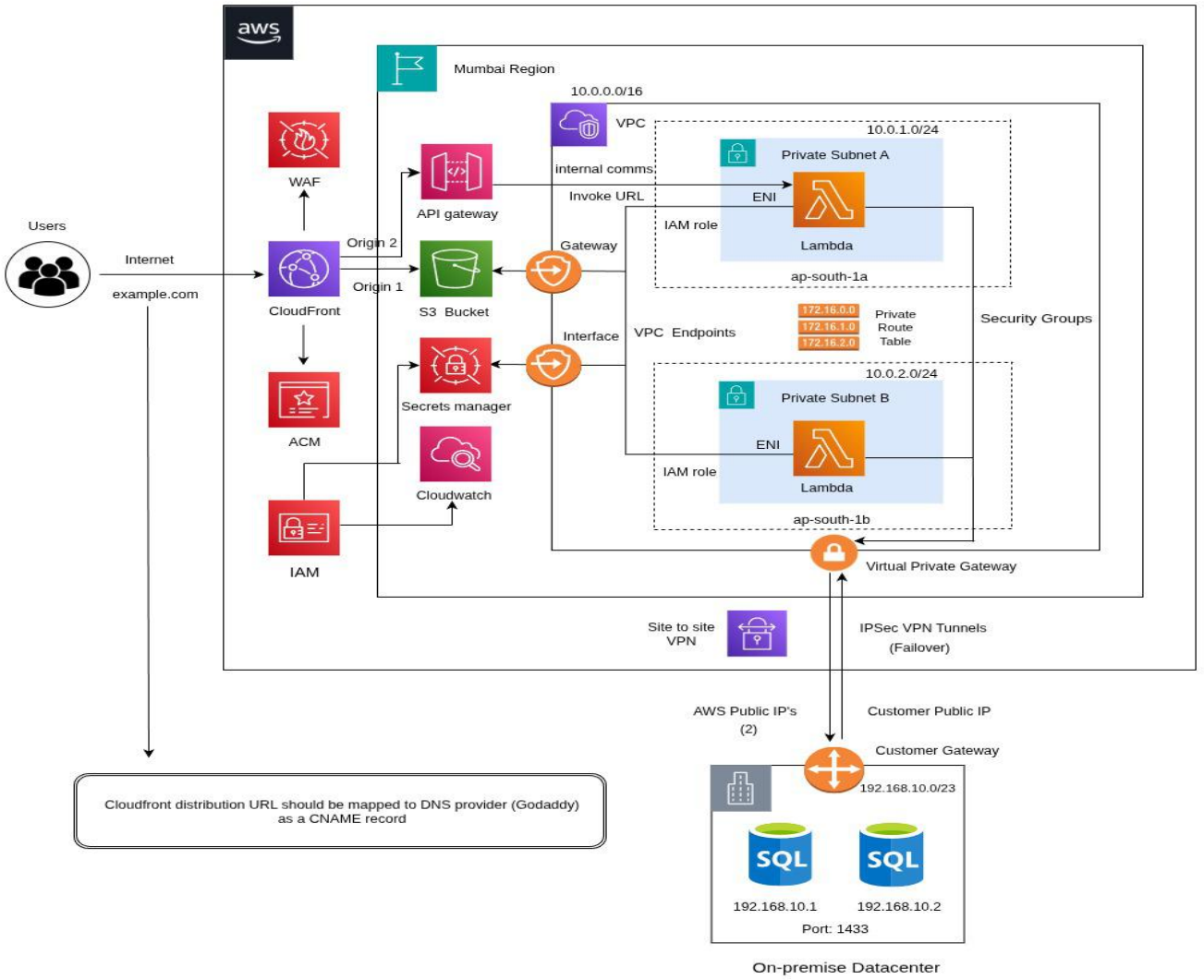
AWS Serverless Architecture for Secure Web Application with On-Premise SQL Connectivity

Project Overview & Implementation Steps

Business Need / Problem Statement

- *Need to host a secure web application that allows downloading of employee details.*
- *Store static assets centrally and serve globally with low latency.*
- *Access on-premises SQL Server securely from the cloud that contains employee data's.*
- *Ensure data security, high availability, and cost efficiency.*
- *Provide monitoring, logging, and alerts for operations.*

Architecture Diagram



Key AWS Services Used

- *Godaddy or Route53 - For DNS management*
- *S3 bucket - Storing static frontend webfiles*
- *CloudFront - CDN for low-latency global delivery through edge location*
- *ACM certificate - SSL/TLS certificate for HTTPS*
- *AWS WAF - Protects against common web attacks*
- *API Gateway (HTTP) - Private API backed by Lambda functions*
- *Lambda - Serverless compute, runs app logic*
- *Pymssql driver - Driver to connect SQL server with lambda*
- *MSSQL server - Database for storing the user details*
- *VPC with Private Subnets - Secure networking for Lambda*
- *VPC endpoints - Private connection to AWS services for S3 (gateway), CloudWatch / Secrets Manager (interface endpoints)*
- *Secrets Manager - Secure storage of DB credentials*
- *CloudWatch - Logging, metrics, alarms*
- *Virtual Private Gateway with Site to site VPN connection - Secure tunnel to on-prem SQL Server*

1. SQL server Database Setup (On-premise)

A. Install SQL Server

To configure SQL Server on Ubuntu, run the following commands in a terminal to install the mssql-server package.

1. Download the public key, convert from ASCII to GPG format, and write it to the required location:

curl -fsSL https://packages.microsoft.com/keys/microsoft.asc | sudo gpg --dearmor -o /usr/share/keyrings/microsoft-prod.gpg

2. Manually download and register the SQL Server Ubuntu repository:

curl -fsSL https://packages.microsoft.com/config/ubuntu/24.04/mssql-server-preview.list | sudo tee /etc/apt/sources.list.d/mssql-server-preview.list

3. Run the following commands to install SQL Server:

sudo apt-get update

sudo apt-get install -y mssql-server

4. After the package installation finishes, run **mssql-conf setup** for the following configurations

sudo /opt/mssql/bin/mssql-conf setup

5. Once the configuration is done, verify that the service is running:

systemctl start mssql-server

systemctl enable mssql-server

systemctl status mssql-server

B. Install and connect the SQL Server using command-line tools

To create a database, you need to connect with a tool that can run Transact-SQL statements on SQL Server. The following steps install the SQL Server command-line tools:

Use the following steps to install the mssql-tools18 for SQL Server 2025 (17.x) Preview on Ubuntu 24.04 in preview.

1. Register the Microsoft repository for Ubuntu 24.04.

   curl -sSL -O https://packages.microsoft.com/config/ubuntu/24.04/packages-microsoft-prod.deb

2. Install the repository package:

   sudo dpkg -i packages-microsoft-prod.deb

3. Update the sources list and run the installation command with the unixODBC developer package.

   sudo apt-get update

   sudo apt-get install mssql-tools18 unixodbc-dev

4. Optional: Add /opt/mssql-tools18/bin/ to your PATH environment variable in a bash shell.

   echo 'export PATH="$PATH:/opt/mssql-tools18/bin"' >> ~/.bash_profile

   source ~/.bash_profile

   echo 'export PATH="$PATH:/opt/mssql-tools18/bin"' >> ~/.bashrc

   source ~/.bashrc

5. use sqlcmd to locally connect to your new SQL Server instance.

   sqlcmd -S localhost -U sa -P -C

C. Create a employee database & table with following commands

1. To view a list of databases on an SQL Server

```
SELECT name, database_id, create_date
FROM sys.databases;
GO
```

2. Create a new database

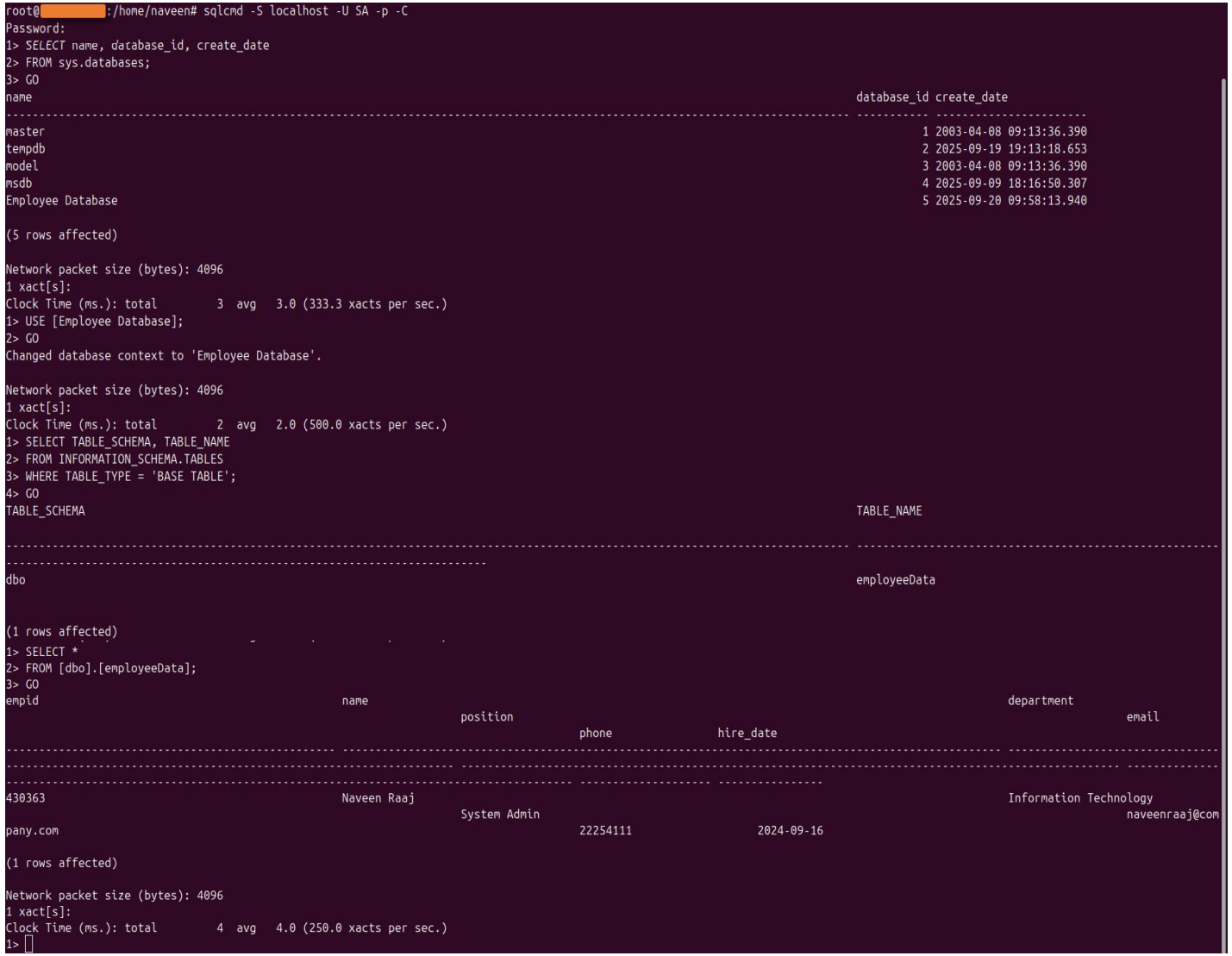
```
CREATE DATABASE Employee Database;
GO
```

3. Switch to Employee Database

```
USE [Employee Database];
GO
```

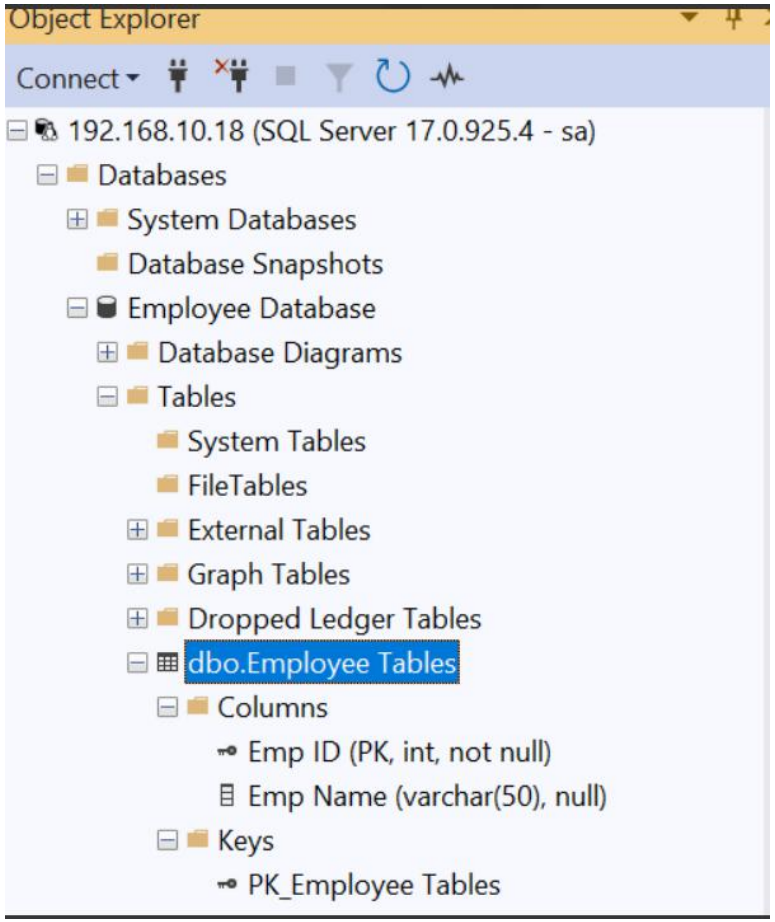
4. Create a new table (employeeData)

```
SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE';
GO
```



5. Create a new table (employeeData)

```
CREATE TABLE employeeData (
    empid INT PRIMARY KEY,
    name NVARCHAR(100) NOT NULL,
    department NVARCHAR(100),
    position NVARCHAR(100),
    email NVARCHAR(255),
    phone NVARCHAR(50),
    hire_date DATE
);
GO
```



6. View All Data (Columns + Rows)

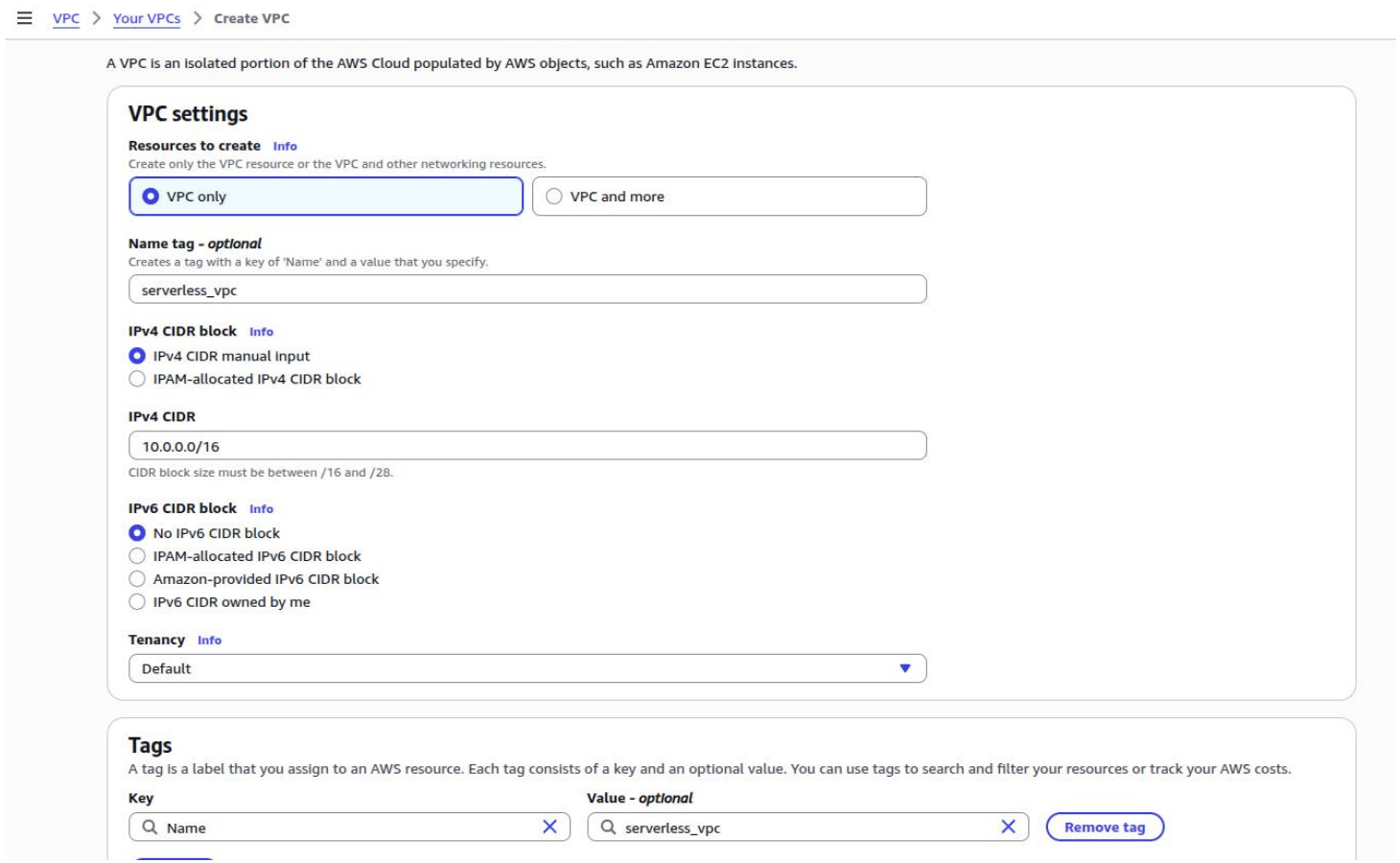
```
SELECT *
FROM [dbo].[employeeData];
GO
```

7. Manually insert the data's into the table.

```
INSERT INTO [dbo].[employeeData] (empid, name, department, position, email, phone, hire_date)
VALUES (101, 'John Doe', 'IT', 'Developer', 'john.doe@example.com', '9876543210', '2025-09-22');
GO
```

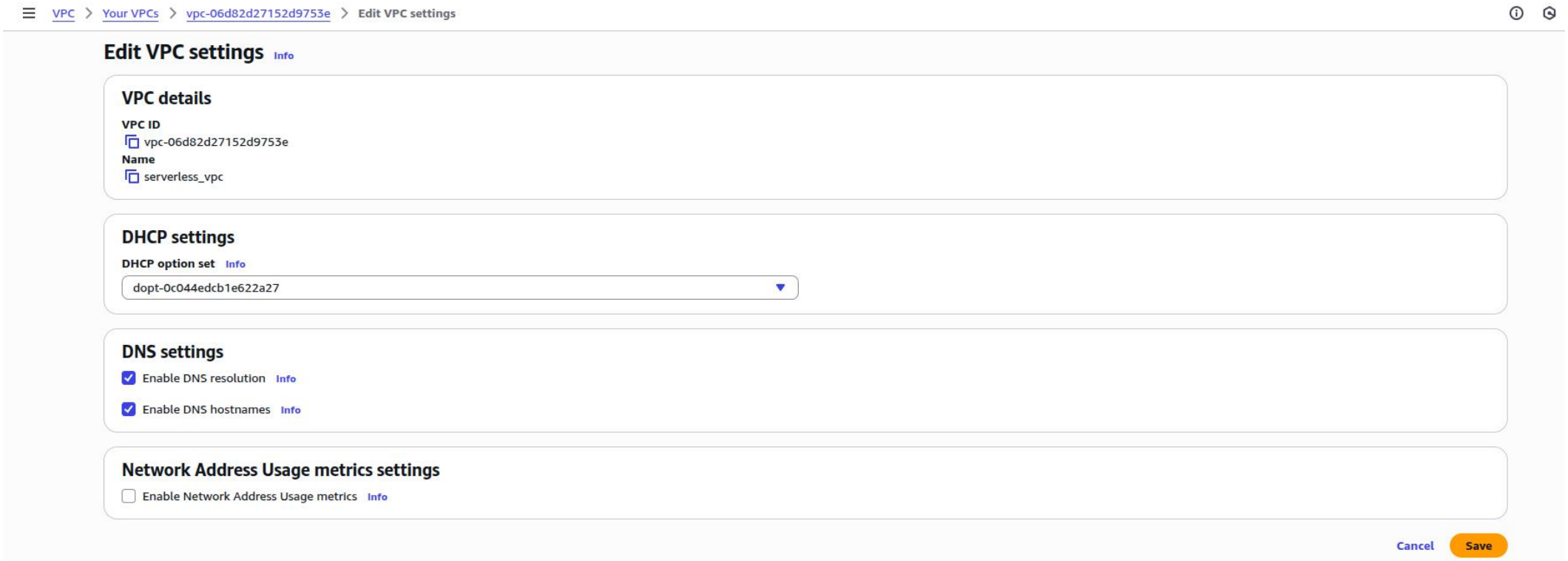
2. Foundation: VPC, subnets, route tables and security groups

1. Go to AWS console --> Virtual private cloud --> Your VPC's --> Create VPC **(CIDR e.g. 10.0.0.0/16)**



(b).Make sure to enable DNS resolution and hostname. Allows resources in your VPC to resolve public DNS names (like amazon.com, api.github.com) to IP addresses.

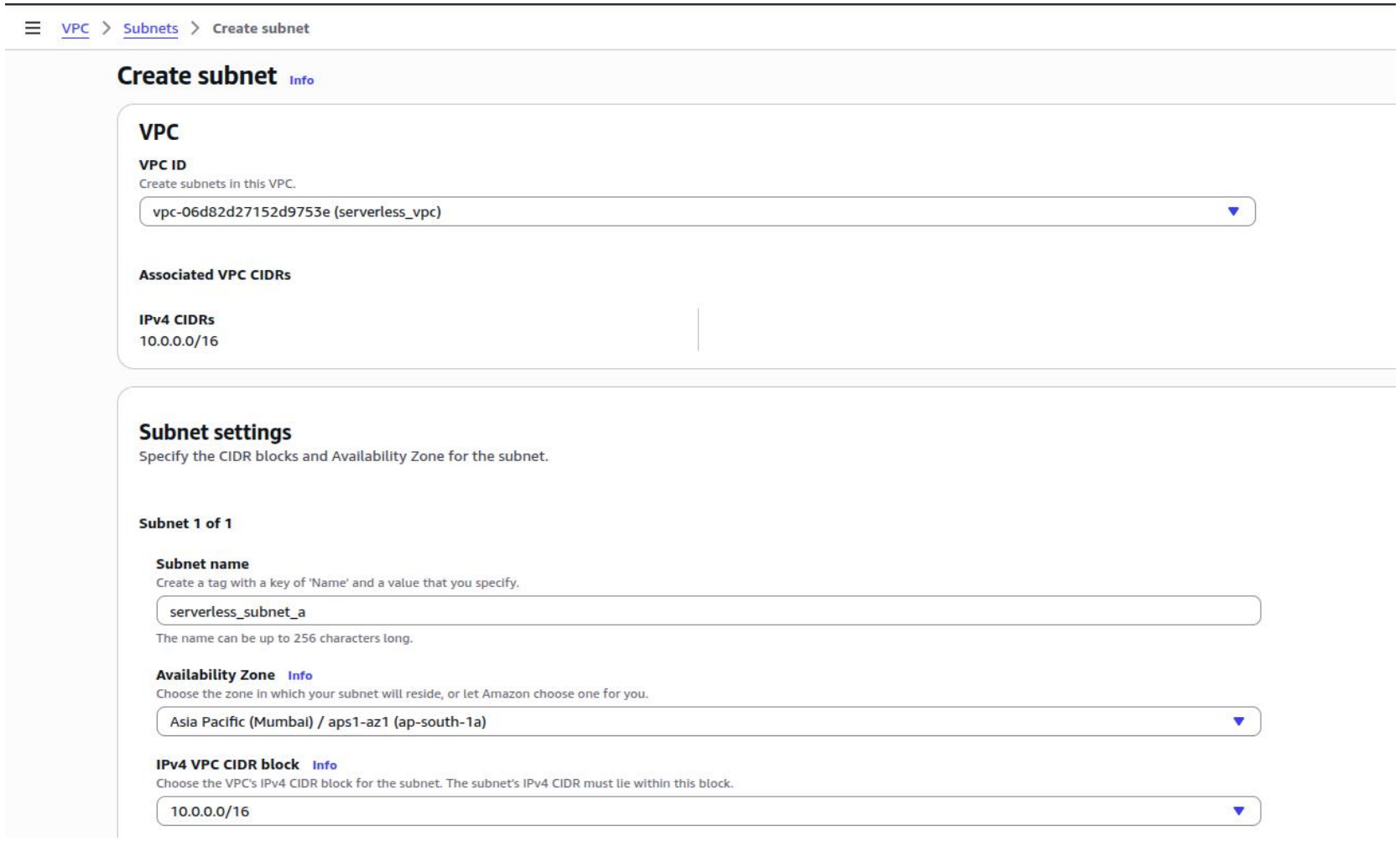
VPC Endpoints (they rely on DNS to route traffic internally)



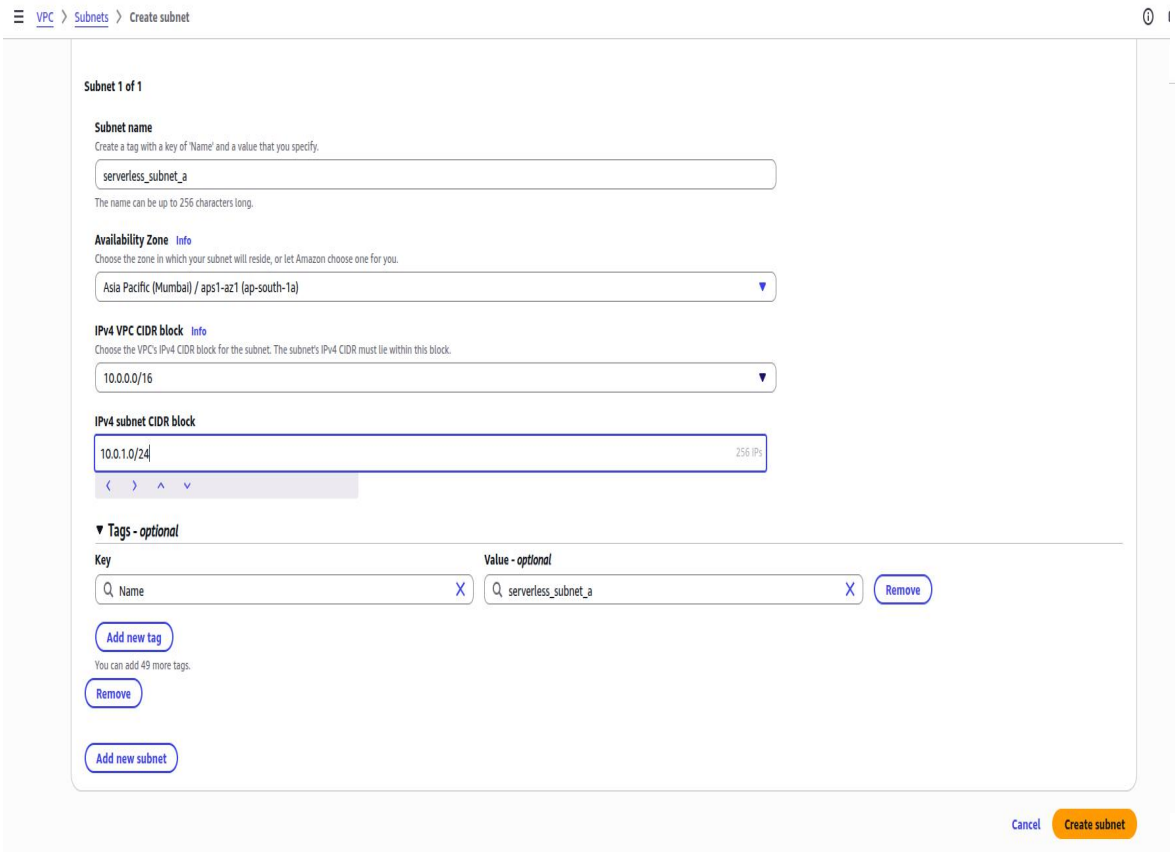
2. Create subnets:

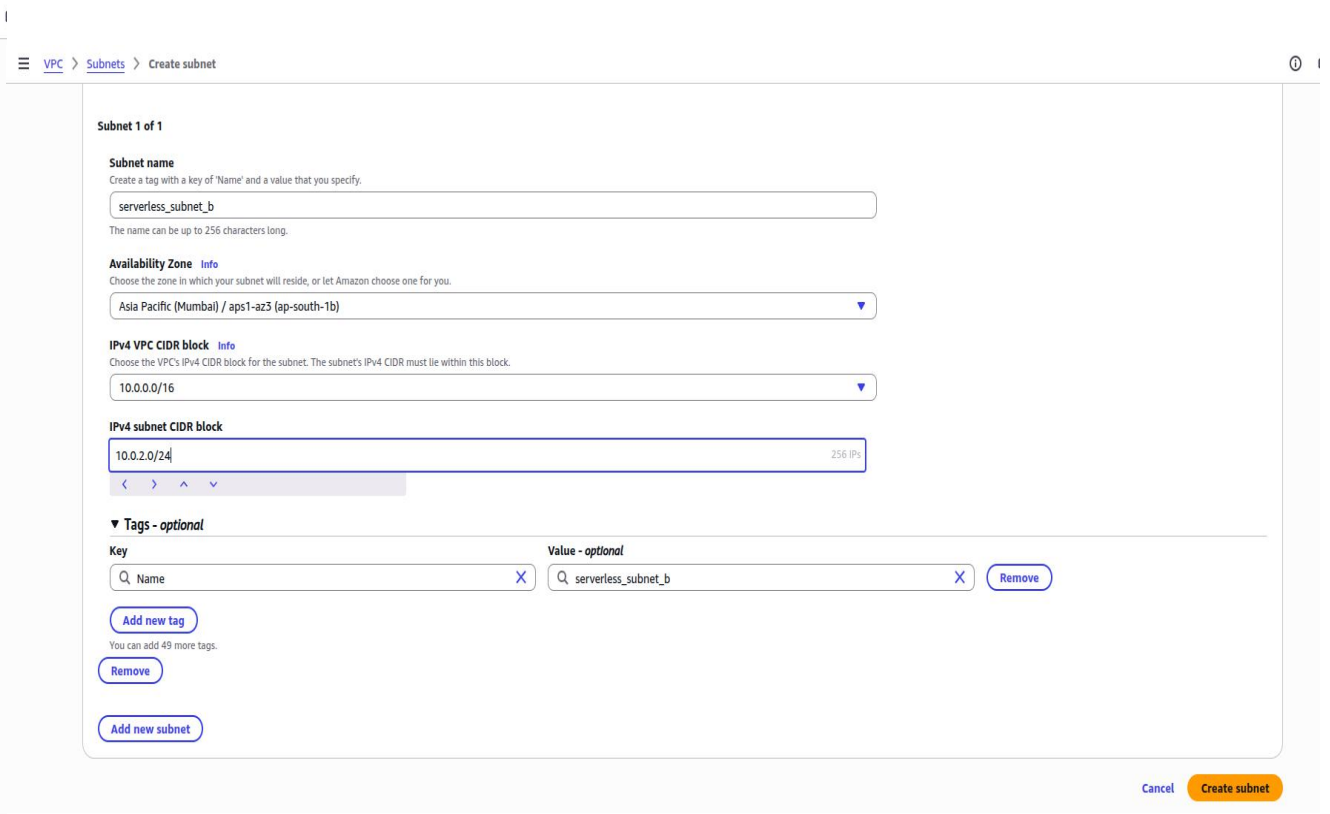
(a). Private subnets in 2 AZ's for Lambda ENIs **(e.g., 10.0.1.0/24, 10.0.2.0/24)**

Note: This architecture uses no public subnets. API Gateway and CloudFront handle public access, while Lambda and on-premise databases run in private subnets with no direct internet exposure.



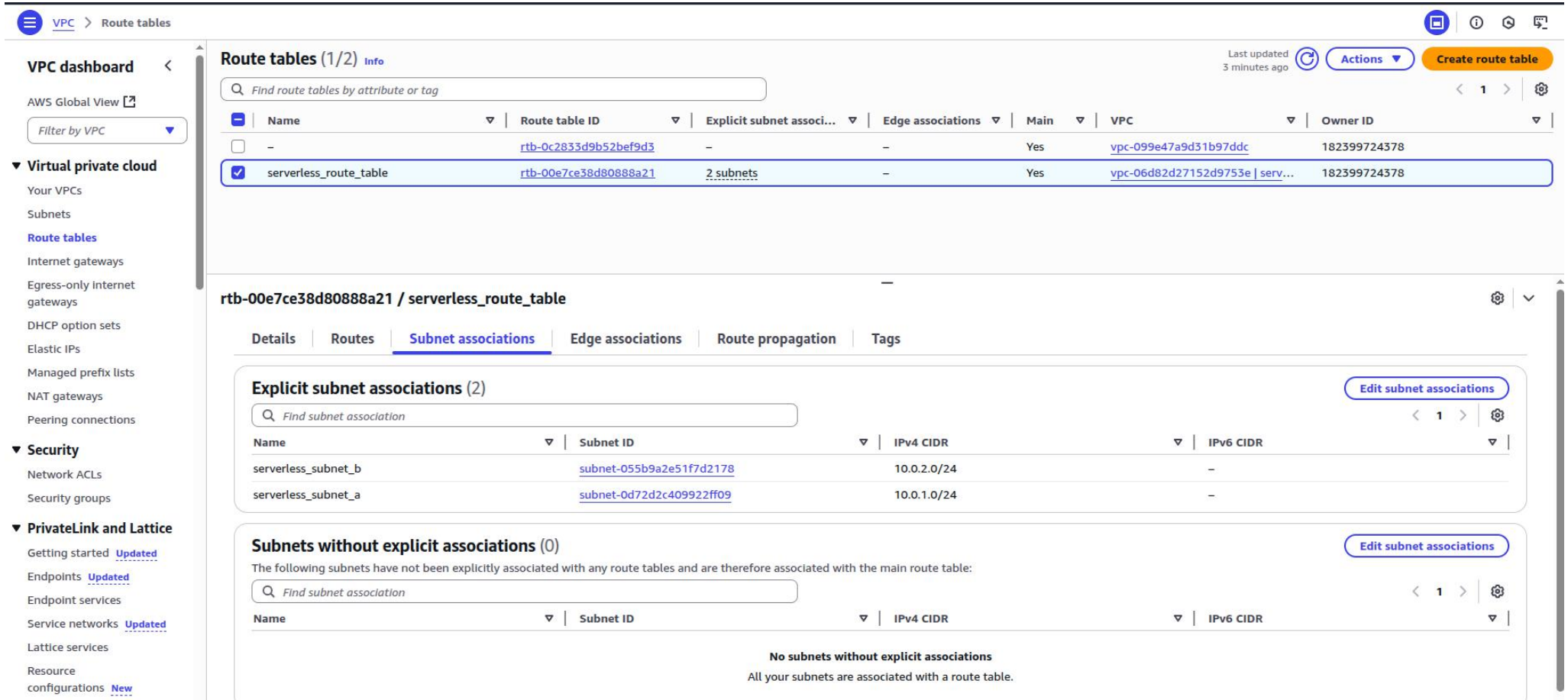
(b). Let's create subnet_a**(10.0.1.0/24)** in private availability zone **(ap-south-1a)** and subnet_b **(10.0.2.0/24)** in private availability zone **(ap-south-1b)**





3. Create Route tables:

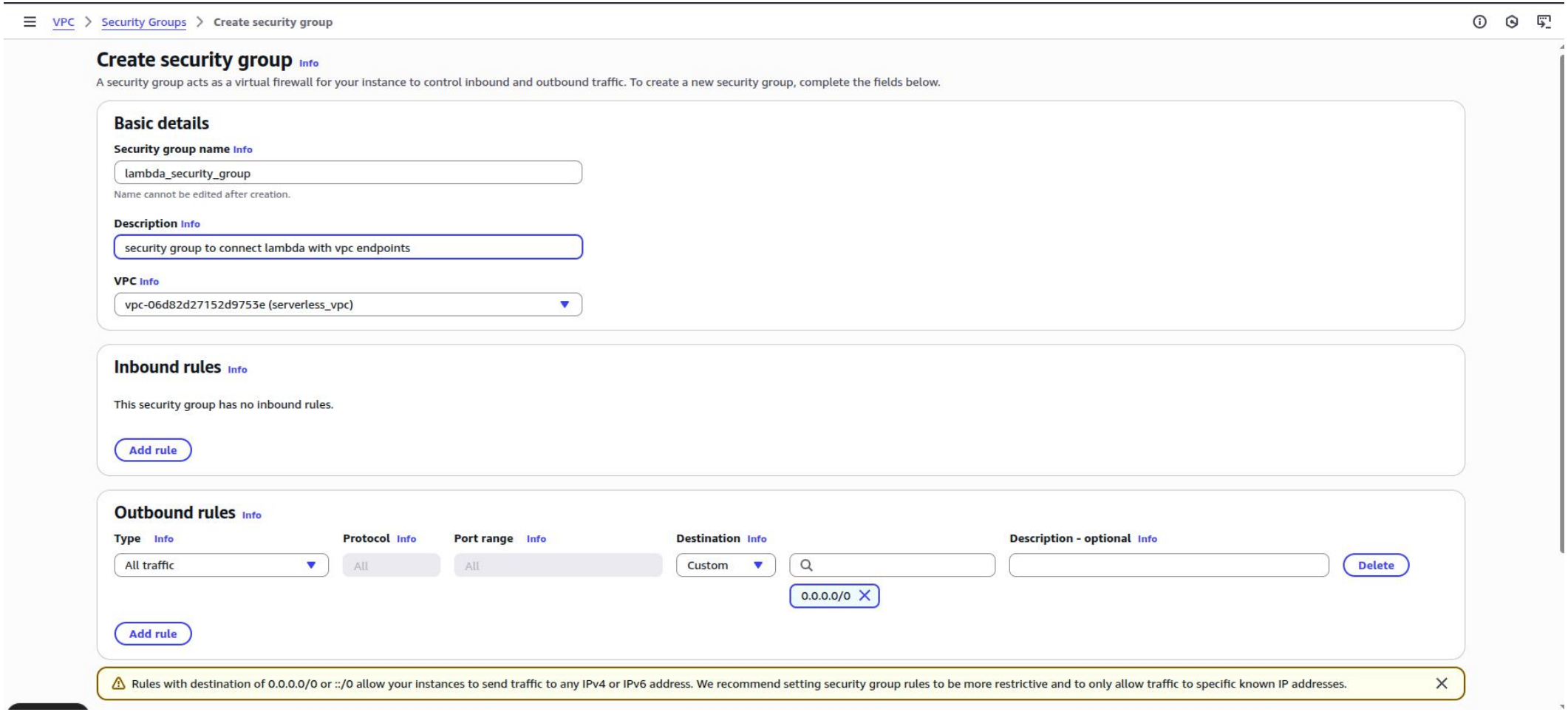
Private subnet route table: Target **(local)** to Destination Private VPC CIDR **(10.0.0.0/16)** and add explicit subnet associations



4. Create security group for accessing **lambda** inside the private VPC

Go to AWS console --> **VPC** --> **Security group** --> **Create security group**

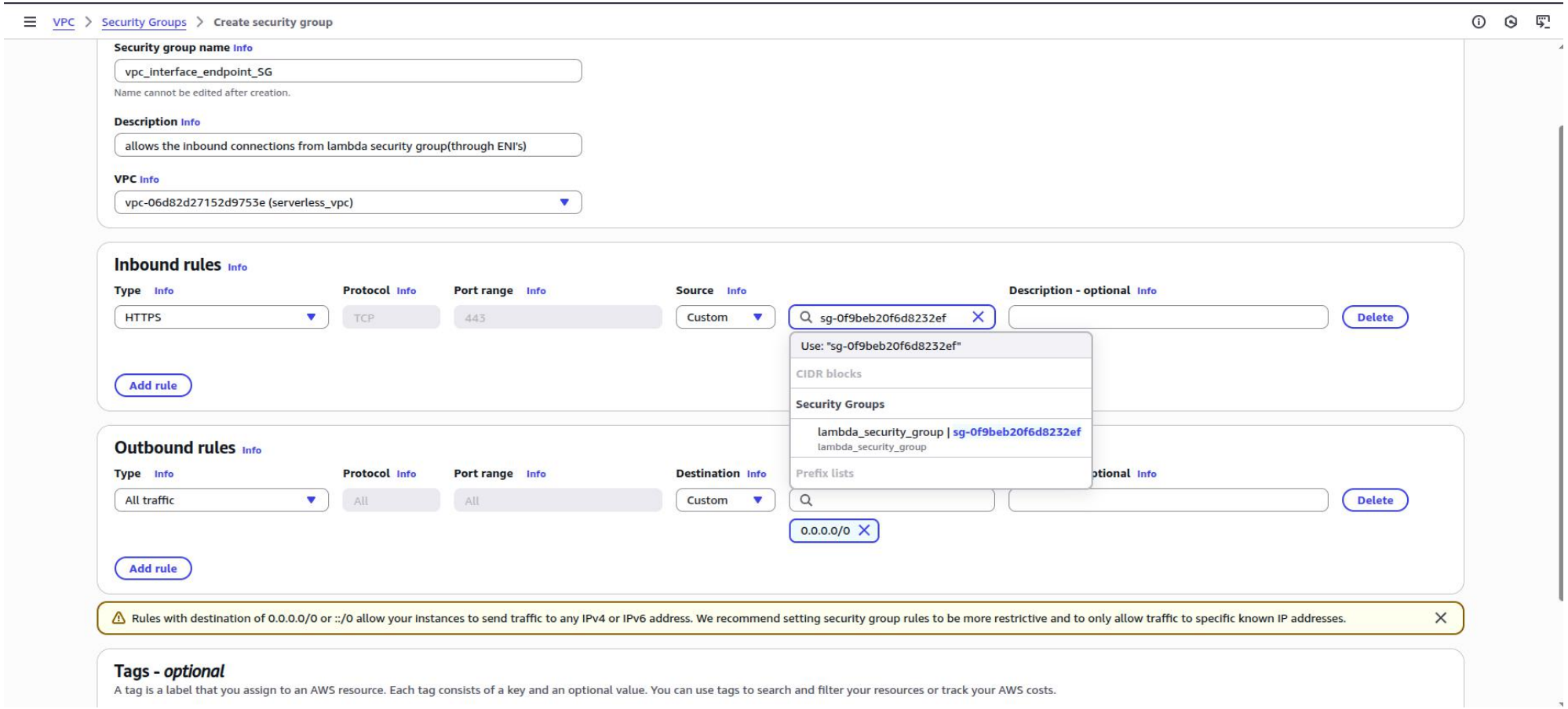
This security group is used to connect resources outside the VPC (eg., VPC endpoints, MSSQL server)



(b). Create another security group for allowing connections from lambda inside the private VPC to the **VPC endpoints**.

