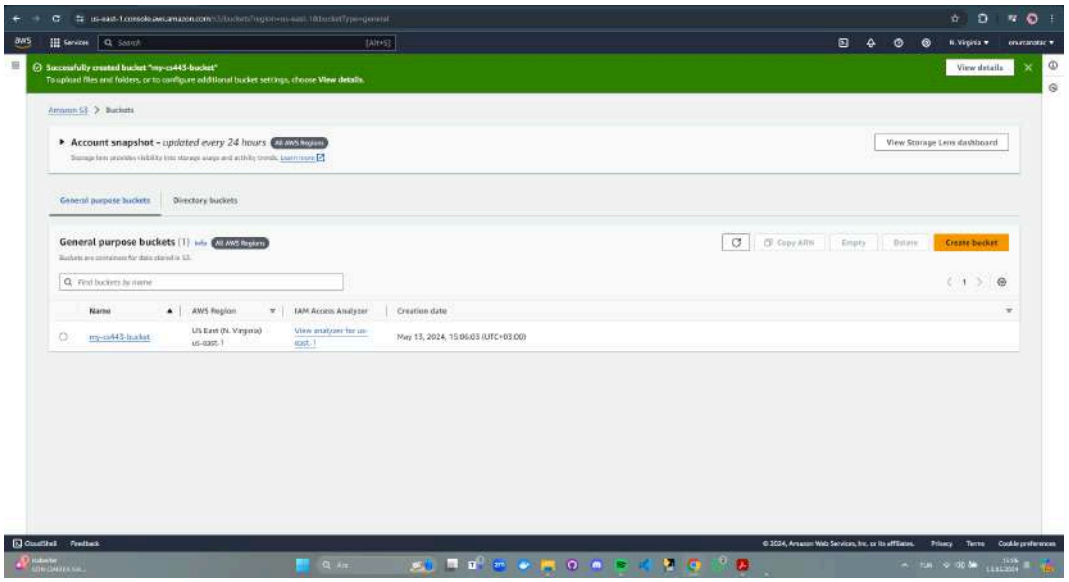
CloudTrail is also crucial for account control in AWS. It's very important to be able to trace the operations when an unwanted operation is made from the account. It also helps different users of the same account to understand what changes were exactly made.

