

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

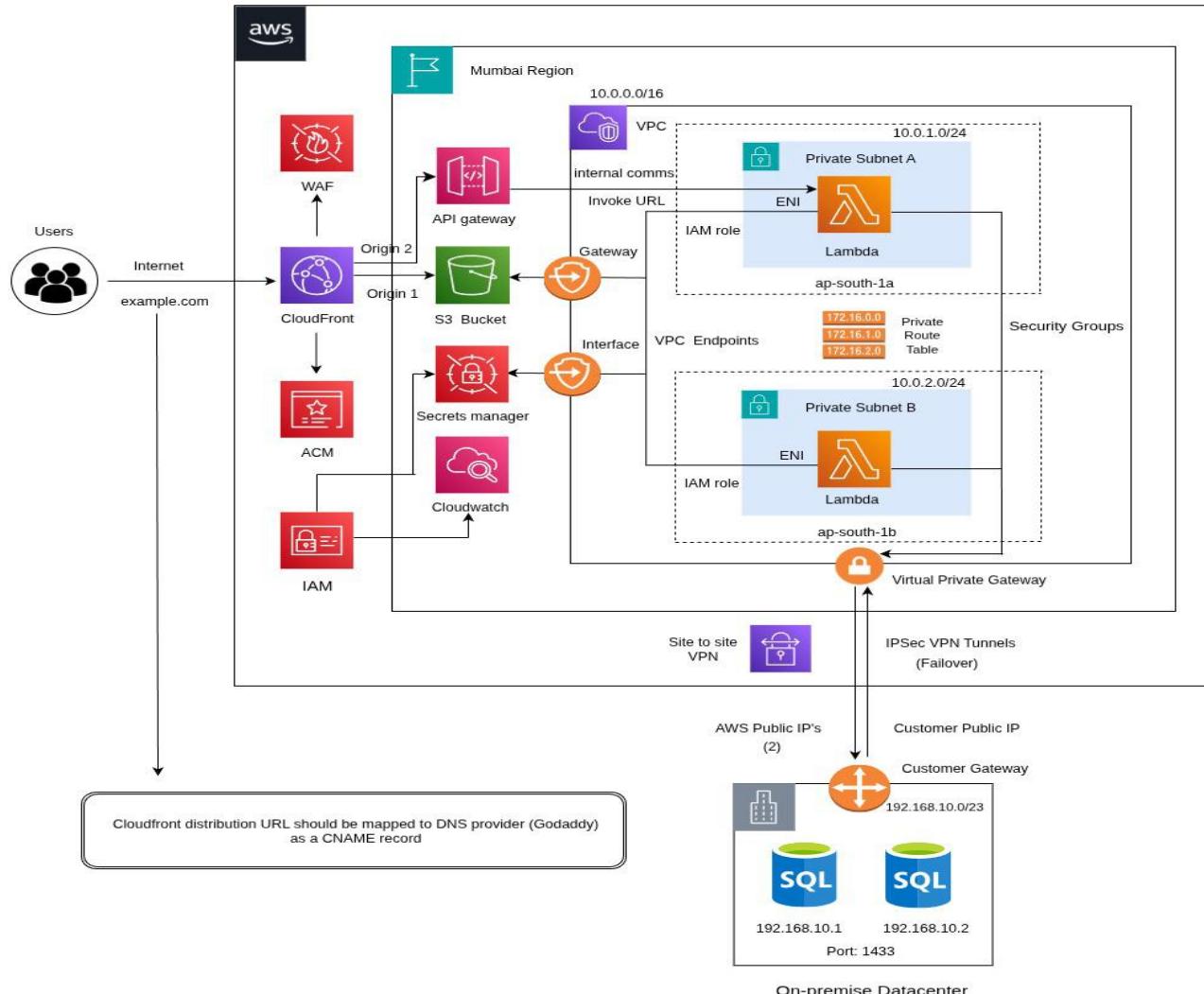
Project Overview & Implementation Steps

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

Naveen Raaj



Key AWS Services Used

Naveen Raaj

- *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

Naveen Raaj

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
```

```
root@naveen: /home/naveen# sqlcmd -S localhost -U SA -p -C  
Password:  
1> SELECT name, database_id, create_date  
2> FROM sys.databases;  
3> GO  
name  
-----  
master  
tempdb  
model  
msdb  
Employee Database  
-----  
(5 rows affected)  
  
Network packet size (bytes): 4096  
1 xact[s]:  
Clock Time (ms.): total      3 avg   3.0 (333.3 xacts per sec.)  
1> USE [Employee Database];  
2> GO  
Changed database context to 'Employee Database'.  
  