Inbound Rules --> Type: HTTPS and Source: Custom and select Lambda security group

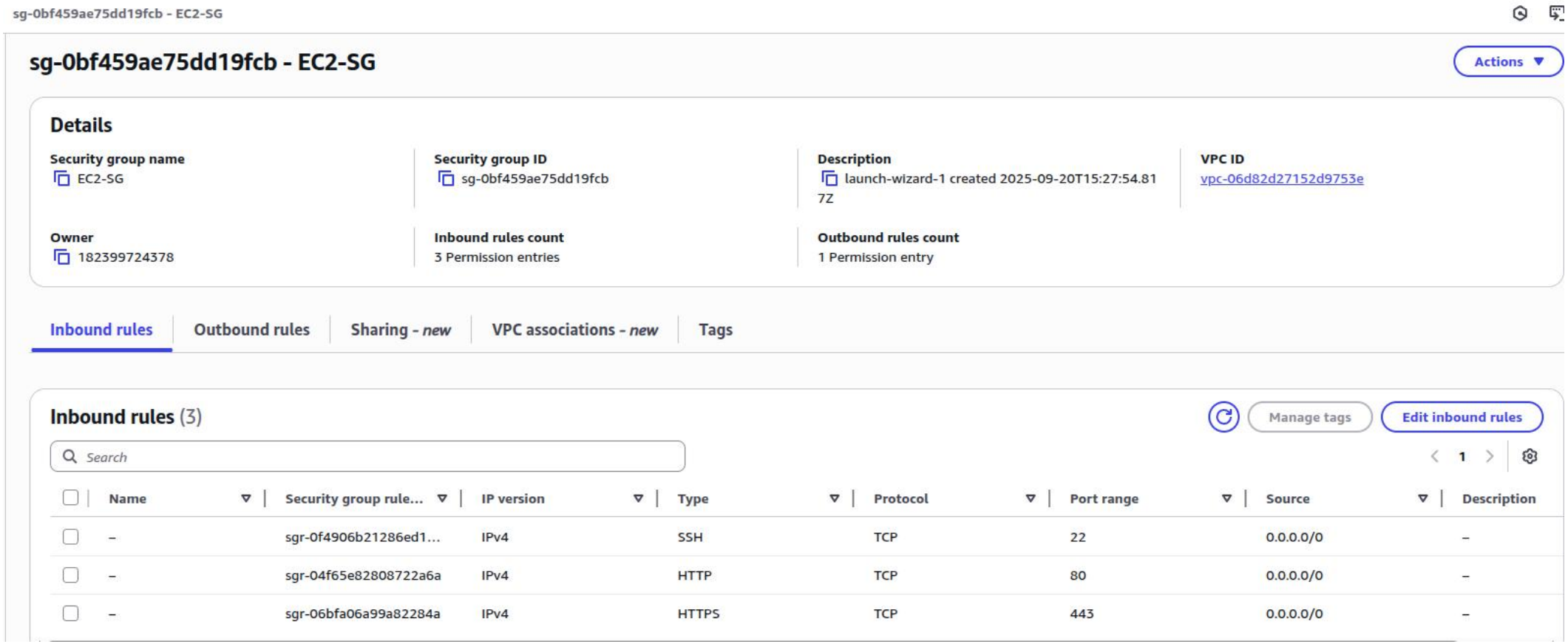
Outbound Rules --> Type: All traffic and Destination: Custom and enter 0.0.0.0/0 as the IP address



(c). Create a security group for **EC2 instance** to access the onpremise datacenter from our private VPC

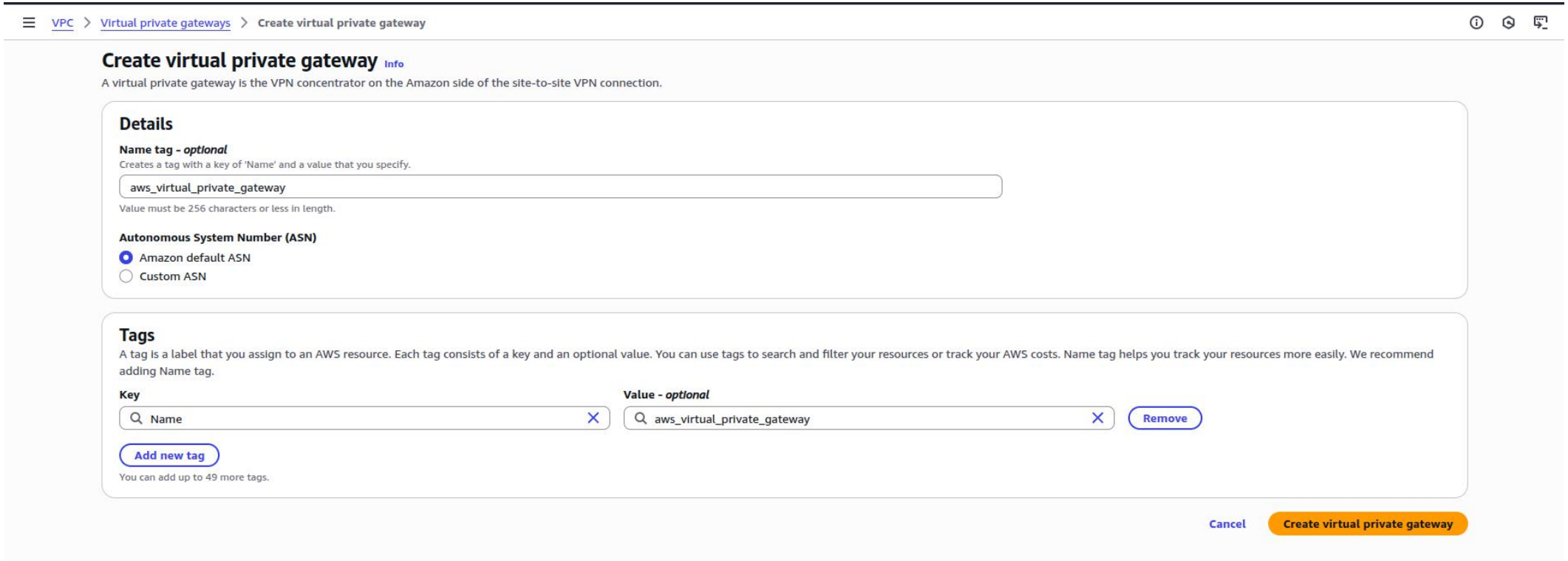
Inbound Rules --> Type: SSH, HTTP & HTTPS and Source: 0.0.0.0/0

Outbound Rules --> Type: All traffic and Destination: 0.0.0.0/0



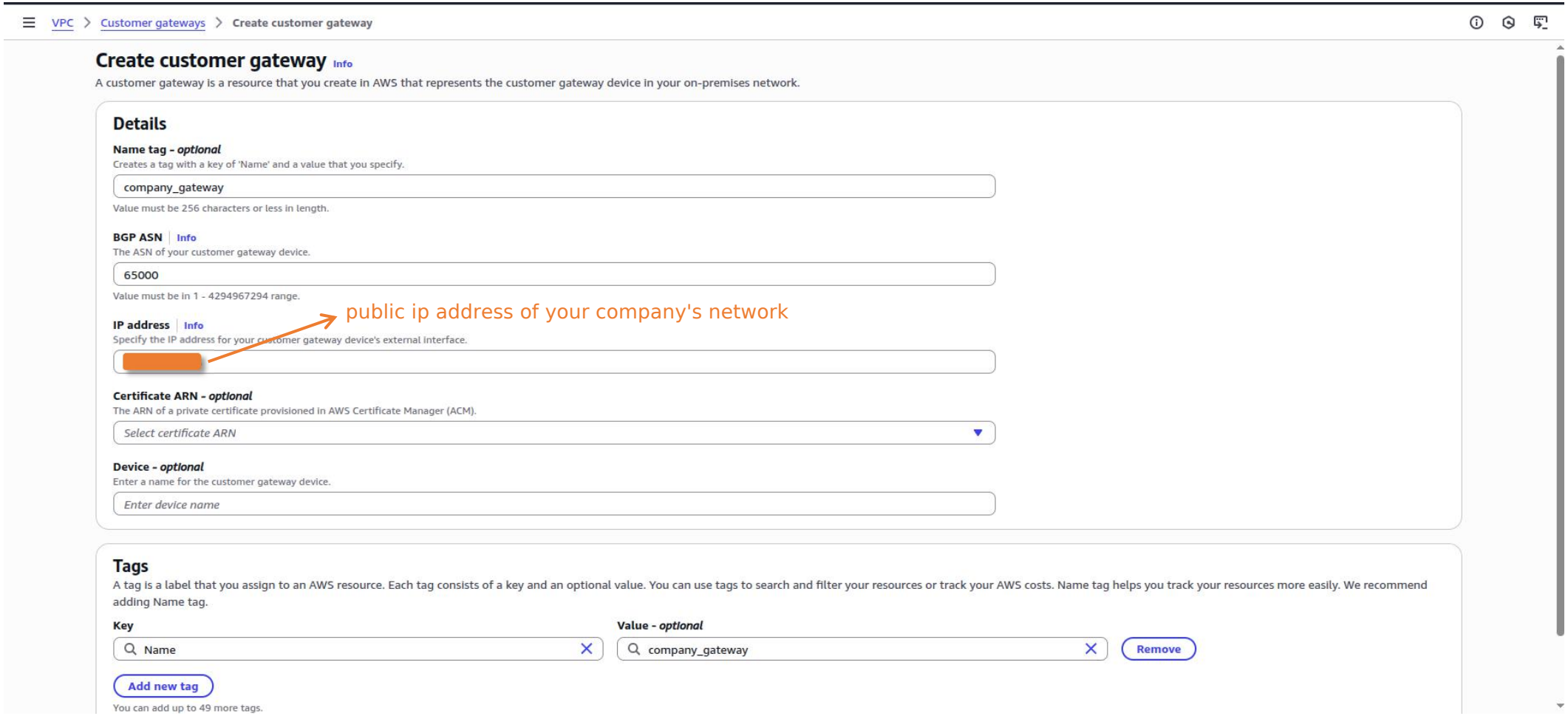
3. Connectivity: Virtual Private Gateway + Site-to-Site VPN connection

1. Create a Virtual Private Gateway (VGW) **aws_virtual_private_gateway** in the VPC resource.

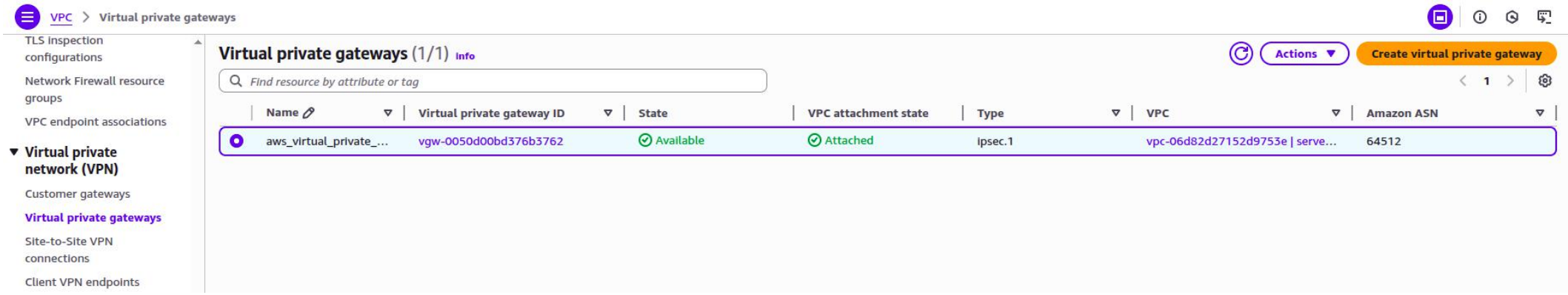


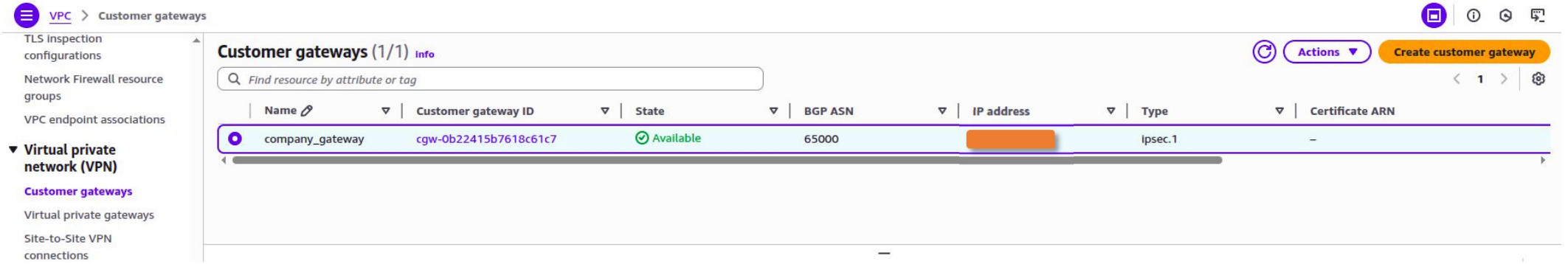
Make sure to attach VGW to the VPC as shown in the image below

2. In your on-prem/firewall, configure the Customer Gateway object (public IP of your customer gateway). and set BGP ASN to default value (65000)

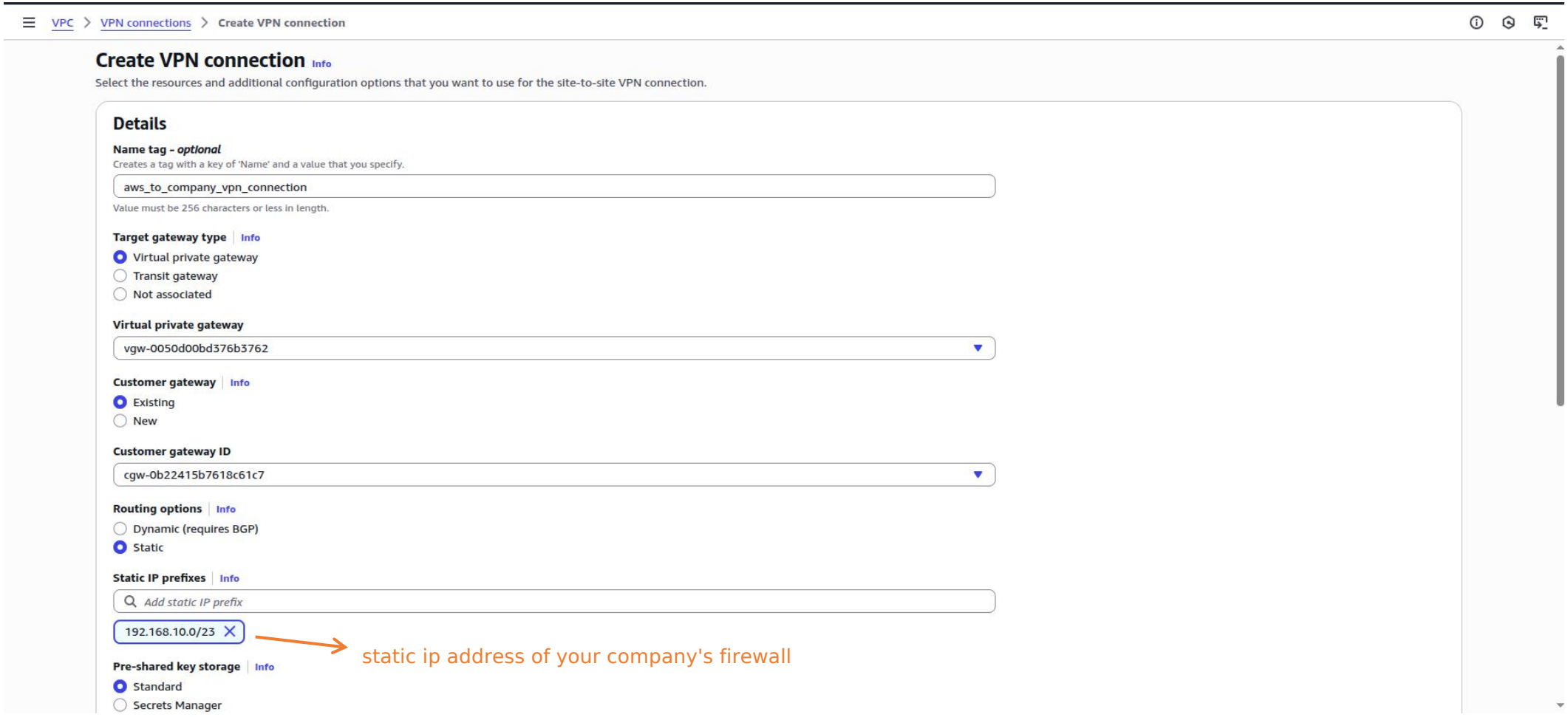


The Virtual Private Gateway and Customer Gateway have been created, but we still need to establish a connection between them.

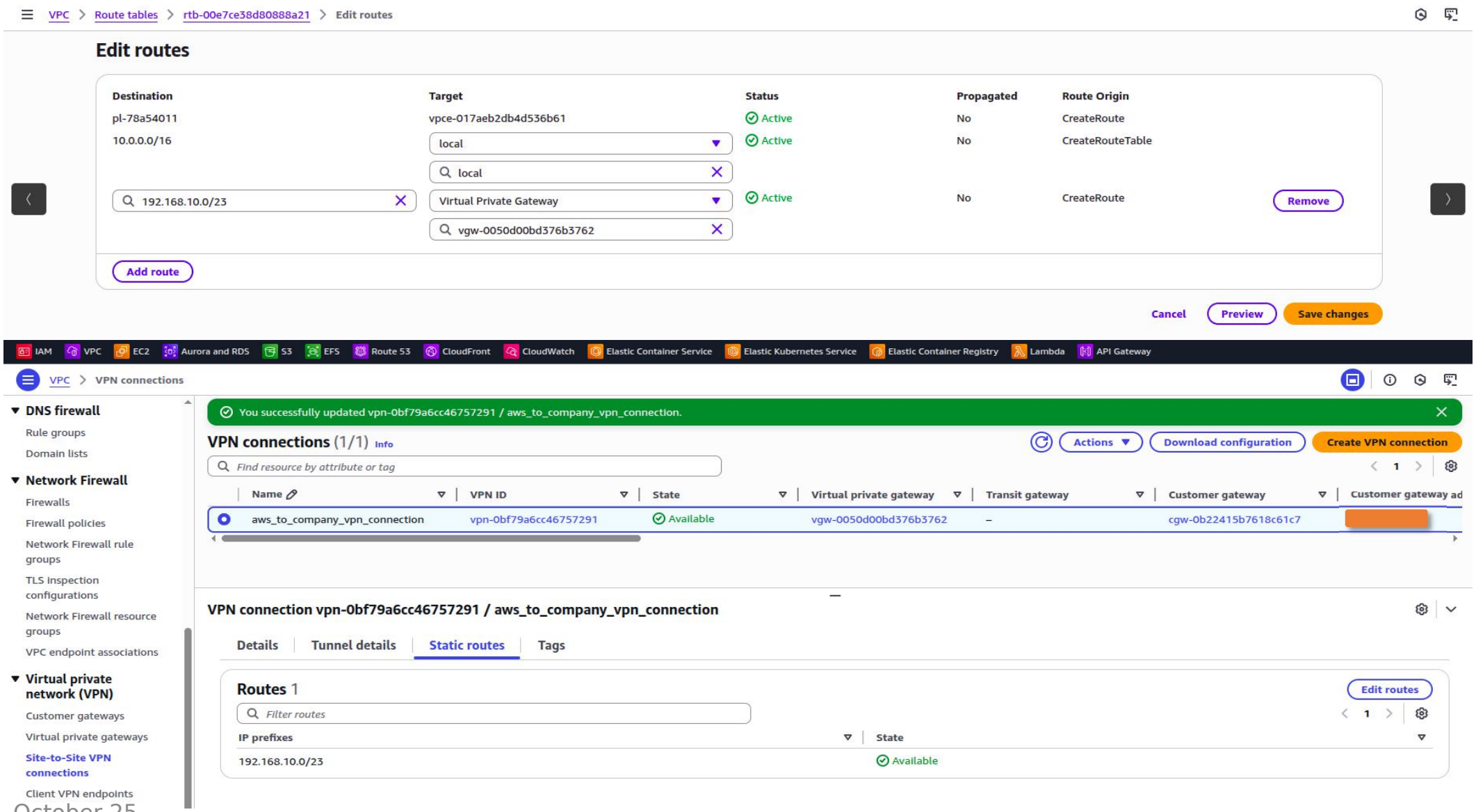




3. Create a Site-to-Site VPN connection (AWS Console → VPC → Create VPN connection), choose static and give your **gateway(firewall) IPv4 address**.

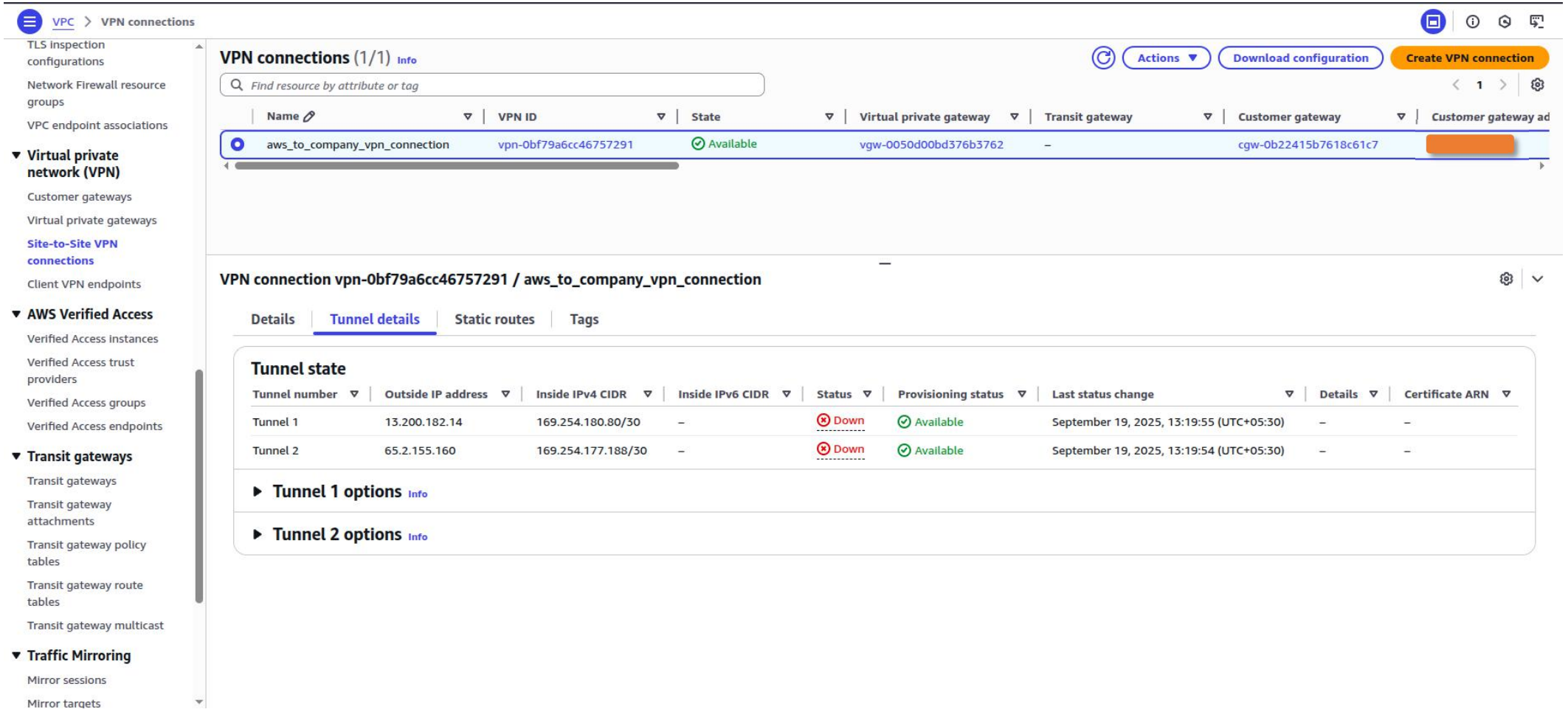


Update VPC route tables for private subnets: add route to on-prem CIDR**(192.168.10.0/23)** via the VGW.



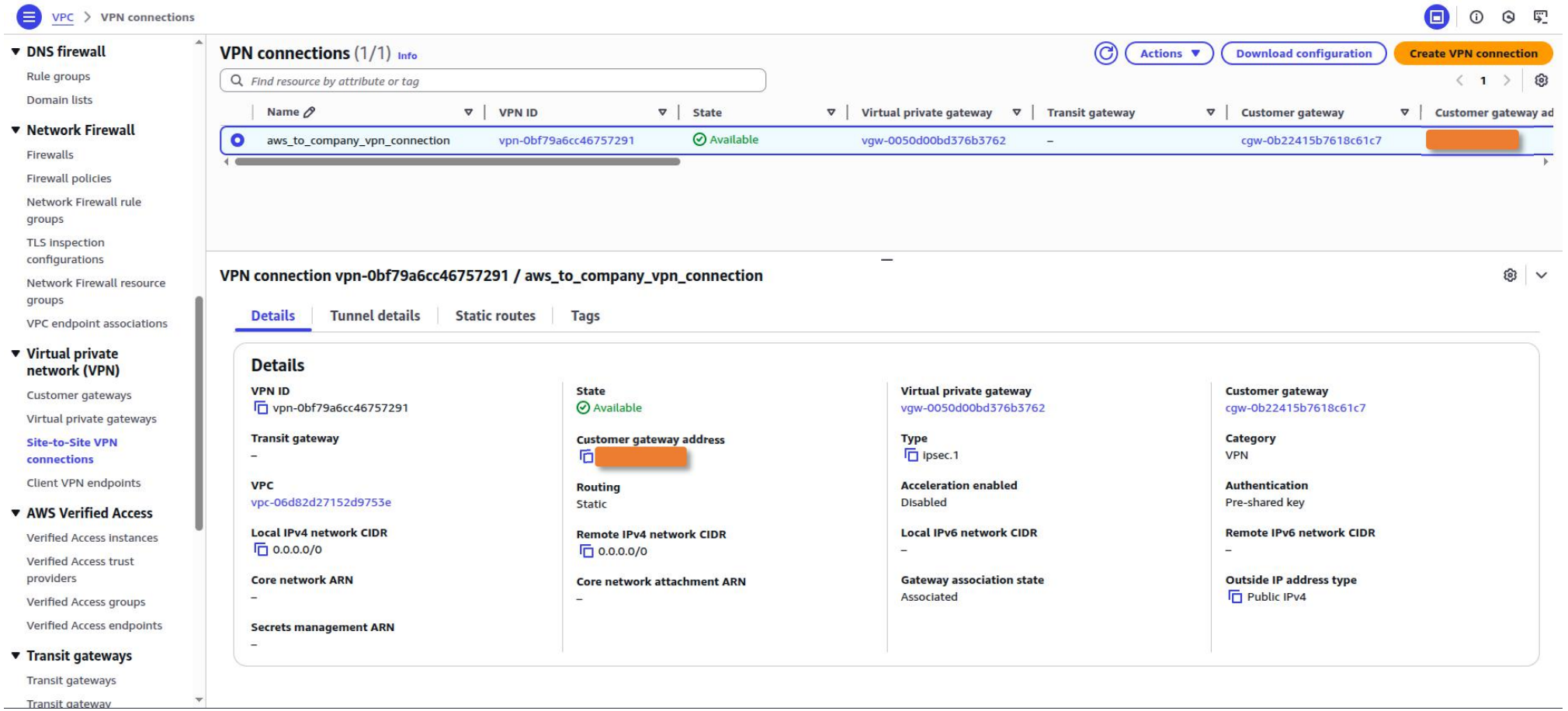
Now the VPN connection has been created but the tunnels are in down state.

Note: AWS will provide **2 public IP addresses** and **2 inside IP address** for the tunnels to be configured in the firewall.



After creating the Site-to-Site VPN connection in AWS, download the **VPN configuration file.**

Note: AWS charges per hour for Site-to-Site VPN while it is provisioned.



4. Since our company is using fortinet device as a firewall.

Choose Vendor: Fortinet

Platform: FortiGate 40+ series

Software: FortiOS latest version depends on devices's firmware

IKE version: ikev2

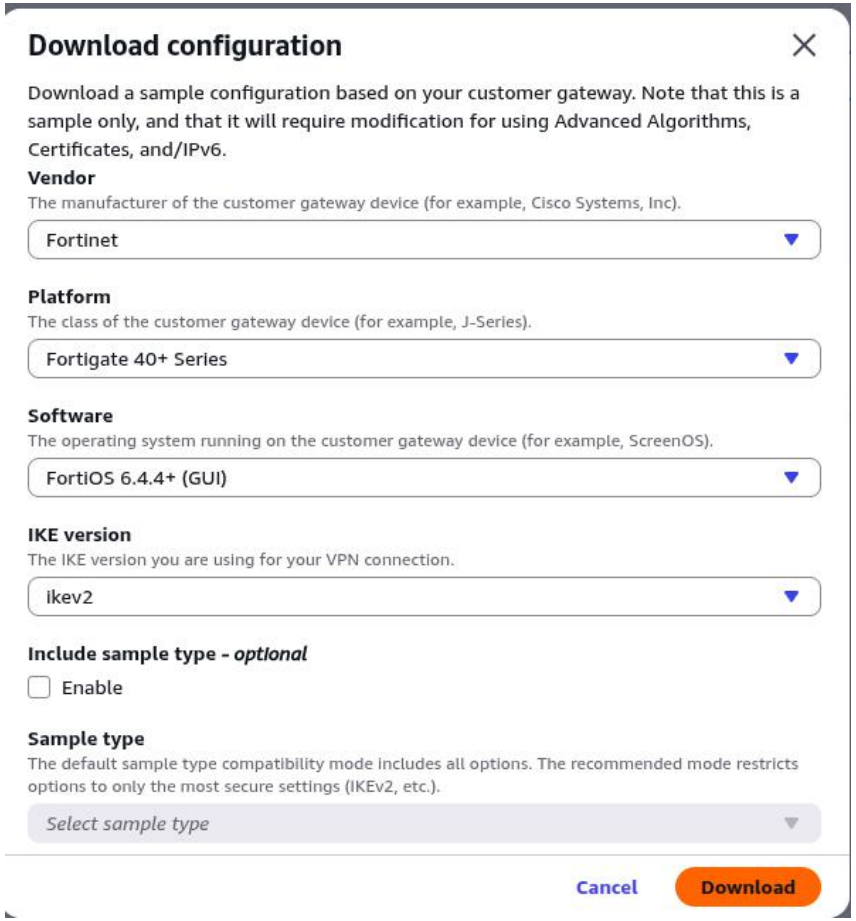
This file will have:

Customer Gateway IP

Virtual Private Gateway IP(s) (two tunnels)

Pre-shared keys

IKE and IPsec parameters

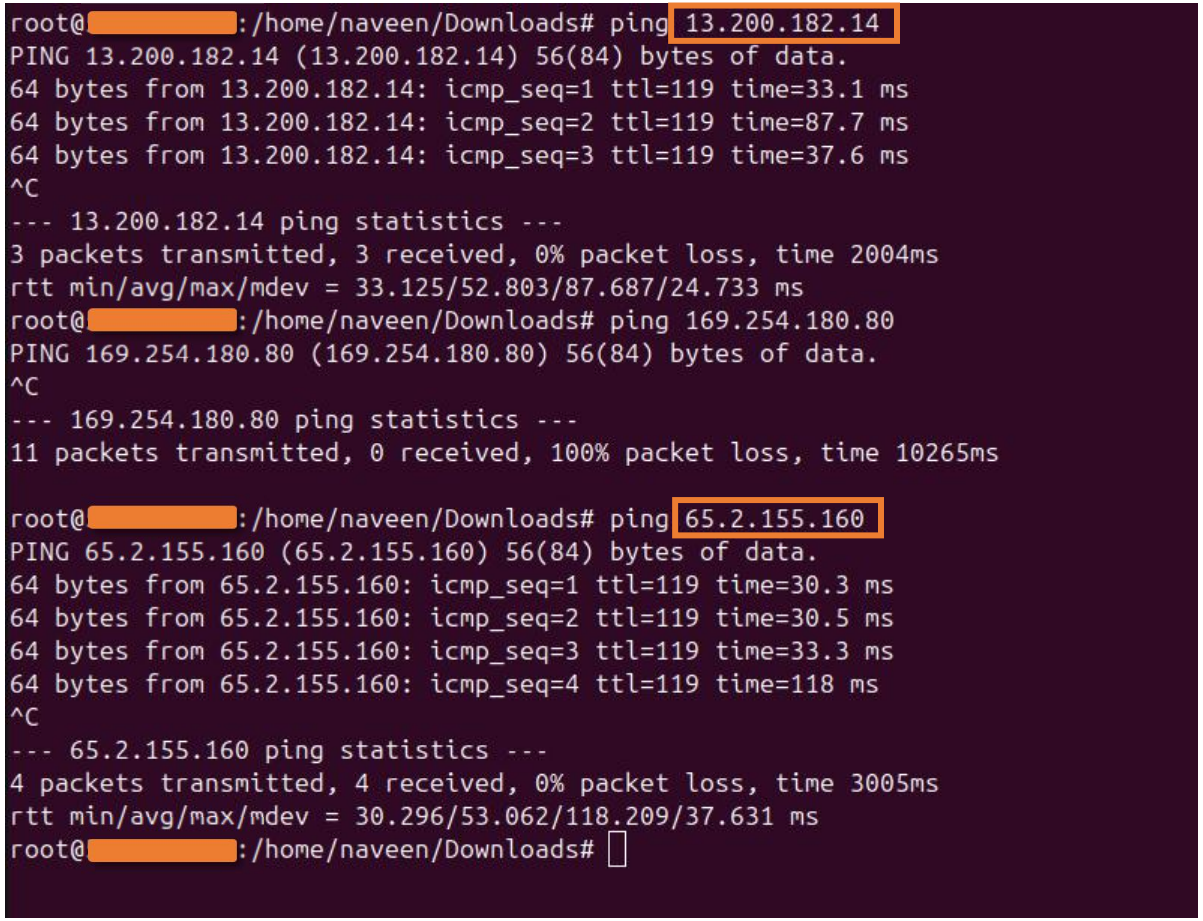


Note: If you don't have any firewall device, you can choose strongSwan

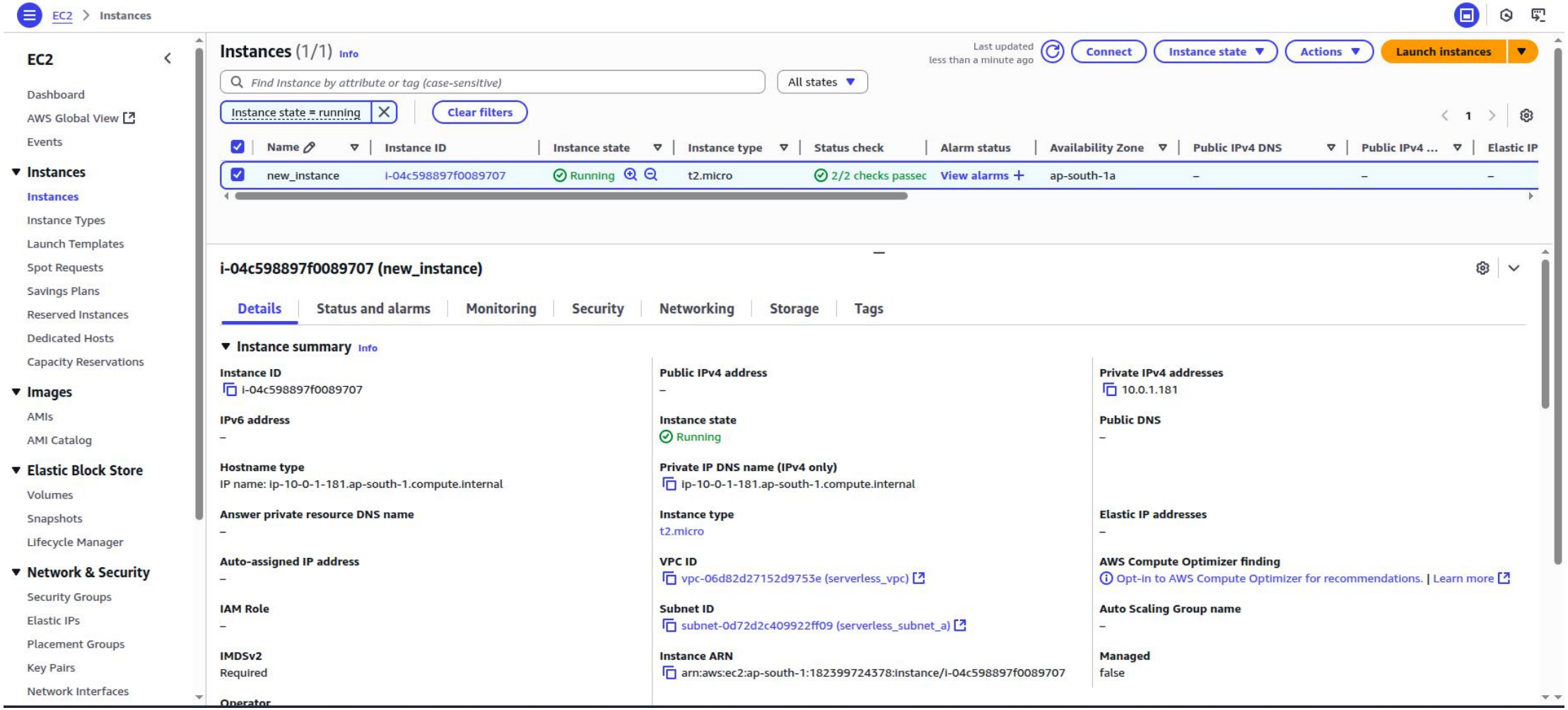
and install it in your device (it will act as a software firewall)

5.Test the connectivity

(a).Now test the connectivity from your local machine by pinging the AWS provided outside IPv4 address

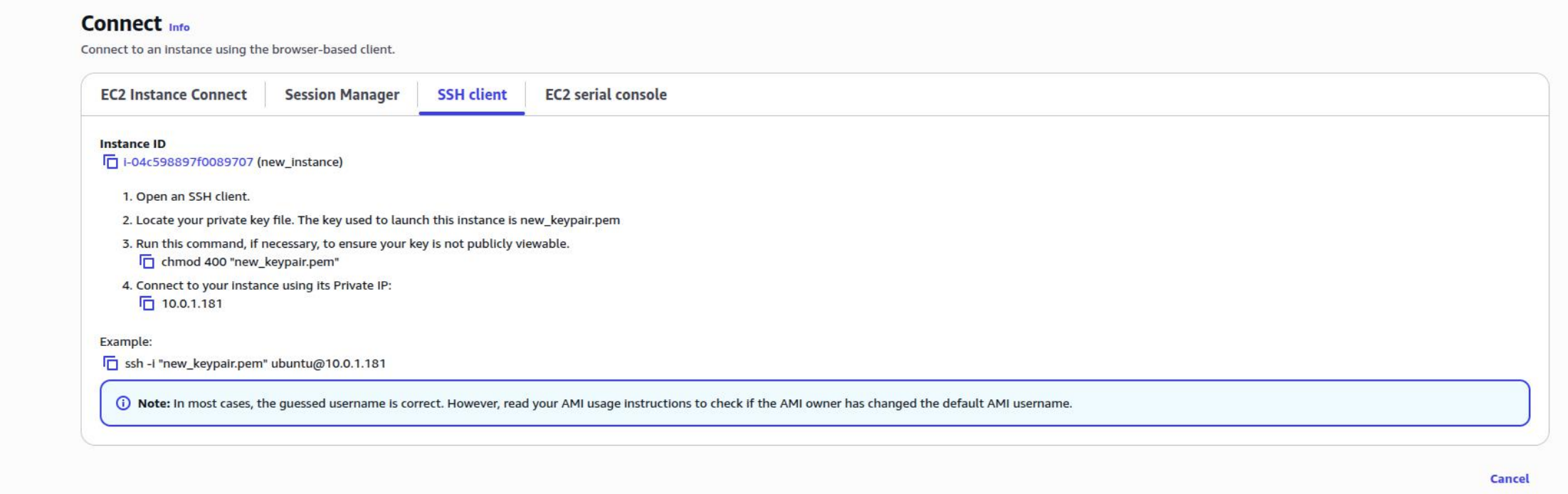


(b). If needed we can do a ping test from the private VPC in the AWS network, for that we need to launch EC2 instance in the private subnet without assigning public IPv4 address. The instance Private IP address is **10.0.1.181**



(c). Connect to the EC2 instance through SSH client.

