CS443
Cloud Computing
Homework 5 Report

Onurcan Ataç
22002194

Date: 15.05.2024

Introduction

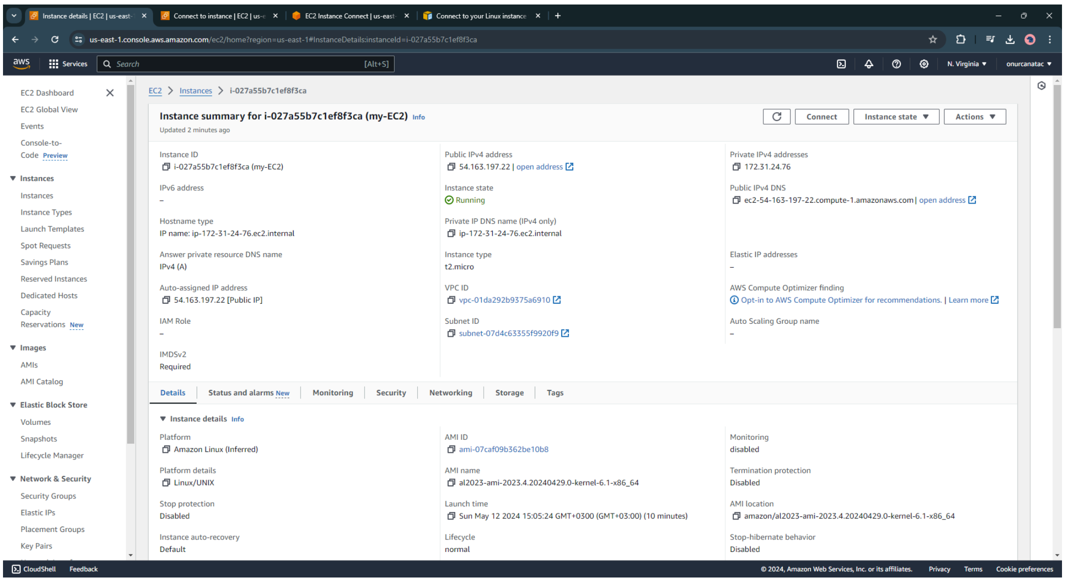
I have answered the questions and completed the demonstrations that are asked in the homework document in this report. I added the necessary screenshots and images of the work I have done to their related sections. I have completed the requirements using a free tier account in my name, “Onurcan Ataç”,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



Instance summary for i-027a55b7c1ef8f3ca (my-EC2) Info

Updated 2 minutes ago

Instance ID
i-027a55b7c1ef8f3ca (my-EC2)

IPv6 address
–

Hostname type
IP name: ip-172-31-24-76.ec2.internal

Answer private resource DNS name
IPv4 (A)

Auto-assigned IP address
54.163.197.22 [Public IP]

IAM Role
–

IMDSv2
Required

Public IPv4 address
54.163.197.22 | open address

Instance state
Running

Private IP DNS name (IPv4 only)
ip-172-31-24-76.ec2.internal

Instance type
t2.micro

VPC ID
vpc-01da292b9375a6910

Subnet ID
subnet-07d4c63355f9920f9

Private IPv4 addresses
172.31.24.76

Public IPv4 DNS
ec2-54-163-197-22.compute-1.amazonaws.com | open address

Elastic IP addresses
–

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more

Auto Scaling Group name
–

Instance details Info

Platform
Amazon Linux (Inferred)

Platform details
Linux/UNIX

Stop protection
Disabled

Instance auto-recovery
Default

AMI ID
ami-07caf09b362be10b8

AMI name
al2023-ami-2023.4.20240429.0-kernel-6.1-x86_64

Launch time
Sun May 12 2024 15:05:24 GMT+0300 (GMT+03:00) (10 minutes)

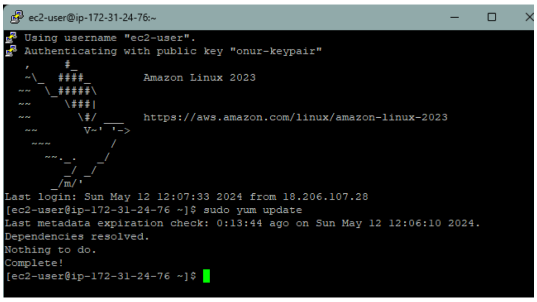
Lifecycle
normal

Monitoring
disabled

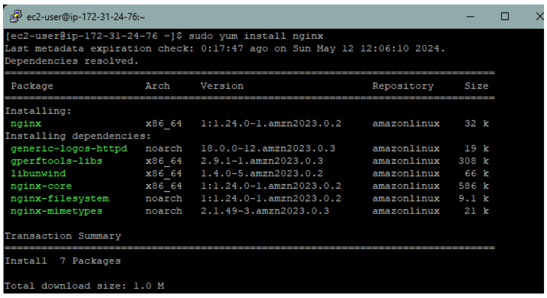
Termination protection
Disabled

AMI location
amazon/al2023-ami-2023.4.20240429.0-kernel-6.1-x86_64

Stop-hibernate behavior
Disabled



```
 Using username "ec2-user".
 Authenticating with public key "onur-keypair"
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Sun May 12 12:07:33 2024 from 18.206.107.28
[ec2-user@ip-172-31-24-76 ~]$ sudo yum update
Last metadata expiration check: 0:13:44 ago on Sun May 12 12:06:10 2024.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-24-76 ~]$
```



```
[ec2-user@ip-172-31-24-76 ~]$ sudo yum install nginx
Last metadata expiration check: 0:17:47 ago on Sun May 12 12:06:10 2024.
Dependencies resolved.
================================================================================
 Package                Arch     Version                    Repository     Size
================================================================================
Installing:
 nginx                  x86_64   1:1.24.0-1.amzn2023.0.2    amazonlinux    32 k
Installing dependencies:
 generic-logos-httpd    noarch   18.0.0-12.amzn2023.0.3     amazonlinux    19 k
 gperftools-libs        x86_64   2.9.1-1.amzn2023.0.3       amazonlinux   308 k
 libunwind              x86_64   1.4.0-5.amzn2023.0.2       amazonlinux    66 k
 nginx-core             x86_64   1:1.24.0-1.amzn2023.0.2    amazonlinux   586 k
 nginx-filesystem       noarch   1:1.24.0-1.amzn2023.0.2    amazonlinux   9.1 k
 nginx-mimetypes        noarch   2.1.49-3.amzn2023.0.3      amazonlinux    21 k

Transaction Summary
================================================================================
Install  7 Packages

Total download size: 1.0 M
```