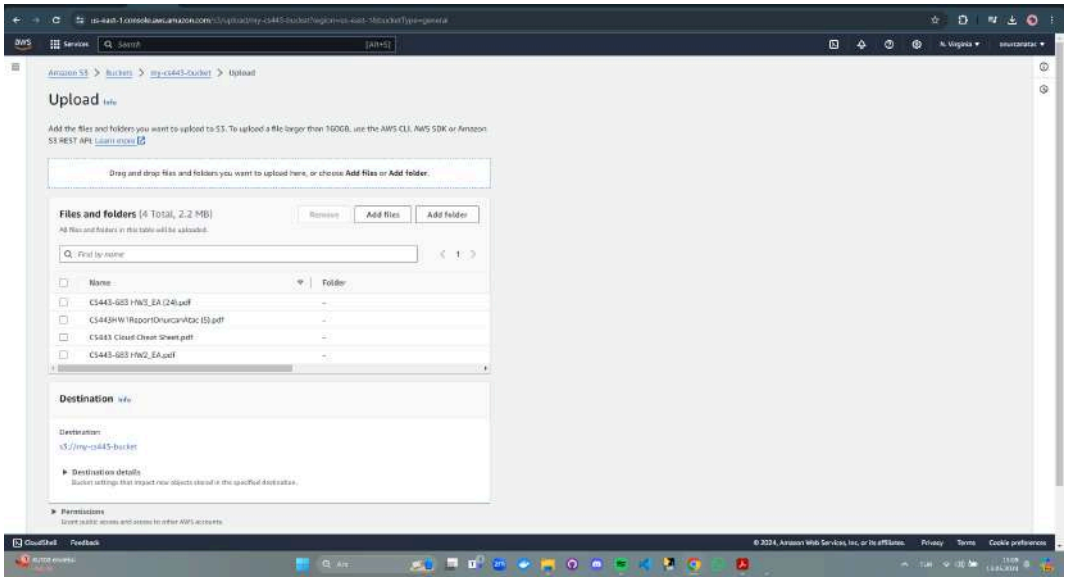
6. **Storage**

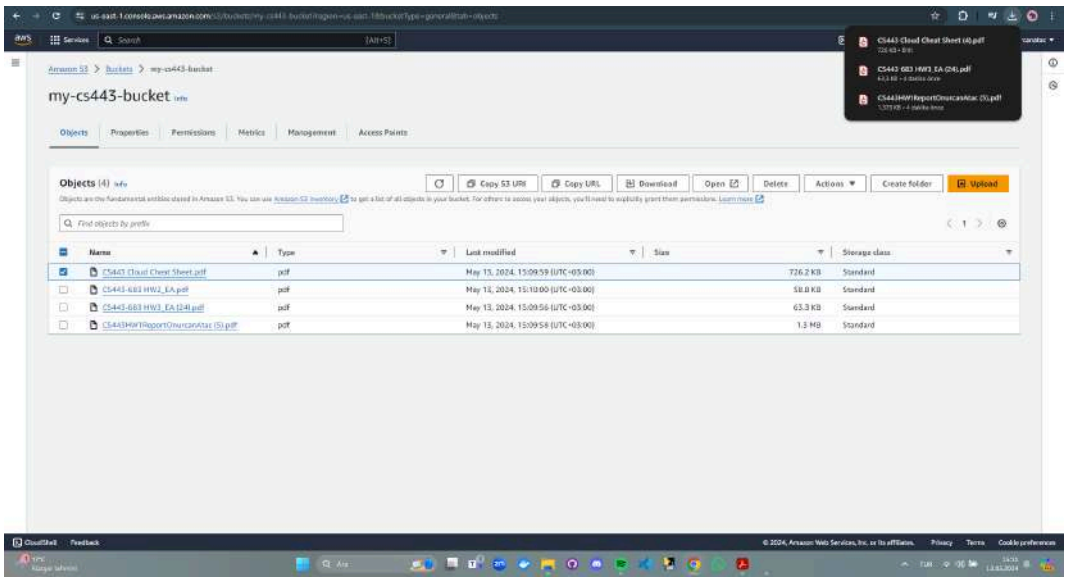
Among the AWS storage options, S3 is the most popular one. It is mostly used for storing large unstructured data, for example video files and other multimedia files. It presents very high durability (99.999999999%). EBS is a block-level storage and is used with EC2s. In EBS, there are different options such as throughput or IO optimized versions. EFS is used to conveniently store data that are in the structure of a file system, that requires shared access from different machines. A development environment might be given as an example.

An S3 bucket is created from the S3 page at the AWS Console. After clicking on the create button, configurations such as object ownership, public access, and encryption options can be set from there. After the creation of the S3 bucket, I uploaded some of the documents from this course to my bucket. I then checked to download the documents and the

documents were successfully downloaded. The file management options can be seen when the files are selected and the actions button is clicked.