Open the terminal or ssh client
change the keypair permission **chmod 400 “keypair**
connect your instance **ssh -i “key_pair”ubuntu@10.0.1.181**



(d). Now the connection has been eshtablished as shown in the image 1, now we are going to ping the IP address as shown in the image 2.

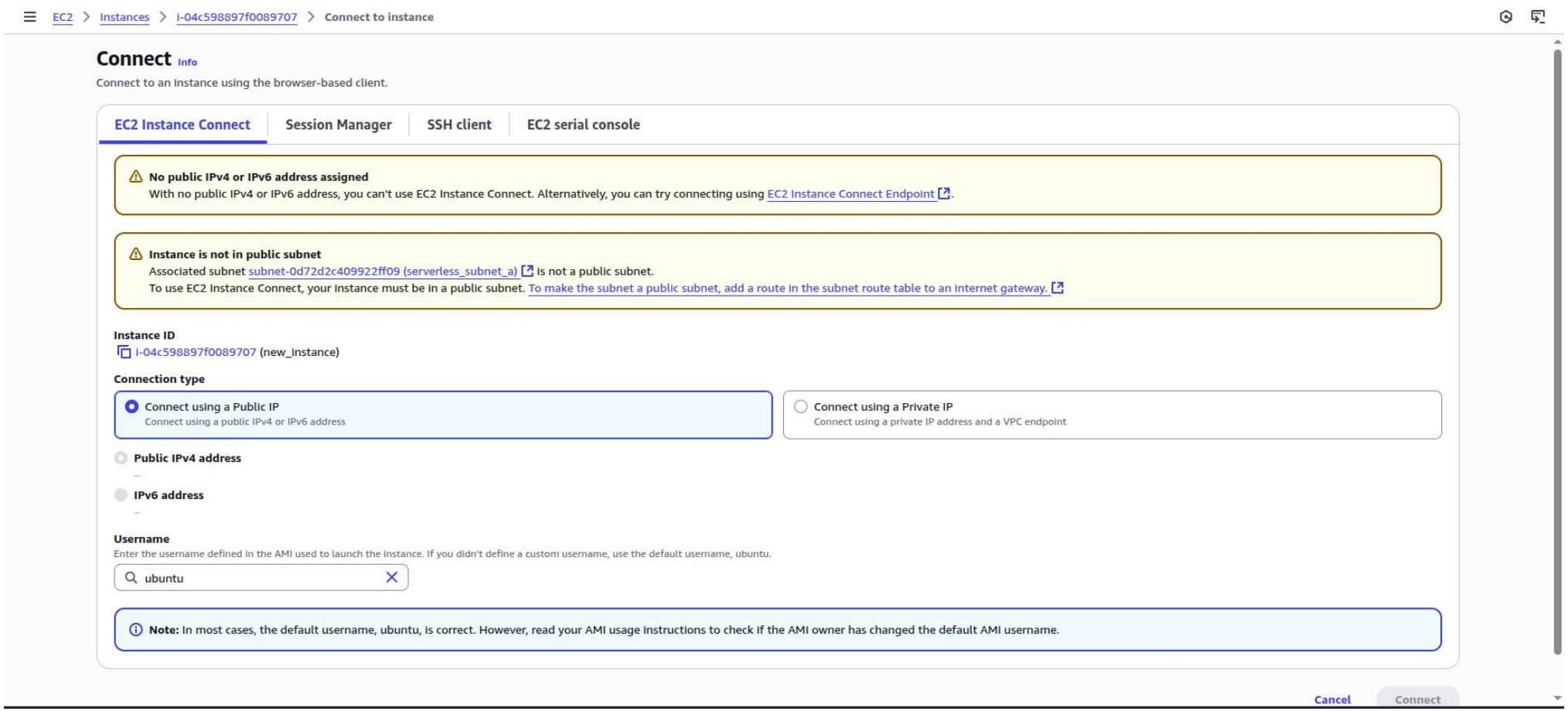

```
root@[redacted]:/home/naveen/Downloads# ssh -i "new_keypair.pem" ubuntu@10.0.1.181
The authenticity of host '10.0.1.181 (10.0.1.181)' can't be established.
ED25519 key fingerprint is SHA256:sCXGisDeebKpYH2BjKeHKP3//aVkU9MBG0XJaBvVIS8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.1.181' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1011-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Mon Sep 22 04:32:10 UTC 2025

  System load:  0.24              Processes:             109
  Usage of /:   25.6% of 6.71GB   Users logged in:       0
  Memory usage: 20%               IPv4 address for enX0: 10.0.1.181
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-0-1-181:~$
```

```
naveen@[redacted]:~$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 0a:b6:ca:0b:36:d7  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 384 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 204715  bytes 103065625 (103.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 204715  bytes 103065625 (103.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlo1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.10.18  netmask [redacted]  broadcast [redacted]
        inet6 fe80::1936:4221:43ef:40a2  prefixlen 64  scopeid 0x20<link>
        ether [redacted]  txqueuelen 1000  (Ethernet)
        RX packets 2721297  bytes 2486978071 (2.4 GB)
        RX errors 0  dropped 54934  overruns 0  frame 0
        TX packets 1123735  bytes 541590357 (541.5 MB)
        TX errors 0  dropped 88 overruns 0  carrier 0  collisions 0
```

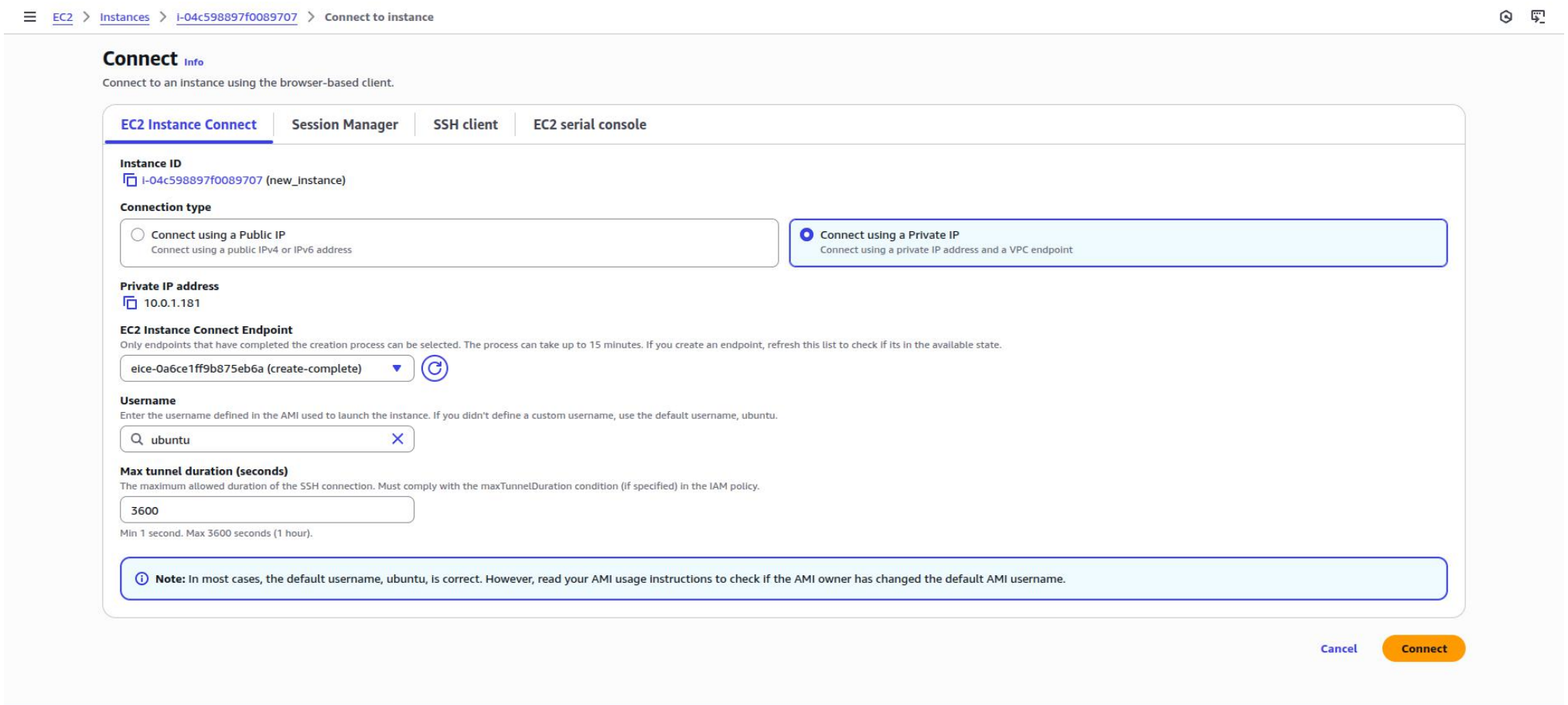
(e). Ping your laptop private IP address **192.168.10.18** from the AWS EC2 private instance which is **10.0.1.181**


```
ubuntu@ip-10-0-1-181:~$ ping 192.168.10.18
PING 192.168.10.18 (192.168.10.18) 56(84) bytes of data.
64 bytes from 192.168.10.18: icmp_seq=1 ttl=63 time=33.7 ms
64 bytes from 192.168.10.18: icmp_seq=2 ttl=63 time=35.4 ms
64 bytes from 192.168.10.18: icmp_seq=3 ttl=63 time=35.8 ms
64 bytes from 192.168.10.18: icmp_seq=4 ttl=63 time=34.2 ms
64 bytes from 192.168.10.18: icmp_seq=5 ttl=63 time=34.5 ms
64 bytes from 192.168.10.18: icmp_seq=6 ttl=63 time=42.3 ms
^C
--- 192.168.10.18 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 33.691/35.987/42.296/2.912 ms
ubuntu@ip-10-0-1-181:~$
```

(f). We have tested the connection through ssh method, To do it from AWS console **EC2 Instance connect** is the option.

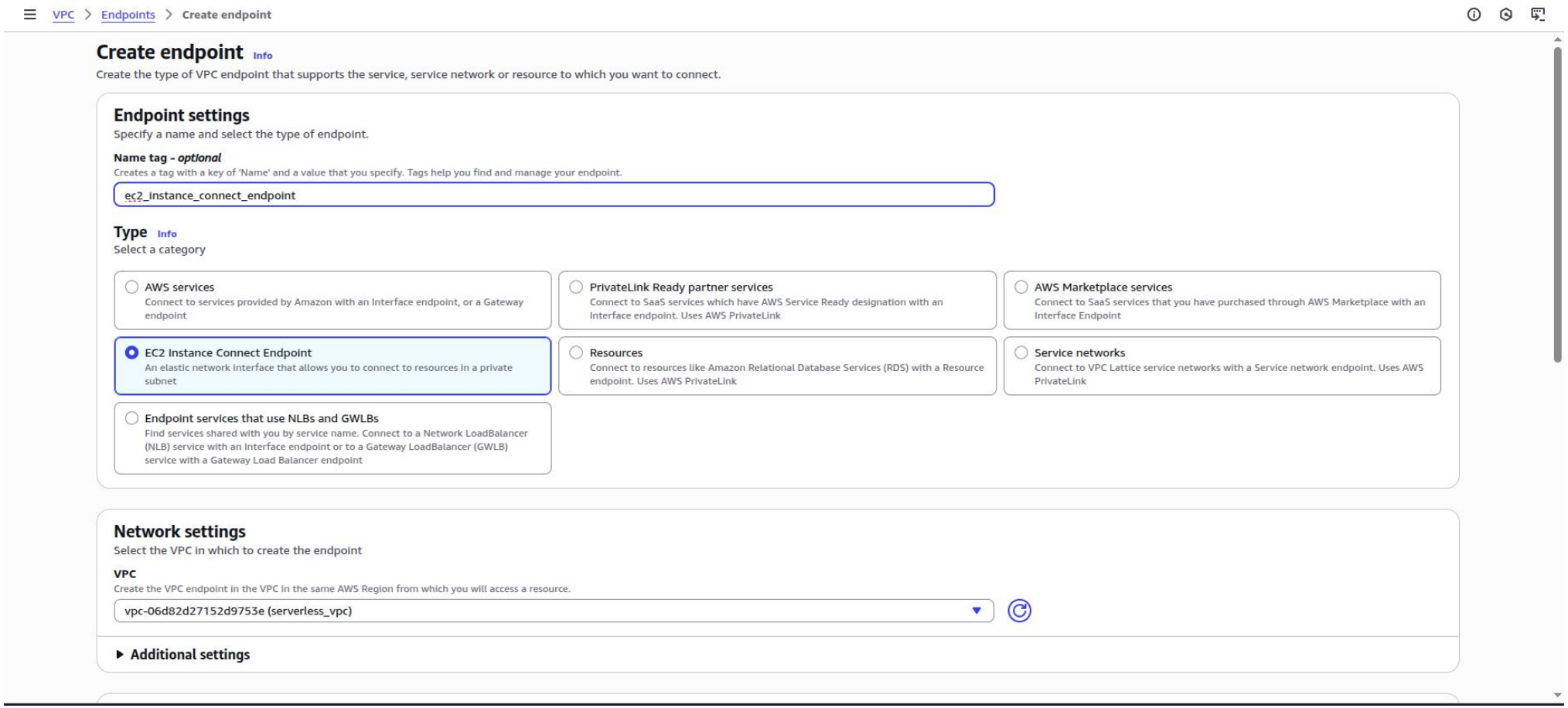


(g). Since we don't have any public IPv4 address to connect, use private IPv4 address. But before that we need network connectivity, So we need to use EC2 instance connect endpoint.



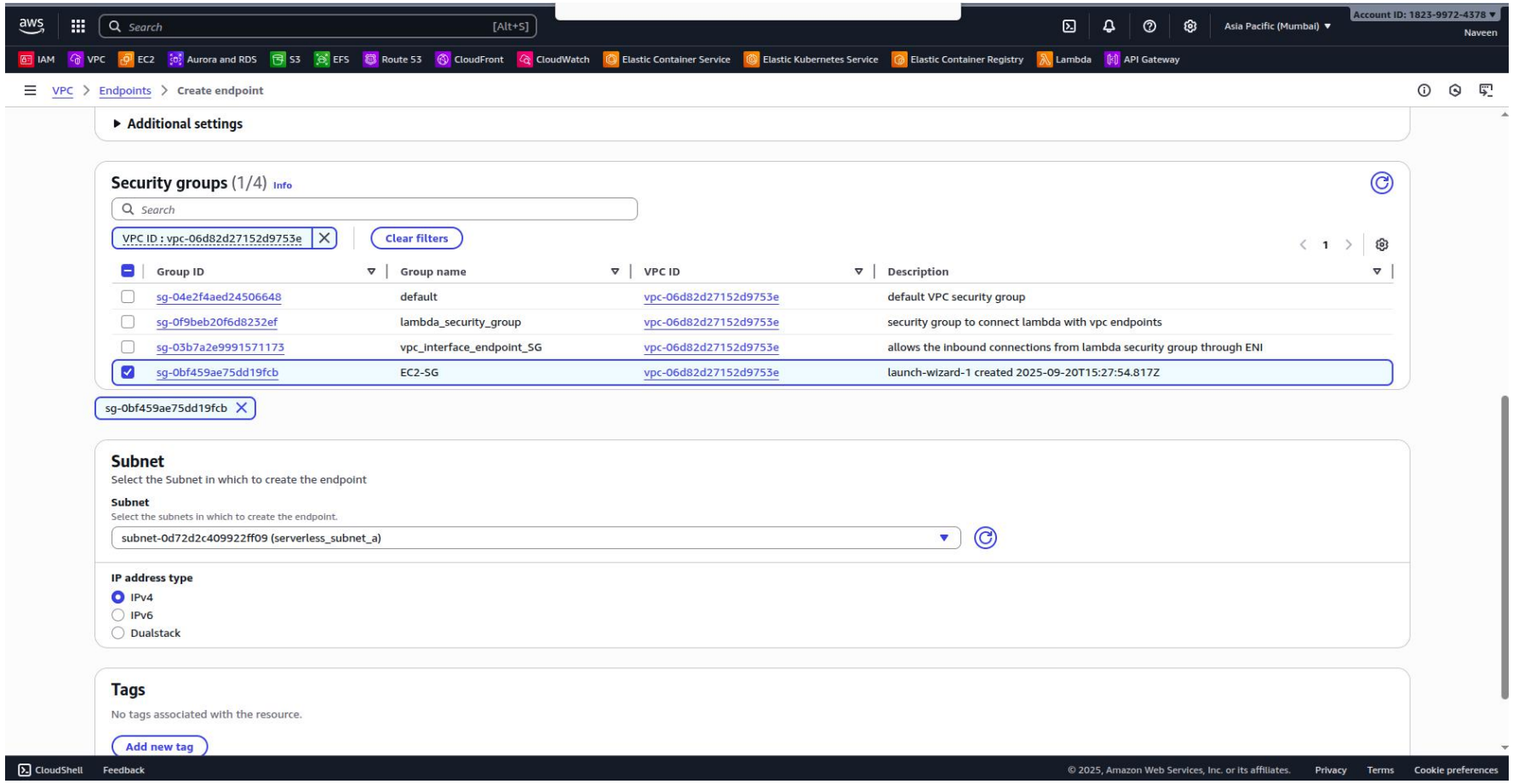
(h). Create EC2 instance connect endpoint in the **VPC** --> **Endpoints** --> **Create Endpoint** --> **EC2 Instance connect Endpoint**

Choose your private VPC in the network settings



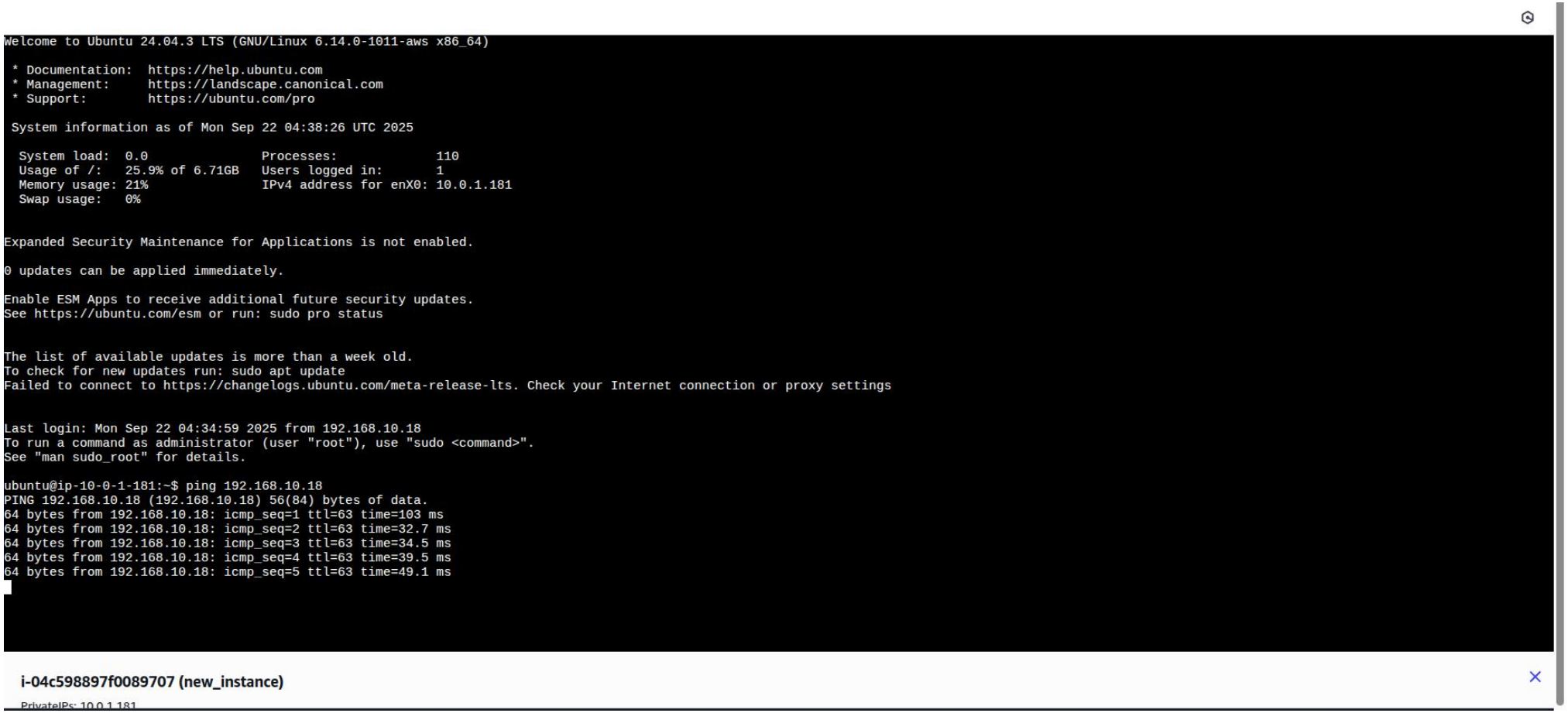
(i). Now choose the security group we have created it for the EC2 instance to access the Onpremise datacenter.

Select one private subnet where your instance is launched and click create and connect the EC2 instance.

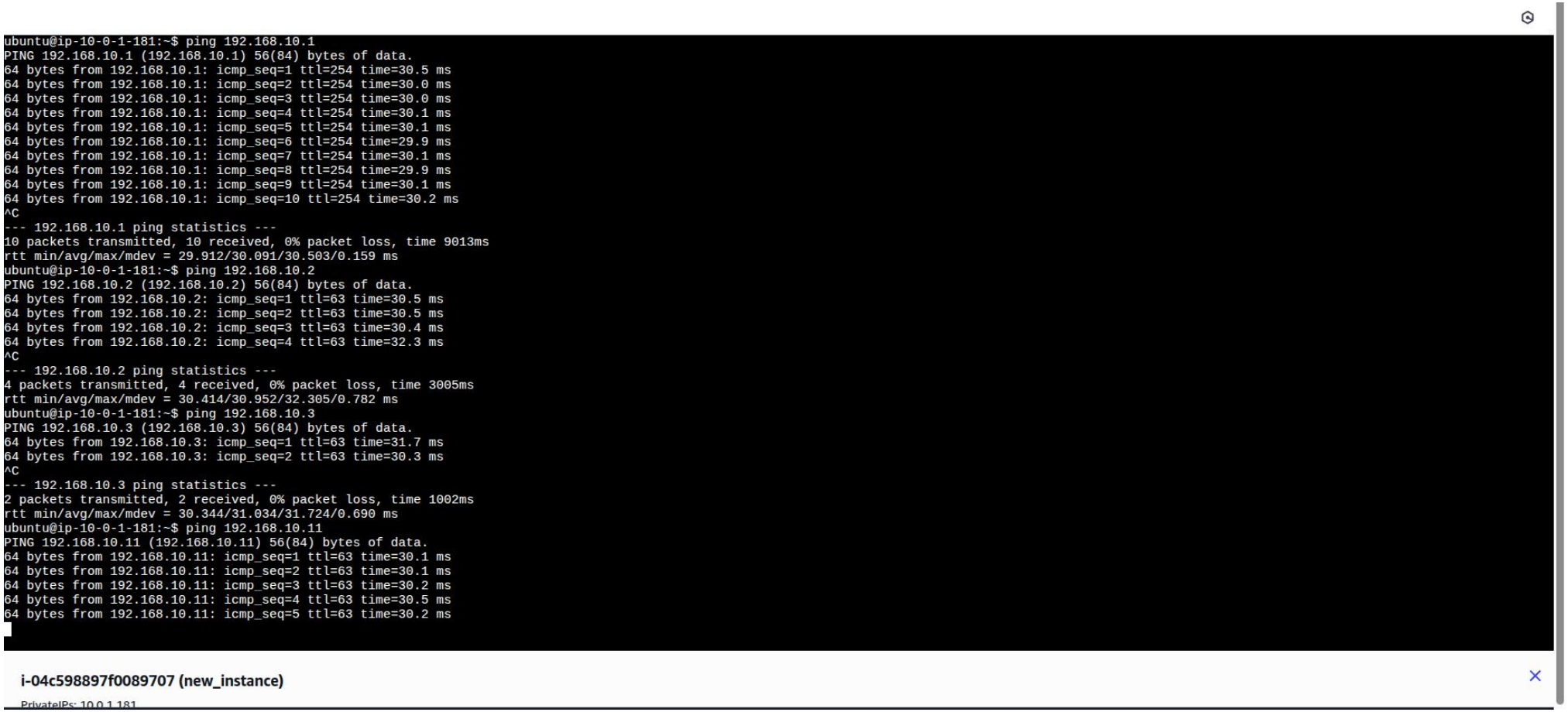


(j). Now the EC2 instance is connected through the EC2 instance connect with the help of endpoint.

Now ping your server ip address **192.168.10.18** in the terminal



(k). After the successful response from your on-premise server. Meanwhile you can also test the other servers or devices ip addresses for the connectivity check.



6. VPC Endpoints (so you don’t need NAT)

When your Lambda is placed in a private subnet within a VPC, it loses its default access to the public internet. To reach any external public service (like the Secrets Manager API endpoint or the S3 API endpoint), the traffic must have a route out of the private subnet.

You have two main options for access from your private subnet:

VPC Endpoint (Recommended for private connectivity):

- Secrets Manager: You need an Interface Endpoint (com.amazonaws.<region>.secretsmanager).
- S3: You can use either a Gateway Endpoint (com.amazonaws.<region>.s3) or an Interface Endpoint (com.amazonaws.<region>.s3). The Gateway Endpoint is free and is generally used for S3.

NAT Gateway (For general internet access):

- Your private subnet's route table must route all outbound internet traffic (0.0.0.0/0) to a NAT Gateway in a public subnet. This allows your Lambda's code to reach the public endpoints for Secrets Manager, S3, and any other internet APIs, but the traffic leaves your private network.

Service/Action	Traffic Origin	Traffic Type	Network Requirement
CloudWatch Logs (automatic logging)	**AWS Lambda Service** (Control Plane)	Service-to-Service Control	**No VPC Endpoint** needed (Bypasses your VPC)
API Gateway Invocation (inbound call)	**AWS API Gateway Service** (Control Plane)	Service-to-Service Control	**No VPC Endpoint** needed (Bypasses your VPC)
Secrets Manager API Call (from your code)	**Your Lambda's ENI** (Data Plane)	Outbound HTTPS API Call	**VPC Endpoint** or **NAT Gateway** required
S3 API Call (from your code)	**Your Lambda's ENI** (Data Plane)	Outbound HTTPS API Call	**VPC Endpoint** or **NAT Gateway** required

Flow of Communication

1. Secrets Manager (Interface Endpoint)

- Lambda needs to retrieve secrets (e.g., DB credentials)
- You create a VPC Interface Endpoint for secretsmanager
- Lambda uses AWS PrivateLink (via the interface endpoint) to make private API calls to Secrets Manager.
- The DNS for Secrets Manager resolves to the private IP address of the interface endpoint within the subnet.

Flow:

Lambda (private subnet) → Secrets Manager Interface Endpoint → Secrets Manager (AWS internal)

2. S3 (Gateway Endpoint)

- You create a VPC Gateway Endpoint for S3, which updates the route tables for the private subnets.
- S3 traffic (e.g., downloads/uploads from/to S3) goes through this gateway, not via NAT or internet.
- Unlike interface endpoints, no IP address is provisioned, but the route table handles redirection.

Flow:

Lambda (private subnet) → Route Table → S3 Gateway Endpoint → Amazon S3

Why does lamdba doesn't need interface endpoint for accessing cloudwatch?

The mechanism for sending Lambda execution logs to CloudWatch Logs is handled by the AWS Lambda service itself, and this traffic does not originate from your Lambda function's Elastic Network Interface (ENI) within your VPC.

Here's a breakdown of why this happens:

Logs are part of the Lambda Service's overhead: When a Lambda function is invoked, the function's standard output (like print statements) and other runtime information are captured by the **Lambda service's control plane**.

Out-of-band communication: The Lambda service sends these execution logs to CloudWatch Logs outside of your VPC's network path. This is a built-in, **service-to-service communication channel** managed by AWS, effectively bypassing the network configuration of your VPC, including the need for a NAT Gateway or a VPC interface endpoint for logs.

Interface Endpoints for API Calls: The CloudWatch interface endpoint you configured (com.amazonaws.<region>.logs) is only necessary if your Lambda function's code were to make explicit API calls to the CloudWatch Logs service (e.g., using the AWS SDK to manually create log groups, streams, or put log events). Since the standard logging is handled automatically, that specific traffic doesn't need the endpoint.

AWS Service Action	Traffic Origin	Endpoint Required (in private subnet)
Sending execution logs to CloudWatch Logs (automatic logging)	AWS Lambda Service	**No** (Handled internally by AWS)
Accessing Secrets Manager (from function code)	Your Lambda ENI in your VPC	**Yes** (We have already set up the endpoint)
Making explicit API calls to CloudWatch Logs (from function code)	Your Lambda ENI in your VPC	**Yes** (Endpoint com.amazonaws.<region>.logs needed)

Why does lamdba doesn't need interface endpoint for accessing API gateway?

This is possible because the invocation of your Lambda function by API Gateway is an internal, **service-to-service communication** that uses the **AWS control plane**, which is separate from your application's data plane traffic within your Virtual Private Cloud (VPC).

Here is a breakdown of the network path:

Lambda Invocation is an API Call: When API Gateway receives a request, it uses the AWS Lambda Invoke API to call your function. This Invoke API is a publicly accessible, regional service.

API Gateway is Outside Your VPC: The API Gateway service itself is an AWS-managed service and sits outside of your VPC. When it calls the Lambda service, it does so over the AWS backbone network, which doesn't go through your VPC's subnets or routing.

VPC Configuration for Lambda's Outbound Traffic: Your Lambda's VPC configuration (being in a private subnet with an ENI) is only for outbound traffic originating from your function's code when it tries to access resources like a database in your VPC or external services via a NAT Gateway/VPC Endpoint.

VPC Endpoints are Not Needed for Invocations:

You don't need a VPC endpoint for API Gateway to invoke Lambda because the invocation is handled by the AWS service layer.

You would only need an API Gateway VPC endpoint if your Lambda function's code needed to make an outbound call back to a Private API Gateway endpoint.

In short, your Lambda function is always invoked via its public API endpoint by the API Gateway service; the VPC configuration only governs the function's network egress (outbound traffic) when the code is running.

Permissions for accessing Cloudwatch and Network interface:

The permissions you see are included to ensure the Lambda function can operate and, if needed, connect to your private network resources.

The permissions are present for two distinct and essential reasons:

1. CloudWatch Permissions (Essential for Logging)

The **AWSLambdaBasicExecutionRole** policy is designed to give the function the absolute minimum permissions required to run and be observable. CloudWatch Logs permissions are at the core of this:

Necessity for Debugging: Lambda is a transient compute service. Without the ability to log to CloudWatch, you would have no visibility into the function's output, errors, or performance, making debugging and monitoring impossible.

The Specific Permissions: This default role typically grants permissions for the actions:

logs:CreateLogGroup

logs:CreateLogStream

logs:PutLogEvents

These actions ensure the Lambda service can automatically create a dedicated log group for your function and write the output/errors to it.

2. EC2 Permissions (Required for VPC Connectivity)

The EC2 permissions are not part of the AWSLambdaBasicExecutionRole itself, but they are automatically added to the execution role when you configure your Lambda function to run inside a Virtual Private Cloud (VPC).

Lambda requires these permissions to manage the network connection to your private subnets:

Network Interface Management: When you put a Lambda function into a VPC, AWS must create an Elastic Network Interface (ENI) within that VPC/subnet to give the function a private IP address.

The Specific Permissions: The managed policy **AWSLambdaVPCAccessExecutionRole** (which is attached when you configure VPC access) contains the following EC2 actions:

ec2:CreateNetworkInterface: To establish the connection when the function starts.

ec2:DescribeNetworkInterfaces: To retrieve information about the ENI.

ec2:DeleteNetworkInterface: To clean up the ENI when the function's execution environment is terminated.

In summary:

CloudWatch permissions are in the **Basic Execution Role** so the function can tell you what it's doing (logging).

EC2 permissions are in the **VPC Access Role** so the function can connect to your private resources (networking).

Elastic Network Interface:

The ENIs (Elastic Network Interfaces) are created in the private subnets you specify when you create the Interface VPC Endpoints.

Here's What's Happening:

When you create an Interface VPC Endpoints (for services like Secrets Manager or CloudWatch Logs), AWS provisions one or more ENIs in your VPC, inside the subnets you choose.

These ENIs:

- Live inside your private subnets
- Are owned and managed by AWS
- Appear as AWS PrivateLink interfaces
- Have private IP addresses from the subnet’s CIDR block
- Are connected to the corresponding AWS service over the AWS internal network

How Interface Endpoints Work (IP Allocation)

When you create an Interface VPC Endpoint, each endpoint provisions one **Elastic Network Interface (ENI)** per subnet that you associate with the endpoint.

Each ENI consumes **1 private IP address** from the subnet’s CIDR block.

Example:

Interface Endpoints	Subnet A	Subnet B	Total IPs
Secrets Manager	1 ENI IP	1 ENI IP	2 ENI IP's
Cloudwatch	1 ENI IP	1 ENI IP	2 ENI IP's
			4 ENI IP's

You create a VPC with two private subnets:

subnet-a (10.0.1.0/24)

subnet-b (10.0.2.0/24)

A Lambda function in 2 private subnets (let's say subnet-a in az1 and subnet-b in az2)

You create an Interface Endpoint for Secrets Manager and cloudwatch (auto create by lambda)

You specify both private subnets during endpoint creation

AWS will:

Create one ENI in subnet-a **(e.g., 10.0.1.72)**

Create one ENI in subnet-b **(e.g., 10.0.2.11)**

These ENIs are:

- The actual access points for your Lambda and other VPC resources to reach Secrets Manager
- What the DNS name (e.g., secretsmanager.us-east-1.amazonaws.com) will resolve to — if private DNS is enabled

Where to See These ENIs

You can find them:

- In the **EC2 console > Network Interfaces**
- They will have a description like VPC Endpoint Interface for com.amazonaws.ap-south-1.secretsmanager
- You'll see them associated with your specified subnets, and attached to the VPC endpoint

Not Created in "Interface Endpoints" Subnet (because...)

"Interface endpoint" isn't a location — it's a type of VPC endpoint. The ENIs are the endpoint from your VPC's point of view.

So when someone says, "the ENIs are created in the interface endpoint" - they're technically misunderstanding you pick the subnet, and the ENIs are created there.

Component	Lives In	Notes
ENIs for endpoint	**Your specified subnets**	These are the interface endpoints, technically
Gateway endpoints	N/A (no ENI created)	Use route table entries only
Private IPs	From your subnets' CIDR	Consumes subnet IP space
DNS names	Resolve to those ENI IPs	If private DNS + VPC DNS settings are enabled

DNS Hostname and DNS Resolution:

Enabling DNS hostnames and DNS resolution allows your Lambda (or any resource in the VPC) to use standard AWS service names (like secretsmanager.<region>.amazonaws.com) and have them resolve to the private IPs of your interface endpoints — not the public AWS endpoints.

1. DNS Resolution (VPC setting)

- This allows the VPC to resolve domain names to IP addresses using AmazonProvidedDNS.
- Required for any DNS-based communication, including internal AWS services via VPC endpoints.

2. DNS Hostnames (VPC setting)

- This allows AWS to assign hostnames to instances and endpoints using private DNS.
- Needed if you want private DNS names for VPC interface endpoints to resolve correctly.