2. Databases

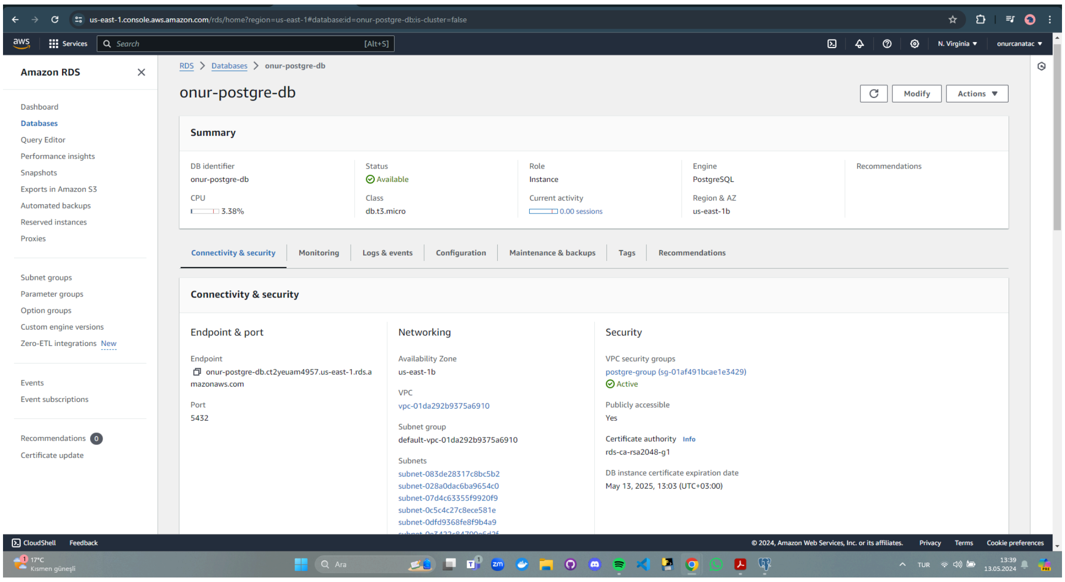
The most popular databases that AWS provides are Amazon RDS, Aurora, and DynamoDB. RDS and Aurora are databases that are specialized for storing relational data. RDS enables the customer to create database services using various popular database engines such as PostgreSQL, MySQL, Oracle, etc. It provides 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.



us-east-1.console.aws.amazon.com/rds/home?region=us-east-1#database:id=onur-postgre-db;is-cluster=false

RDS > Databases > onur-postgre-db

onur-postgre-db

Summary

DB identifier	Status	Role	Engine	Recommendations
onur-postgre-db	Available	Instance	PostgreSQL	
CPU	Class	Current activity	Region & AZ	
3.38%	db.t3.micro	0.00 sessions	us-east-1b	

Connectivity & security

Endpoint & port

Endpoint: onur-postgre-db.ct2yeuam4957.us-east-1.rds.amazonaws.com

Port: 5432

Networking

Availability Zone: us-east-1b

VPC: vpc-01da292b9375a6910

Subnet group: default-vpc-01da292b9375a6910

Subnets: subnet-083de28317c8bc5b2, subnet-028a0dac6ba9654c0, subnet-07d4c63355f9920f9, subnet-0c5c4c27c8ece581e, subnet-0dfd9368fe8f9b4a9

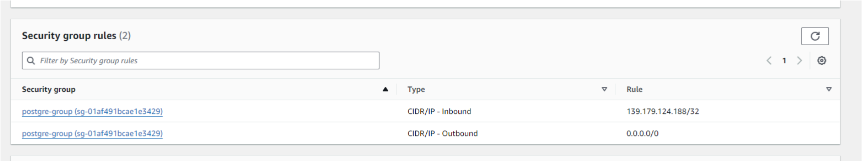
Security

VPC security groups: postgre-group (sg-01af491bcae1e3429) Active

Publicly accessible: Yes

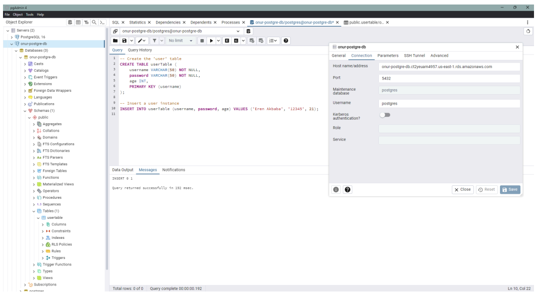
Certificate authority: rds-ca-rsa2048-g1

DB instance certificate expiration date: May 13, 2025, 13:03 (UTC+03:00)



Security group rules (2)

Security group	Type	Rule
postgre-group (sg-01af491bcae1e3429)	CIDR/IP - Inbound	139.179.124.188/32
postgre-group (sg-01af491bcae1e3429)	CIDR/IP - Outbound	0.0.0.0/0



```
-- Create the 'user' table
CREATE TABLE userTable (
    username VARCHAR(50) NOT NULL,
    password VARCHAR(50) NOT NULL,
    age INT,
    PRIMARY KEY (username)
);

-- Insert a user instance
INSERT INTO userTable (username, password, age) VALUES ('Eren Akbaba', '12345', 21);
```

```
INSERT 0 1

Query returned successfully in 192 msec.
```

onur-postgre-db — Connection

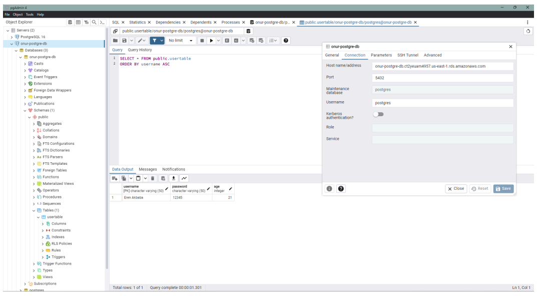
Host name/address: onur-postgre-db.ct2yeuam4957.us-east-1.rds.amazonaws.com

Port: 5432

Maintenance database: postgres

Username: postgres

Total rows: 0 of 0 Query complete 00:00:00.192 Ln 10, Col 22



```
SELECT * FROM public.usertable
ORDER BY username ASC
```

	username [PK] character varying (50)	password character varying (50)	age integer
1	Eren Akbaba	12345	21

onur-postgre-db — Connection

Host name/address: onur-postgre-db.ct2yeuam4957.us-east-1.rds.amazonaws.com

Port: 5432

Maintenance database: postgres

Username: postgres

Total rows: 1 of 1 Query complete 00:00:01.301 Ln 1, Col 1

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.