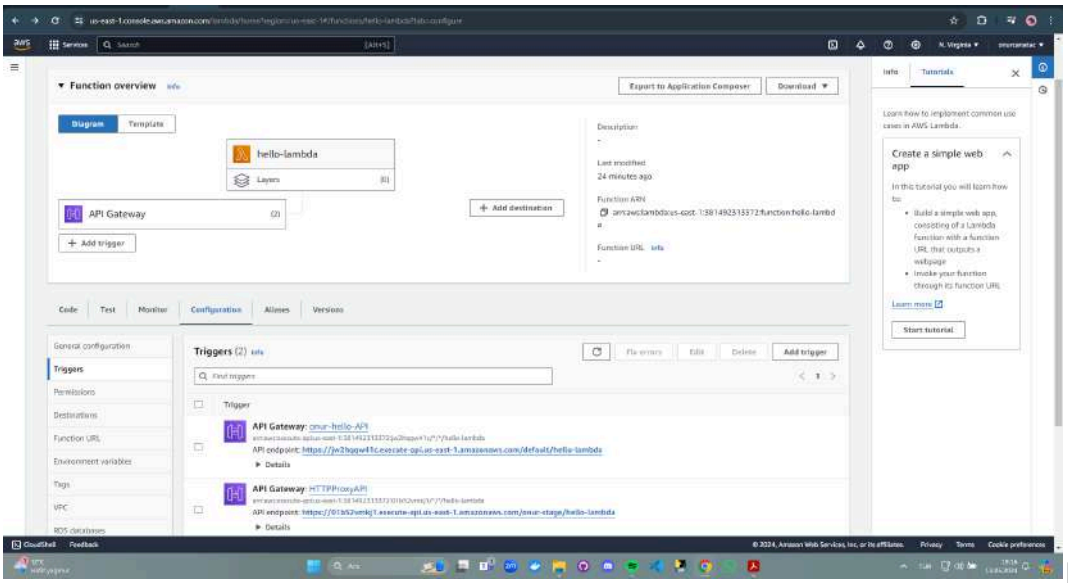


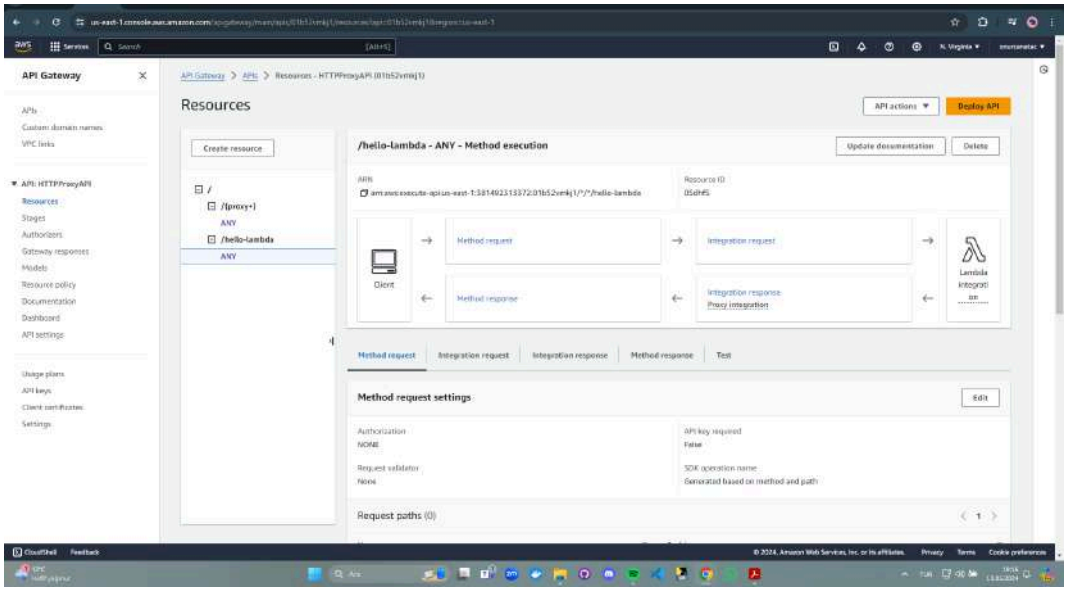


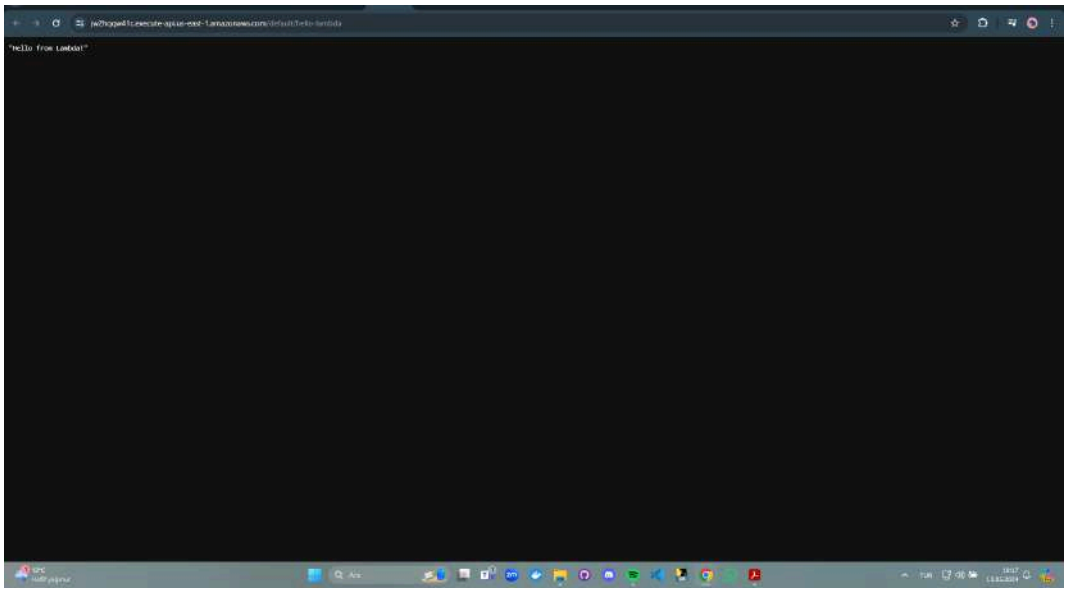
7. API Gateway

AWS API Gateway is mainly used for clients to properly interact with the backend, in order to use an application or a service. API Gateway is a powerful service, it enables integration with AWS services such as Lambda, authentication, and authorization using IAM, supports WAF, and increases security besides handling and transforming requests.

I created a basic API Gateway along with a Lambda function that logs “Hello from Lambda” when the endpoint is called. I also added a proxy by following the AWS tutorial. When I invoke the URL, the expected output shows up.





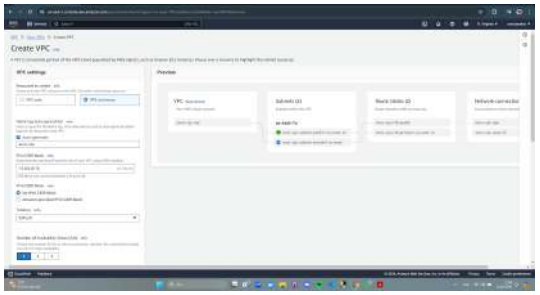
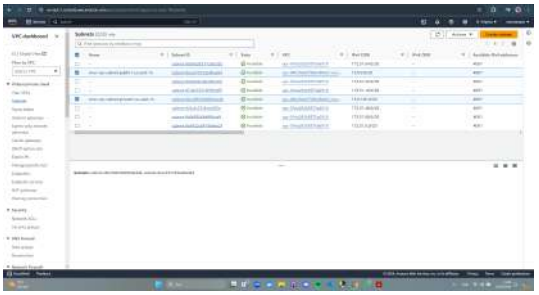
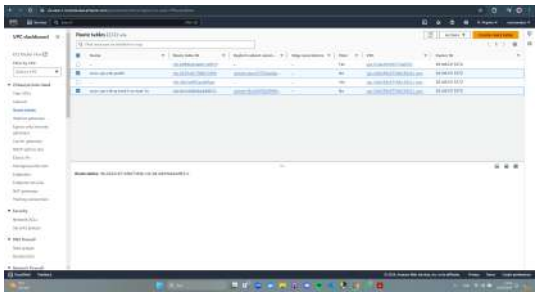
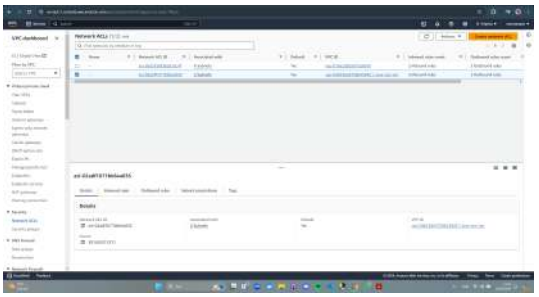