Why It Matters for Interface Endpoints
When you create an interface endpoint, AWS sets up a private DNS name for the service. For example:

Service	Public DNS	Private DNS (via Interface Endpoint)
Secrets Manager	secretsmanager.ap-south-1.amazonaws.com	Resolves to private IP of endpoint ENI

To make this work:

AWS overrides the public DNS with a private one — only if:

- Private DNS is enabled on the interface endpoint, AND
- Your VPC has DNS hostnames & resolution enabled

Without those VPC DNS settings:

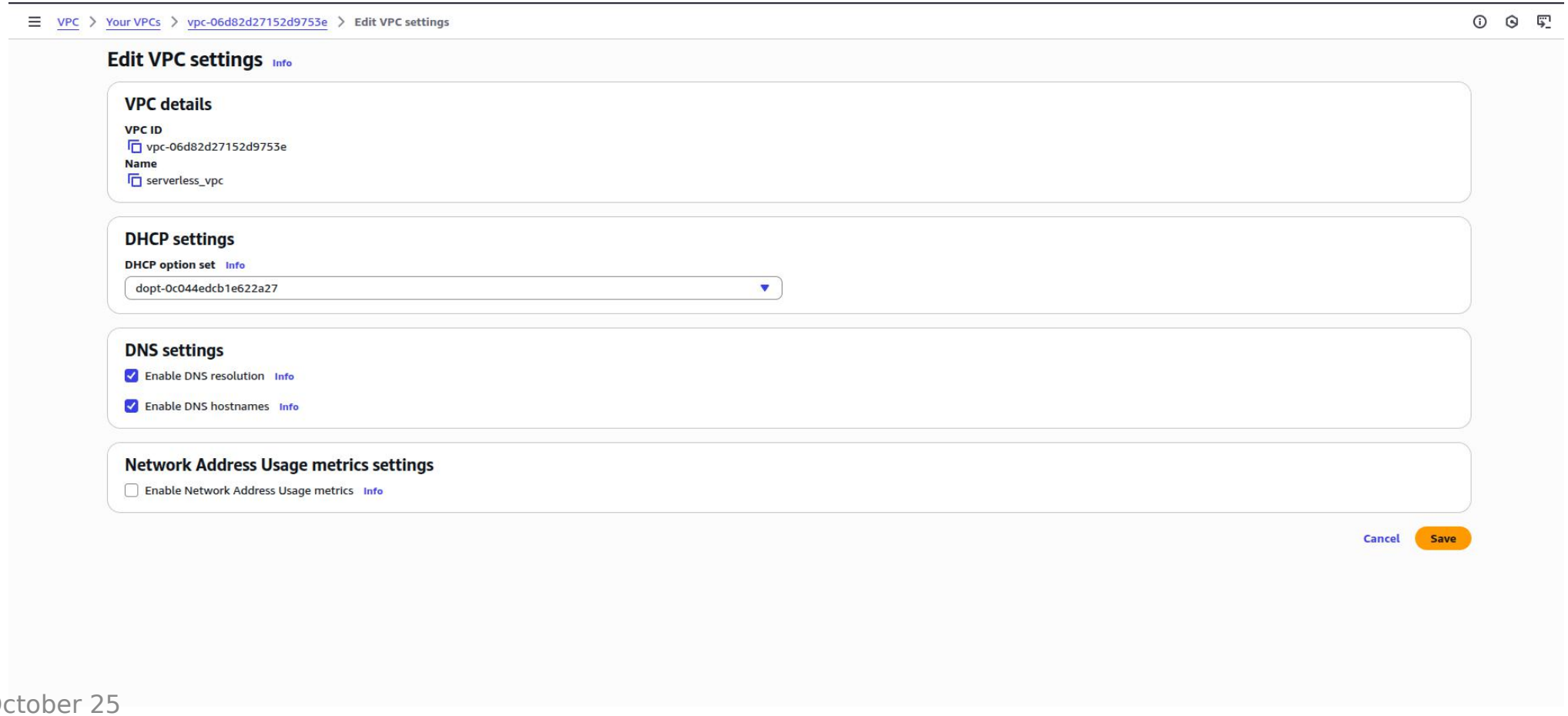
- DNS lookups from your Lambda or EC2 would resolve to the public IP addresses of AWS services
- Since your Lambda is in a private subnet with no internet access, it would fail to connect
- Even though the interface endpoint exists, it won't be used
- If either of these is disabled, your interface endpoints may not be used, and your traffic may fail or try to go through NAT (if present).

Setting	Required?	Why
DNS Resolution	Yes	Enables name resolution inside the VPC
DNS Hostnames	Yes	Allows use of private service DNS names via interface endpoints

How to Check If It's Enabled

You can check in the VPC settings:

- Go to **VPC** --> **Your VPC's** --> **Edit VPC settings** --> **DNS settings** --> **Enable DNS resolution** & **hostnames**
- Ensure both are enabled



Key Notes

No NAT Gateway required: Since you're using both interface and gateway endpoints, Lambda can access these services without NAT or internet access.

Each endpoint provisions one **Elastic Network Interface (ENI)** per subnet that you associate with the endpoint.

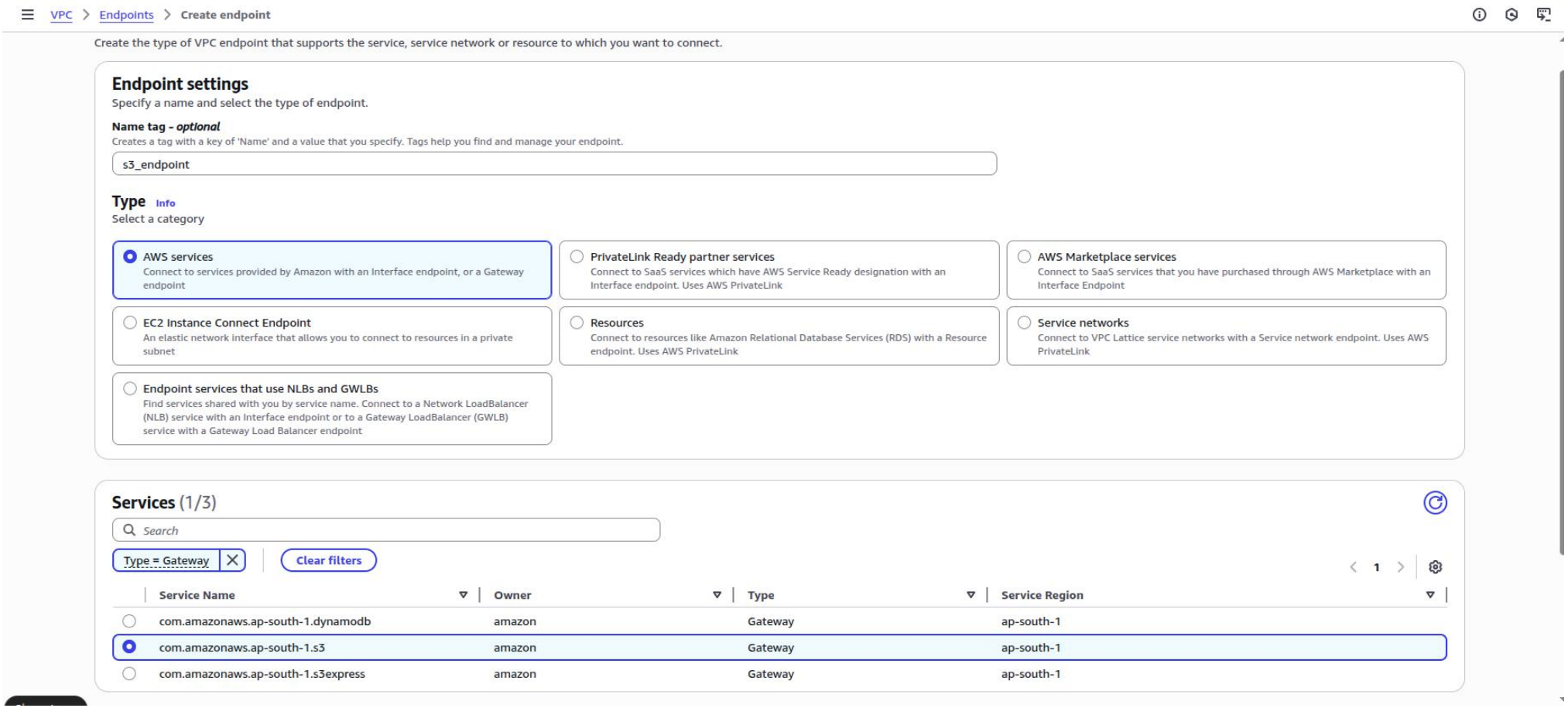
DNS resolution is important: The interface endpoint uses private DNS to override the public AWS service endpoint DNS with a private IP from the VPC.

IAM permissions: Your Lambda must still have IAM permissions to access Secrets Manager, S3, and CloudWatch.

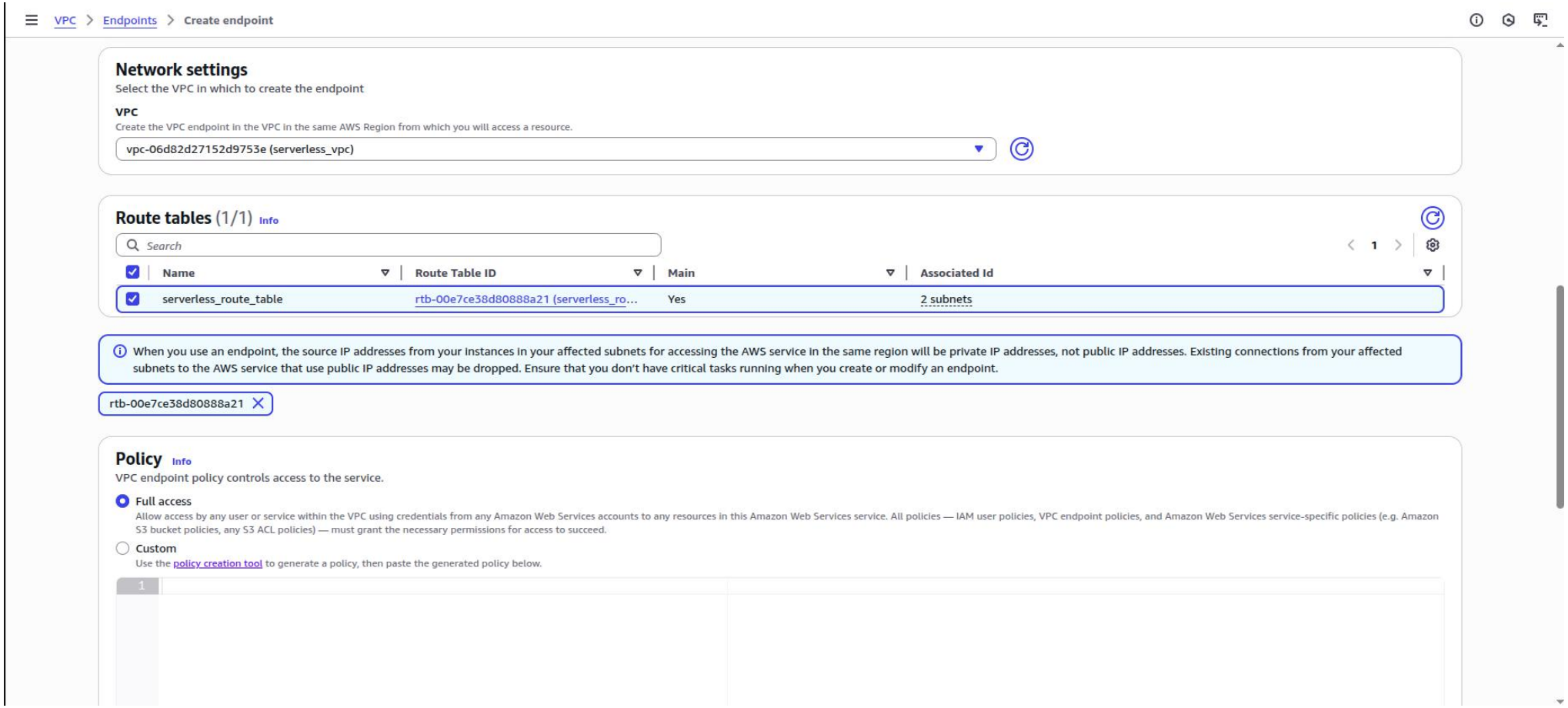
Security groups: Interface endpoints use security groups—make sure they allow inbound traffic from your Lambda subnets.

1(a). We need to create **S3 Gateway Endpoint** to eshtablish a connection between lambda in private subnet to access the S3 service present outside the VPC.

VPC --> Endpoints --> Create Endpoint --> AWS services --> Type = Gateway --> S3

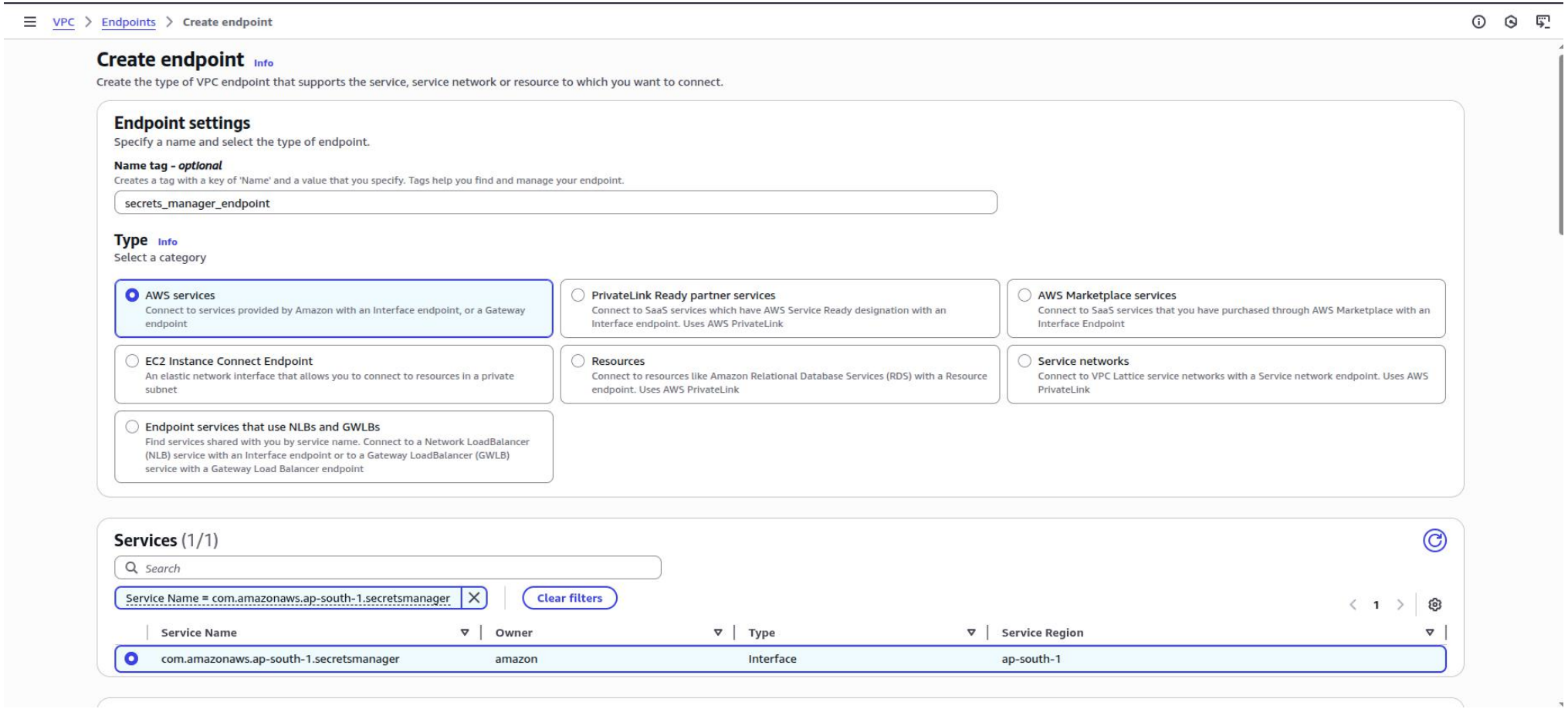


(b). Choose your custom VPC and attach to your route tables for private subnets. This allows Lambda to talk to S3 without internet NAT Gateway.



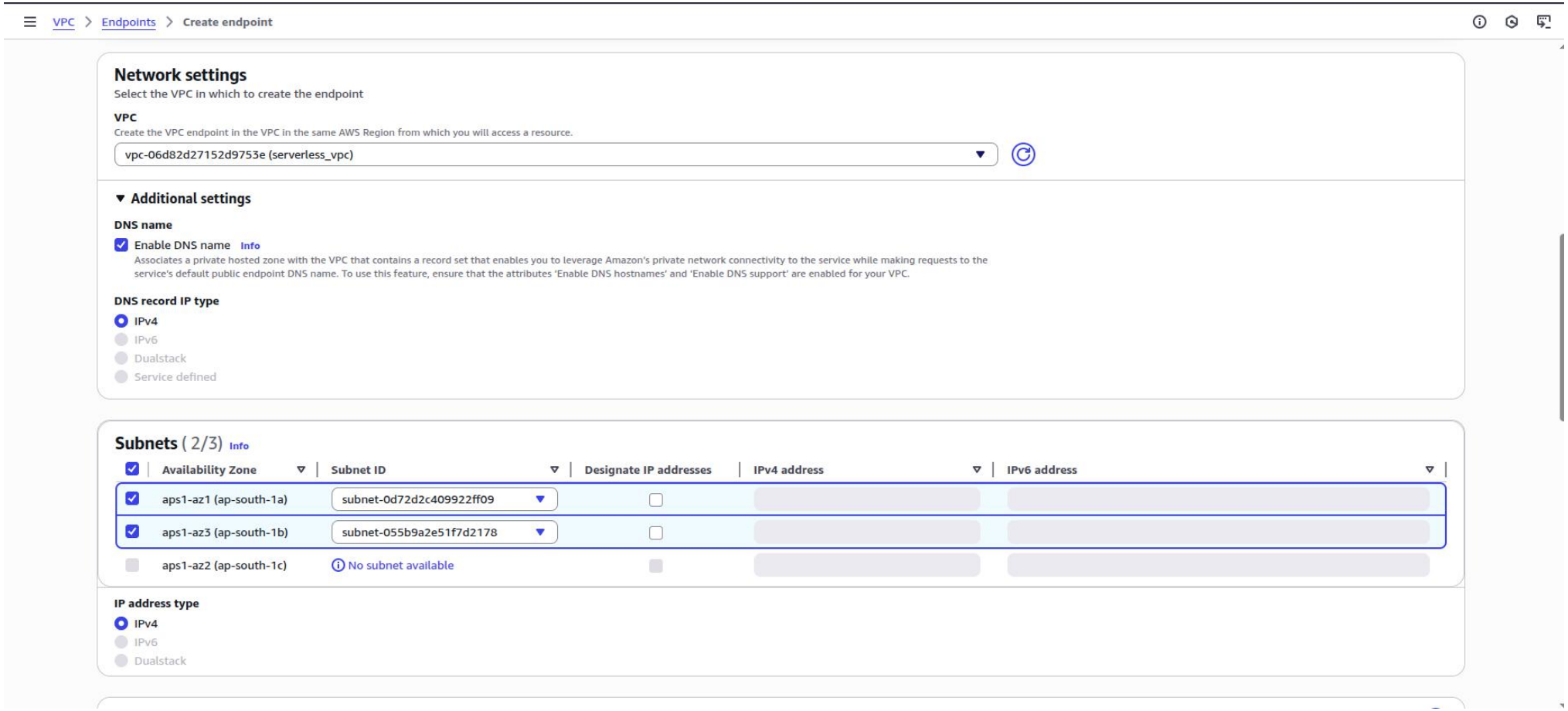
2(a). Create **Interface Endpoints (AWS PrivateLink)** for lambda which needs to retrieve secrets from the secrets manager (e.g., DB credentials)

Service Name = com.amazonaws.ap-south-1.secretsmanager (Secrets Manager)



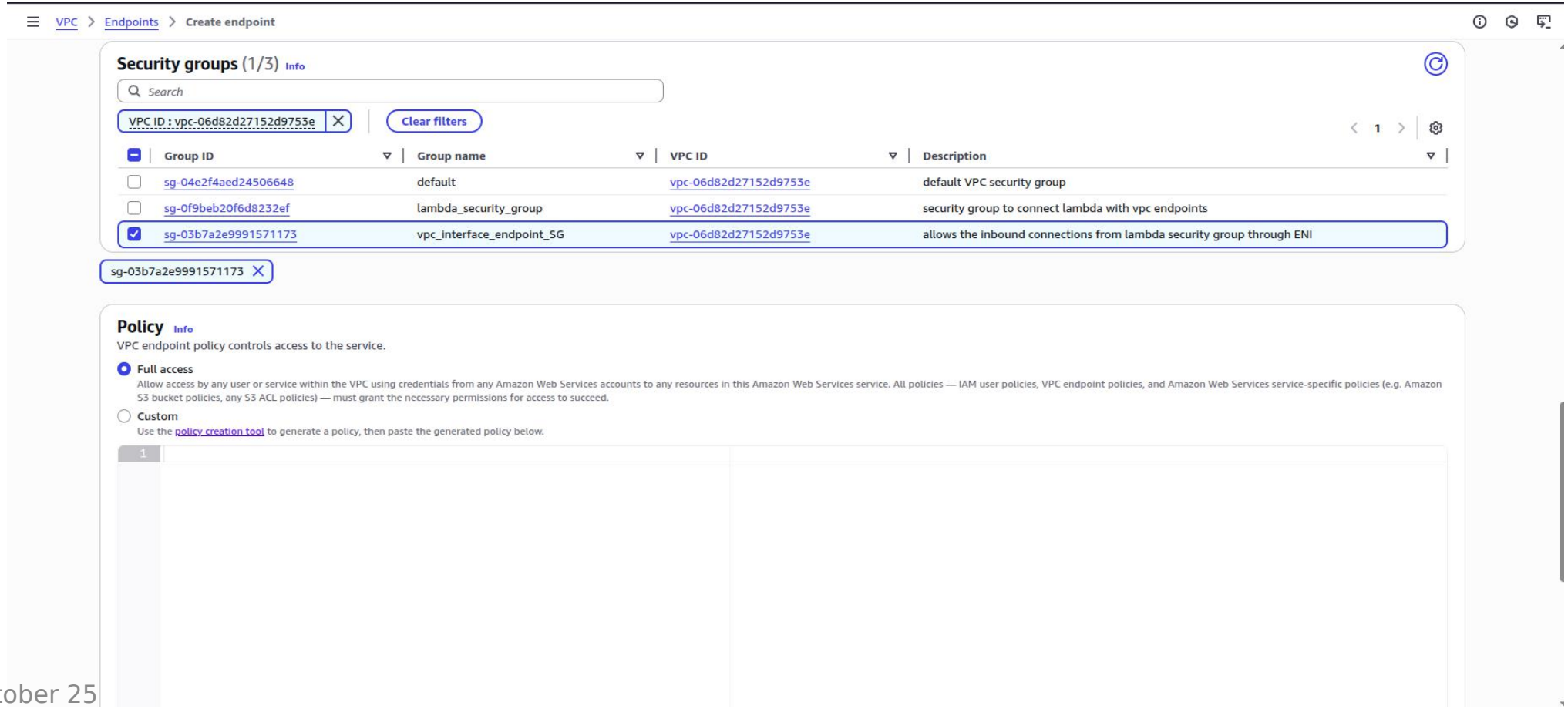
(b). Choose your custom VPC, click on enable **DNS name** and enable the availability zones and private subnets.

Note: Check on the **designate IP addresses** for assiging your custom IP addresses to the **ENI**

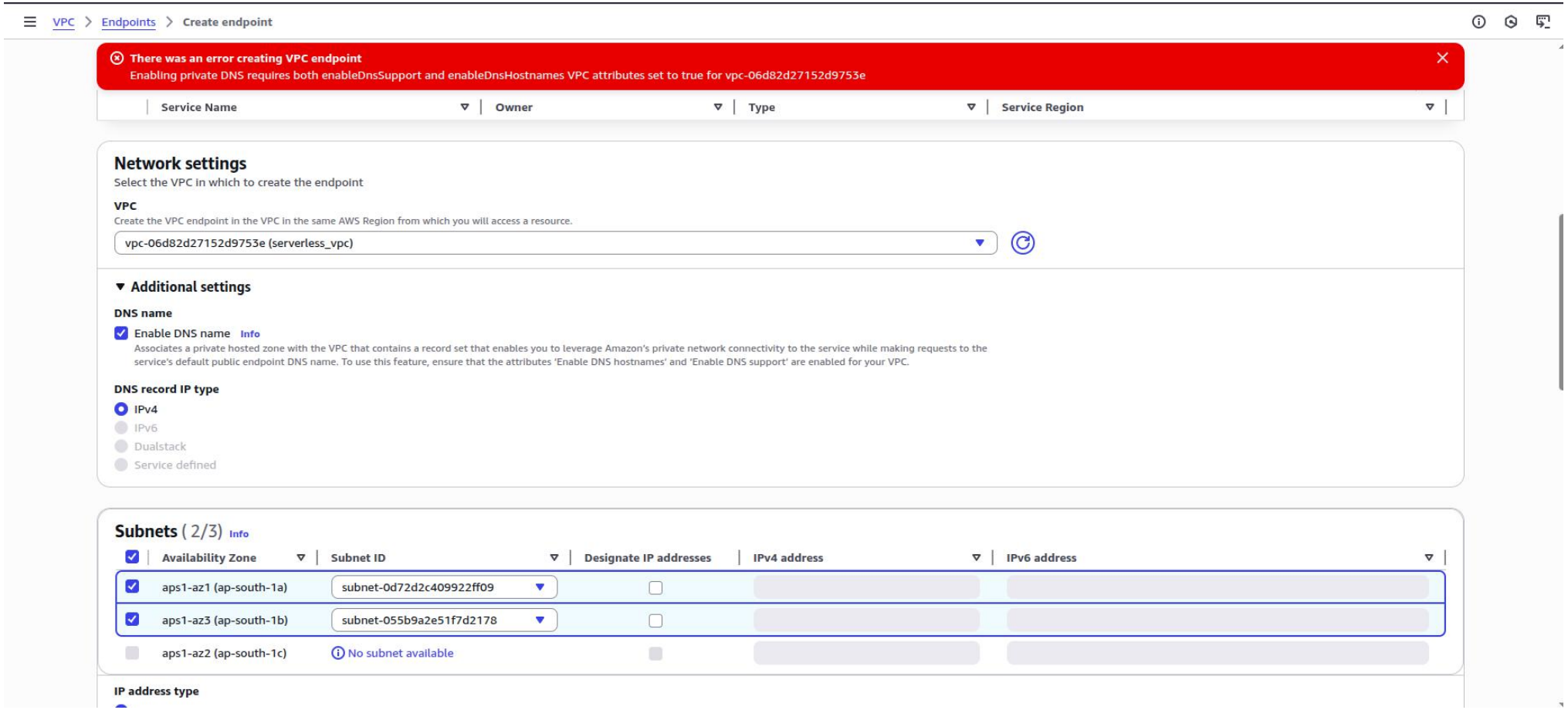


(c). Now choose the security group we have created it for the **VPC interface endpoints** to allow connections from the lambda.

If you don't want the full permission to access cloudwatch, you can attach your own custom policies with limited access.

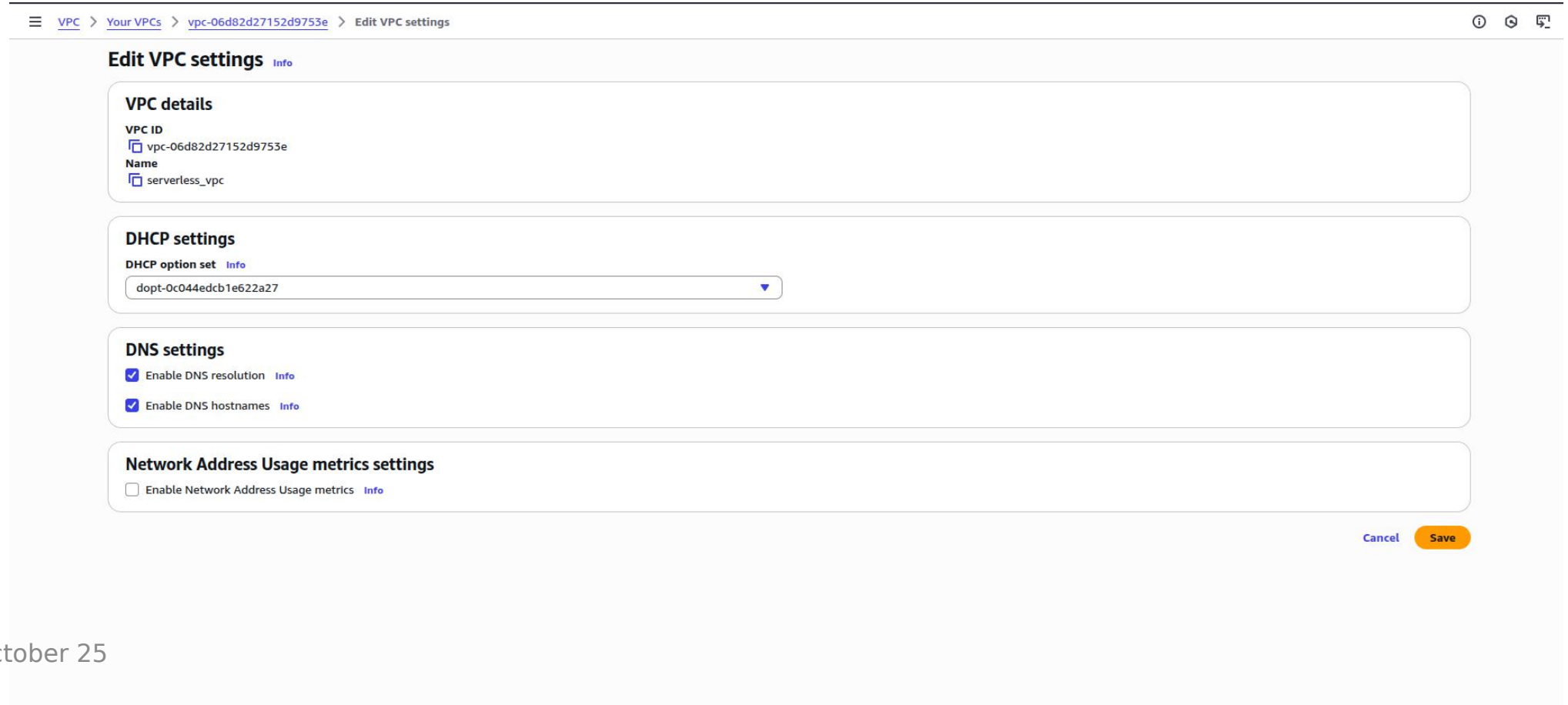


This error shows the DNS resolution and hostname to be enabled.
By default only DNS resolution is enabled, we need to enable DNS hostname manually in VPC settings.

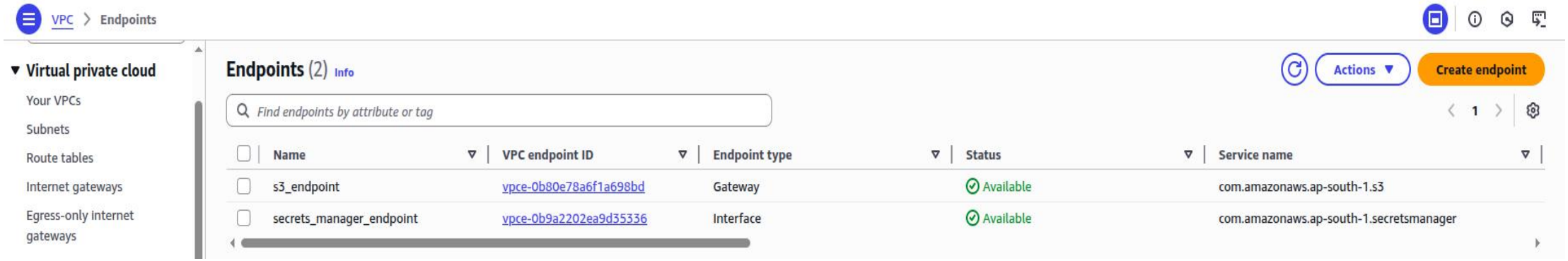


(d).Make sure to enable both DNS resolution and hostname. Allows resources in your VPC (eg,. lambda, EC2) to resolve public DNS names (like amazon.com, api.github.com) to IP addresses.
VPC Endpoints (they rely on DNS to route traffic internally)

VPC --> Your VPC's --> Edit VPC settings --> DNS settings --> Enable DNS resolution & hostnames



3. We have created two endpoints (Gateway and Interface) for accessing S3 bucket and secrets manager.

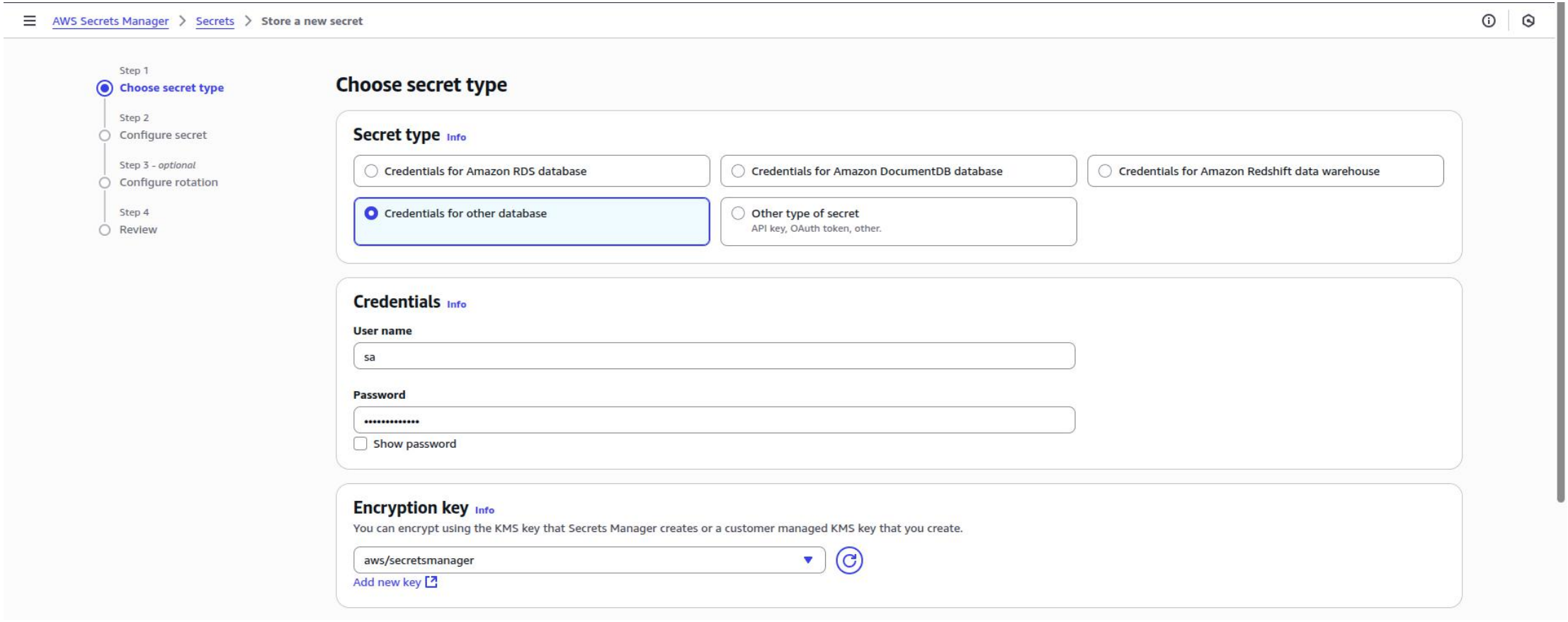


Also we can see four ENI's created in the network interface for secrets manager & cloudwatch.

7. Secrets Manager for DB credentials

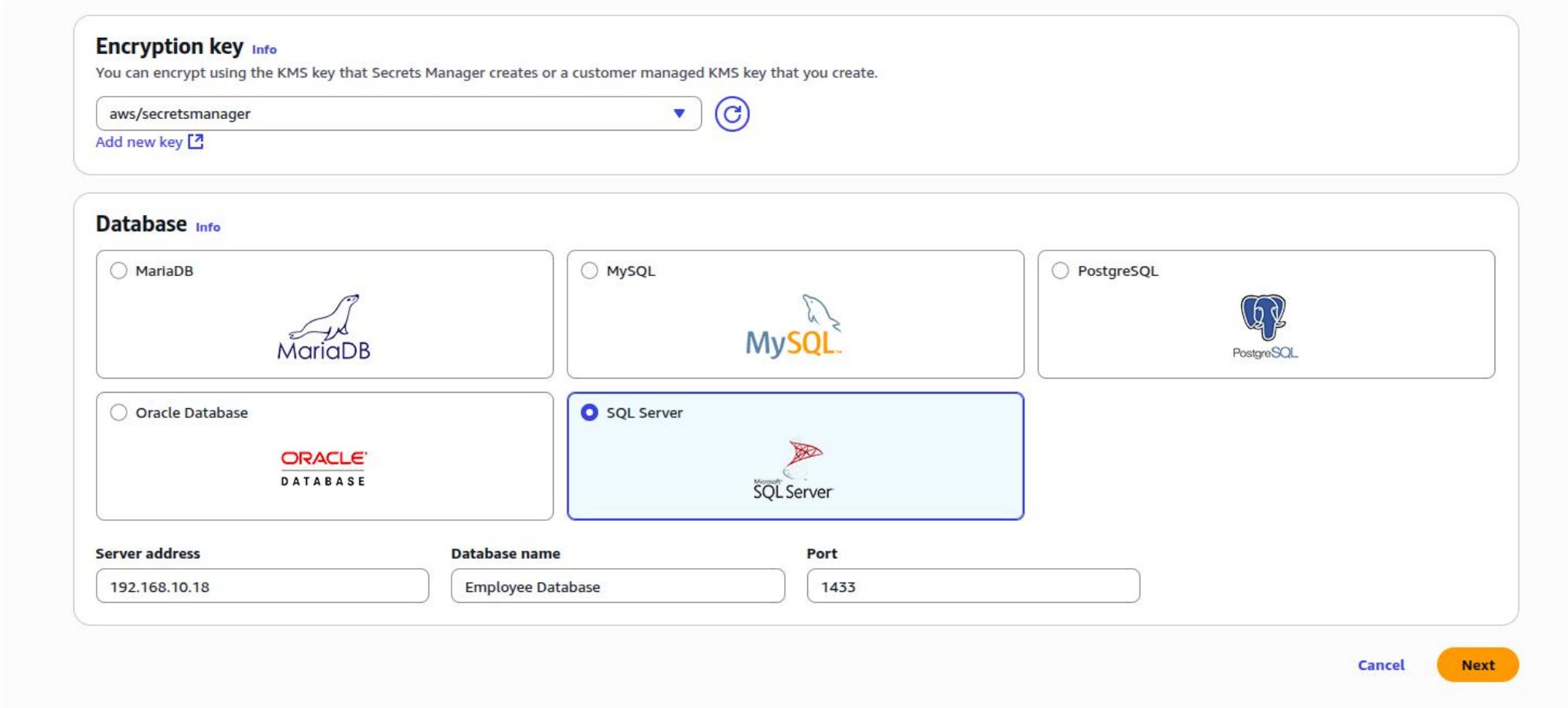
1. Create a secret for your on-prem SQL credentials (username/password).