Network packet size (bytes): 4096  
1 xact[s]:  
Clock Time (ms.): total      2 avg   2.0 (500.0 xacts per sec.)  
1> SELECT TABLE_SCHEMA, TABLE_NAME  
2> FROM INFORMATION_SCHEMA.TABLES  
3> WHERE TABLE_TYPE = 'BASE TABLE';  
4> GO  
TABLE_SCHEMA  
-----  
dbo  
-----  
employeeData  
  
(1 rows affected)  
1> SELECT *  
2> FROM [dbo].[employeeData];  
3> GO  
empid          name           position        phone       hire_date    department      email  
-----  
430363         Naveen Raaj  System Admin  22254111  2024-09-16  Information Technology  naveenraaj@com  
pany.com  
  
(1 rows affected)  
  
Network packet size (bytes): 4096  
1 xact[s]:  
Clock Time (ms.): total      4 avg   4.0 (250.0 xacts per sec.)  
1> █
```

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

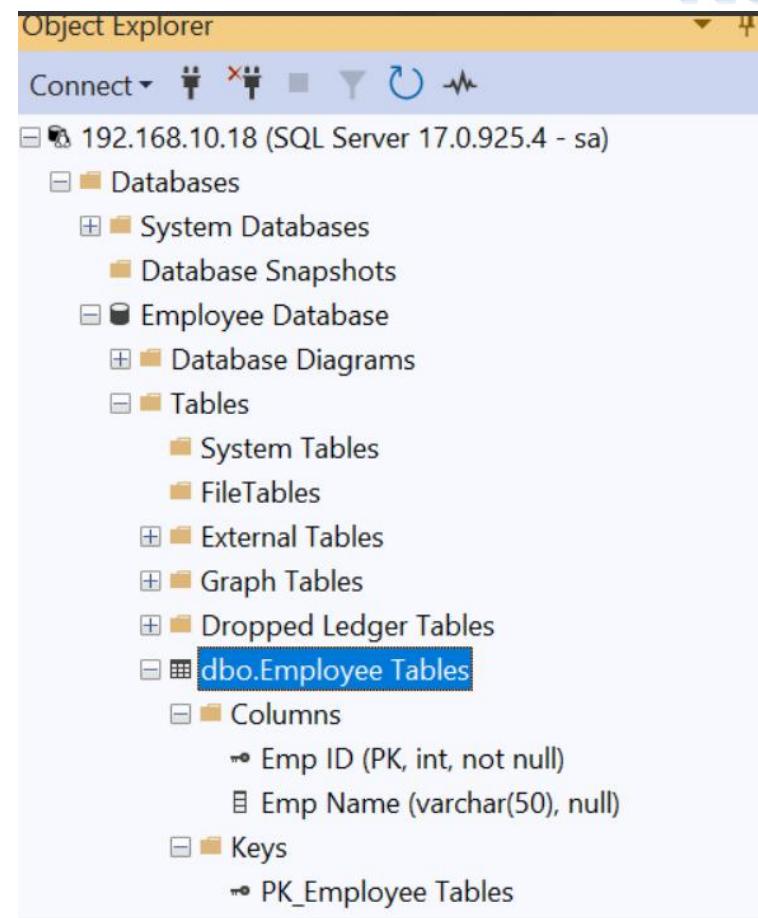
6. View All Data (Columns + Rows)

```
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**)

A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Amazon EC2 instances.

VPC settings

Resources to create [Info](#)
Create only the VPC resource or the VPC and other networking resources.

VPC only VPC and more

Name tag - optional
Creates a tag with a key of 'Name' and a value that you specify.

IPv4 CIDR block [Info](#)
 IPv4 CIDR manual input IPAM-allocated IPv4 CIDR block

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

IPv6 CIDR block [Info](#)
 No IPv6 CIDR block IPAM-allocated IPv6 CIDR block Amazon-provided IPv6 CIDR block IPv6 CIDR owned by me

Tenancy [Info](#)

Tags

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

| Key | Value - optional |
|--|---|
| <input type="text" value="Name"/> | <input type="text" value="serverless_vpc"/> Remove tag |

(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)

The screenshot shows the 'Edit VPC settings' page for a VPC with ID `vpc-06d82d27152d9753e` and name `serverless_vpc`. The page is divided into four main sections: **VPC details**, **DHCP settings**, **DNS settings**, and **Network Address Usage metrics settings**.