API Gateway in AWS is especially useful since it can be integrated with other AWS services. It can be integrated with IAM in order to perform authentication and authorization. It can also be integrated with AWS Lambda, which is huge since Lambda can offer easy functionality to API Gateway. Through AWS CloudWatch, API Gateway provides extensive monitoring options as well. By the same way, the API Gateway can be configured so that it sends notifications or alarms when some predetermined conditions are met. By throttling, API Gateway can control the number of incoming requests to the API. Hence, API Gateway enables the API owner to be flexible and open-minded with the options it provides, while also being crucial for the security of the API.

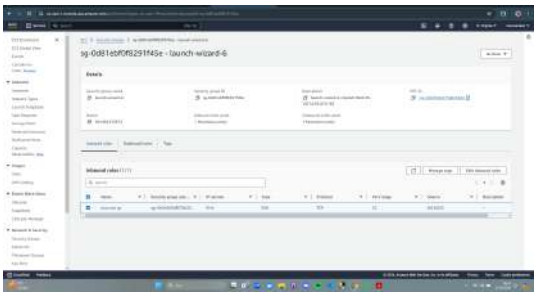
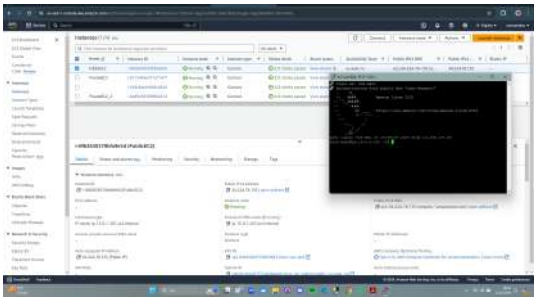
8. **Networking**