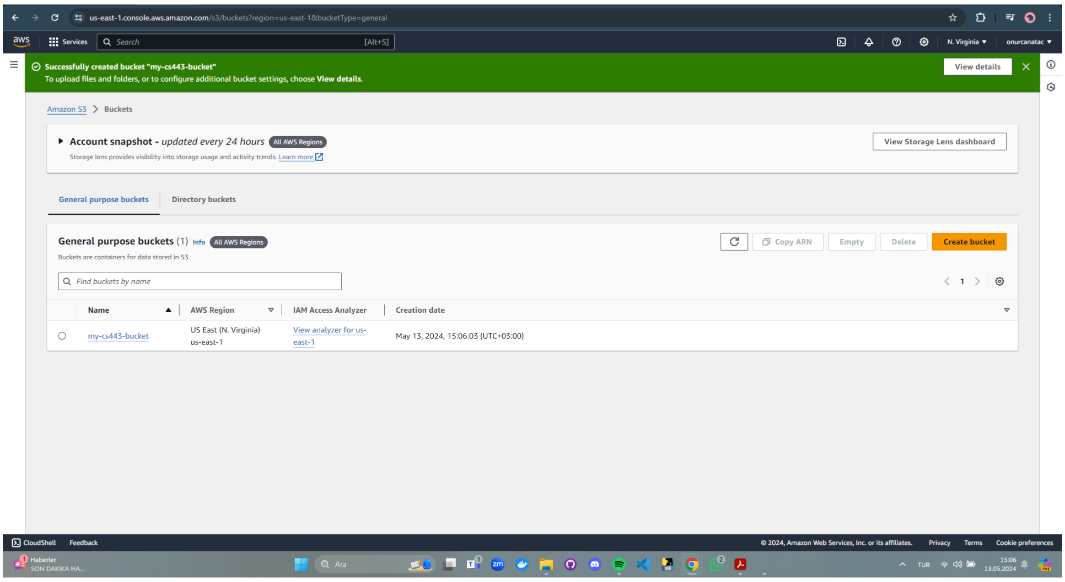
CloudWatch is very useful for incident response since it keeps track of metrics, and can send notifications or alarms when a preset emergency issue occurs. It can also take direct preset actions. These are extremely valuable, especially at night time when the customer is not aware.

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.



us-east-1.console.aws.amazon.com/s3/buckets?region=us-east-1&bucketType=general

Successfully created bucket "my-cs443-bucket"
To upload files and folders, or to configure additional bucket settings, choose **View details**.

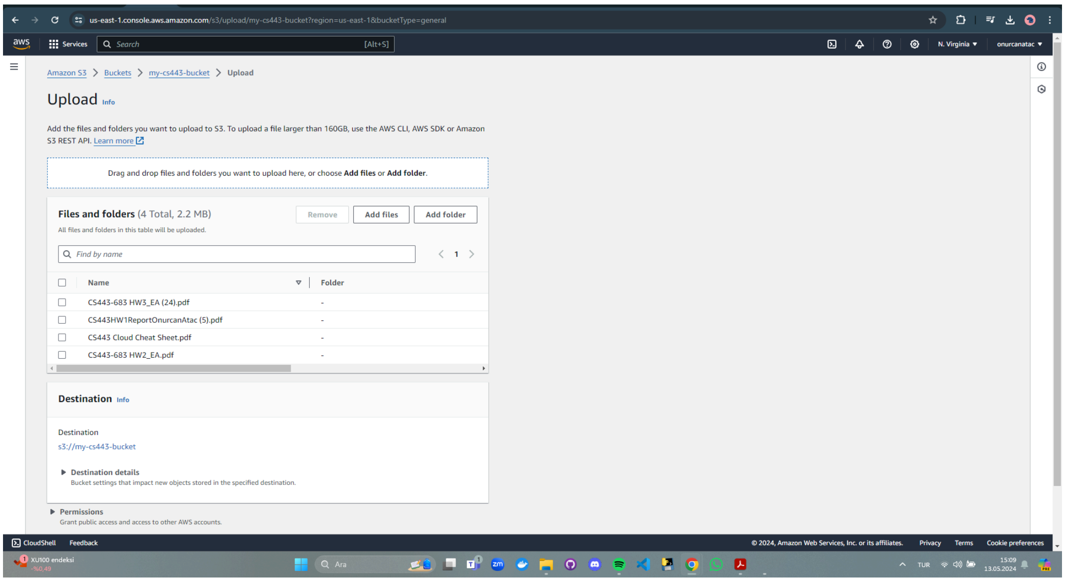
Amazon S3 > Buckets

Account snapshot - *updated every 24 hours* All AWS Regions
Storage lens provides visibility into storage usage and activity trends. Learn more

General purpose buckets (1) Info All AWS Regions
Buckets are containers for data stored in S3.

Name	AWS Region	IAM Access Analyzer	Creation date
○ my-cs443-bucket	US East (N. Virginia) us-east-1	View analyzer for us-east-1	May 13, 2024, 15:06:03 (UTC+03:00)

© 2024, Amazon Web Services, Inc. or its affiliates.



us-east-1.console.aws.amazon.com/s3/upload/my-cs443-bucket?region=us-east-1&bucketType=general

Amazon S3 > Buckets > my-cs443-bucket > Upload

Upload Info

Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. Learn more

Drag and drop files and folders you want to upload here, or choose **Add files** or **Add folder**.