Go to AWS secrets manager --> Secrets --> Store new secret --> credentials for other database

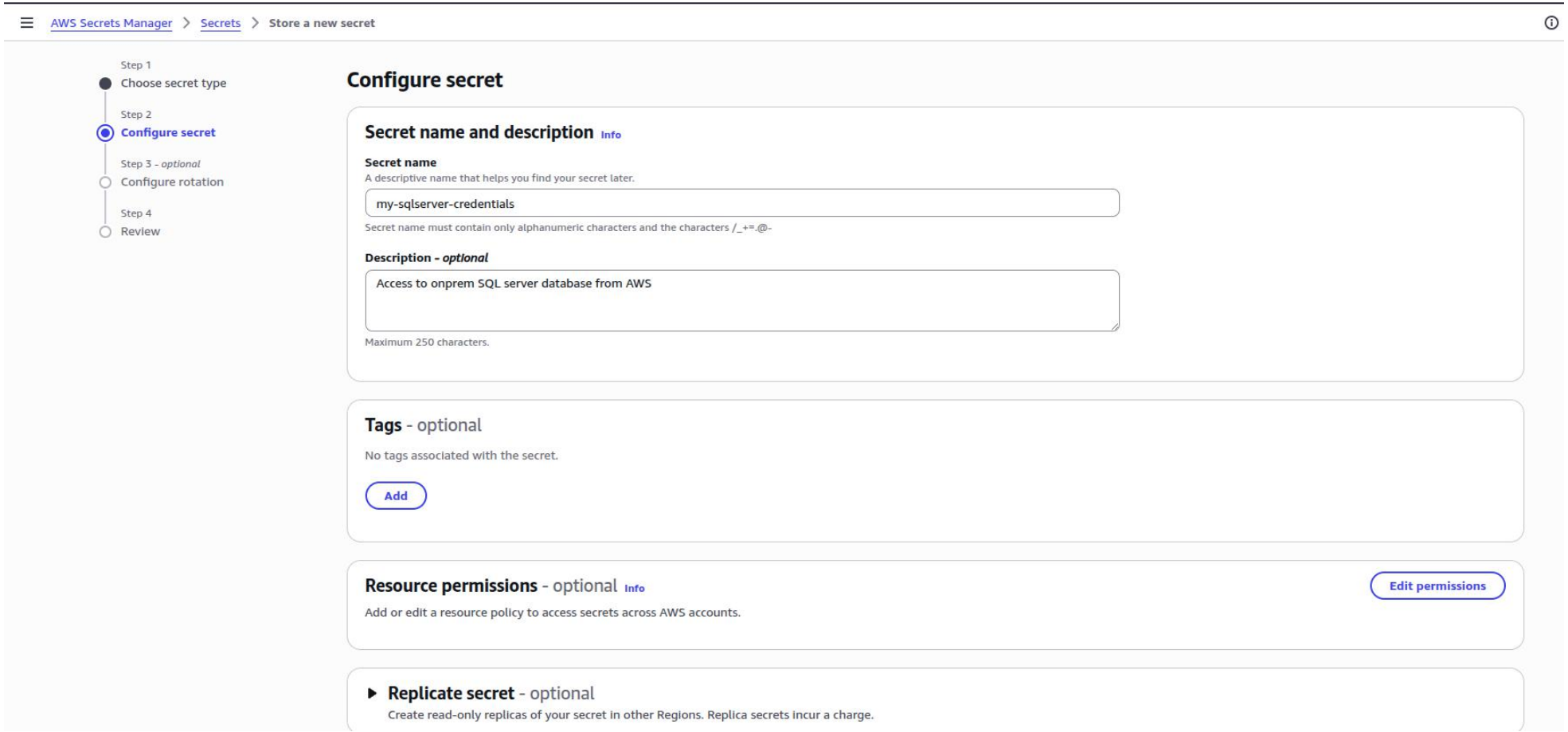


2. Choose default encryption key and select your database, in our case we are using SQL server as the database

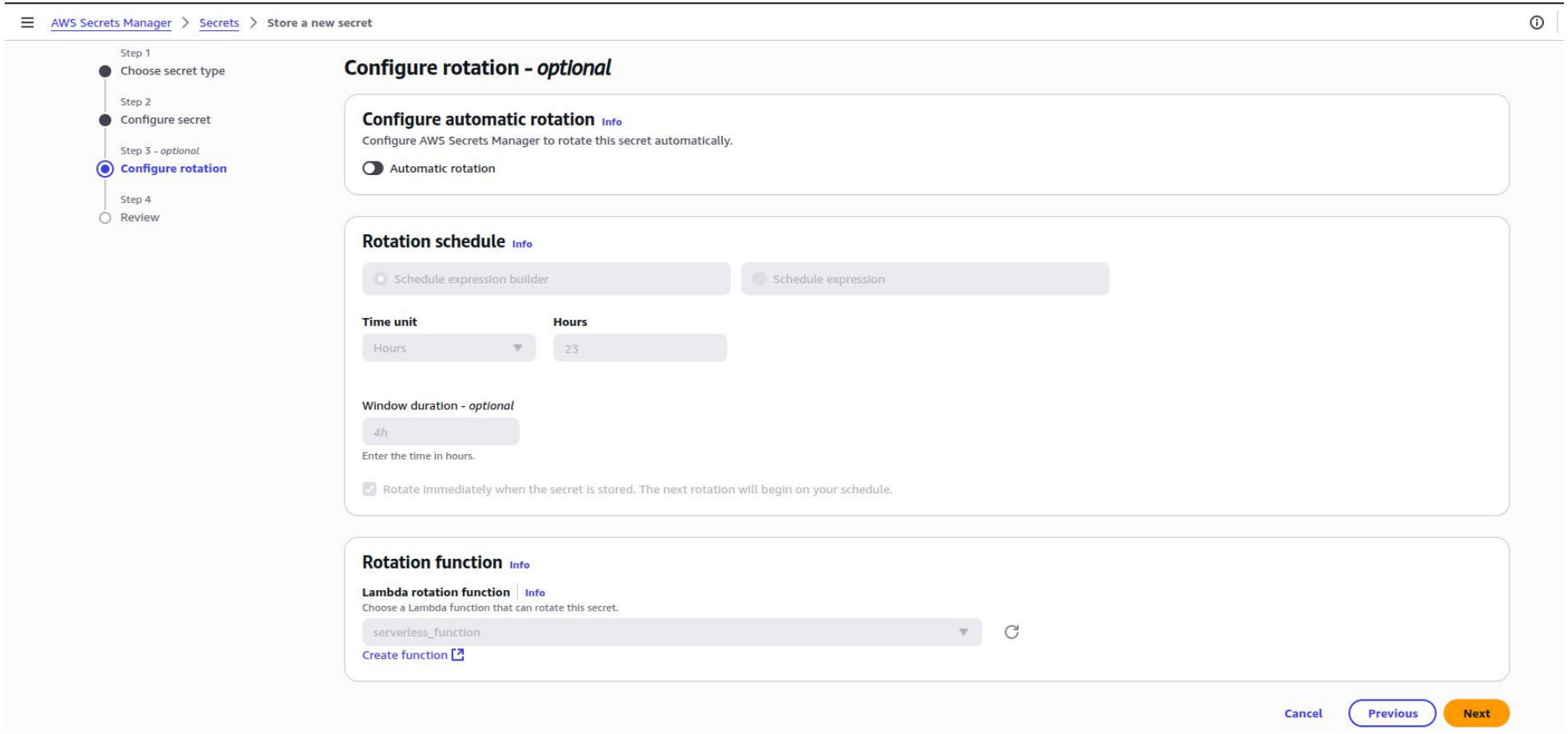
Server address: 192.168.10.18
Database name: Employee Database
Port: 1433



3. Name your secret name **my-sqlserver-credentials** and click next.

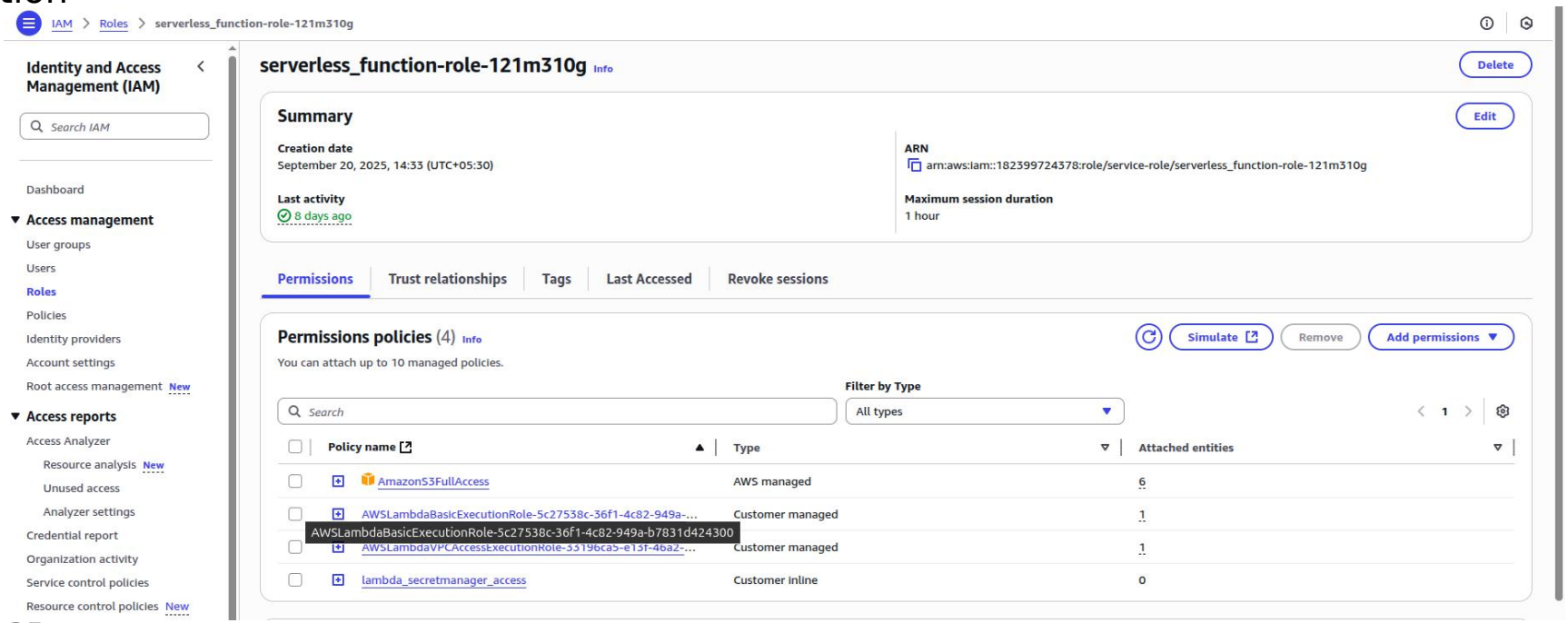


4. Leave the rotation configuration as it is and select our **Lambda functio**n, which we will cover in the next section.



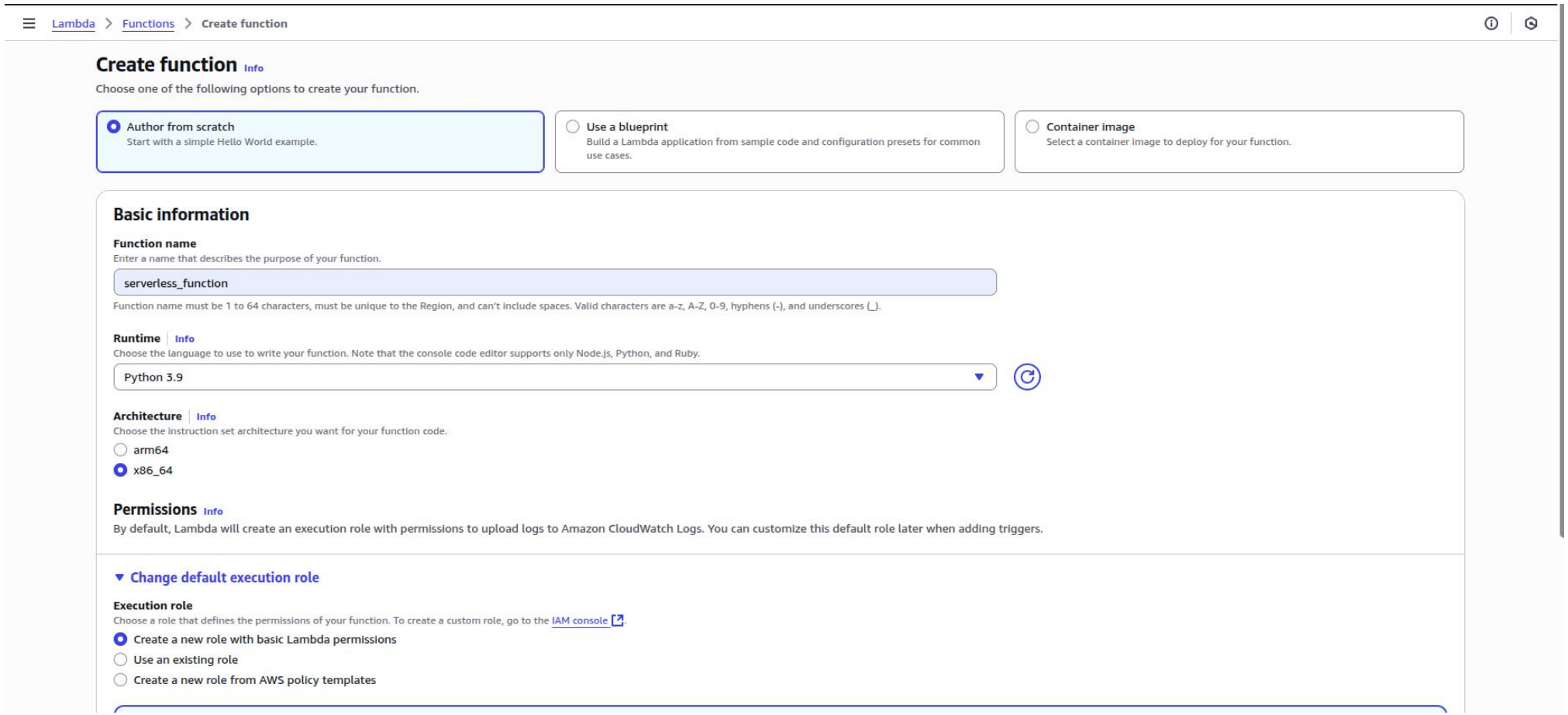
5. Now the DB credentials stored in the AWS secrets manager

Give the **secrets manager access** and **S3 full access** permissions in the IAM role created by lambda function

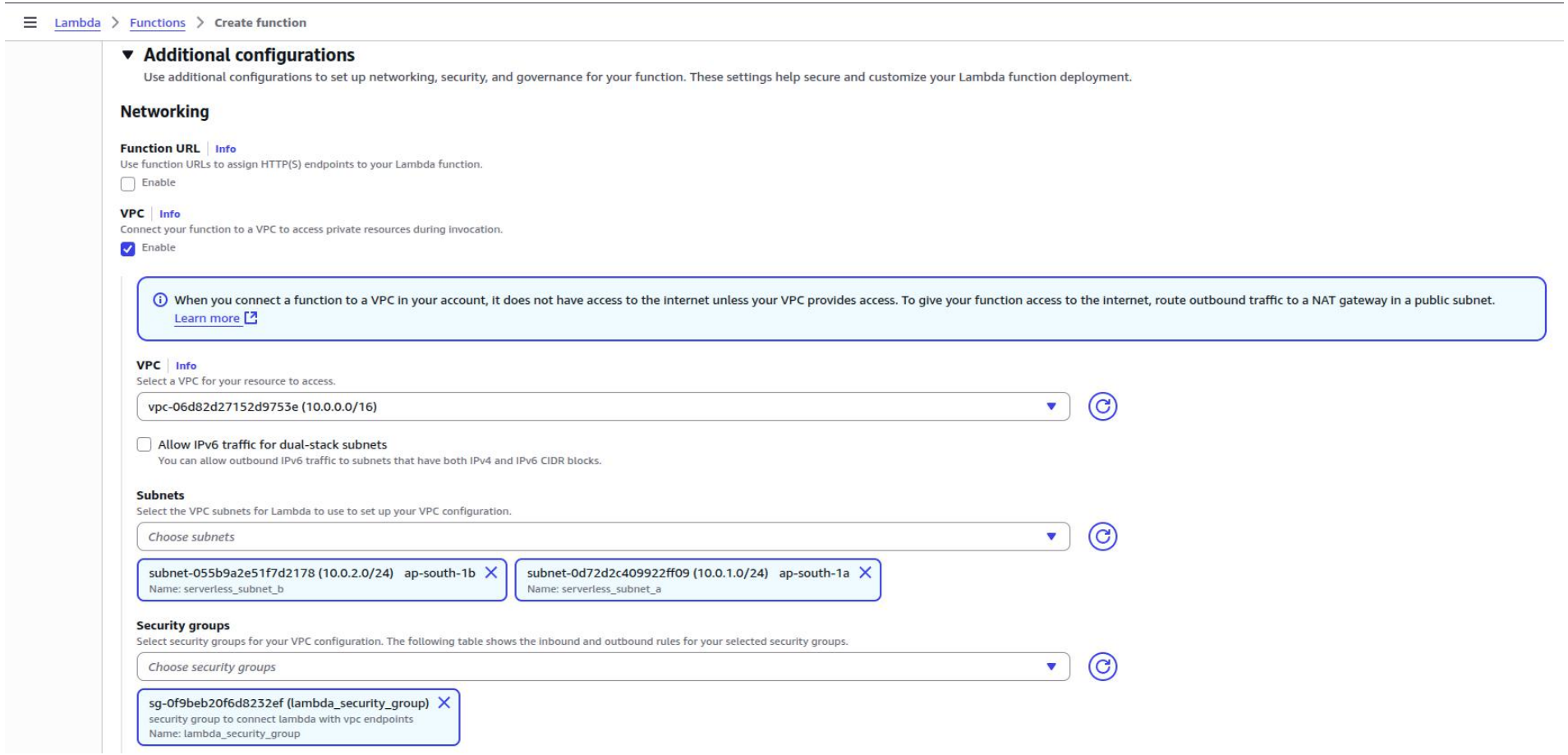


8. Lambda inside VPC

1. Choose **author from scratch**, create a function name and choose your runtime **(eg., Python 3.9**)
select create new role with **basic lambda permissions**

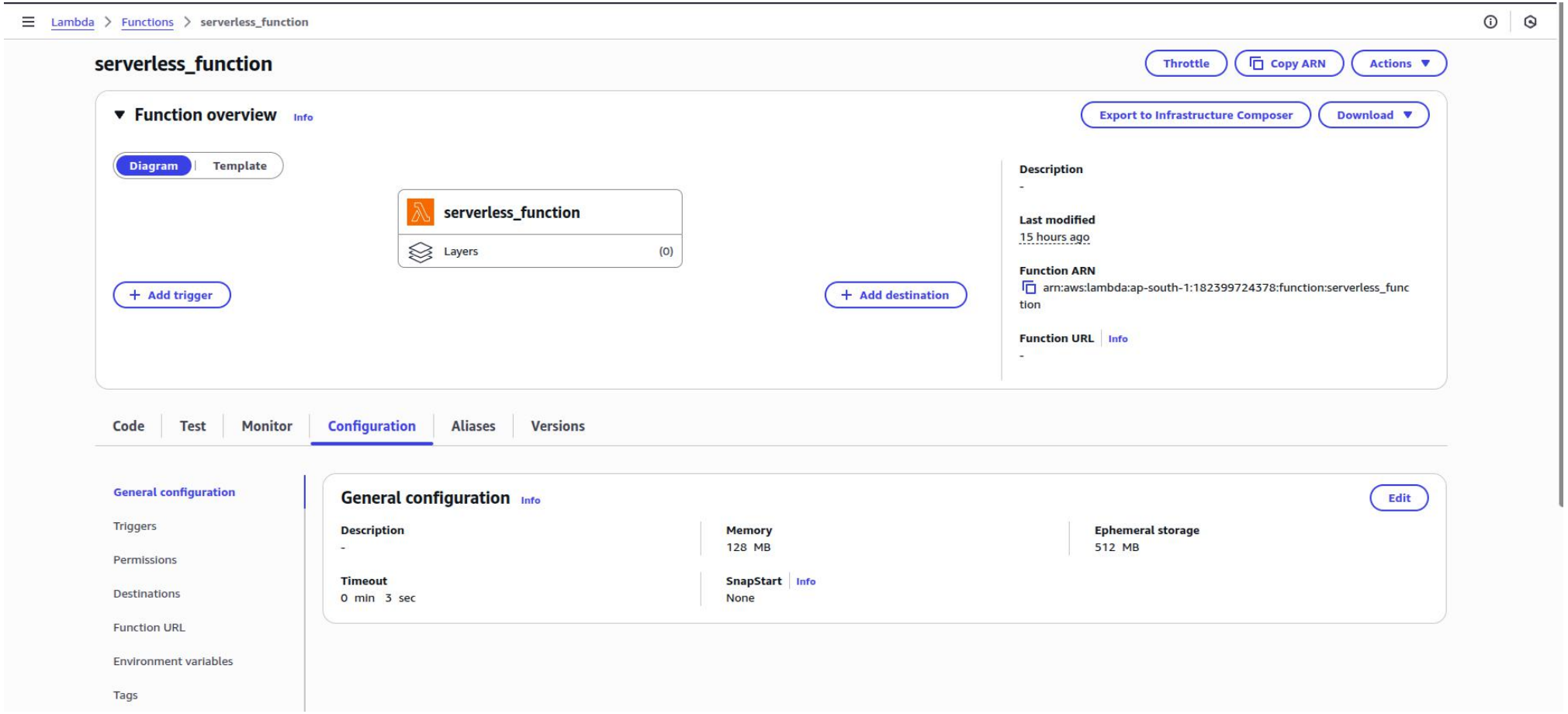


2. Click on **enable VPC** and configure VPC networking: select both the **private subnets** and **Lambda Security group**. For resilience AWS recommends at least two private subnets in different AZ's because if one AZ goes down, your lambda can still run.

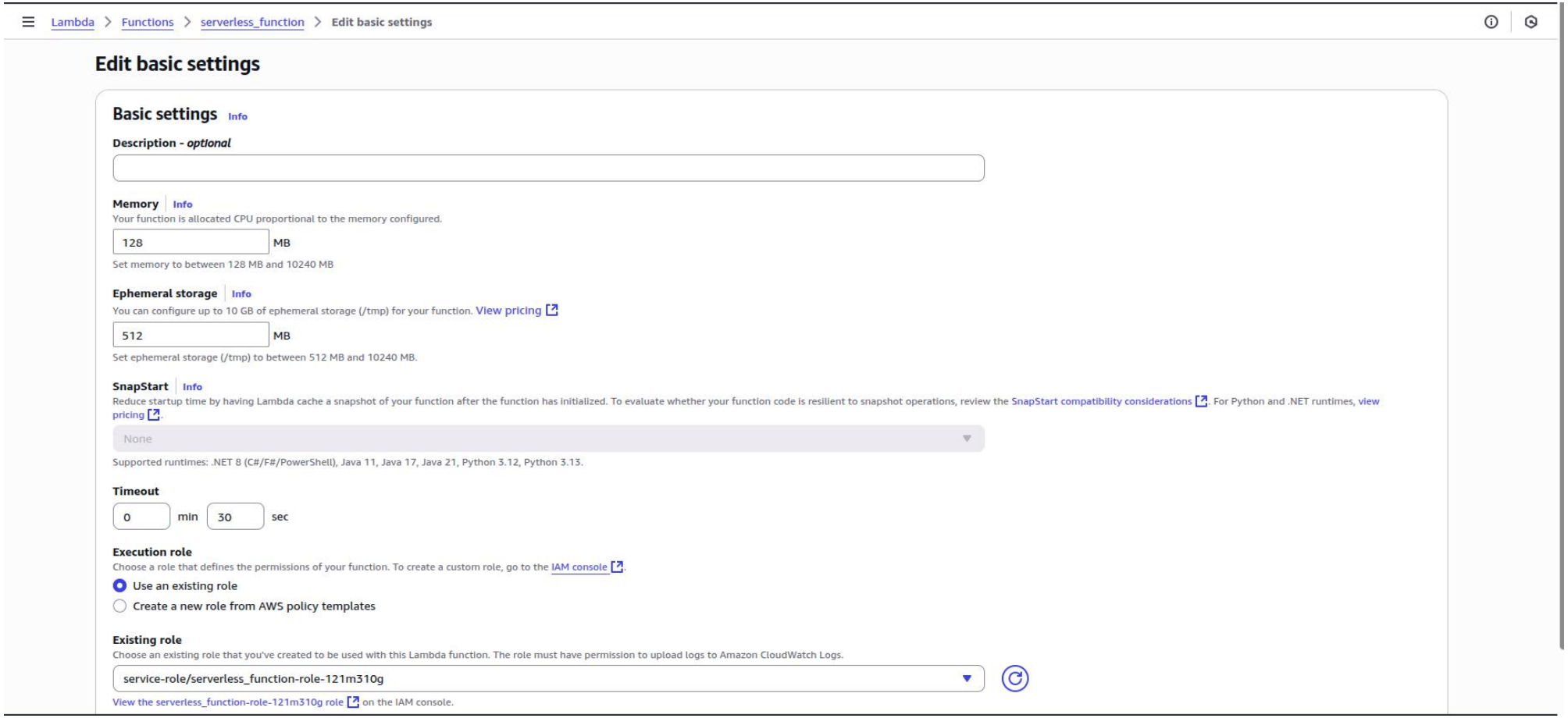


3. Now the function has been created and you can edit the configurations

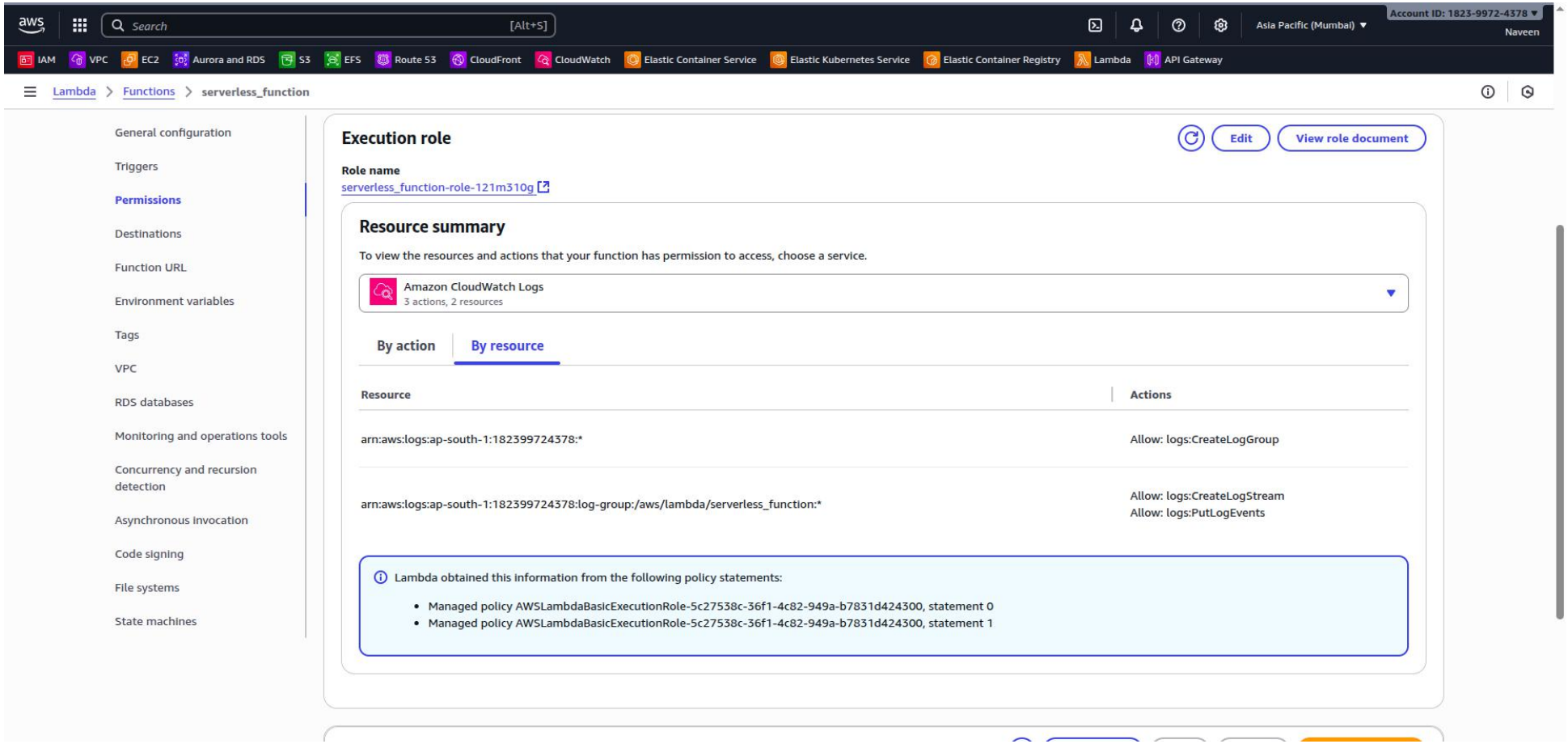
Lambda --> Functions --> Configurations



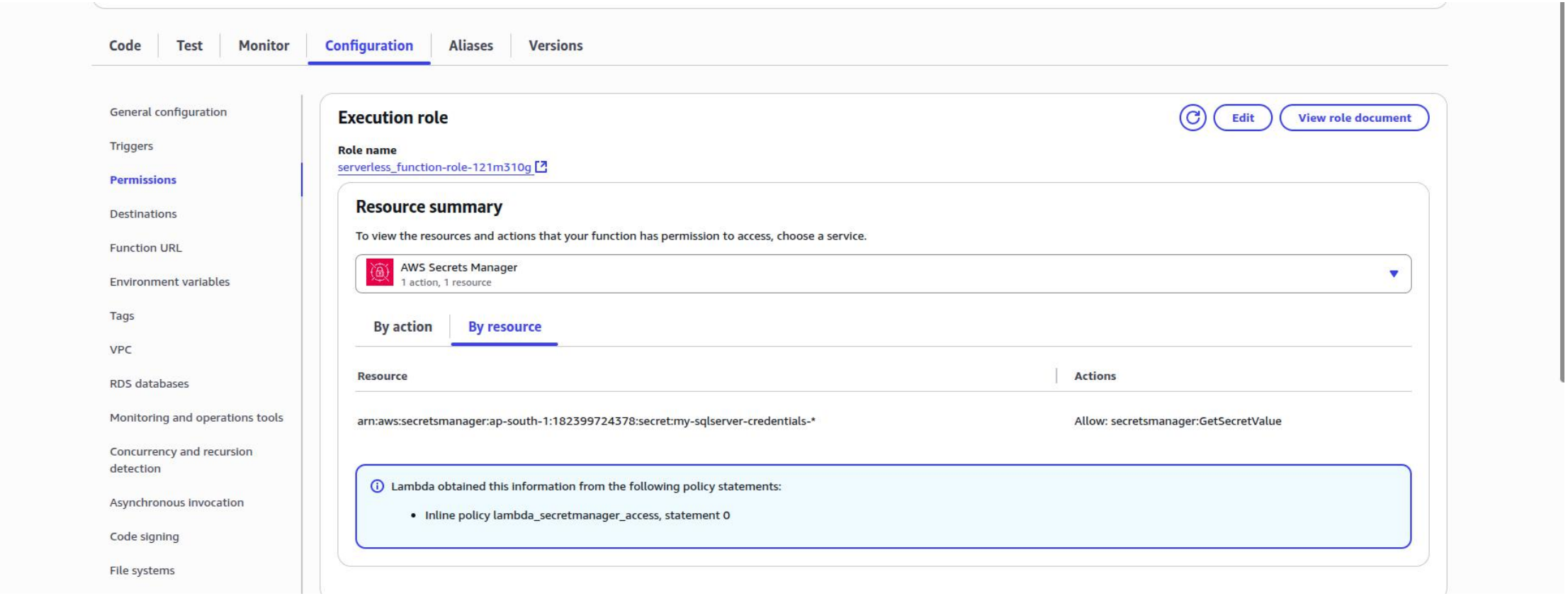
4. Make sure the memory has been set to **128MB** by default value and change the timeout to min **30 secs** for making lambda to connect with your on premise datacenter.



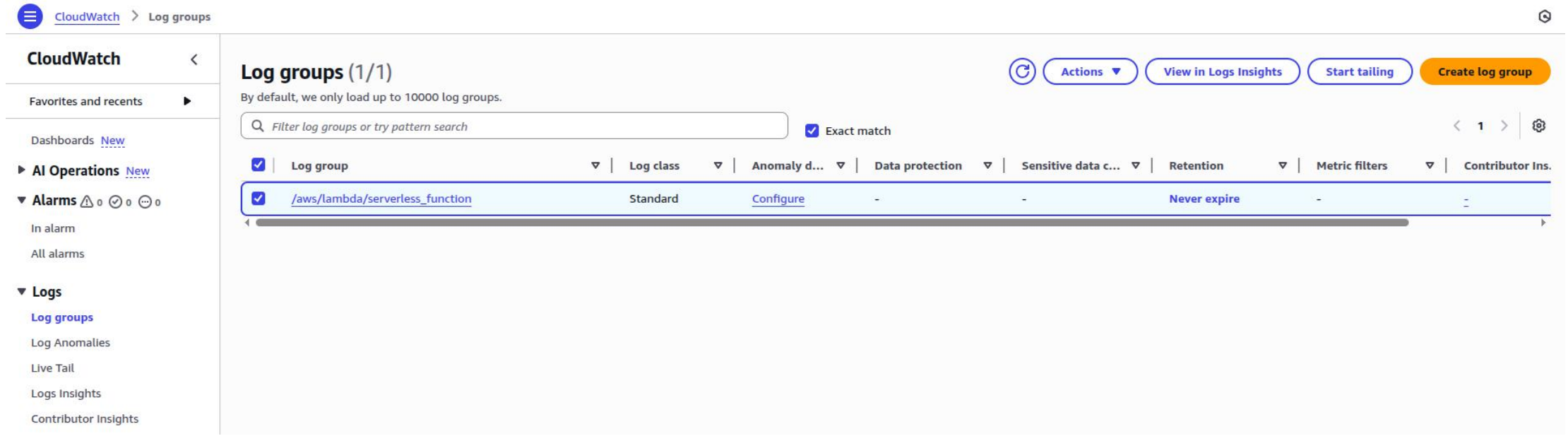
5. Go to **Lambda --> Functions --> Configuration --> Permissions** to check the permissions assigned by lambda, by default the access will be provided for cloudwatch and EC2 network interfaces.



6. Ensure IAM role has permissions: Secrets Manager, CloudWatch Logs, necessary S3/GetObject if function reads static assets, and network access as needed. It can be viewed in resource summary in lambda function.

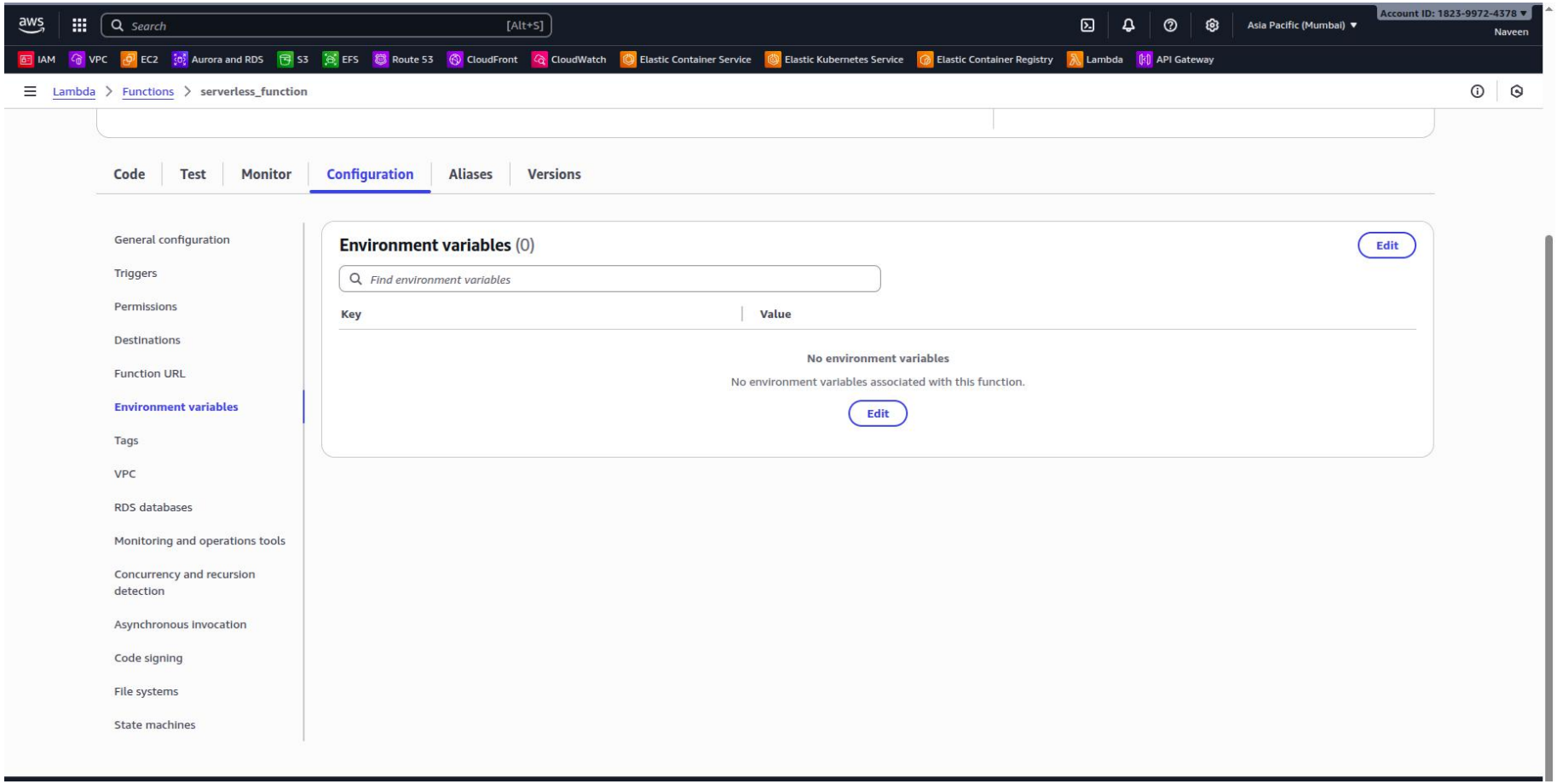


7. Meanwhile you can check cloudwatch service, **a log group** will be created automatically by lambda function to store the exection logs.