- VPC details:** Displays the VPC ID (`vpc-06d82d27152d9753e`) and Name (`serverless_vpc`).
- DHCP settings:** Shows the DHCP option set `dopt-0c044edcb1e622a27`.
- DNS settings:** Contains two checked checkboxes:
 - Enable DNS resolution Info
 - Enable DNS hostnames Info
- Network Address Usage metrics settings:** Contains one unchecked checkbox:
 - Enable Network Address Usage metrics Info

At the bottom right, there are 'Cancel' and 'Save' buttons.

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.

The screenshot shows the 'Create subnet' wizard in the AWS VPC console. The top navigation bar includes 'VPC' > 'Subnets' > 'Create subnet'. The main section is titled 'Create subnet' with an 'Info' link. It has two tabs: 'VPC' (selected) and 'Subnet settings'.
VPC Tab:

- VPC ID:** A dropdown menu is set to 'vpc-06d82d27152d9753e (serverless_vpc)'.
- Associated VPC CIDRs:** Shows 'IPv4 CIDRs' as '10.0.0.0/16'.

Subnet settings Tab:

- Subnet 1 of 1:**
 - Subnet name:** An input field contains 'serverless_subnet_a'. A note says 'The name can be up to 256 characters long.'
 - Availability Zone:** A dropdown menu is set to 'Asia Pacific (Mumbai) / aps1-az1 (ap-south-1a)'.
- IPv4 VPC CIDR block:** A dropdown menu is set to '10.0.0.0/16'.

(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**)

The image shows two separate screenshots of the AWS VPC 'Create subnet' interface. Both screenshots are nearly identical, showing the configuration for creating two subnets: subnet_a and subnet_b.

Subnet 1 of 1

Subnet name: serverless_subnet_a

Availability Zone: Asia Pacific (Mumbai) / aps1-az1 (ap-south-1a)

IPv4 VPC CIDR block: 10.0.0.0/16

IPv4 subnet CIDR block: 10.0.1.0/24

Tags - optional:

- Key: Name, Value: serverless_subnet_a
- Add new tag
- Remove

Subnet 1 of 1

Subnet name: serverless_subnet_b

Availability Zone: Asia Pacific (Mumbai) / aps1-az3 (ap-south-1b)

IPv4 VPC CIDR block: 10.0.0.0/16

IPv4 subnet CIDR block: 10.0.2.0/24

Tags - optional:

- Key: Name, Value: serverless_subnet_b
- Add new tag
- Remove

Create subnet

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

The screenshot shows the AWS VPC Route Tables page. On the left, there's a navigation sidebar with sections like VPC dashboard, Virtual private cloud, Security, and PrivateLink and Lattice. The main area displays a table of route tables, with one row selected: "serverless_route_table" (rtb-00e7ce38d80888a21). Below this, a detailed view of the selected route table is shown, specifically the "Subnet associations" tab. It lists two explicit subnet associations: "serverless_subnet_b" (subnet-055b9a2e51f7d2178) with IPv4 CIDR 10.0.2.0/24 and "serverless_subnet_a" (subnet-0d72d2c409922ff09) with IPv4 CIDR 10.0.1.0/24. There are also sections for "Subnets without explicit associations" (0) and a note stating "No subnets without explicit associations" and "All your subnets are associated with a route table."

| Name | Route table ID | Explicit subnet assoc... | Edge associations | Main | VPC | Owner ID |
|--|-----------------------|--------------------------|-------------------|------|---------------------------------|--------------|
| - | rtb-0c2833d9b52bef9d3 | - | - | Yes | vpc-099e47a9d31b97ddc | 182399724378 |
| <input checked="" type="checkbox"/> serverless_route_table | rtb-00e7ce38d80888a21 | 2 subnets | - | Yes | vpc-06d82d27152d9753e serv... | 182399724378 |

rtb-00e7ce38d80888a21 / serverless_route_table

- Details
- Routes
- Subnet associations**
- Edge associations
- Route propagation
- Tags

Explicit subnet associations (2)

| Name | Subnet ID | IPv4 CIDR | IPv6 CIDR |
|---------------------|--------------------------|-------------|-----------|
| serverless_subnet_b | subnet-055b9a2e51f7d2178 | 10.0.2.0/24 | - |
| serverless_subnet_a | subnet-0d72d2c409922ff09 | 10.0.1.0/24 | - |

Subnets without explicit associations (0)

The following subnets have not been explicitly associated with any route tables and are therefore associated with the main route table:

| Name | Subnet ID | IPv4 CIDR | IPv6 CIDR |
|------|-----------|-----------|-----------|
|------|-----------|-----------|-----------|

No subnets without explicit associations
All your subnets are associated with a route table.