Files and folders (4 Total, 2.2 MB)
All files and folders in this table will be uploaded.

	Name	Folder
□	CS443-683 HW3_EA (24).pdf	-
□	CS443HW1ReportOnurcanAtac (5).pdf	-
□	CS443 Cloud Cheat Sheet.pdf	-
□	CS443-683 HW2_EA.pdf	-

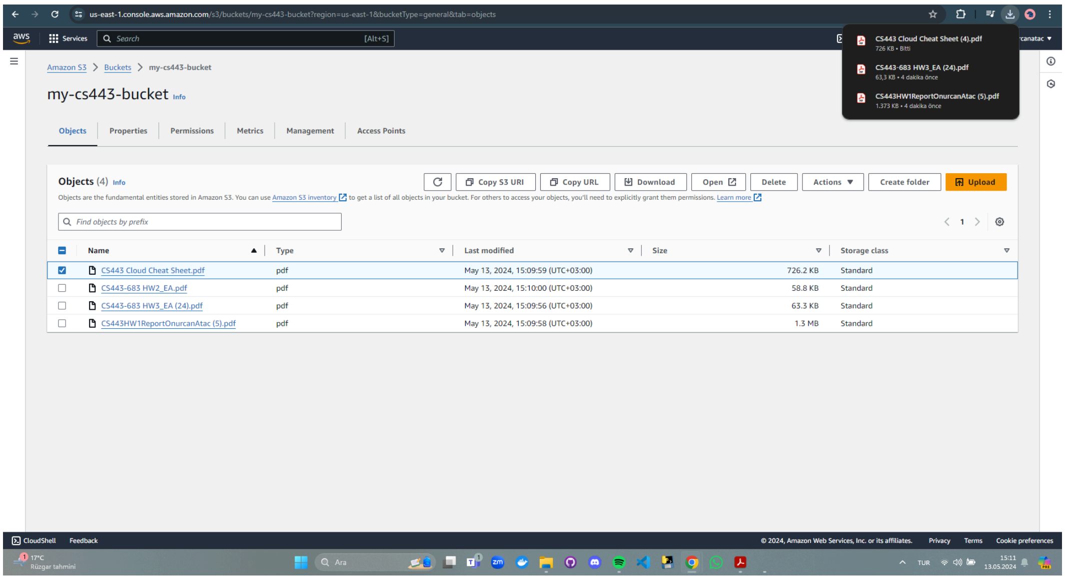
Destination Info

Destination
s3://my-cs443-bucket

Destination details
Bucket settings that impact new objects stored in the specified destination.

Permissions
Grant public access and access to other AWS accounts.

© 2024, Amazon Web Services, Inc. or its affiliates.



us-east-1.console.aws.amazon.com/s3/buckets/my-cs443-bucket?region=us-east-1&bucketType=general&tab=objects

Amazon S3 > Buckets > my-cs443-bucket

my-cs443-bucket Info

Objects (4) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

	Name	Type	Last modified	Size	Storage class
☑	CS443 Cloud Cheat Sheet.pdf	pdf	May 13, 2024, 15:09:59 (UTC+03:00)	726.2 KB	Standard
□	CS443-683 HW2_EA.pdf	pdf	May 13, 2024, 15:10:00 (UTC+03:00)	58.8 KB	Standard
□	CS443-683 HW3_EA (24).pdf	pdf	May 13, 2024, 15:09:56 (UTC+03:00)	63.3 KB	Standard
□	CS443HW1ReportOnurcanAtac (5).pdf	pdf	May 13, 2024, 15:09:58 (UTC+03:00)	1.3 MB	Standard

CS443 Cloud Cheat Sheet (4).pdf
726 KB • Bitti

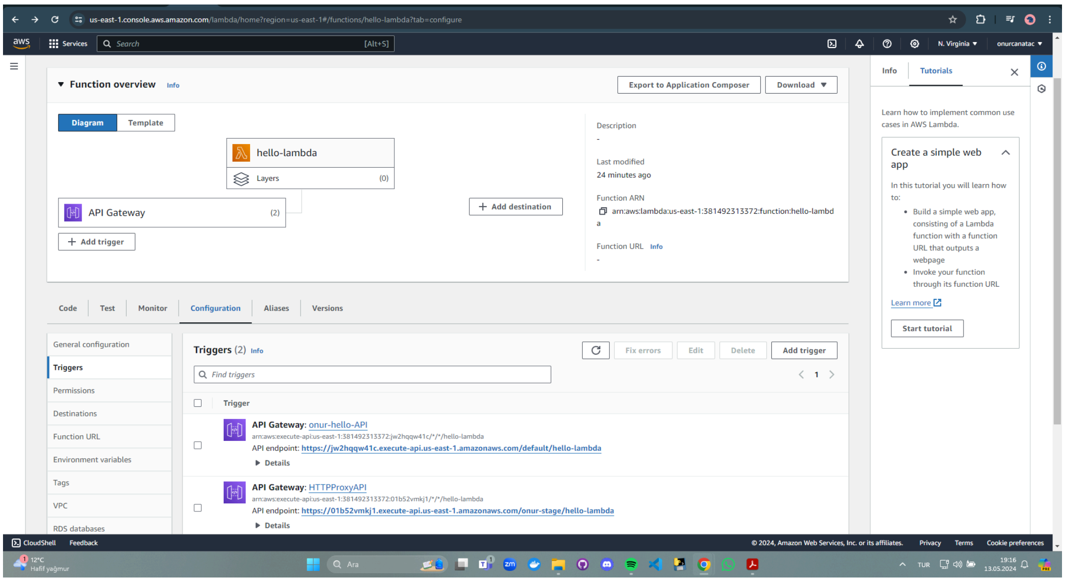
CS443-683 HW3_EA (24).pdf
63,3 KB • 4 dakika önce

CS443HW1ReportOnurcanAtac (5).pdf
1.373 KB • 4 dakika önce

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.



us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/hello-lambda?tab=configure

Function overview Info

hello-lambda

Layers (0)

API Gateway (2)