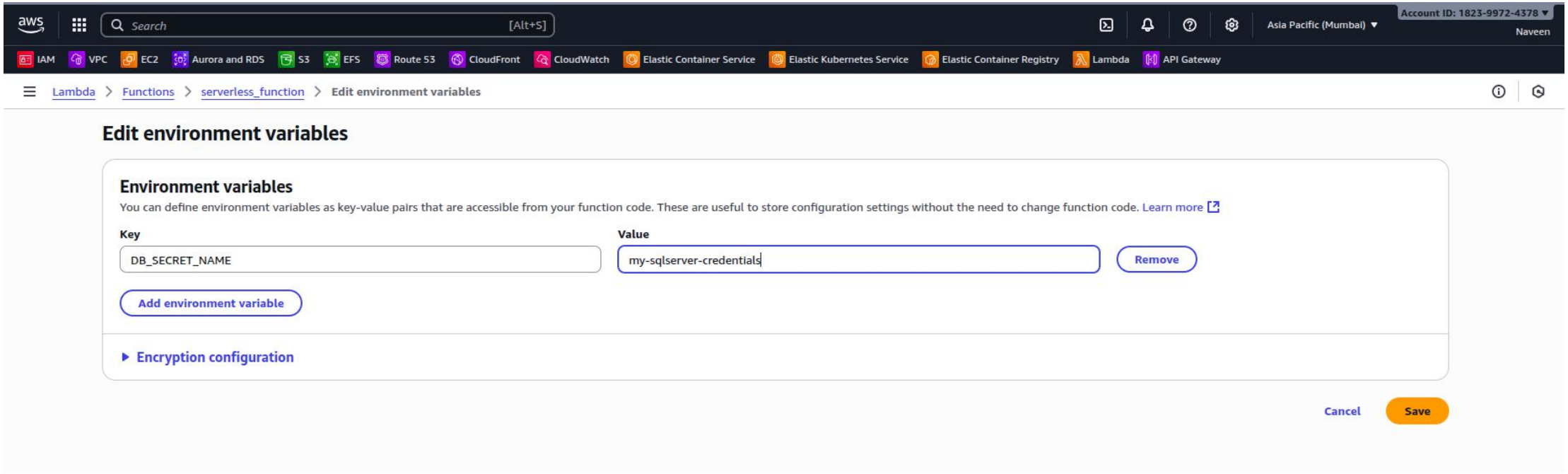


8. Now edit your environment variables for lambda to access the secrets manager for the DB credentials

Lambda --> Functions --> Configurations --> Environment Variables

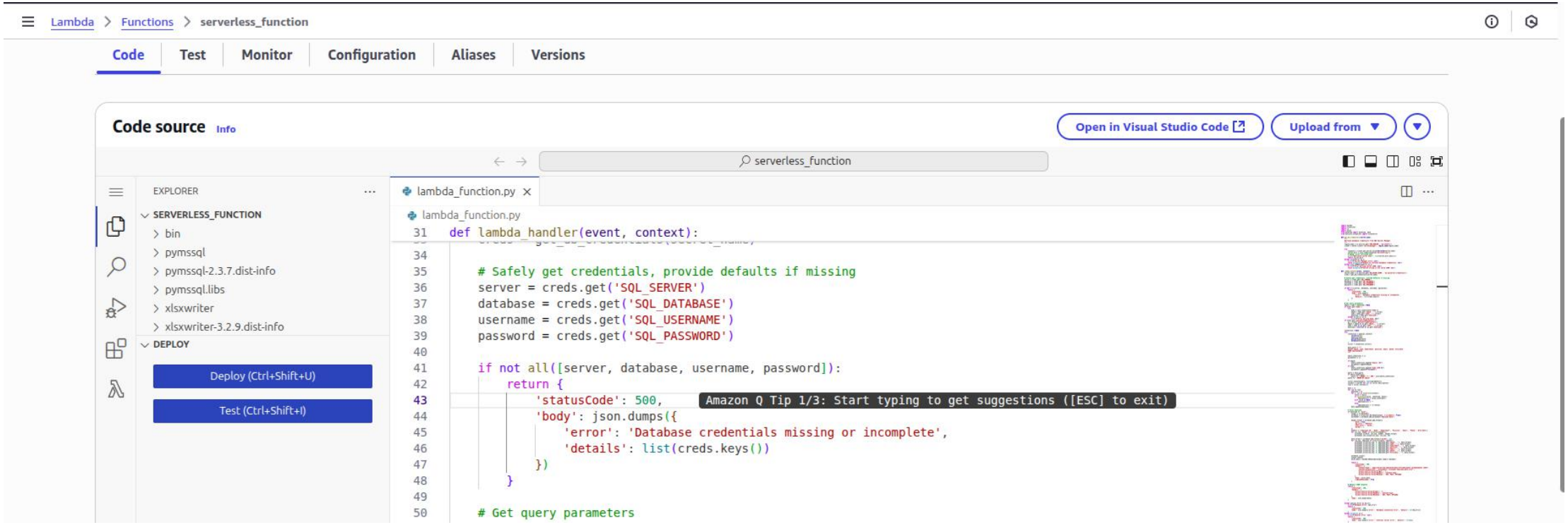


9. This key and value name should be same as you mentioned in the lambda function code and in the Secrets manager name.

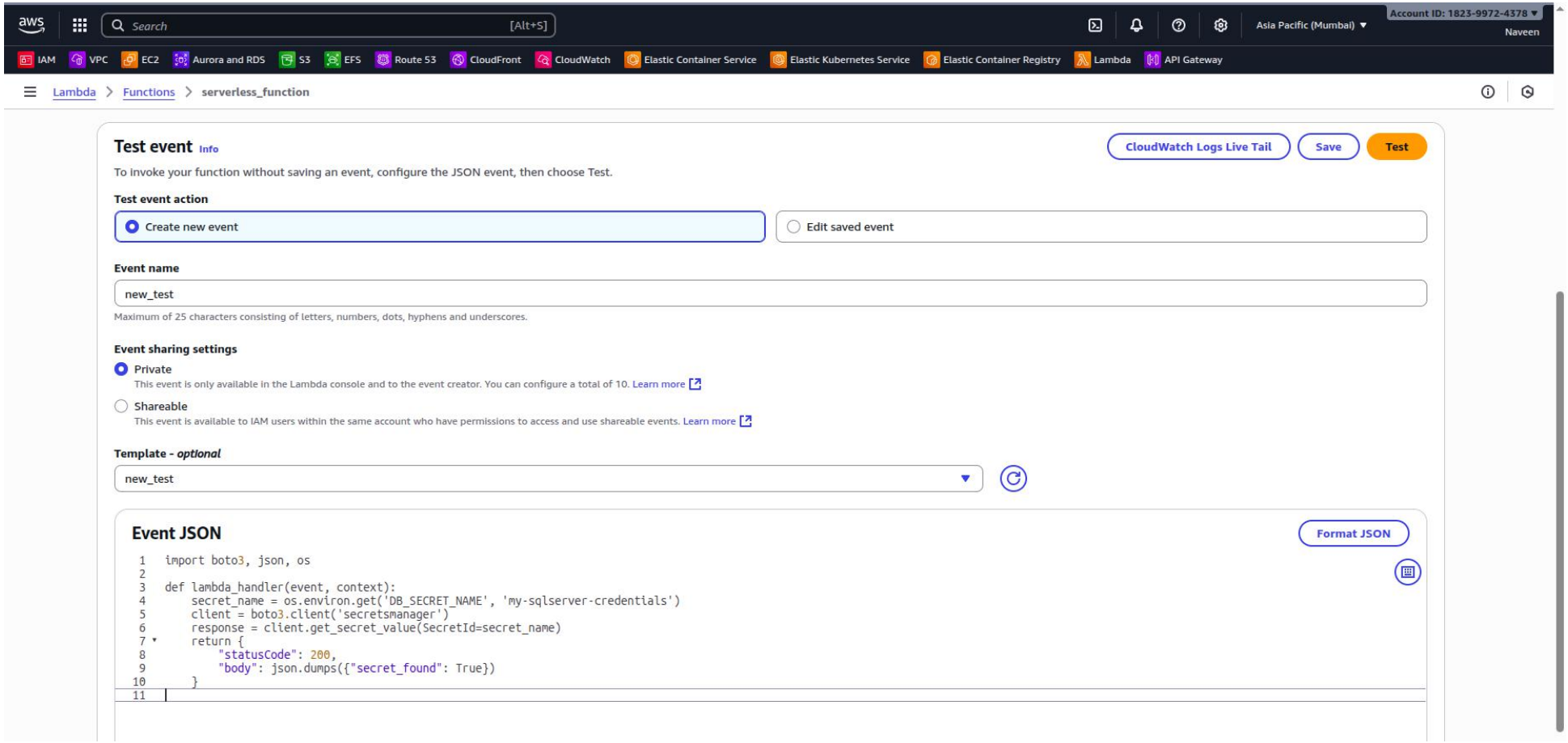


10. Create your backend script **.py** file with the dependecy packages as a **.zip** file and upload those files in the code source and **deploy (Ctrl+shift+U)**

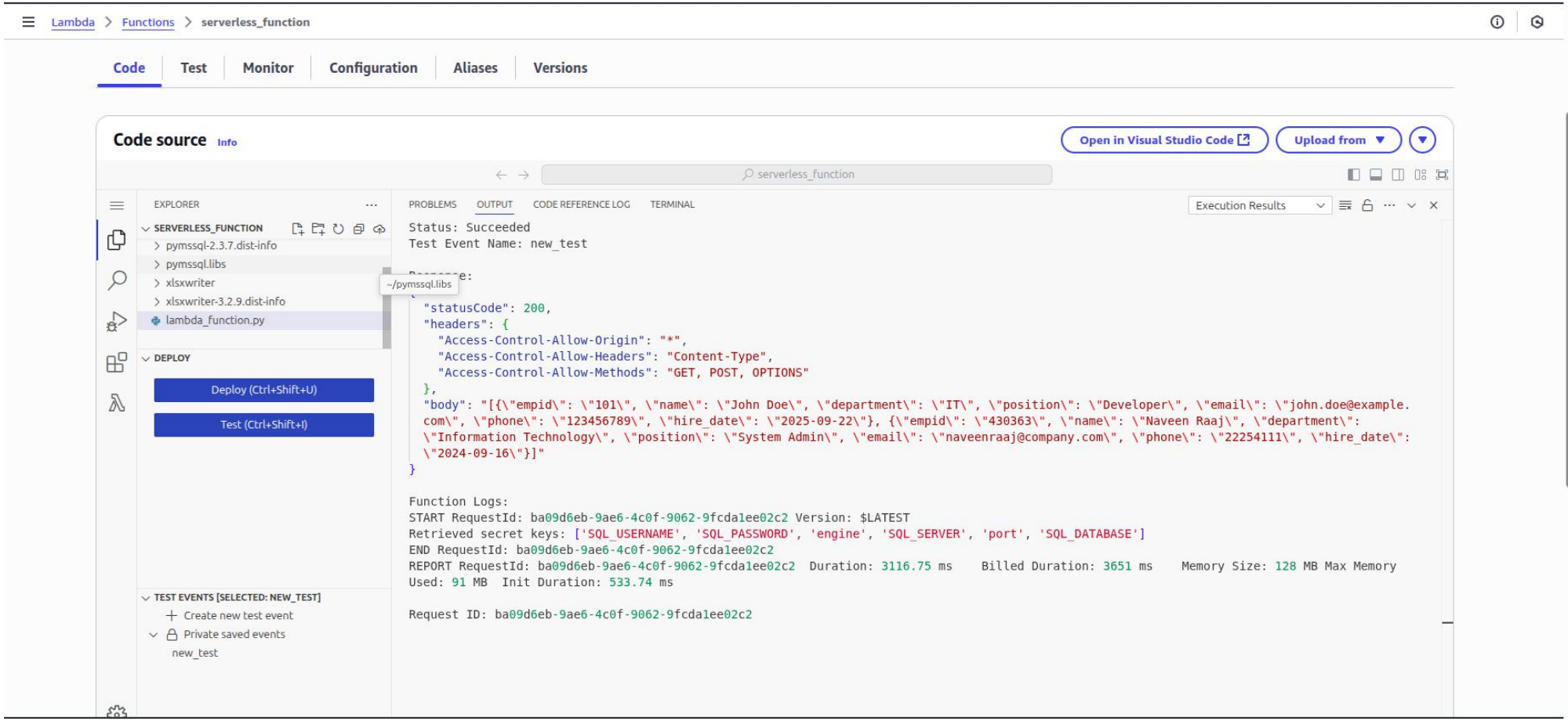
Note: This architecture requires connecting to a SQL Server database and exporting the data in Excel format. For this, we need two dependency libraries (pymssql and xlsxwriter). Unfortunately, AWS Lambda does not provide these by default. These libraries must be packaged manually from an Amazon Linux environment, since Lambda runs on Amazon Linux and only supports binaries built for that OS.



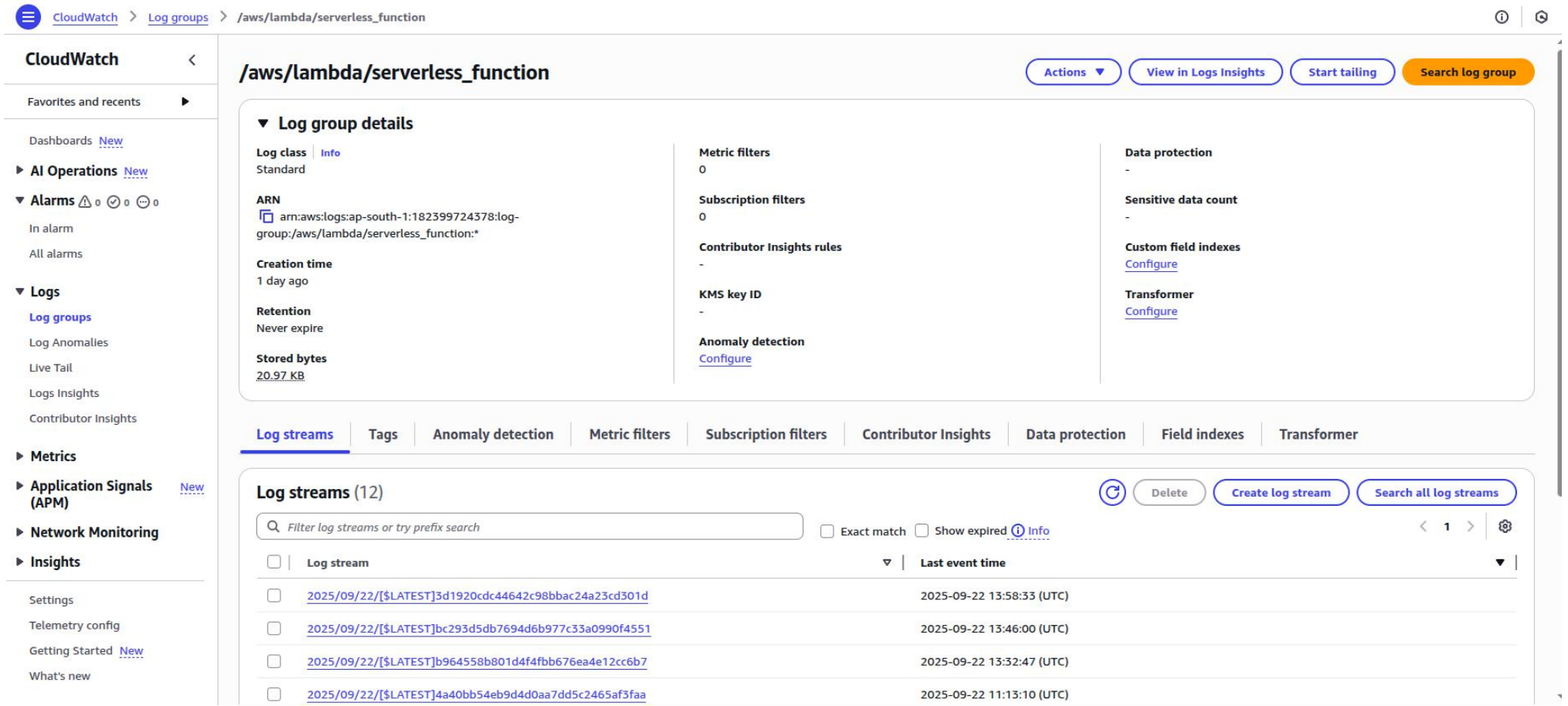
11. Once the code is deployed, you can do the code test by creating a **new test event** as JSON template and check the output for errors.



12. Click on **Test (Ctrl+Shift+I).** Now the lambda's console gives me the output of my SQL server database!



13. You can also check the logs from the Log groups in the cloudwatch.



9.API Gateway (Regional)

API Endpoint Types: (Regional V Private)

1. If You Use a Regional API (Recommended for Internet-facing API)

This is the most common and easiest way:

a) Create a Regional API Gateway endpoint (not Edge-optimized, not Private).

b) Deploy your API stage and get the Invoke URL (e.g. https://abcd1234.execute-api.ap-south-1.amazonaws.com/prod).

c) In CloudFront:

- Create an Origin with the API Gateway Invoke URL.
- Set Origin Protocol Policy to HTTPS Only.
- Forward required headers (Host, Authorization, etc.) via cache behavior.

a) Attach AWS WAF (optional but recommended) to CloudFront to protect API traffic.

Best for: Public APIs secured by WAF + Cognito/Auth.

Simple: No VPC endpoint needed.

2. If You Use a Private API (Inside VPC)

This requires more setup, because CloudFront is public and your API is private.

Steps:

- Create a VPC Endpoint for API Gateway (com.amazonaws.<region>.execute-api).
- Deploy your API as Private and associate it with the VPC endpoint.
- Use an internal ALB or NLB + AWS PrivateLink or a CloudFront feature called Origin Access Control with PrivateLink (newer feature) to connect CloudFront to your private API.

This ensures traffic never leaves AWS network, but it is more complex and usually used for internal-facing applications.

Feature	Regional API (Our choice)	Private API
Publicly Accessible	Yes (over HTTPS)	No (VPC-only)
Requires VPC Endpoint for Clients	No	Yes
Lambda Integration	Works	Works
Best Use Case	Public APIs (with WAF/Auth)	Internal APIs (internal dashboards, corporate network)

HTTP API v REST API

Feature / Capability	HTTP API	REST API
Cost	70% cheaper per million requests	Higher cost
Latency	Lower (optimized, faster)	Higher
WAF Support	Direct WAF association not supported → must front with **CloudFront + WAF**	Can attach WAF **directly**to REST API (regional/edge)
Integrations	Lambda, HTTP backends, Step Functions, EventBridge	Lambda, HTTP, Step Functions, DynamoDB, SQS, Kinesis, etc.
Authorization	JWT/OIDC, Lambda authorizer	IAM auth, Cognito user pools, JWT, Lambda
Usage Plans / API Keys	Limited support	Full support
Caching	Not supported	Supported
Best Use Case	Cost-sensitive apps, microservices, backend for SPA/mobile	Legacy APIs, APIs needing WAF directly, caching, API keys, heavy customization

Notes:

HTTP API is far cheaper for typical serverless backends. Choose HTTP API unless you need REST features (API keys, complex authorizers, request/response transformations)

How API Gateway Connects to Lambda in Private Subnet

Lambda in VPC:

- Our Lambda is attached to a VPC (private subnets), AWS creates ENIs (Elastic Network Interfaces) in those subnets.
- The Lambda function now has a private IP address inside your VPC.

API Gateway → Lambda:

- Regional API Gateway **does not need to be in the VPC**.
- This API Gateway invokes Lambda using **AWS's internal control plane network**, not the public internet.
- This means **no NAT Gateway or VPC Endpoint** is required just to allow regional API Gateway to talk to Lambda.
- So your Lambda can stay in private subnets (for DB/VPN access), and API Gateway can stay **Regional/Public** — they work together seamlessly.

Security Considerations

1. CORS (Cross-Origin Resource Sharing):

- If your frontend is served via CloudFront (say, https://app.example.com) and your API Gateway uses its default URL (https://abc123.execute-api.region.amazonaws.com), you must enable CORS on API Gateway to allow the browser to call it.
- Best practice: Use a Custom Domain for API Gateway (e.g., api.example.com) so both frontend and API share the same root domain — this simplifies CORS.

2. Exposing API Endpoint Publicly:

- Because you're embedding the API URL in JavaScript, anyone can see it in browser DevTools.
- Protect your API with authentication/authorization (JWT, Cognito, or custom Lambda authorizer) and use WAF to mitigate abuse.

3. WAF Coverage:

- Currently, your WAF is attached to CloudFront, so it only protects your static website — not your API calls.
- To protect the API, you either:
- Attach WAF directly to API Gateway (Regional API), OR
- Put CloudFront in front of API Gateway as a second distribution.

4. Authentication/Authorization:

- You can use API Gateway's built-in authorizers (JWT, Cognito, Lambda authorizer).

Option 1: Keep API URL in JavaScript (simple method)

Pros

- Simple to set up — no changes to CloudFront needed.
- Works even if you have multiple APIs — you just hardcode the URLs in JS.
- Easy to test locally (you can call the API Gateway endpoint directly).

Cons

- API Gateway URL is visible to anyone who views your site (inspect → sources → entire javascript).
- Requires CORS configuration in API Gateway (to allow browser calls).
- WAF on CloudFront does not protect API traffic — you must attach WAF to API Gateway separately.
- Harder to switch between environments (dev, staging, prod) unless you use build-time environment variables.

Option 2: Route API Traffic via CloudFront (Recommended)

Here, you configure CloudFront with two origins:

Origin 1: S3 bucket (for static files)
Origin 2: API Gateway (for /api/* paths)

Then you configure a cache behavior:

Requests for /api/* → CloudFront routes to API Gateway origin.
Requests for everything else → CloudFront routes to S3.

Pros

- Single domain for frontend + API (e.g., https://app.example.com/api/employees):
- No CORS required
- Cleaner architecture
- API Gateway URL hidden (users only see CloudFront domain).
- WAF protects both static files and API (because WAF is attached to CloudFront).
- You can enable API caching at CloudFront for GET requests (better performance & lower cost).
- Easier for blue/green deployments — you can swap API Gateway origin without changing JS.

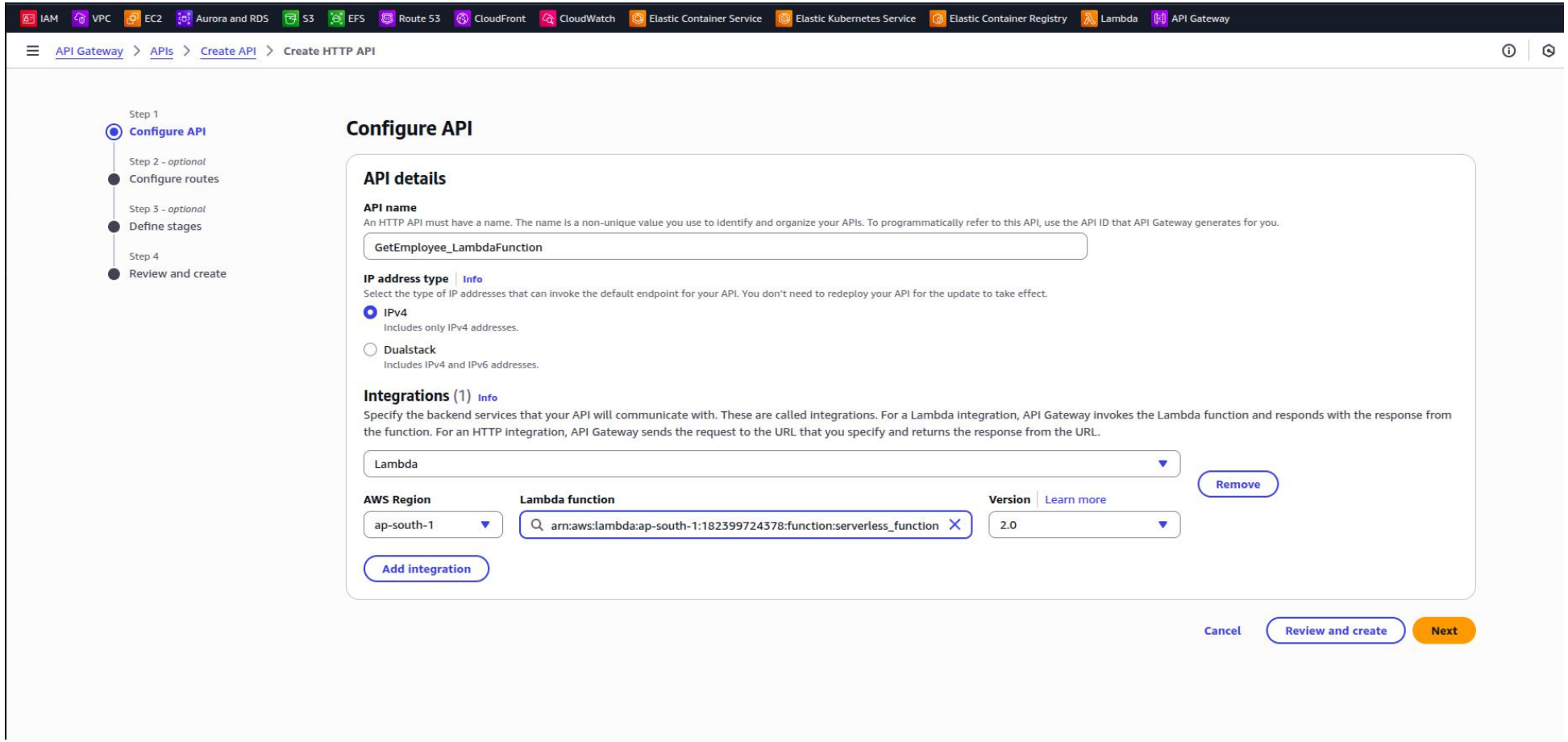
Cons

- Slightly more setup in CloudFront (add second origin, cache behavior).
- Must forward headers/query strings/cookies properly for API calls.

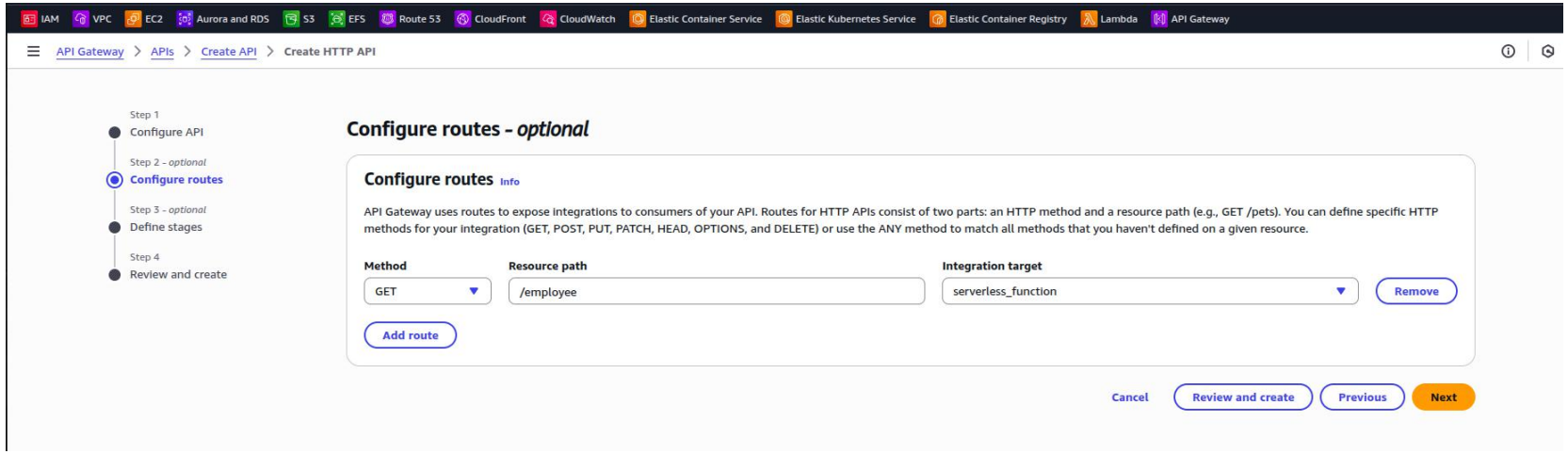
What This Means for Security

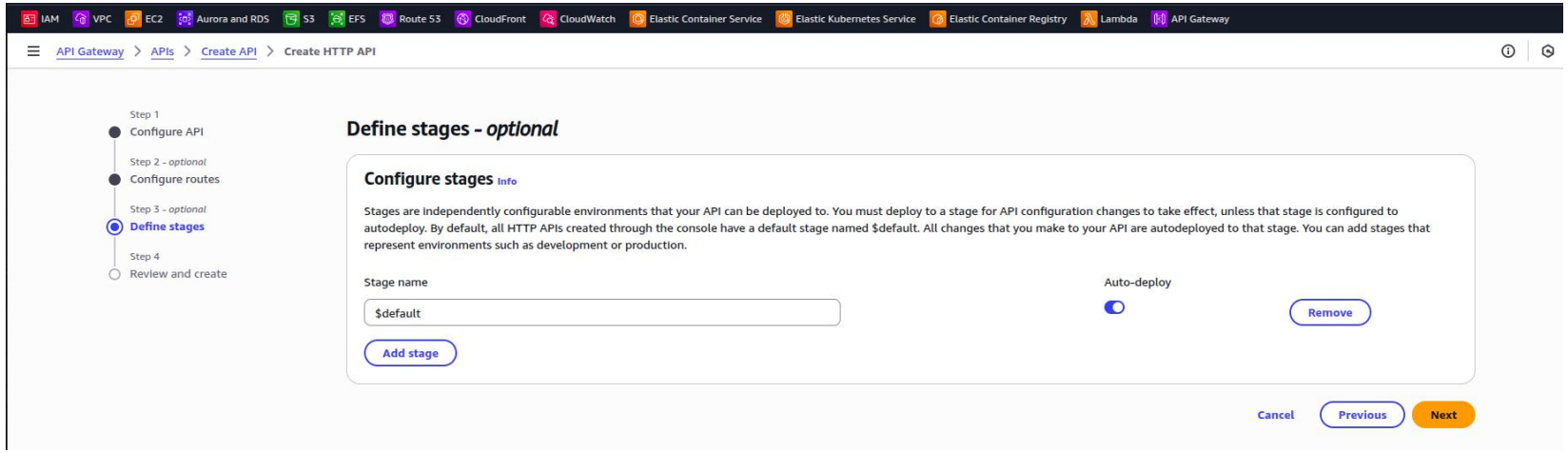
- Do not put secrets (like database passwords, API keys, tokens) in JS.
- Your API Gateway endpoint will always be visible — you must secure it properly:
- Require authentication (JWT/Cognito/Auth header).
- Use WAF to block malicious IPs, SQL injection, bots.

1. Create an API Gateway **HTTP API** (or REST API if you need features). For lower cost & common serverless flows, HTTP API is recommended. Choose our lambda function and the region in the integrations.



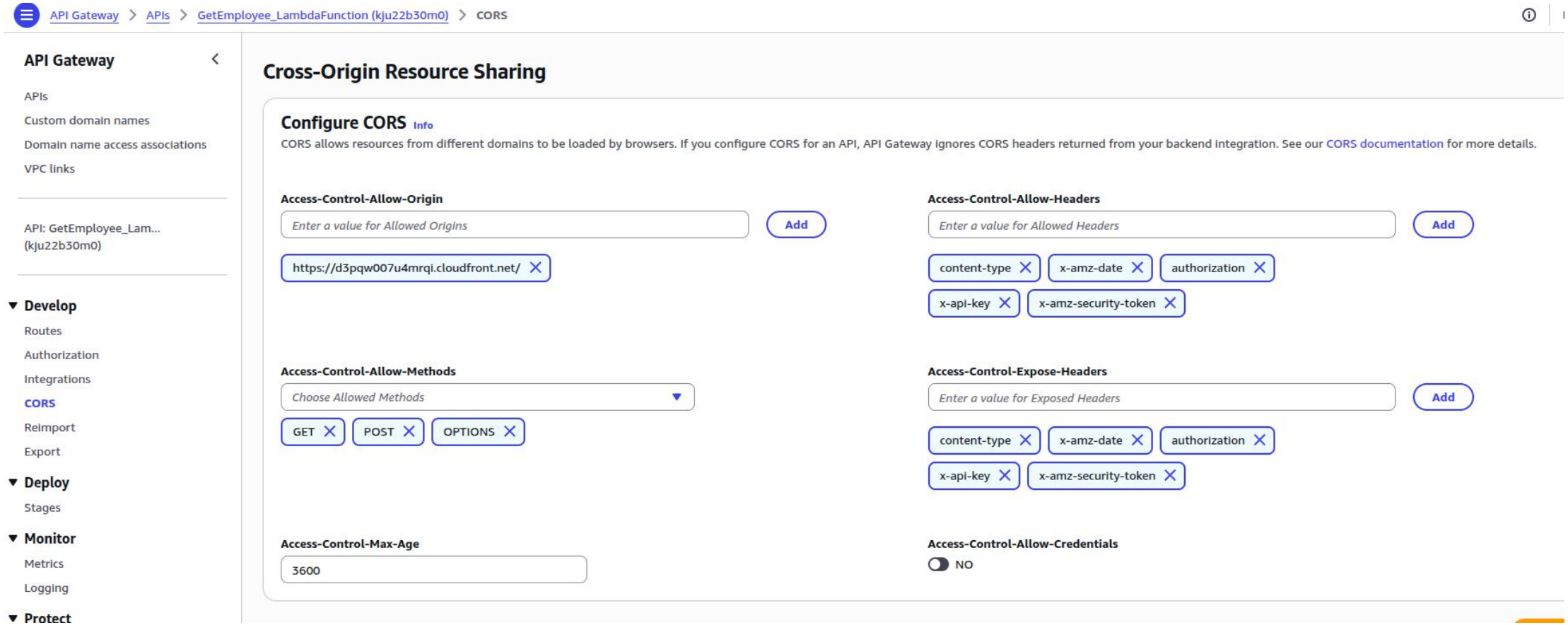
2. Configure routes/integrations: select **GET method** and mention **/employee** as resource path integrate API Gateway routes to Lambda and name the stage and click next.



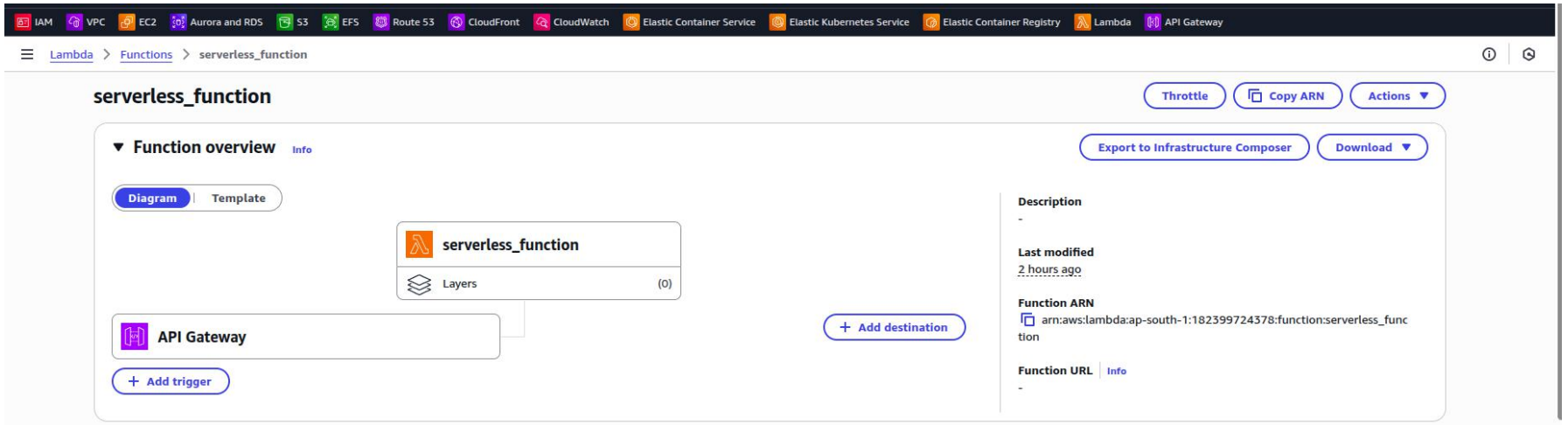


3. Go to **API gateway --> API's --> Develop --> CORS**

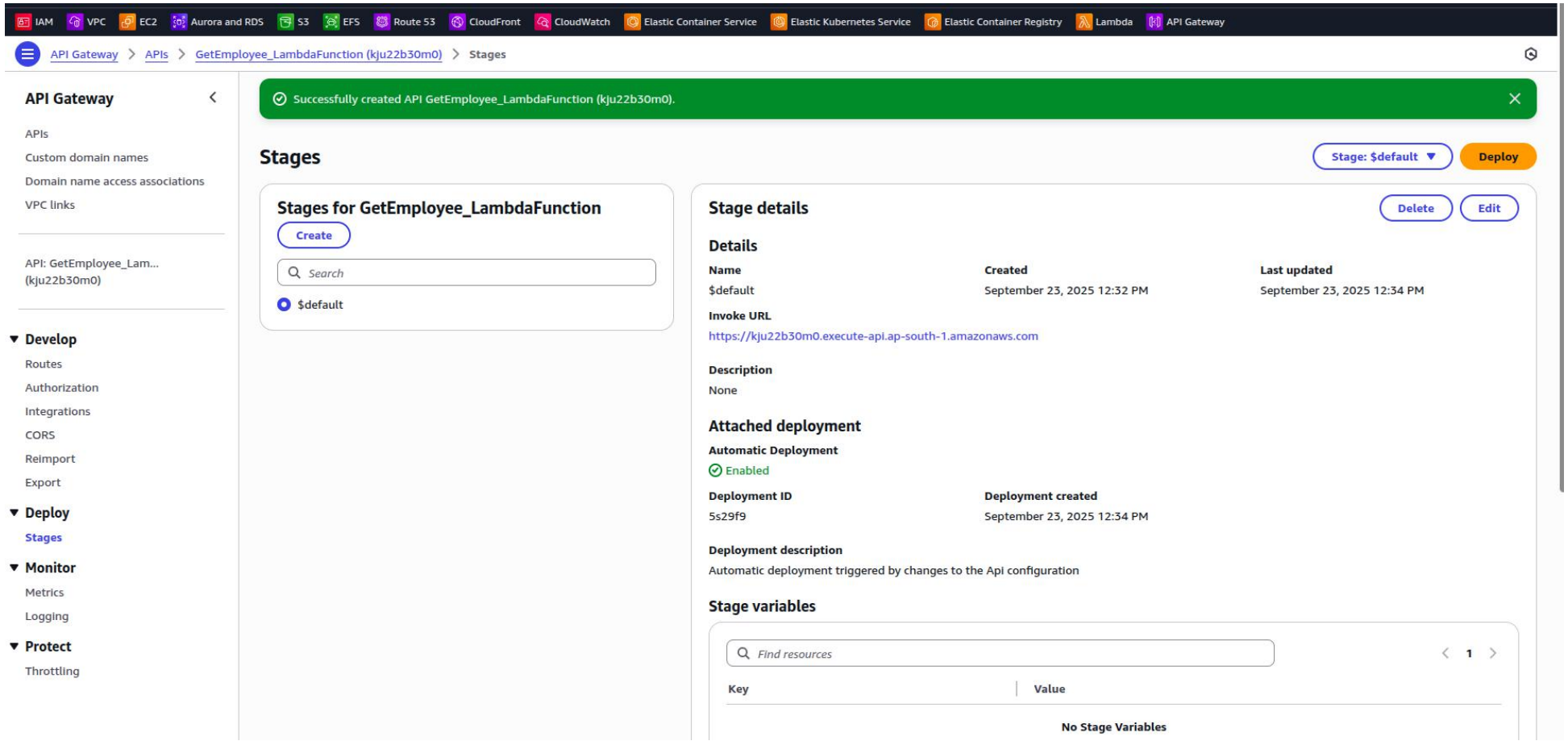
Allow-control-allow-origin: <cloudfront URL>
Access-control-allow-methods: GET POST OPTIONS
Access-control-allow-headers: content-type z-amz-date authorization x-api-key x-amz-security-token
Access-control-expose-headers:
Access-control-max-age: 3600



4. Go to the Lambda function and verify that the **API Gateway trigger** has been added.
A **resource-based policy** will have been automatically attached by Lambda to allow API Gateway to invoke the function.



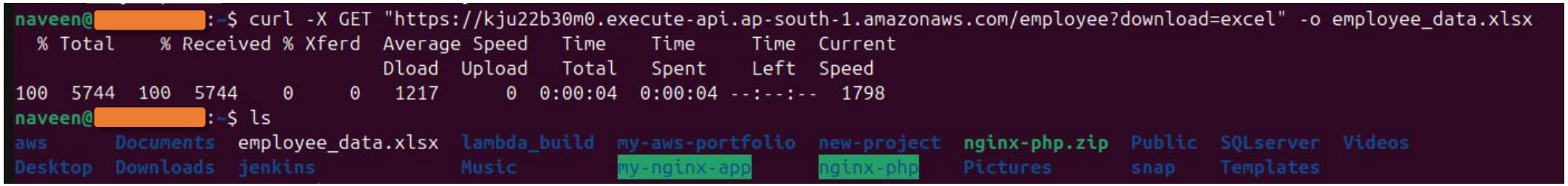
5. Go to **API Gateway → APIs → Deploy → Stages**, and copy the Invoke URL.



6(a). Go to the browser and paste the API invoke URL, you'll get the response.