AWS VPCs allow the user to create a virtual network that is dedicated to the account, the user can create private and public subnets inside his own VPC after specifying the IP address range. VPC is used for the user to launch AWS services. Users can create multiple VPCs with different configurations and they work independently.

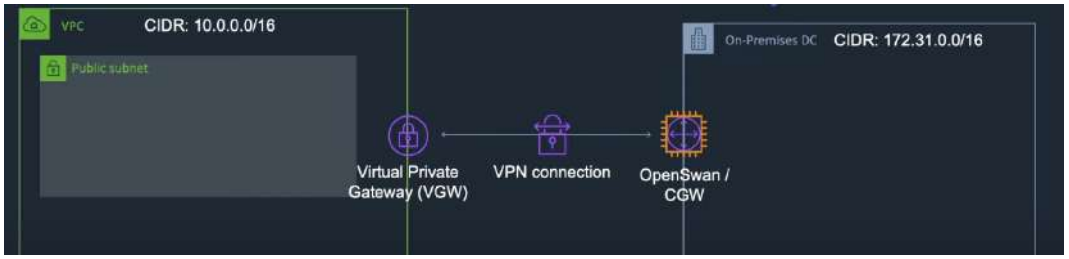
While creating a VPC, AWS provides the option to also configure the subnets and the route tables. After clicking the “VPC and more” button and specifying the name of the VPC, the user can select the IPv4 CIDR block required for the VPC. This block shows the IP addresses enabled to the VPC by CIDR notation. For example, “15.0.0.0/16” means the first 16 bytes are fixed, therefore 65536 IPs are available for the VPC, which is equal to 2 to the power of 16. The number of public and private subnets and the number of AZs can also be specified. The separate addresses of the subnets can also be changed from the creation process. The number of AZs is important depending on the application created, since they may be used for backups when an unforeseeable problem occurs in one AZ. The CIDR addresses of the subnet blocks can also be specified. Route tables and the network ACL for the subnets are also created when the VPC is created along with the Internet Gateway. Routes for the route table and the inbound and outbound rules for the network ACLs are easily modifiable through the left tab on the VPC page of the AWS Console. The instances that I have created can be seen below.

For the demonstration of connecting my EC2 instances within a VPC, I started by creating an EC2 instance on the public subnet of my VPC. I then configured a security group for my EC2 that enables an inbound rule with an SSH connection. By using Putty, I connected to my EC2 instance.

The connection between the VPC and on-premises resources is established by setting up a Virtual Private Gateway to the VPC, and a Customer Gateway to the on-premise network. A VPN connection will be established between them. After setting the VPG and CGW up, the VPN connection is created by setting the IP addresses of both networks and therefore configuring the routing.



9. Boto3 Library

Boto3 is the AWS SDK for Python. It is crucial for the Python developers to use their AWS resources using Python. The control over AWS services includes the creation and management of those services. The services that can be used include S3, EC2, RDS, DynamoDB, Lambda, and more. The SDK also simplifies working with AWS resources since it is intuitive.

In order to use Boto3, I need a user created in IAM, and that user has to have the necessary permissions to do the task that is specified in the code. I started by creating a user named “Onurcann” on the IAM page. In order to create an EC2, I created a user group for the user and provided the following permissions as JSON.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "ec2:DescribeInstances",
            "ec2:DescribeImages",
            "ec2:DescribeInstanceTypes",
            "ec2:DescribeKeyPairs",
            "ec2:DescribeVpcs",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups",
            "ec2:CreateSecurityGroup",
            "ec2:AuthorizeSecurityGroupIngress",
            "ec2:CreateKeyPair"
         ],
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": "ec2:RunInstances",
         "Resource": "*"
      }
   ]
}
```

By the following code, I successfully created an EC2 with the features that I specified in the object.