Description
-

Last modified
24 minutes ago

Function ARN
arn:aws:lambda:us-east-1:381492313372:function:hello-lambda

Function URL Info
-

Triggers (2) Info

API Gateway: onur-hello-API
arn:aws:execute-api:us-east-1:381492313372:jw2hqqw41c/*/*/hello-lambda
API endpoint: https://jw2hqqw41c.execute-api.us-east-1.amazonaws.com/default/hello-lambda

API Gateway: HTTPProxyAPI
arn:aws:execute-api:us-east-1:381492313372:01b52vmkj1/*/*/hello-lambda
API endpoint: https://01b52vmkj1.execute-api.us-east-1.amazonaws.com/onur-stage/hello-lambda

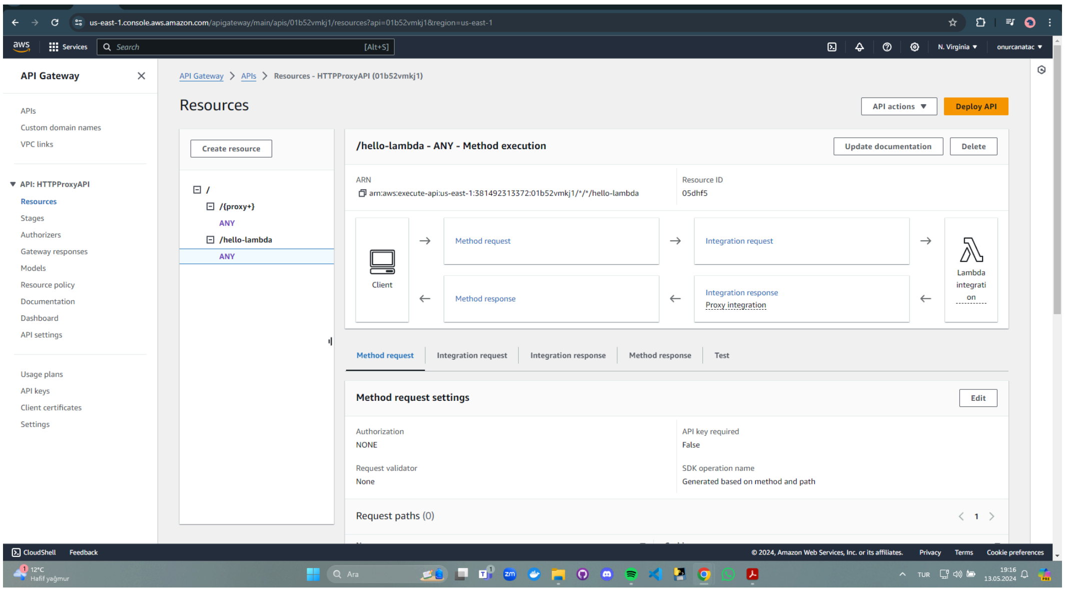
Learn how to implement common use cases in AWS Lambda.

Create a simple web app

In this tutorial you will learn how to:

- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

p.



/hello-lambda - ANY - Method execution

ARN
arn:aws:execute-api:us-east-1:381492313372:01b52vmkj1/*/*/hello-lambda

Resource ID
05dhf5

Method request settings

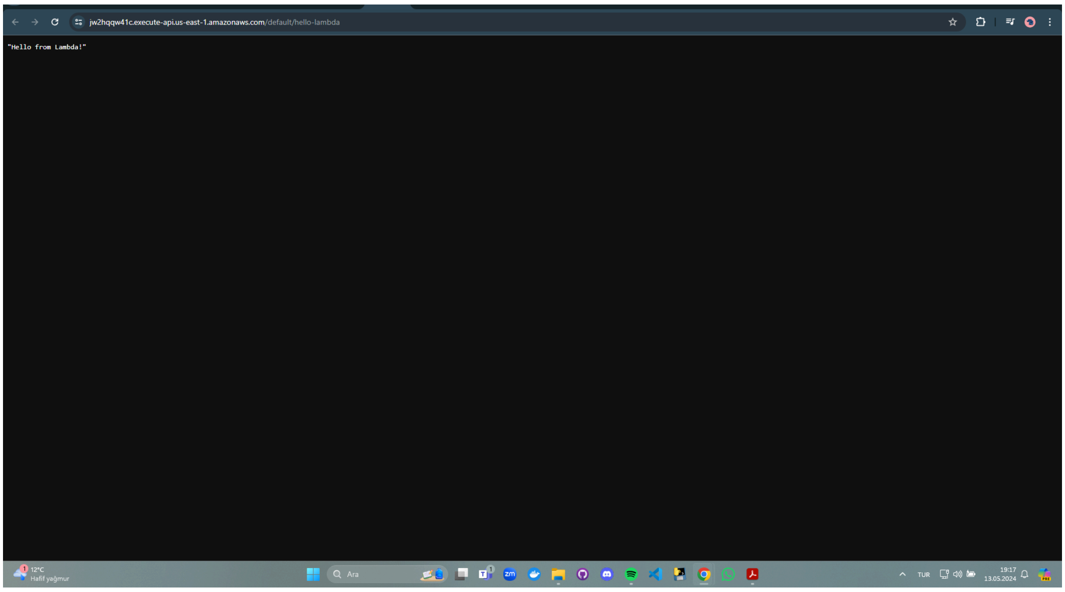
Authorization
NONE

API key required
False

Request validator
None

SDK operation name
Generated based on method and path

Request paths (0)


jw2hqqw41c.execute-api.us-east-1.amazonaws.com/default/hello-lambda
12°C
Hafif yağmur
Ara
TUR
19:17
13.05.2024

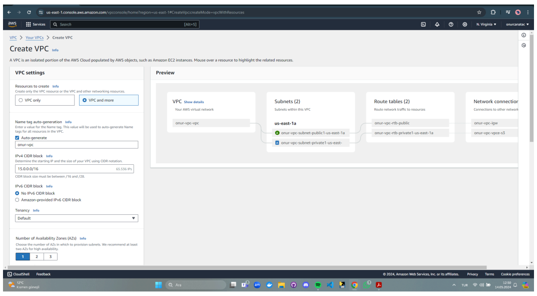
```
"Hello from Lambda!"
```

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.



Create VPC

A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Amazon EC2 instances. Mouse over a resource to highlight the related resources.