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)

The screenshot shows the 'Create security group' wizard in the AWS VPC console. The steps completed are:

- Step 1: Basic details**
 - Security group name:** lambda_security_group
 - Description:** security group to connect lambda with vpc endpoints
 - VPC:** vpc-06d82d27152d9753e (serverless_vpc)
- Step 2: Inbound rules**

This security group has no inbound rules.

Add rule
- Step 3: Outbound rules**

| Type | Protocol | Port range | Destination | Description - optional |
|-------------|----------|------------|------------------|------------------------|
| All traffic | All | All | Custom 0.0.0.0/0 | |

Add rule

Warning: Rules with destination of 0.0.0.0/0 or ::/0 allow your instances to send traffic to any IPv4 or IPv6 address. We recommend setting security group rules to be more restrictive and to only allow traffic to specific known IP addresses.

(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

Security group name [Info](#)
vpc_interface_endpoint_SG
Name cannot be edited after creation.

Description [Info](#)
allows the inbound connections from lambda security group(through ENI's)

VPC Info
vpc-06d82d27152d9753e (serverless_vpc)

Inbound rules [Info](#)

| Type | Protocol | Port range | Source | Description - optional |
|-------|----------|------------|--------|--|
| HTTPS | TCP | 443 | Custom | <input type="text" value="sg-0f9beb20f6d8232ef"/> Delete Use: "sg-0f9beb20f6d8232ef" CIDR blocks Security Groups lambda_security_group sg-0f9beb20f6d8232ef lambda_security_group |

[Add rule](#)

Outbound rules [Info](#)

| Type | Protocol | Port range | Destination | Description - optional |
|-------------|----------|------------|-------------|--|
| All traffic | All | All | Custom | <input type="text" value="0.0.0.0/0"/> Delete 0.0.0.0/0 |

[Add rule](#)

Tags - optional
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

⚠ Rules with destination of 0.0.0.0/0 or ::/0 allow your instances to send traffic to any IPv4 or IPv6 address. We recommend setting security group rules to be more restrictive and to only allow traffic to specific known IP addresses. [X](#)

(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

The screenshot shows the AWS Security Groups console for a security group named "sg-0bf459ae75dd19fcb - EC2-SG".

Details:

- Security group name: EC2-SG
- Security group ID: sg-0bf459ae75dd19fcb
- Description: launch-wizard-1 created 2025-09-20T15:27:54.817Z
- VPC ID: vpc-06d82d27152d9753e
- Owner: 182399724378
- Inbound rules count: 3 Permission entries
- Outbound rules count: 1 Permission entry

Inbound rules (3):

| Name | Security group rule... | IP version | Type | Protocol | Port range | Source | Description |
|------|------------------------|------------|-------|----------|------------|-----------|-------------|
| - | sgr-0f4906b21286ed1... | IPv4 | SSH | TCP | 22 | 0.0.0.0/0 | - |
| - | sgr-04f65e82808722a6a | IPv4 | HTTP | TCP | 80 | 0.0.0.0/0 | - |
| - | sgr-06bfa06a99a82284a | IPv4 | HTTPS | TCP | 443 | 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.

The screenshot shows the 'Create virtual private gateway' page in the AWS VPC console. The top navigation bar includes 'VPC', 'Virtual private gateways', and 'Create virtual private gateway'. The main section is titled 'Create virtual private gateway' with a 'Info' link. A descriptive text states: 'A virtual private gateway is the VPN concentrator on the Amazon side of the site-to-site VPN connection.' Below this, the 'Details' section contains a 'Name tag - optional' field where 'aws_virtual_private_gateway' is entered. An 'Autonomous System Number (ASN)' section shows 'Amazon default ASN' selected. The 'Tags' section allows adding key-value pairs; one tag 'Name' with value 'aws_virtual_private_gateway' is shown. At the bottom right are 'Cancel' and 'Create virtual private gateway' buttons.