```python
import boto3

session = boto3.Session(
    aws_access_key_id='[redacted]',
    aws_secret_access_key='[redacted]',
    region_name='us-east-1'
)

ec2_client = session.client('ec2')

instance_details = {
    'ImageId': 'ami-0bb84b8ffd87024d8',
    'InstanceType': 't2.micro',
    'KeyName': 'onur-keypair',
    'MinCount': 1,
    'MaxCount': 1
}

response = ec2_client.run_instances(**instance_details)
instance_id = response['Instances'][0]['InstanceId']

print(f"Created EC2 ID => {instance_id}")
```

```
[Running] python -u "c:\onur_code\boto3code\boto3code.py"
Created EC2 ID => i-038721b228d7ec8c3

[Done] exited with code=0 in 2.669 seconds
```

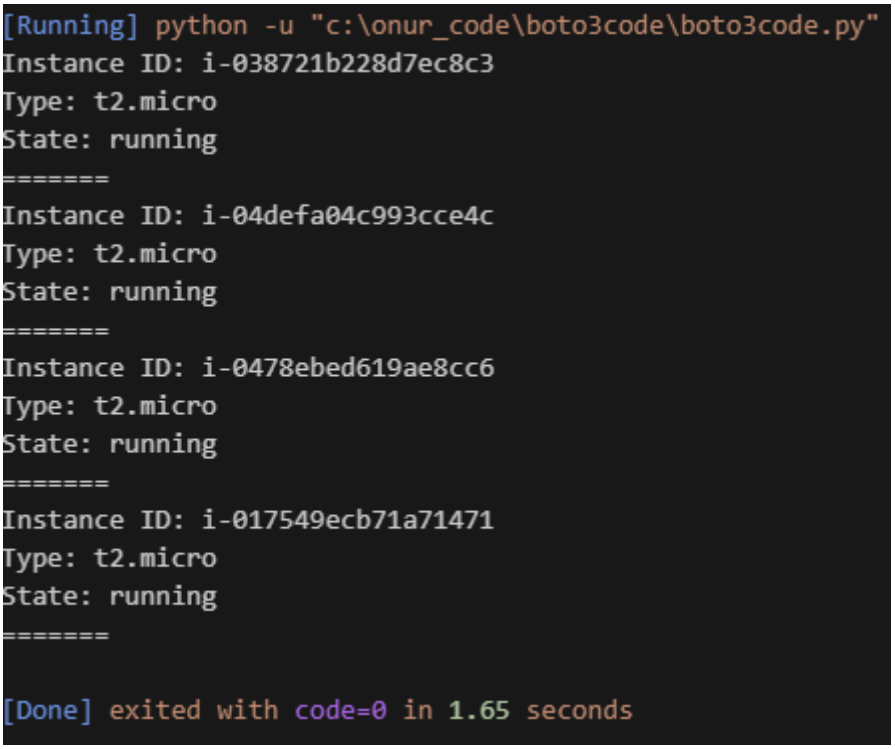
I then used Boto3 to list all the EC2 instances that I have created in the US-east-1 region.

```python
import boto3

session = boto3.Session(
    aws_access_key_id='[redacted]',
    aws_secret_access_key='[redacted]',
    region_name='us-east-1'
)

ec2_client = session.client('ec2')
response = ec2_client.describe_instances()

for reservation in response['Reservations']:
    for instance in reservation['Instances']:
        print(f"Instance ID: {instance['InstanceId']}")
        print(f"Type: {instance['InstanceType']}")
        print(f"State: {instance['State']['Name']}")
        print("=======")
```



Conclusion

This homework was very beneficial for me since it was a summary of many subjects we learned during the lectures. The homework also included both theoretical and practical work and contributed to by overall understanding of AWS and general areas of cloud computing. Even though the other homework assignments were also very helpful, not following a concrete guide while doing tasks enabled me to become more intuitive with the AWS Console and resources.

In this assignment:

- I became more familiar with connecting to the EC2 instances that I create.
- I created and connected to my PostgreSQL instance from Pgadmin.
- I learned how to create users, and roles, giving them permissions by creating groups.
- I learned the process of deploying a Docker container to AWS.
- I learned how to set alarms up from the metrics.
- I became more familiar with using an S3 bucket.
- I created an API Gateway endpoint.
- I became more familiar with the creation of VPC in detail and learned how a VPN connection is established between VPCs and on-premises.
- I learned how to use the boto3 library, how to set the credentials up in the code, and how to give complex permissions to a user.

Overall, I developed my cloud computing and AWS skills comprehensively.