VPC settings

Resources to create: ○ VPC only ● VPC and more

Name tag auto-generation: ☑ Auto-generate: onur-vpc

IPv4 CIDR block: 15.0.0.0/16 (65,536 IPs)

CIDR block size must be between /16 and /28.

IPv6 CIDR block: ● No IPv6 CIDR block ○ Amazon-provided IPv6 CIDR block

Tenancy: Default

Number of Availability Zones (AZs): 1 | 2 | 3

Choose the number of AZs in which to provision subnets. We recommend at least two AZs for high availability.

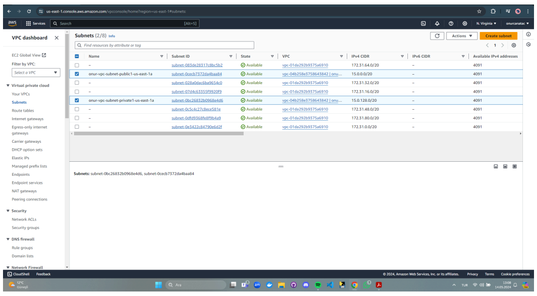
Preview

VPC — Your AWS virtual network: onur-vpc-vpc

Subnets (2) — Subnets within this VPC: us-east-1a: onur-vpc-subnet-public1-us-east-1a, onur-vpc-subnet-private1-us-east-

Route tables (2) — Route network traffic to resources: onur-vpc-rtb-public, onur-vpc-rtb-private1-us-east-1a

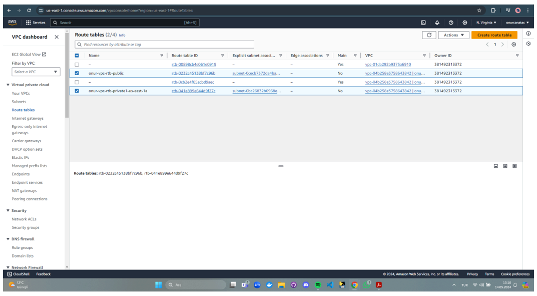
Network connection — Connections to other network: onur-vpc-igw, onur-vpc-vpce-s3



Subnets (2/8)

	Name	Subnet ID	State	VPC	IPv4 CIDR	IPv6 CIDR	Available IPv4 addresses
□	–	subnet-083de28317c8bc5b2	Available	vpc-01da292b9375a6910	172.31.64.0/20	–	4091
☑	onur-vpc-subnet-public1-us-east-1a	subnet-0cecb7372da4baa84	Available	vpc-04b258e3758643842 | onu...	15.0.0.0/20	–	4091
□	–	subnet-028a0dac6ba9654c0	Available	vpc-01da292b9375a6910	172.31.32.0/20	–	4091
□	–	subnet-07d4c63355f9920f9	Available	vpc-01da292b9375a6910	172.31.16.0/20	–	4091
☑	onur-vpc-subnet-private1-us-east-1a	subnet-0bc26832b0968e4d6	Available	vpc-04b258e3758643842 | onu...	15.0.128.0/20	–	4091
□	–	subnet-0c5c4c27c8ece581e	Available	vpc-01da292b9375a6910	172.31.48.0/20	–	4091
□	–	subnet-0dfd9368fe8f9b4a9	Available	vpc-01da292b9375a6910	172.31.80.0/20	–	4091
□	–	subnet-0e3422c84790e6d2f	Available	vpc-01da292b9375a6910	172.31.0.0/20	–	4091

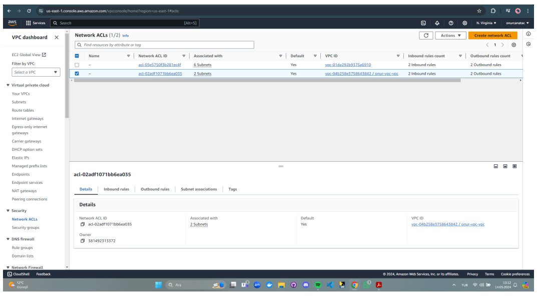
Subnets: subnet-0bc26832b0968e4d6, subnet-0cecb7372da4baa84



Route tables (2/4)

	Name	Route table ID	Explicit subnet associ...	Edge associations	Main	VPC	Owner ID
□	–	rtb-00898cb4e061e0919	–	–	Yes	vpc-01da292b9375a6910	381492313372
☑	onur-vpc-rtb-public	rtb-0232c45138bf7c96b	subnet-0cecb7372da4ba...	–	No	vpc-04b258e3758643842 | onu...	381492313372
□	–	rtb-0cb2e4f05acbd9aec	–	–	Yes	vpc-04b258e3758643842 | onu...	381492313372
☑	onur-vpc-rtb-private1-us-east-1a	rtb-041e899e644d9f27c	subnet-0bc26832b0968e...	–	No	vpc-04b258e3758643842 | onu...	381492313372

Route tables: rtb-0232c45138bf7c96b, rtb-041e899e644d9f27c



Network ACLs (1/2)

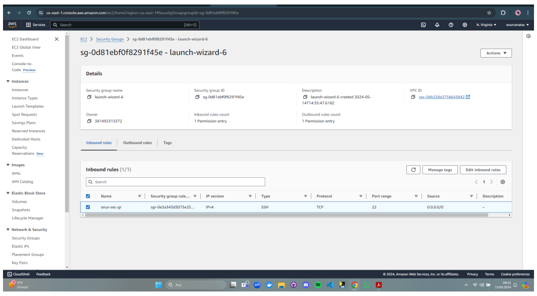
	Name	Network ACL ID	Associated with	Default	VPC ID	Inbound rules count	Outbound rules count
□	–	acl-05e5750f3b281ec4f	6 Subnets	Yes	vpc-01da292b9375a6910	2 Inbound rules	2 Outbound rules
☑	–	acl-02adf1071bb6ea035	2 Subnets	Yes	vpc-04b258e3758643842 / onur-vpc-vpc	2 Inbound rules	2 Outbound rules

acl-02adf1071bb6ea035

Details

Network ACL ID: acl-02adf1071bb6ea035
Associated with: 2 Subnets
Default: Yes
VPC ID: vpc-04b258e3758643842 / onur-vpc-vpc
Owner: 381492313372

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.