VPC > Virtual private gateways > Create virtual private gateway

Create virtual private gateway [Info](#)

A virtual private gateway is the VPN concentrator on the Amazon side of the site-to-site VPN connection.

Details

Name tag - optional
Creates a tag with a key of 'Name' and a value that you specify.

aws_virtual_private_gateway

Value must be 256 characters or less in length.

Autonomous System Number (ASN)

Amazon default ASN
 Custom ASN

Tags
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs. Name tag helps you track your resources more easily. We recommend adding Name tag.

Key **Value - optional** [Remove](#)

[Add new tag](#)

You can add up to 49 more tags.

[Cancel](#) [Create virtual private gateway](#)

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)

Naveen Raaj

VPC > Customer gateways > Create customer gateway

Create customer gateway Info

A customer gateway is a resource that you create in AWS that represents the customer gateway device in your on-premises network.

Details

Name tag - optional
Creates a tag with a key of 'Name' and a value that you specify.

Value must be 256 characters or less in length.

BGP ASN Info
The ASN of your customer gateway device.

Value must be in 1 - 4294967294 range.

IP address Info
Specify the IP address for your customer gateway device's external interface.

Certificate ARN - optional
The ARN of a private certificate provisioned in AWS Certificate Manager (ACM).

Device - optional
Enter a name for the customer gateway device.

Tags
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs. Name tag helps you track your resources more easily. We recommend adding Name tag.

| Key | Value - optional |
|--|---|
| <input type="text" value="Name"/> X | <input type="text" value="company_gateway"/> X Remove |

Add new tag
You can add up to 49 more tags.

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

The screenshot shows two separate AWS VPC console pages. The top page displays the 'Virtual private gateways' list, and the bottom page displays the 'Customer gateways' list. Both lists show a single resource each, with their respective IDs, states, and other details.

Virtual private gateways (1/1) Info

| Name | Virtual private gateway ID | State | VPC attachment state | Type | VPC | Amazon ASN |
|-------------------------|----------------------------|-----------|----------------------|---------|----------------------------------|------------|
| aws_virtual_private_... | vgw-0050d00bd376b3762 | Available | Attached | ipsec.1 | vpc-06d82d27152d9753e serve... | 64512 |

Customer gateways (1/1) Info

| Name | Customer gateway ID | State | BGP ASN | IP address | Type | Certificate ARN |
|-----------------|-----------------------|-----------|---------|------------|---------|-----------------|
| company_gateway | cgw-0b22415b7618c61c7 | Available | 65000 | | ipsec.1 | - |

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

The screenshot shows the 'Create VPN connection' wizard in the AWS VPC console. The 'Details' step is selected, and the configuration is as follows:

- Name tag - optional:** aws_to_company_vpn_connection
- Target gateway type:** Virtual private gateway (selected)
- Virtual private gateway:** vgw-0050d00bd376b3762
- Customer gateway:** Existing (selected)
- Customer gateway ID:** cgw-0b22415b7618c61c7
- Routing options:** Static (selected)
- Static IP prefixes:** 192.168.10.0/23
- Pre-shared key storage:** Standard (selected)

An orange arrow points from the text "static ip address of your company's firewall" to the static IP prefix input field.

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

Naveen Raaj

VPC > Route tables > rtb-00e7ce38d80888a21 > Edit routes

Edit routes

| Destination | Target | Status | Propagated | Route Origin |
|-----------------|-------------------------|--------|------------|------------------|
| pl-78a54011 | vpce-017aeb2db4d536b61 | Active | No | CreateRoute |
| 10.0.0.0/16 | local | Active | No | CreateRouteTable |
| 192.168.10.0/23 | Virtual Private Gateway | Active | No | CreateRoute |
| | vgw-0050d00bd376b3762 | | | |

Add route Remove

Add route Cancel Preview Save changes

IAM VPC EC2 Aurora and RDS S3 EFS Route 53 CloudFront CloudWatch Elastic Container Service Elastic Kubernetes Service Elastic Container Registry Lambda API Gateway

VPC > VPN connections

You successfully updated vpn-0bf79a6cc46757291 / aws_to_company_vpn_connection.