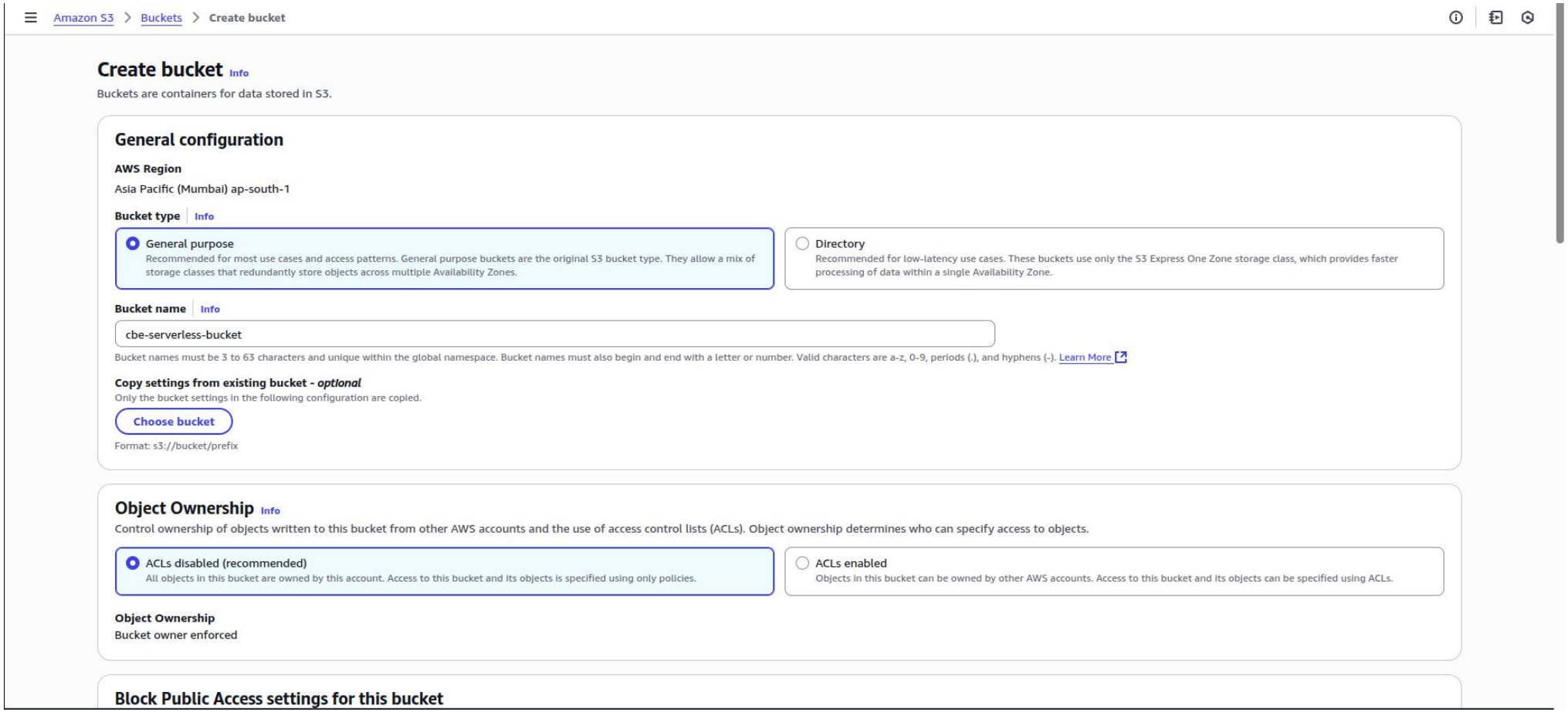
(b). Run the command **curl -X GET https://kju22b30m0.execute-api.ap-south-1.amazonaws.com/employee** to view the database results.

(c). Run the command **curl -X GET "https://kju22b30m0.execute-api.ap-south-1.amazonaws.com/employee?download=excel" -o employee_data.xlsx** to download the database results as a excel format.

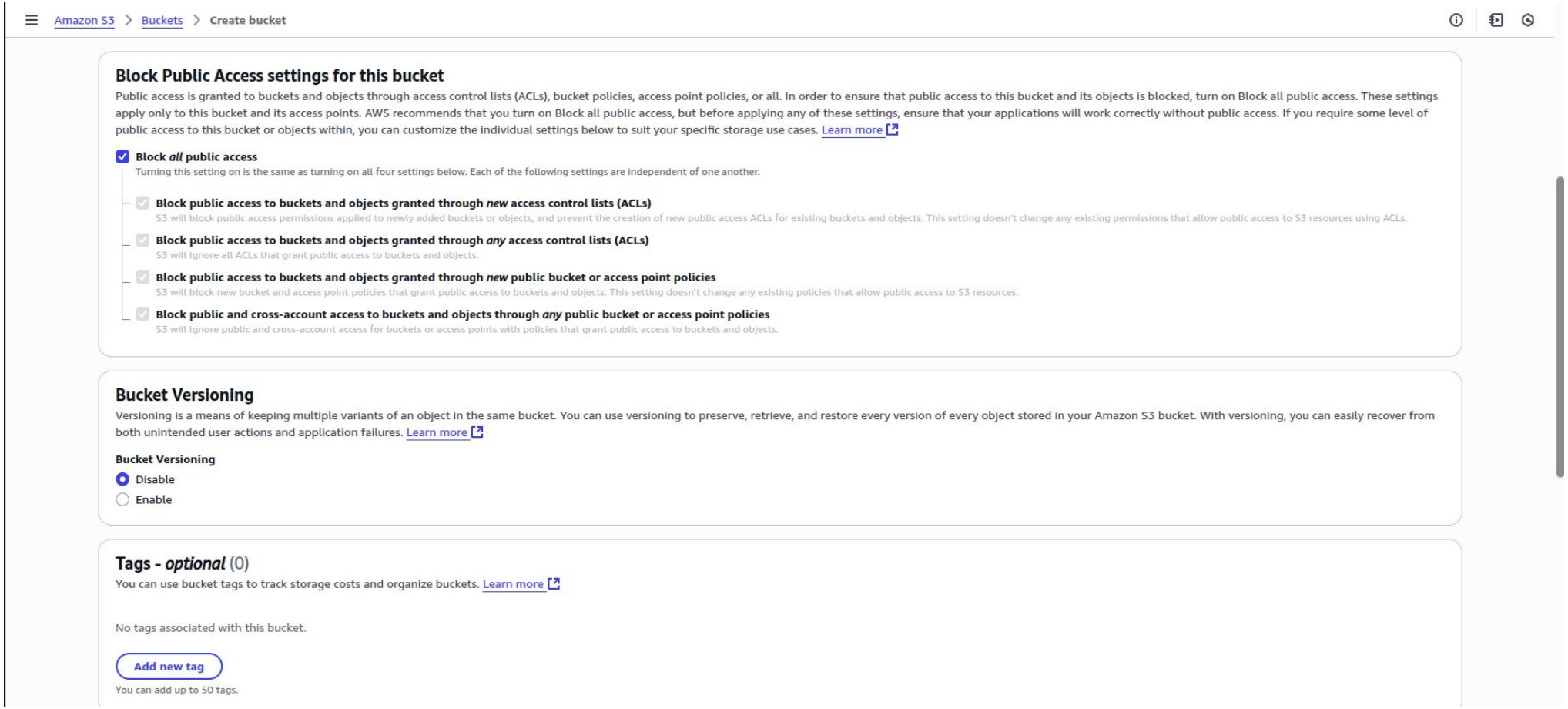


10. S3 static website, CloudFront (WAF) & ACM

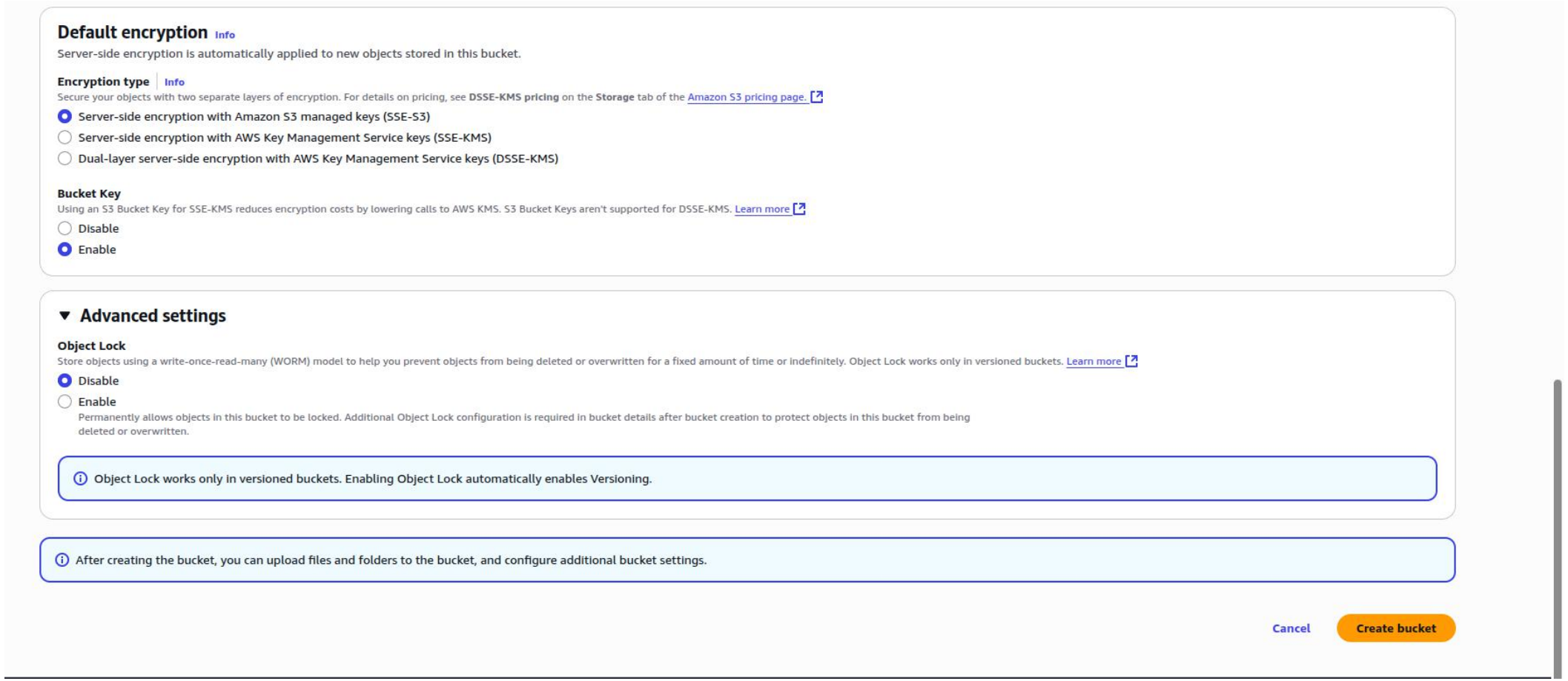
1. Create **S3 bucket** for storing static web files, enable versioning and server-side encryption. Put static files in **index.html, script.js** etc. Make sure bucket is created in your region.



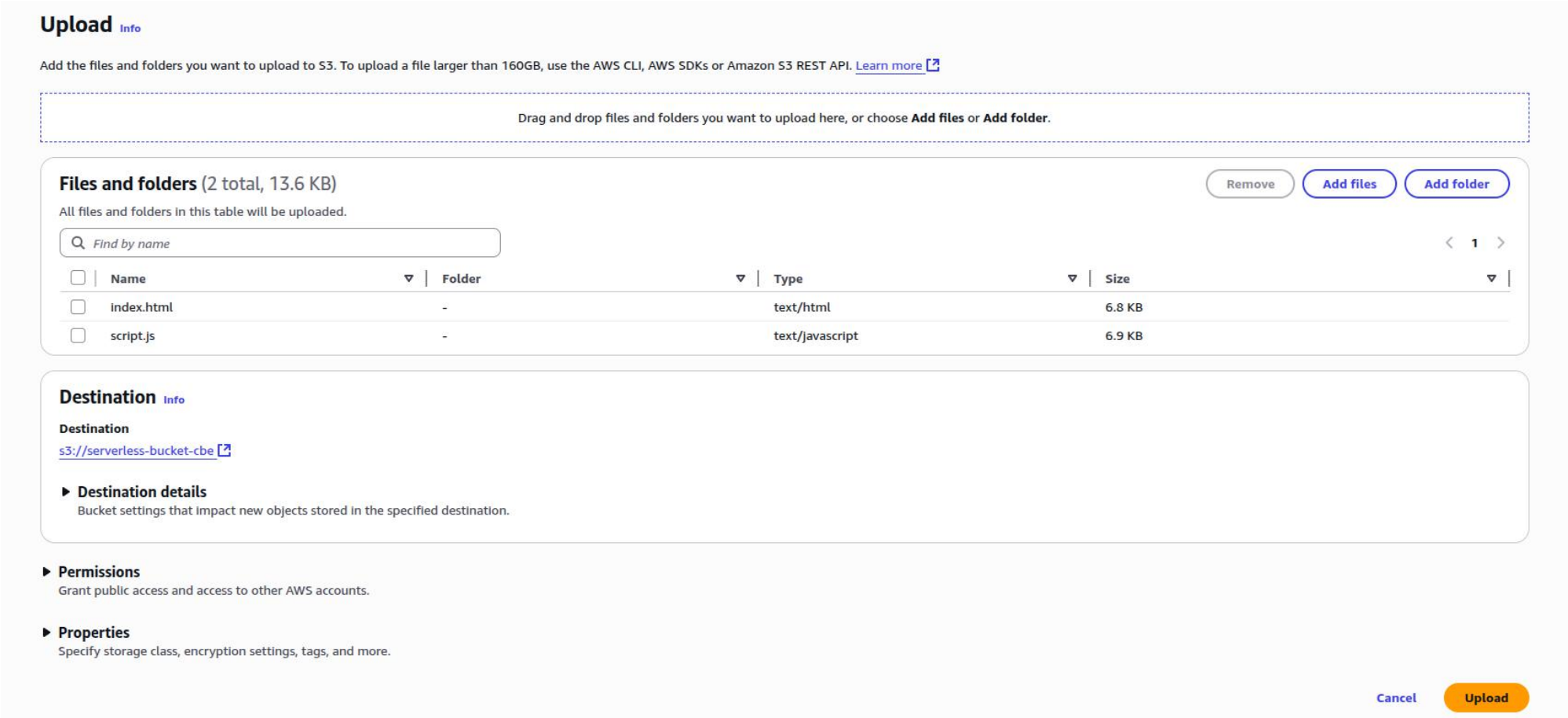
2. Block public access on the bucket; use an Origin Access Control (OAC) or Origin Access Identity (OAI) so only CloudFront can fetch objects. (Recommend OAC for modern flows.)



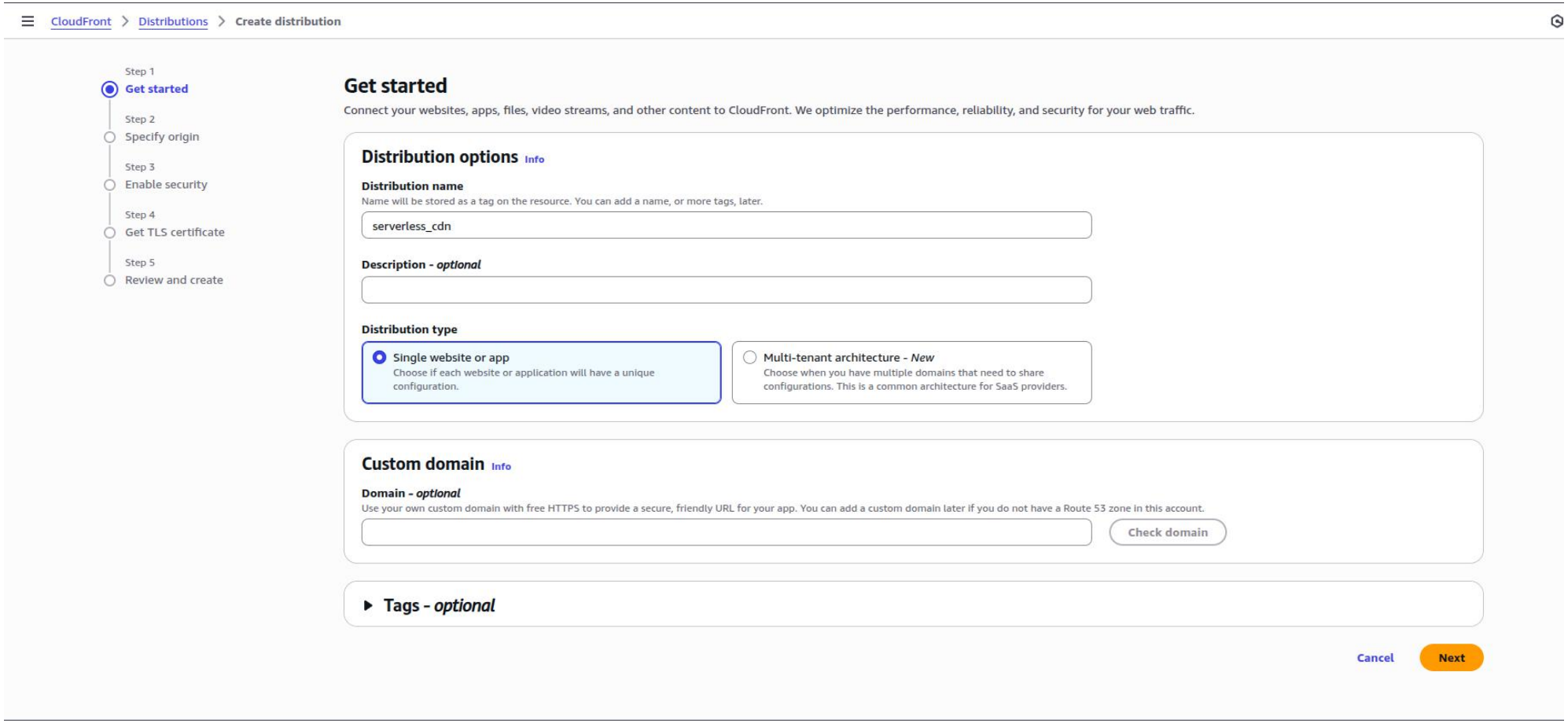
3. As mentioned before, encryption and bucket key is enabled by default for security purpose.



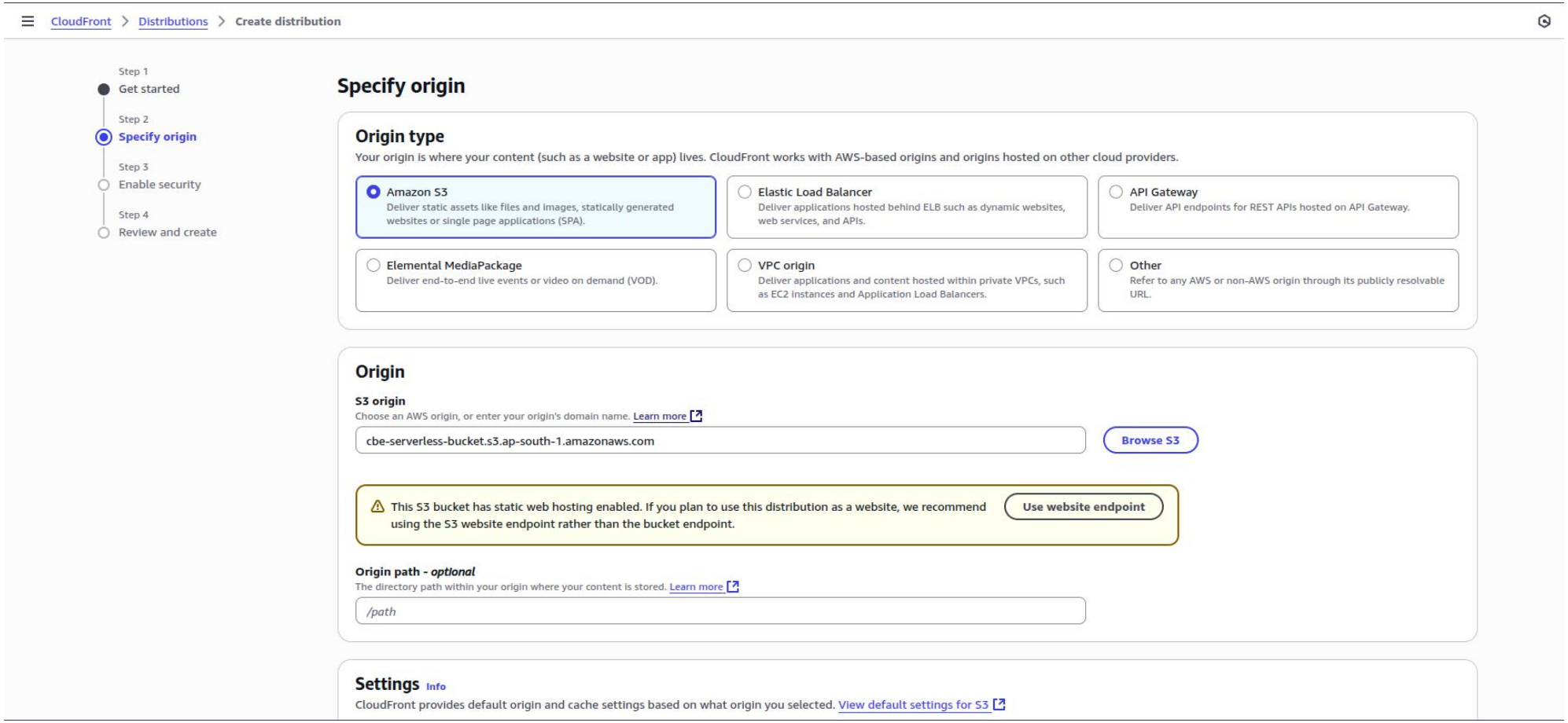
4. Now upload the frontend web files **index.html** and **script.js file** and click **upload**



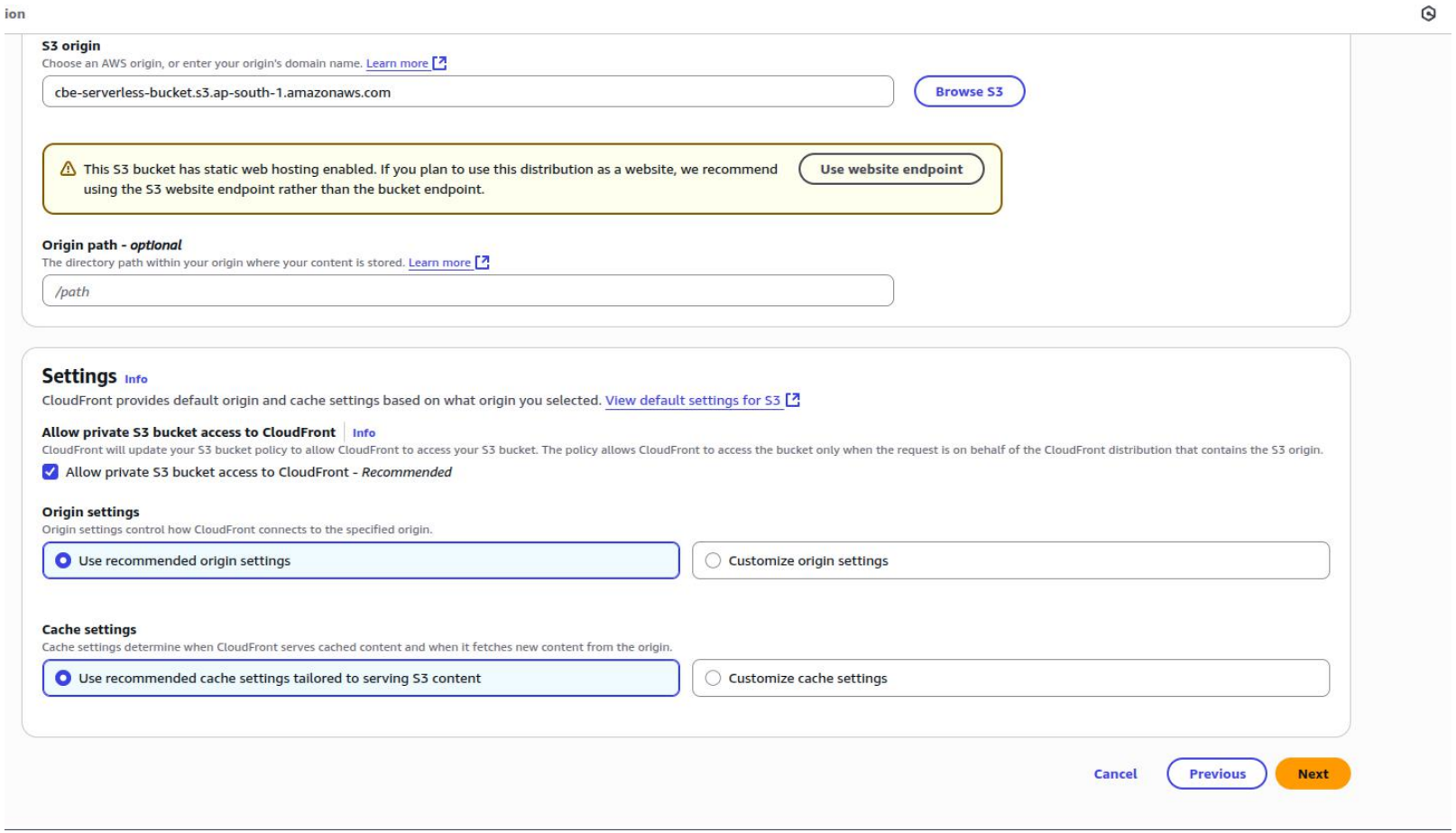
5. Now create a distribution in **cloudfront**, choose single website in the type and click next.



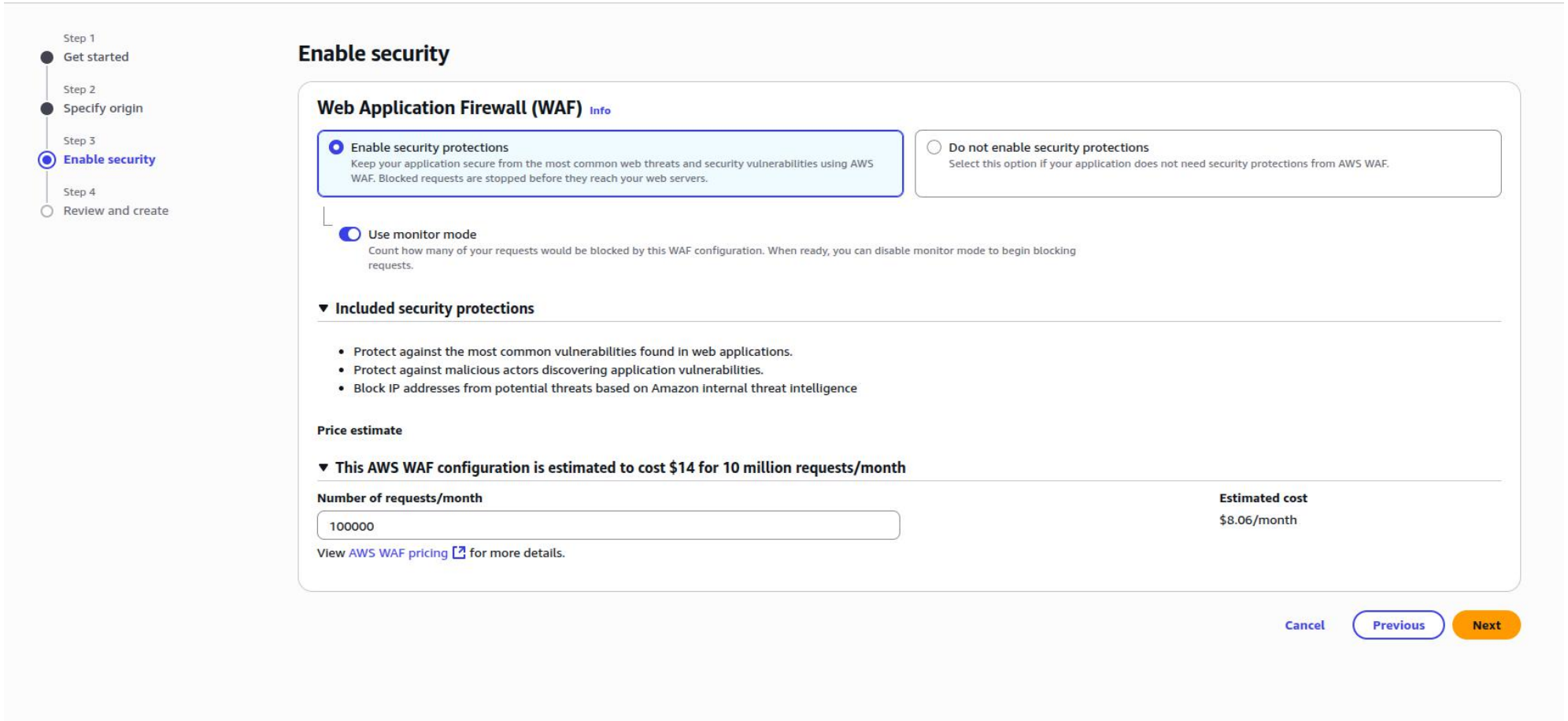
6. Choose origin as **S3 bucket (bucket endpoint)** and click next



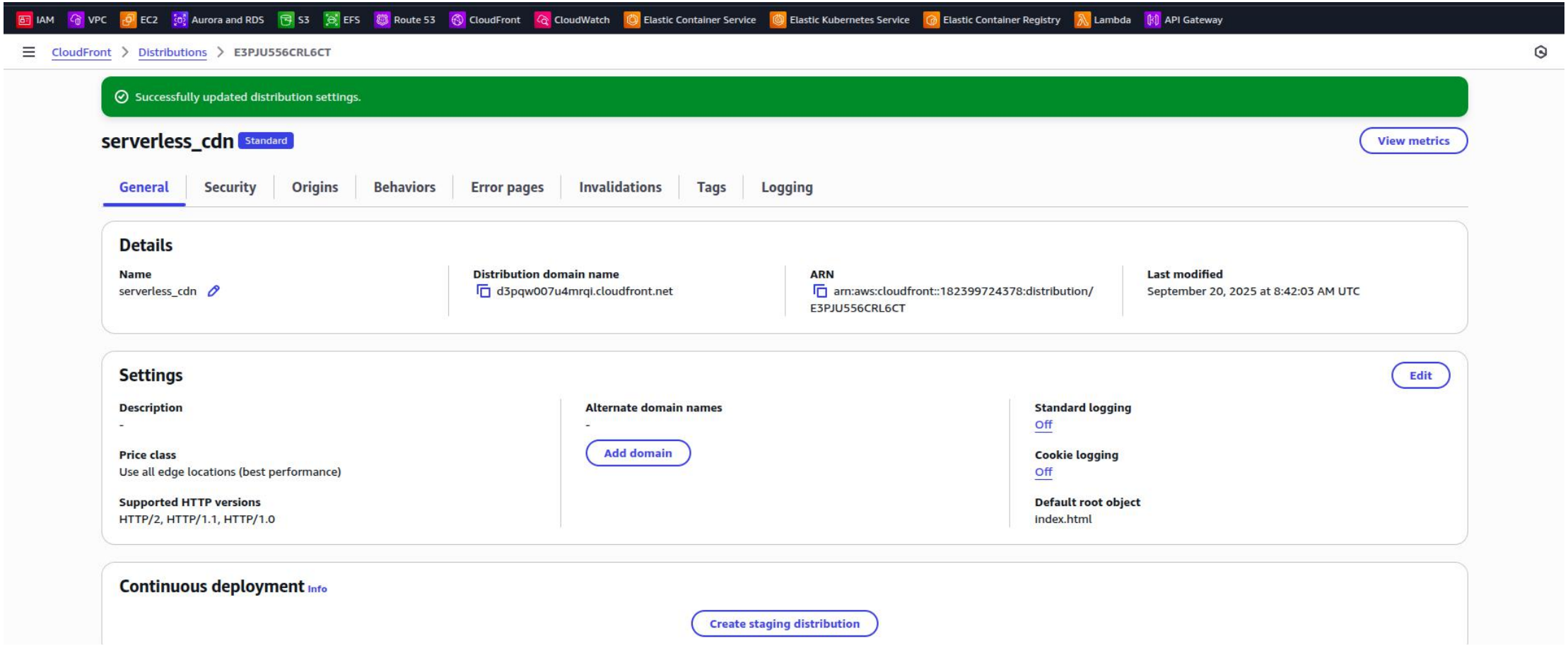
7. Enable allow **private S3 bucket to access cloudfront**, choose the recommended settings for origin and cache settings.



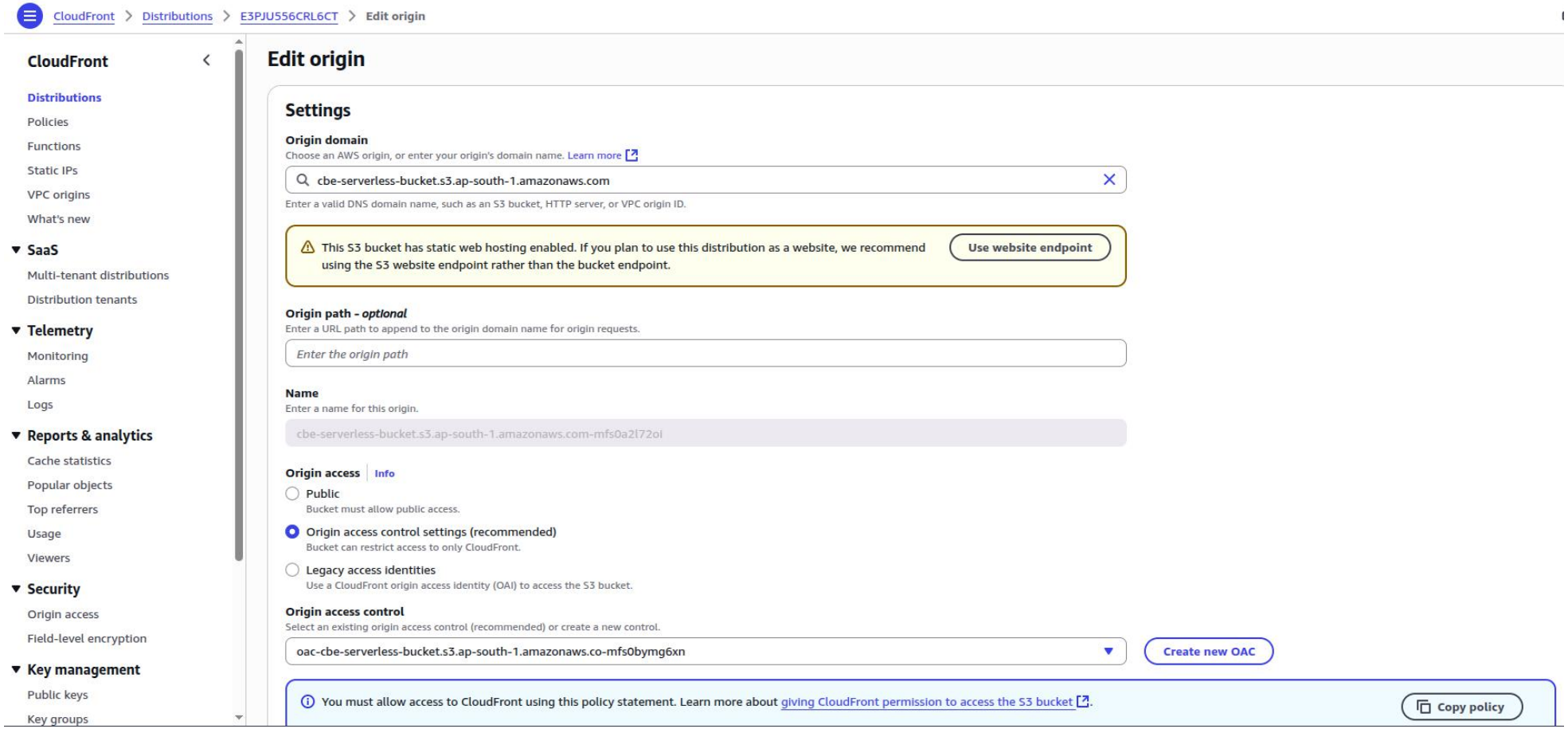
8. Enable **WAF** for application security (recommended) and mention the number of requests per month for your website and click next.