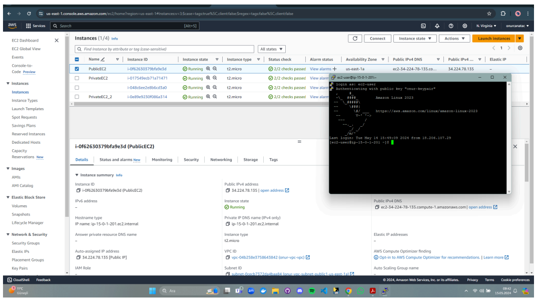
sg-0d81ebf0f8291f45e - launch-wizard-6

Details

Security group name: launch-wizard-6
Security group ID: sg-0d81ebf0f8291f45e
Description: launch-wizard-6 created 2024-05-14T14:35:47.618Z
VPC ID: vpc-04b258e3758643842
Owner: 381492313372
Inbound rules count: 1 Permission entry
Outbound rules count: 1 Permission entry

Inbound rules (1/1)

	Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
☑	onur-sec-gr	sgr-0e3a343d3073e25...	IPv4	SSH	TCP	22	0.0.0.0/0	–



Instances (1/4)

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
☑	PublicEC2	i-0f62630379bfa9e3d	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	ec2-34-224-78-135.co...	34.224.78.135	–
□	PrivateEC2	i-017549ecb71a71471	Running	t2.micro	2/2 checks passed	View alarm				
□		i-048c6ee2e8b6cd3a0	Running	t2.micro	2/2 checks passed	View alarm				
□	PrivateEC2_2	i-0e89e9230f086a314	Running	t2.micro	2/2 checks passed	View alarm				

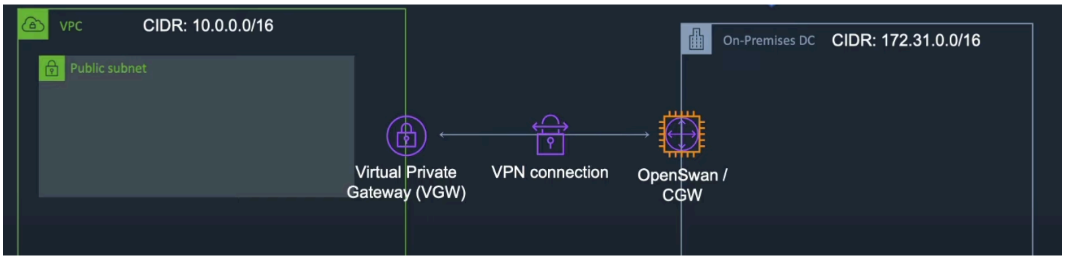
```
login as: ec2-user
Authenticating with public key "onur-keypair"
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Tue May 14 15:49:09 2024 from 18.206.107.29
[ec2-user@ip-15-0-1-201 ~]$
```

i-0f62630379bfa9e3d (PublicEC2)

Instance summary

Instance ID: i-0f62630379bfa9e3d (PublicEC2)
Public IPv4 address: 34.224.78.135 | open address
IPv6 address: –
Instance state: Running
Public IPv4 DNS: ec2-34-224-78-135.compute-1.amazonaws.com | open address
Hostname type: IP name: ip-15-0-1-201.ec2.internal
Private IP DNS name (IPv4 only): ip-15-0-1-201.ec2.internal
Answer private resource DNS name: –
Instance type: t2.micro
Elastic IP addresses: –
Auto-assigned IP address: 34.224.78.135 [Public IP]
VPC ID: vpc-04b258e3758643842 (onur-vpc-vpc)
AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role: –
Subnet ID: subnet-0cecb7372da4baa84 (onur-vpc-subnet-public1-us-east-1a)
Auto Scaling Group name: –

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.



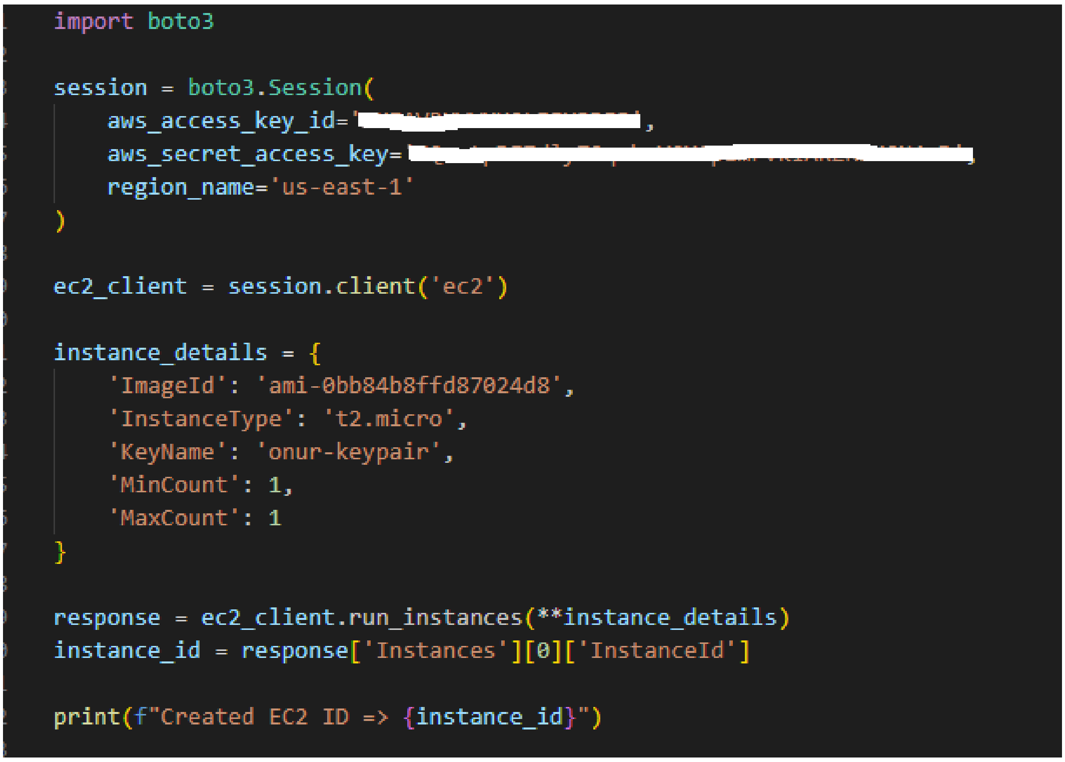
9. Boto3 Library

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.