VPN connections (1/1)

| Name | VPN ID | State | Virtual private gateway | Transit gateway | Customer gateway | Customer gateway ad |
|-------------------------------|-----------------------|-----------|-------------------------|-----------------|-----------------------|---------------------|
| aws_to_company_vpn_connection | vpn-0bf79a6cc46757291 | Available | vgw-0050d00bd376b3762 | - | cgw-0b22415b7618c61c7 | |

VPN connection vpn-0bf79a6cc46757291 / aws_to_company_vpn_connection

Details Tunnel details Static routes Tags

Routes 1

| IP prefixes | State |
|-----------------|-----------|
| 192.168.10.0/23 | Available |

Edit routes

October 25

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

Naveen Raaj

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

The screenshot shows the AWS VPC console with the 'VPN connections' page selected. On the left, a navigation sidebar lists various VPC-related services: TLS inspection configurations, Network Firewall resource groups, VPC endpoint associations, Virtual private network (VPN) (selected), Customer gateways, Virtual private gateways, Site-to-Site VPN connections, Client VPN endpoints, AWS Verified Access (under 'AWS Services'), Transit gateways, and Traffic Mirroring.

The main content area displays the 'VPN connections (1/1)' table. The table has columns for Name, VPN ID, State, Virtual private gateway, Transit gateway, Customer gateway, and Customer gateway address. One row is present, labeled 'aws_to_company_vpn_connection' with VPN ID 'vpn-0bf79a6cc46757291', State 'Available', VPG 'vgw-0050d00bd376b3762', and CGW 'cgw-0b22415b7618c61c7'. A large orange button labeled 'Edit' is visible on the right side of the row.

Below the table, a detailed view for the selected connection is shown. The title is 'VPN connection vpn-0bf79a6cc46757291 / aws_to_company_vpn_connection'. The 'Tunnel details' tab is active, showing the 'Tunnel state' table. This table lists two tunnels: Tunnel 1 (IP 13.200.182.14) and Tunnel 2 (IP 65.2.155.160), both of which are currently 'Down' (indicated by a red circle with a white exclamation mark). The provisioning status is 'Available' for both, and the last status change occurred on September 19, 2025, at 13:19:55 (UTC+05:30).

Under the 'Tunnel details' tab, there are two expandable sections: 'Tunnel 1 options' and 'Tunnel 2 options', each with an 'Info' link.

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.

The screenshot shows the AWS VPC console with the 'VPN connections' page. On the left, a navigation menu includes 'DNS firewall', 'Network Firewall', 'Virtual private network (VPN)', 'AWS Verified Access', and 'Transit gateways'. The main area displays a table for 'VPN connections (1/1)'. The table has columns: Name, VPN ID, State, Virtual private gateway, Transit gateway, Customer gateway, and Customer gateway address. A single row is shown for 'aws_to_company_vpn_connection' with values: vpn-0bf79a6cc46757291, Available, vgw-0050d00bd376b3762, -, cgw-0b22415b7618c61c7, and a redacted address. Below the table, a detailed view for 'aws_to_company_vpn_connection' is expanded. It has tabs for 'Details', 'Tunnel details', 'Static routes', and 'Tags'. The 'Details' tab is selected and shows the following configuration:

| Details | Tunnel details | Static routes | Tags |
|---|---|---|--|
| VPN ID vpn-0bf79a6cc46757291 | State Available | Virtual private gateway vgw-0050d00bd376b3762 | Customer gateway cgw-0b22415b7618c61c7 |
| Transit gateway - | Customer gateway address [Redacted] | Type ipsec.1 | Category VPN |
| VPC vpc-06d82d27152d9753e | Routing Static | Acceleration enabled Disabled | Authentication Pre-shared key |
| Local IPv4 network CIDR 0.0.0.0/0 | Remote IPv4 network CIDR 0.0.0.0/0 | Local IPv6 network CIDR - | Remote IPv6 network CIDR - |
| Core network ARN - | Core network attachment ARN - | Gateway association state Associated | Outside IP address type Public IPv4 |
| Secrets management ARN - | | | |

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)

Download configuration

Download a sample configuration based on your customer gateway. Note that this is a sample only, and that it will require modification for using Advanced Algorithms, Certificates, and/IPv6.

Vendor
The manufacturer of the customer gateway device (for example, Cisco Systems, Inc).

Fortinet

Platform
The class of the customer gateway device (for example, J-Series).

Fortigate 40+ Series

Software
The operating system running on the customer gateway device (for example, ScreenOS).