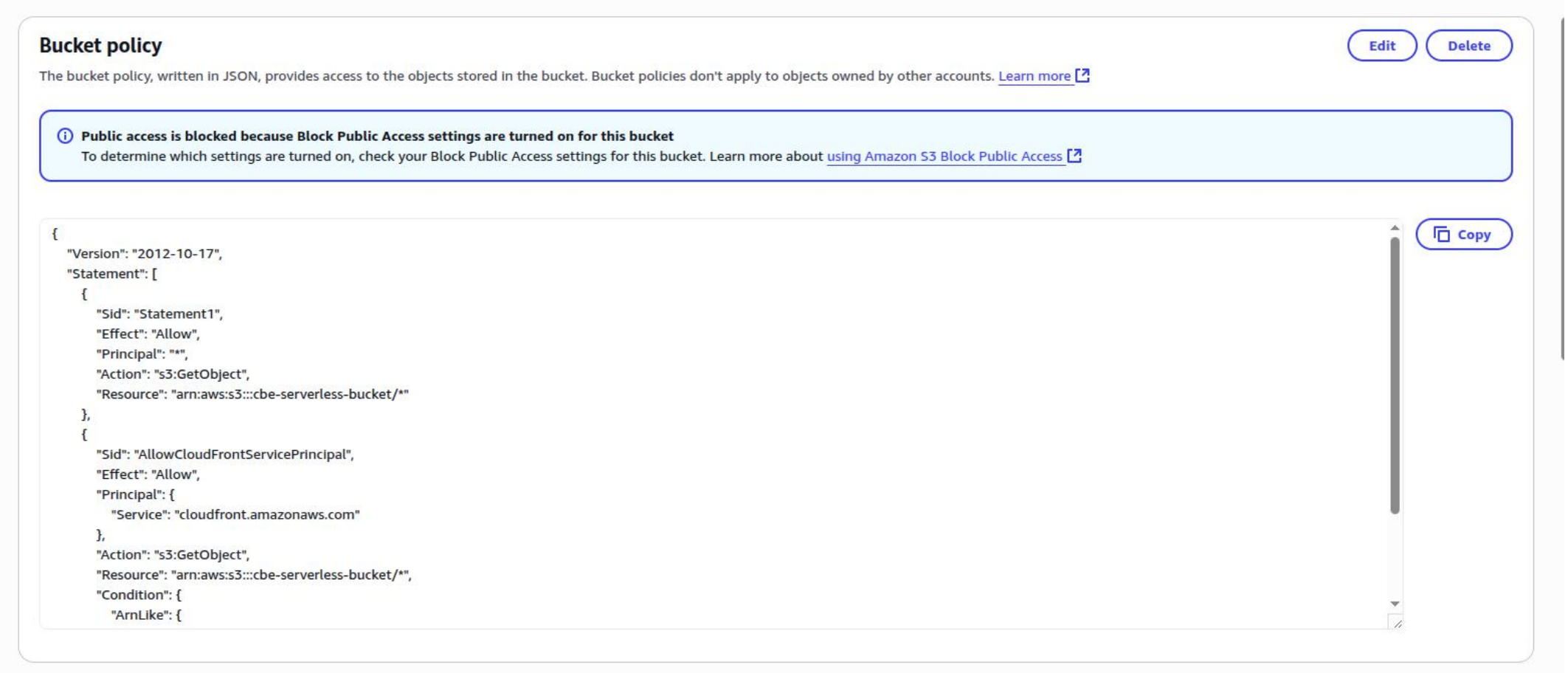
9. Deploying the cloudfront distribution will take some time, now click edit and set the **default root object** as **index.html** and go to **Origins**



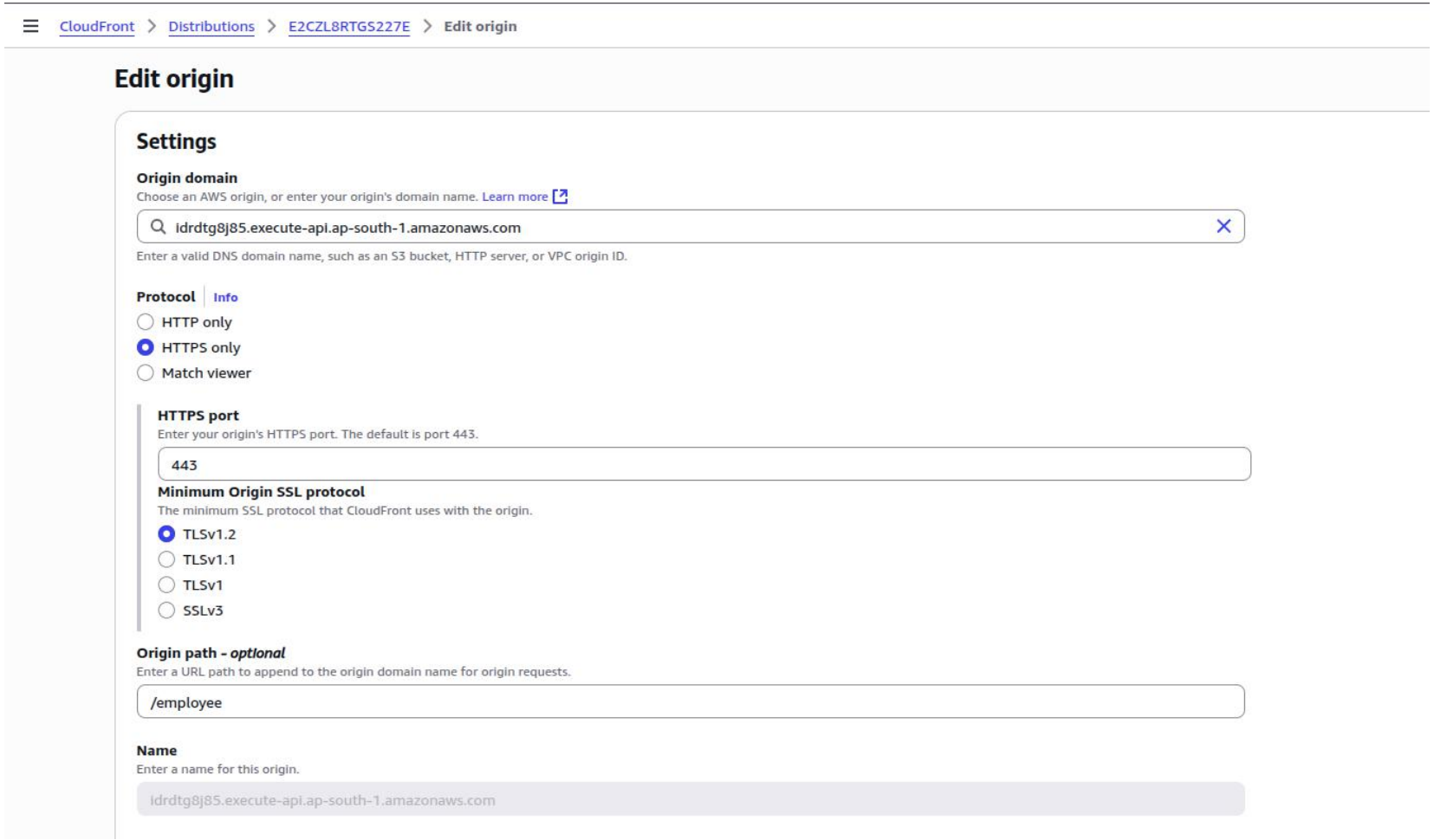
10. Go to **Origin access control** and you can find the policy created by CDN itself to access s3 bucket privately (only CloudFront can fetch objects using OAC)



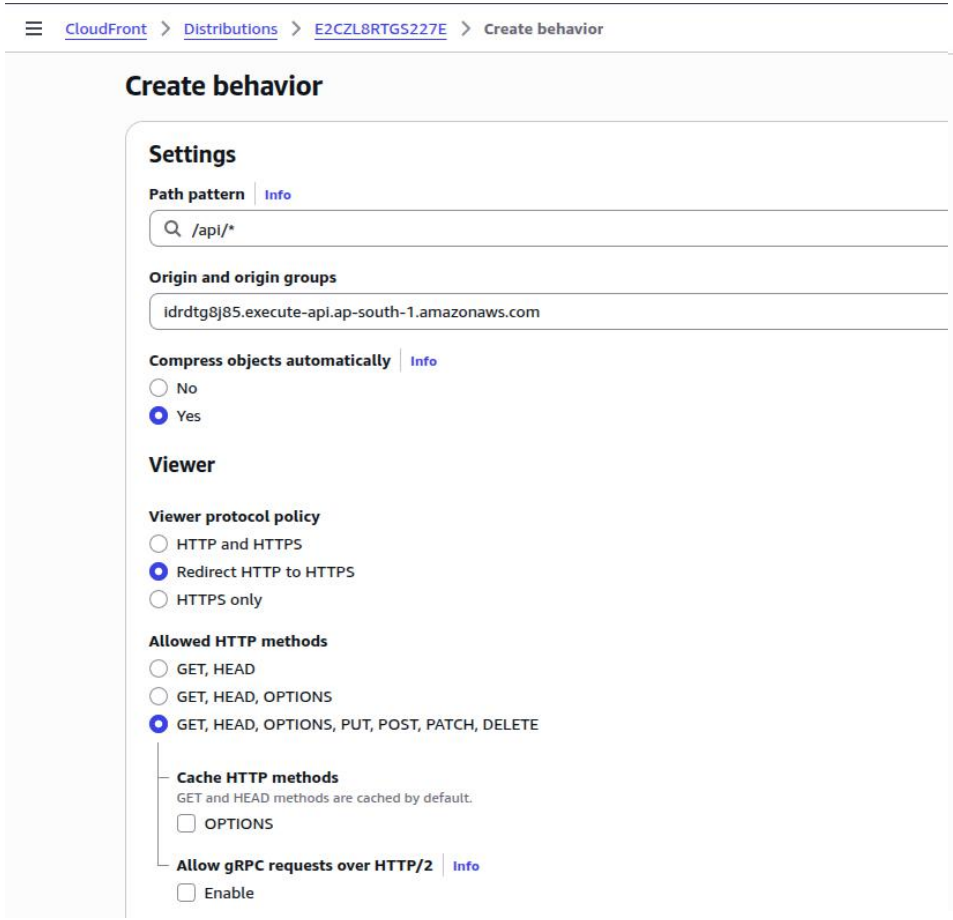
11. Meanwhile you can check it in S3 bucket, a bucket policy was added by cloudfront automatically using OAC.

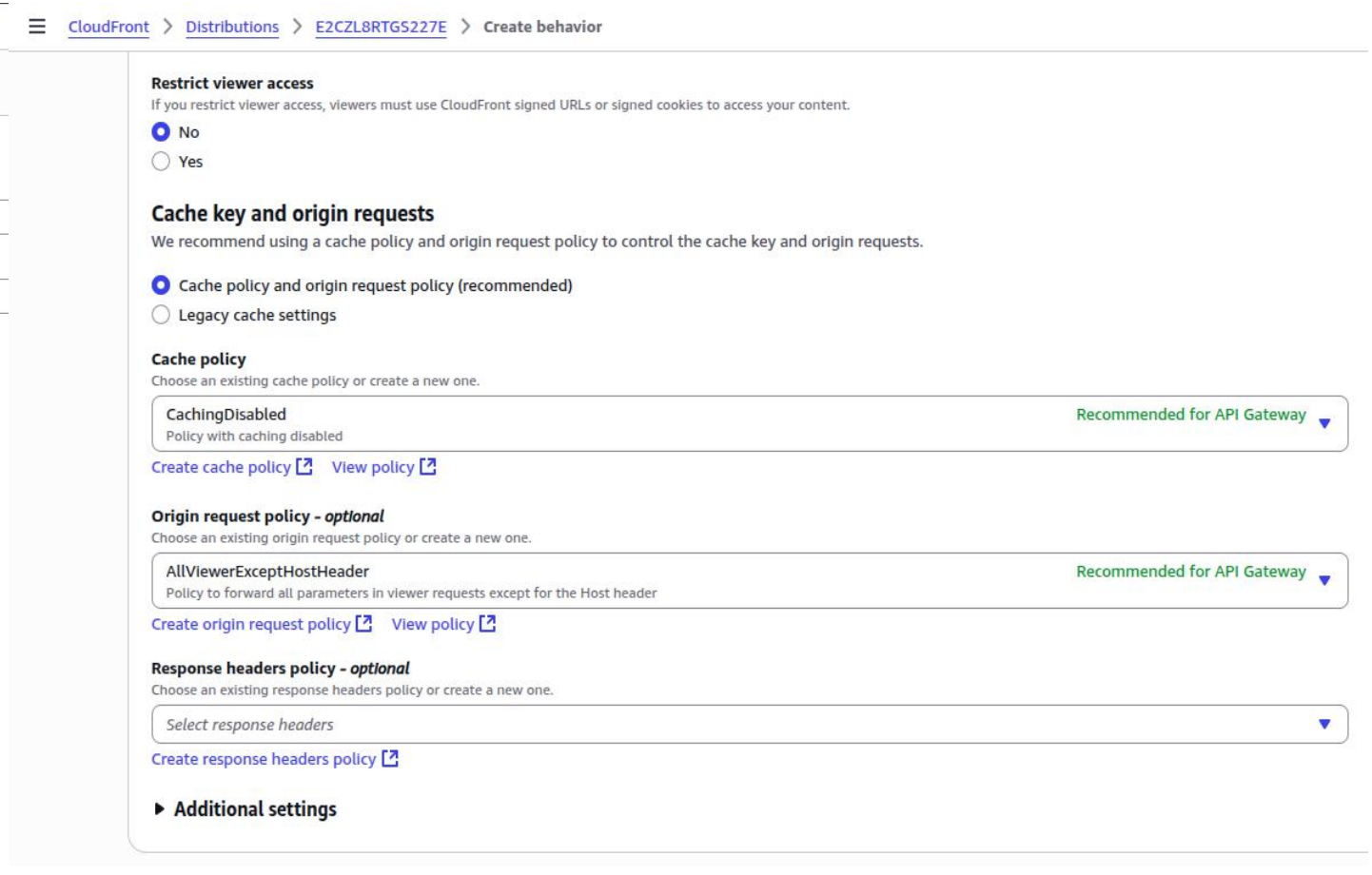


12. Create a new origin for CDN to access api gateway for the backend, select the origin domain as **API gateway endpoint** and origin path as **/employee**



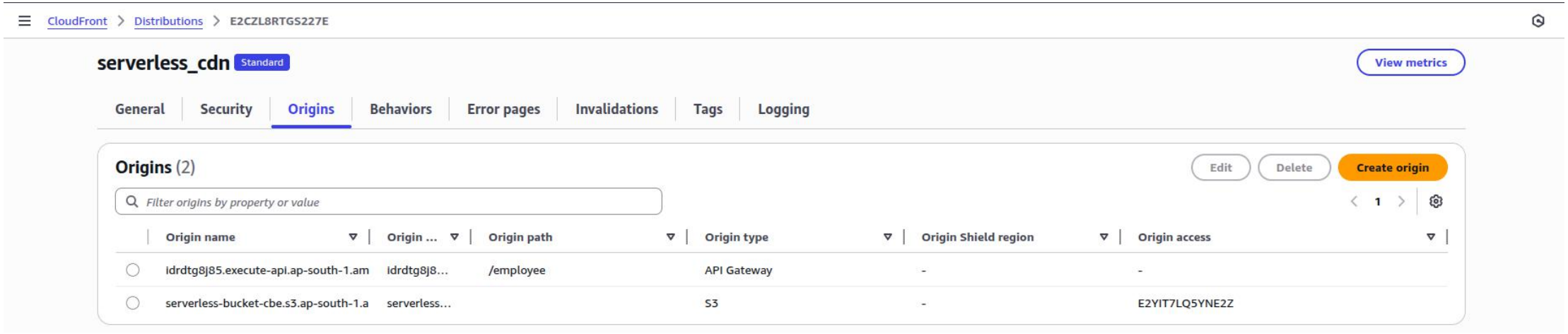
13. Create a new behavior for **API gateway**, set path pattern **/api/*** and select the origin we created before and left the default values.

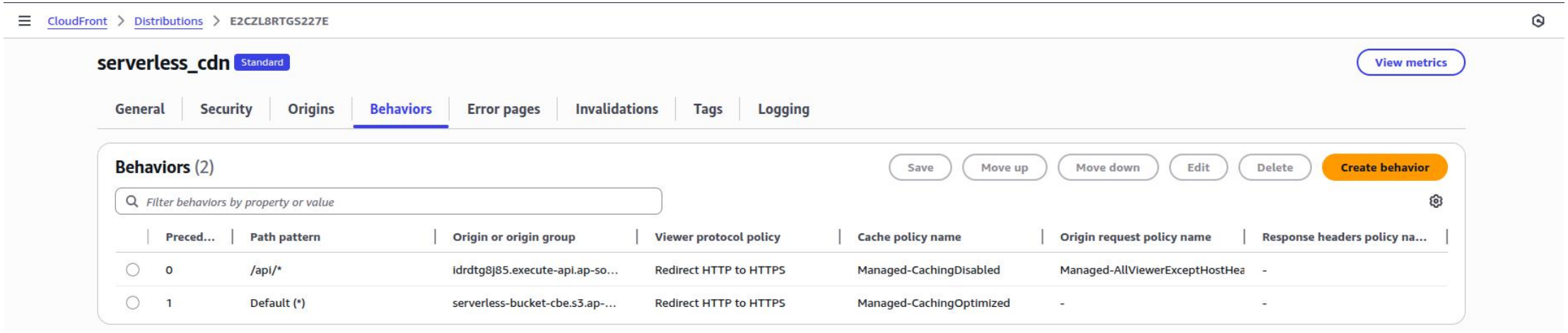




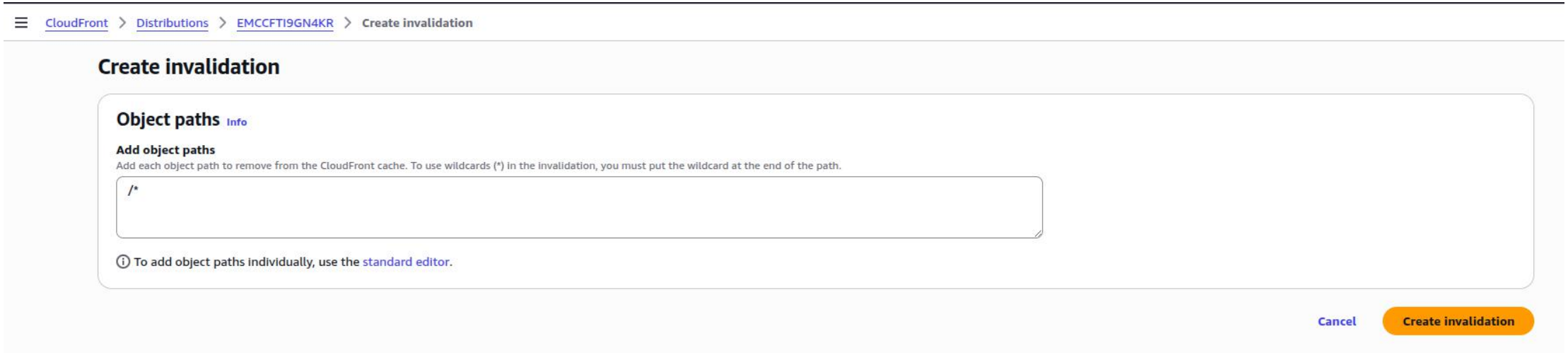
14. Now the cloudfront has two origins and two behaviors.

Origin 1: S3 bucket for static frontend file (use OAC)
Origin 2: API gateway (that will trigger lambda for backend)

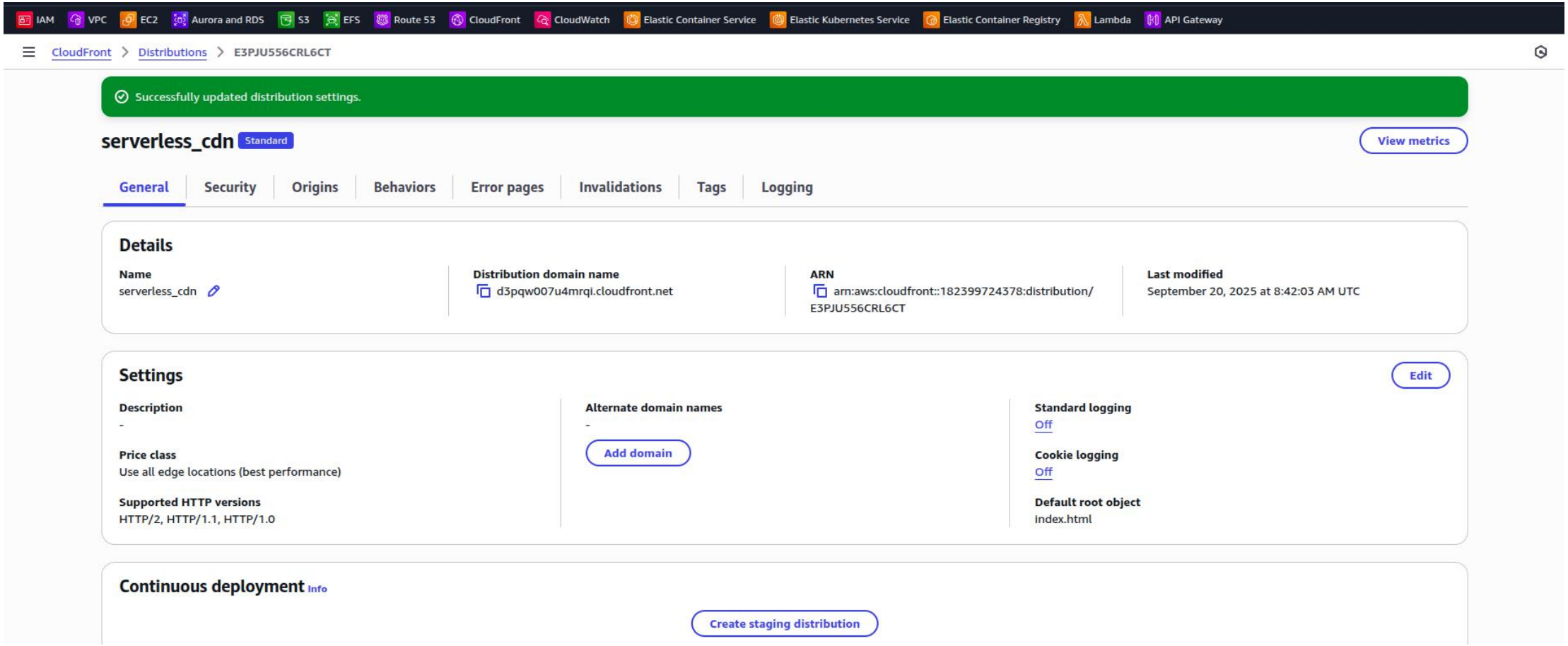




15. By default the cache TTL is set to one day, you can set the cache invalidation here by mentioning the /* path of the files.

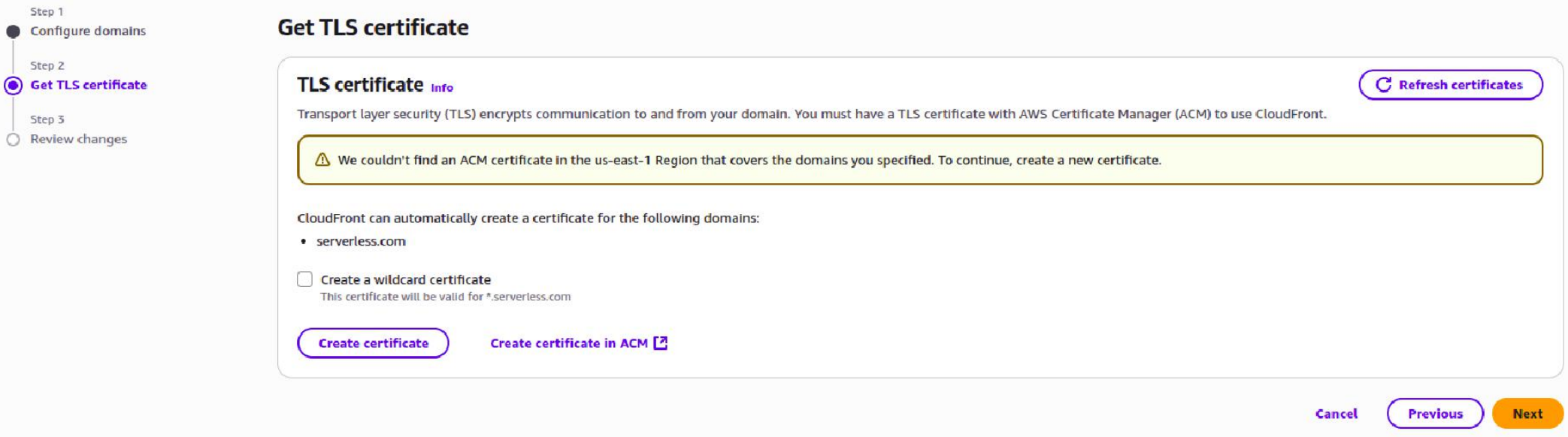


16. Under alternate domain names, click on **Add domain** to add a custom domain name in the cloudfront.

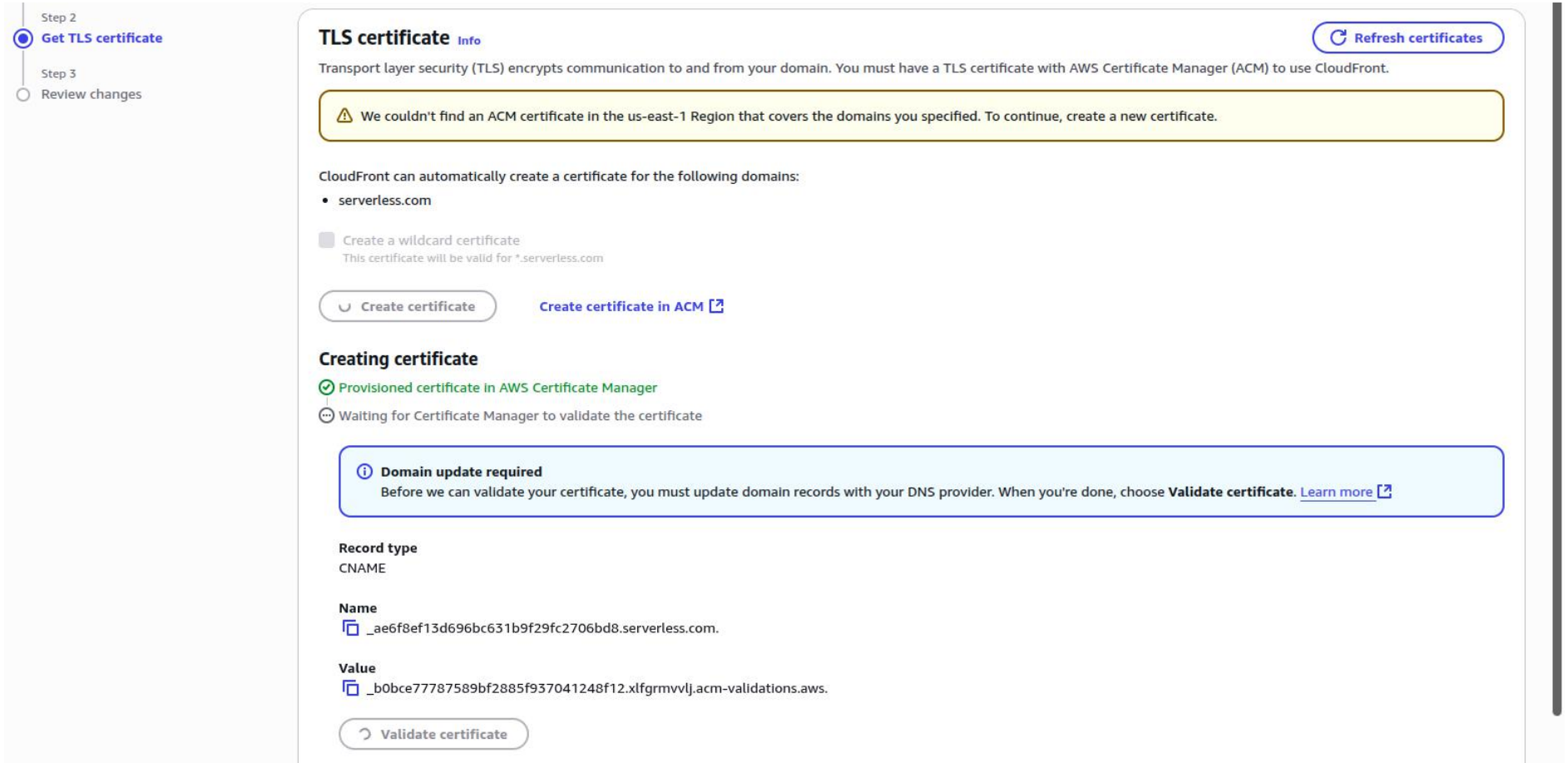


17. Mention your **domain name** and click next to attach **ACM certificate** to CloudFront for your domain.

Note: ACM cert must be in us-east-1 for CloudFront

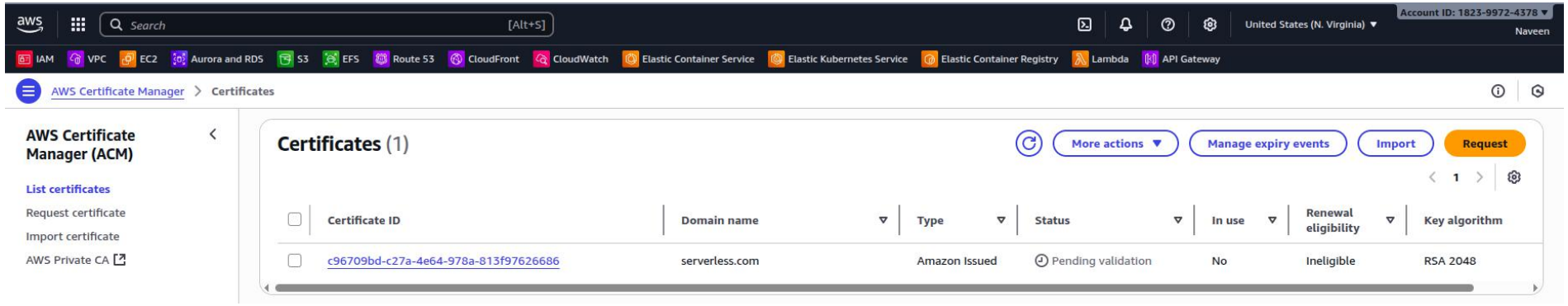


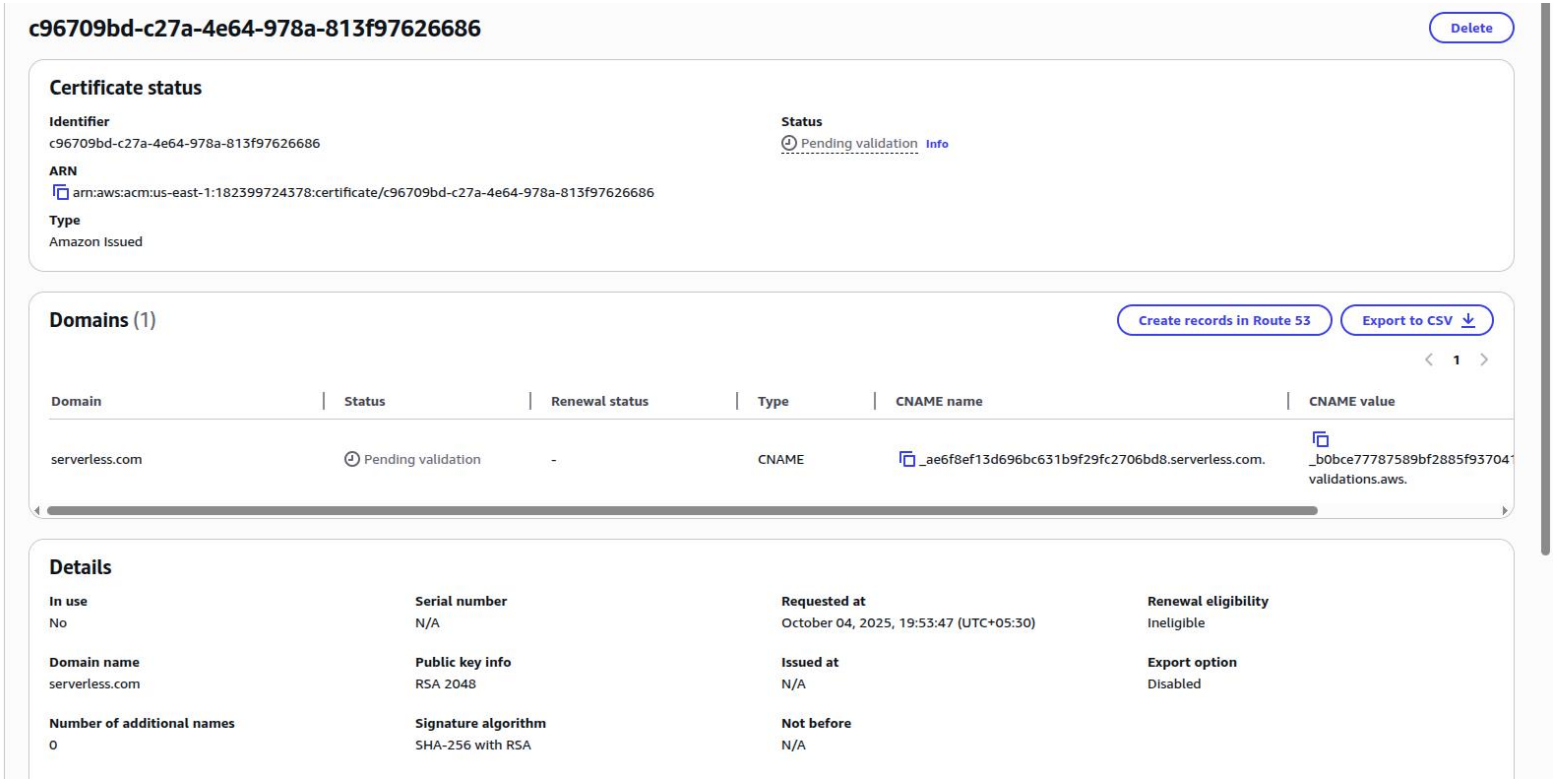
18. Either you can **create a SSL/TLS certificate** in CDN and get validated in ACM or you can **import** your own cert into ACM and attach.
Note: AWS can also provision a public certificate for free.



19. Now the certificate created in cloudfront is reflected to ACM (us-east-1) and its pending for validation.

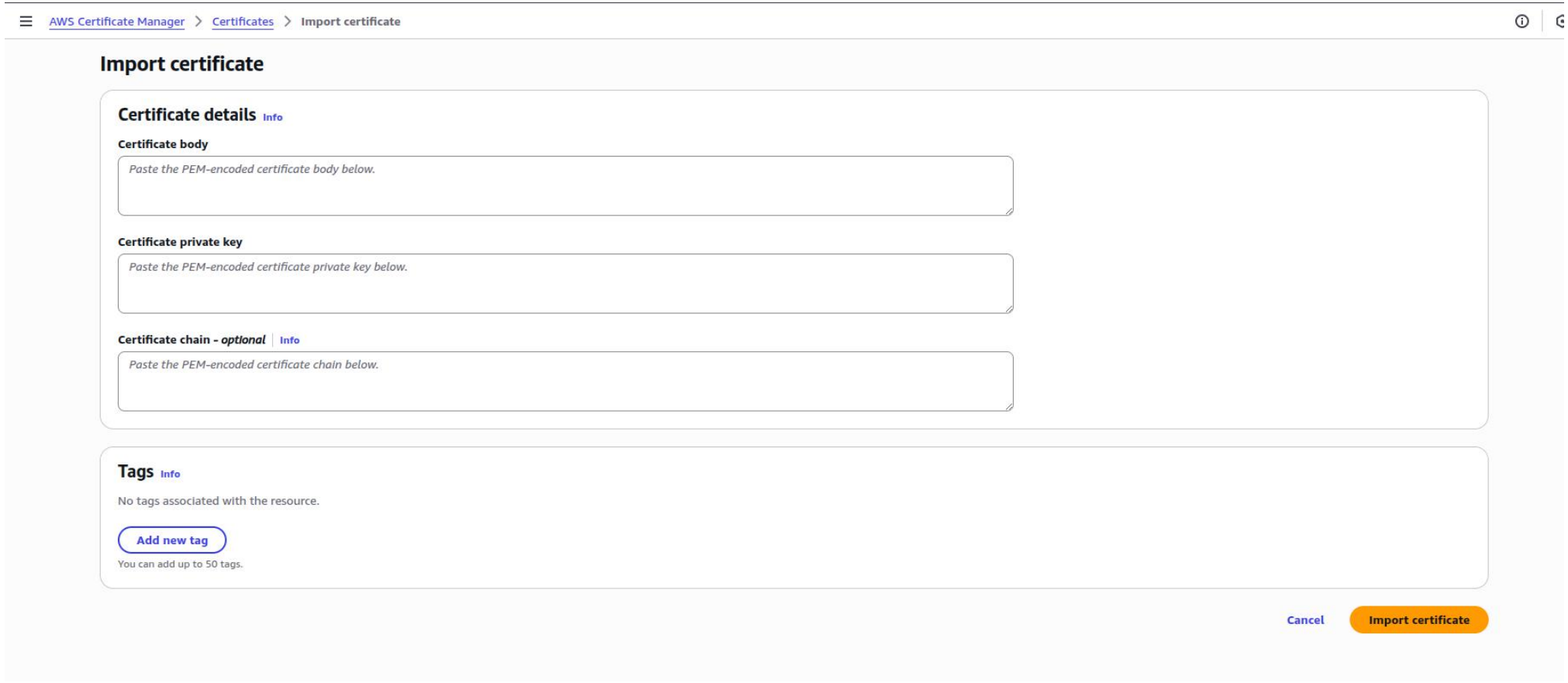
Go to **AWS certificate manager --> List certificates**





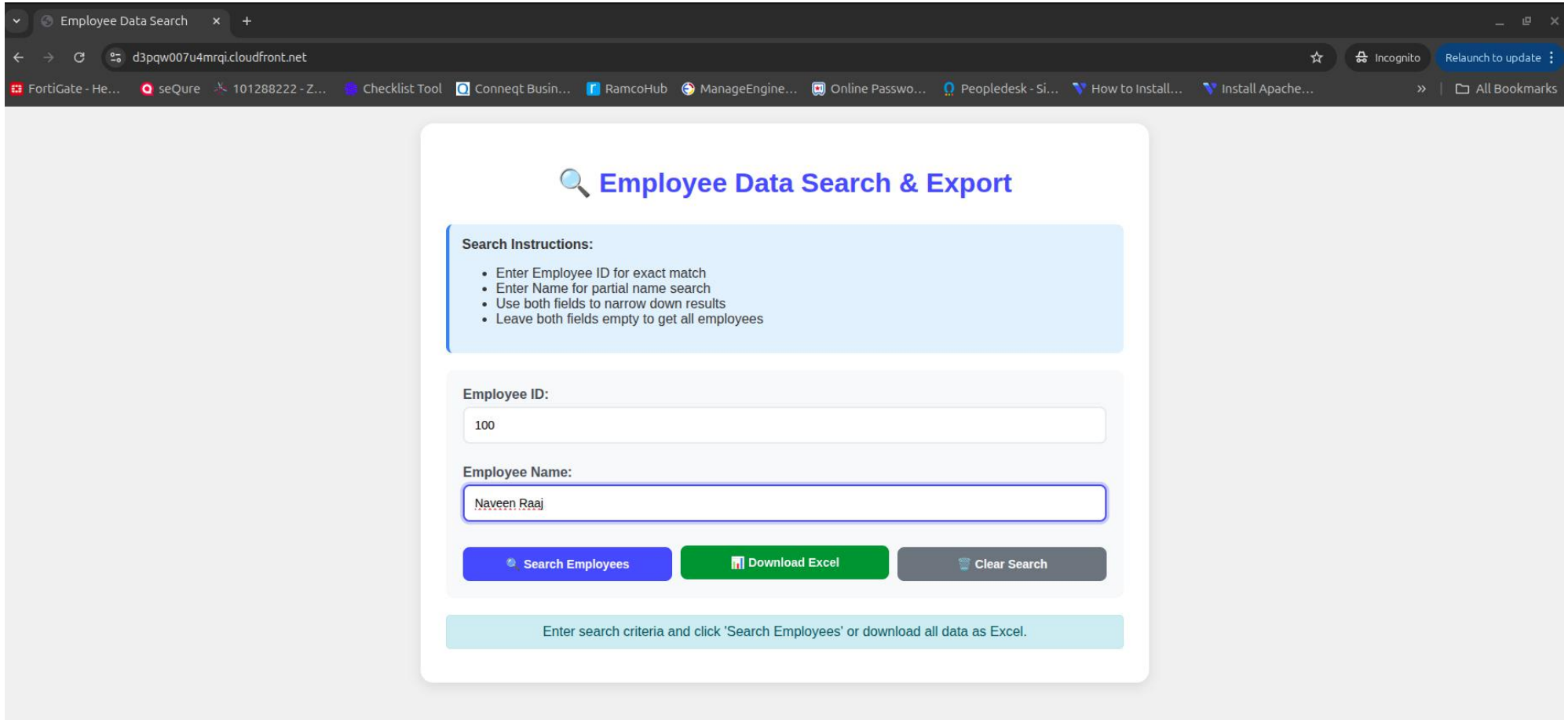
20. If you already have a SSL/TLS certificate for your domain then you can import the cert private key and cert chain over here.

Go to **AWS certificate manager --> Import certificate,** copy and paste your keys and click **import certificate**

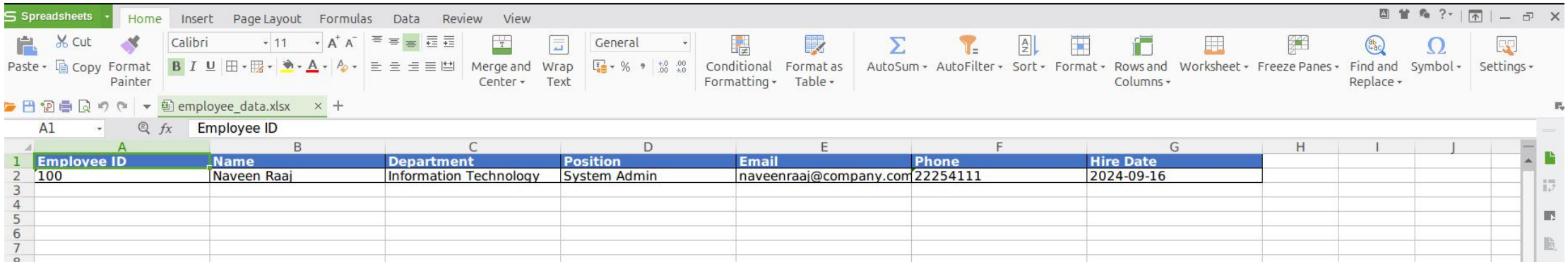


Note: In Godaddy or any other DNS provider, create alias record (CNAME) pointing your domain to CloudFront distribution. Ensure CloudFront uses the ACM cert for that domain.

21. Hit the domain name in the browser URL to get the website. Give the inputs and click on download.



22. Now the excel file was downloaded successfully and you can view the data's stored in the file.



Employee ID	Name	Department	Position	Email	Phone	Hire Date
100	Naveen Raaj	Information Technology	System Admin	naveenraaj@company.com	22254111	2024-09-16

Thank You