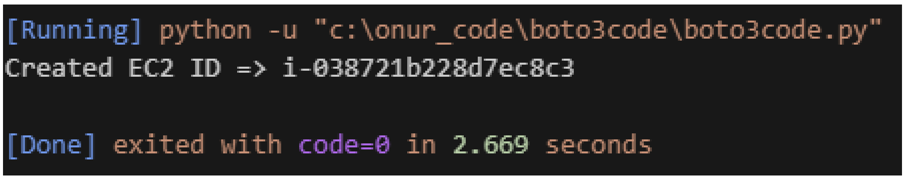
In order to use Boto3, I need a user created in IAM, and that user has to have the necessary permissions to do the task that is specified in the code. I started by creating a user named “Onurcann” on the IAM page. In order to create an EC2, I created a user group for the user and provided the following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

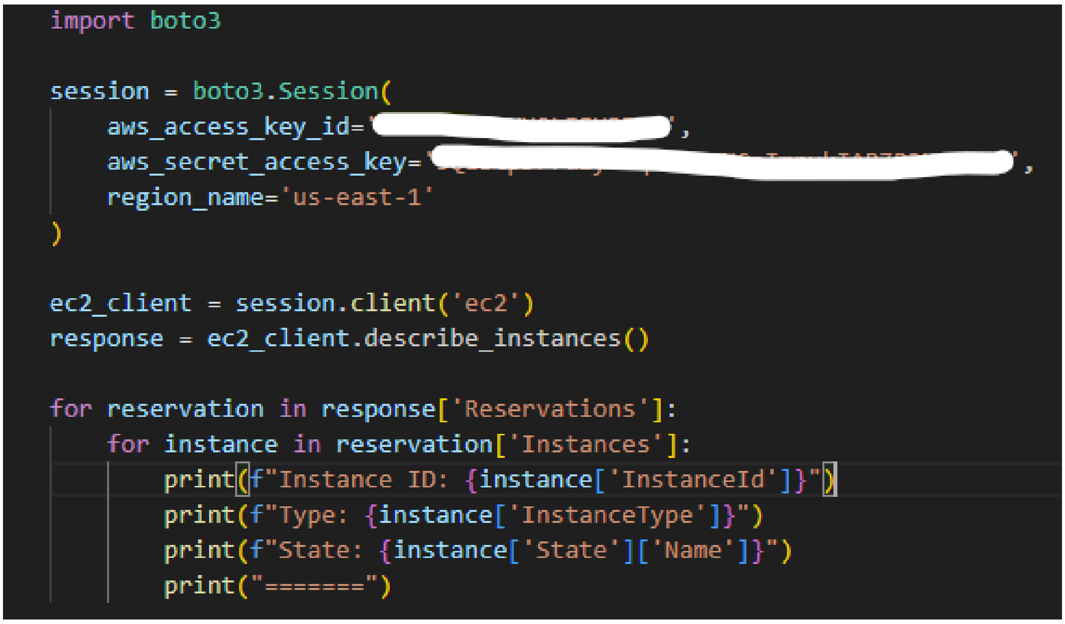


```
import boto3

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}")
```

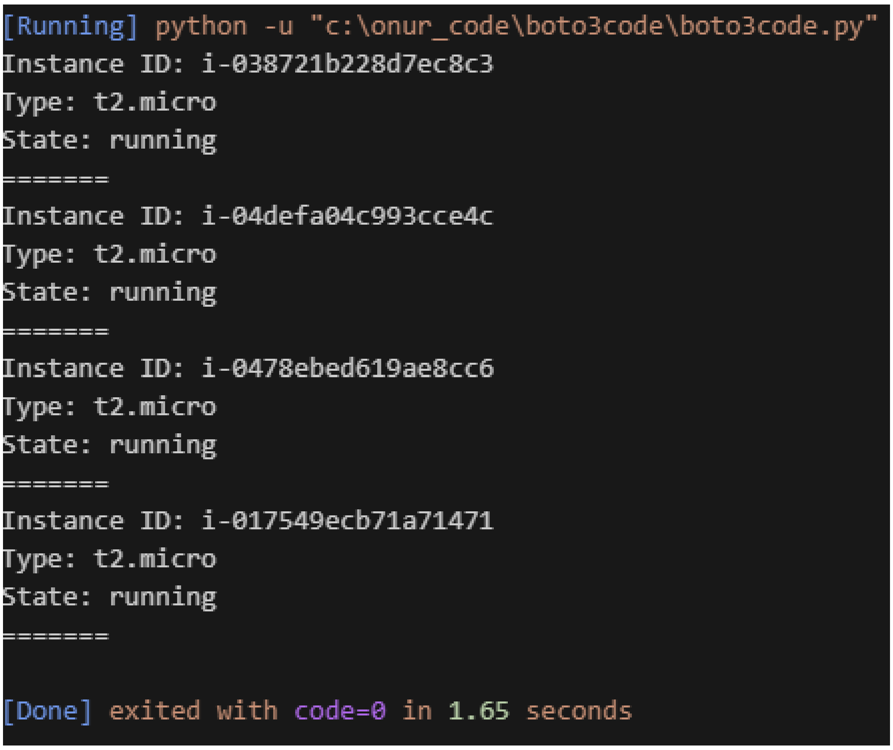


```
[Running] python -u "c:\onur_code\boto3code\boto3code.py"
Created EC2 ID => i-038721b228d7ec8c3

[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.



```
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("=======")
```



```
[Running] python -u "c:\onur_code\boto3code\boto3code.py"
Instance ID: i-038721b228d7ec8c3
Type: t2.micro
State: running
=======
Instance ID: i-04defa04c993cce4c
Type: t2.micro
State: running
=======
Instance ID: i-0478ebed619ae8cc6
Type: t2.micro
State: running
=======
Instance ID: i-017549ecb71a71471
Type: t2.micro
State: running
=======

[Done] exited with code=0 in 1.65 seconds
```

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.