FortiOS 6.4.4+ (GUI)

IKE version
The IKE version you are using for your VPN connection.

ikev2

Include sample type - optional

Enable

Sample type
The default sample type compatibility mode includes all options. The recommended mode restricts options to only the most secure settings (IKEv2, etc.).

Select sample type

Cancel **Download**

5. Test the connectivity

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

```
root@[REDACTED]:/home/naveen/Downloads# ping 13.200.182.14
PING 13.200.182.14 (13.200.182.14) 56(84) bytes of data.
64 bytes from 13.200.182.14: icmp_seq=1 ttl=119 time=33.1 ms
64 bytes from 13.200.182.14: icmp_seq=2 ttl=119 time=87.7 ms
64 bytes from 13.200.182.14: icmp_seq=3 ttl=119 time=37.6 ms
^C
--- 13.200.182.14 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 33.125/52.803/87.687/24.733 ms
root@[REDACTED]:/home/naveen/Downloads# ping 169.254.180.80
PING 169.254.180.80 (169.254.180.80) 56(84) bytes of data.
^C
--- 169.254.180.80 ping statistics ---
11 packets transmitted, 0 received, 100% packet loss, time 10265ms

root@[REDACTED]:/home/naveen/Downloads# ping 65.2.155.160
PING 65.2.155.160 (65.2.155.160) 56(84) bytes of data.
64 bytes from 65.2.155.160: icmp_seq=1 ttl=119 time=30.3 ms
64 bytes from 65.2.155.160: icmp_seq=2 ttl=119 time=30.5 ms
64 bytes from 65.2.155.160: icmp_seq=3 ttl=119 time=33.3 ms
64 bytes from 65.2.155.160: icmp_seq=4 ttl=119 time=118 ms
^C
--- 65.2.155.160 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 30.296/53.062/118.209/37.631 ms
root@[REDACTED]:/home/naveen/Downloads# █
```

(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**

The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with sections like EC2, Dashboard, AWS Global View, Events, Instances (selected), Instances Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, Placement Groups, Key Pairs, and Network Interfaces.

The main content area displays "Instances (1/1) Info". A search bar says "Find Instance by attribute or tag (case-sensitive)" and a filter button says "Instance state = running". A table lists one instance: "new_instance" (Instance ID: i-04c598897f0089707, State: Running, Type: t2.micro, Status check: 2/2 checks passed, Availability Zone: ap-south-1a). Buttons for "Connect", "Actions", and "Launch instances" are at the top right.

A detailed view for "i-04c598897f0089707 (new_instance)" is shown. The "Details" tab is selected. Under "Instance summary", fields include:

| Field | Value |
|----------------------------------|--|
| Instance ID | i-04c598897f0089707 |
| IPv6 address | - |
| Hostname type | IP name: ip-10-0-1-181.ap-south-1.compute.internal |
| Answer private resource DNS name | - |
| Auto-assigned IP address | - |
| IAM Role | - |
| IMDSv2 | Required |
| Operator | - |

Under "Networking", fields include:

| Field | Value |
|---------------------------------|--|
| Public IPv4 address | - |
| Private IP DNS name (IPv4 only) | ip-10-0-1-181.ap-south-1.compute.internal |
| Instance type | t2.micro |
| VPC ID | vpc-06d82d27152d9753e (serverless_vpc) |
| Subnet ID | subnet-0d72d2c409922ff09 (serverless_subnet_a) |
| Instance ARN | arn:aws:ec2:ap-south-1:182399724378:instance/i-04c598897f0089707 |

Under "Security", fields include:

| Field | Value |
|-------------------------------|--|
| Public IPv4 addresses | 10.0.1.181 |
| Public DNS | - |
| Elastic IP addresses | - |
| AWS Compute Optimizer finding | <small>Opt-in to AWS Compute Optimizer for recommendations. Learn more</small> |
| Auto Scaling Group name | - |
| Managed | false |

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

Naveen Raaj

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**

Connect Info

Connect to an instance using the browser-based client.

EC2 Instance Connect Session Manager **SSH client** EC2 serial console

Instance ID
 [i-04c598897f0089707 \(new_instance\)](#)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is new_keypair.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
 `chmod 400 "new_keypair.pem"`
4. Connect to your instance using its Private IP:
 `10.0.1.181`

Example:
 `ssh -i "new_keypair.pem" ubuntu@10.0.1.181`

Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

[Cancel](#)

(d). Now the connection has been established 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

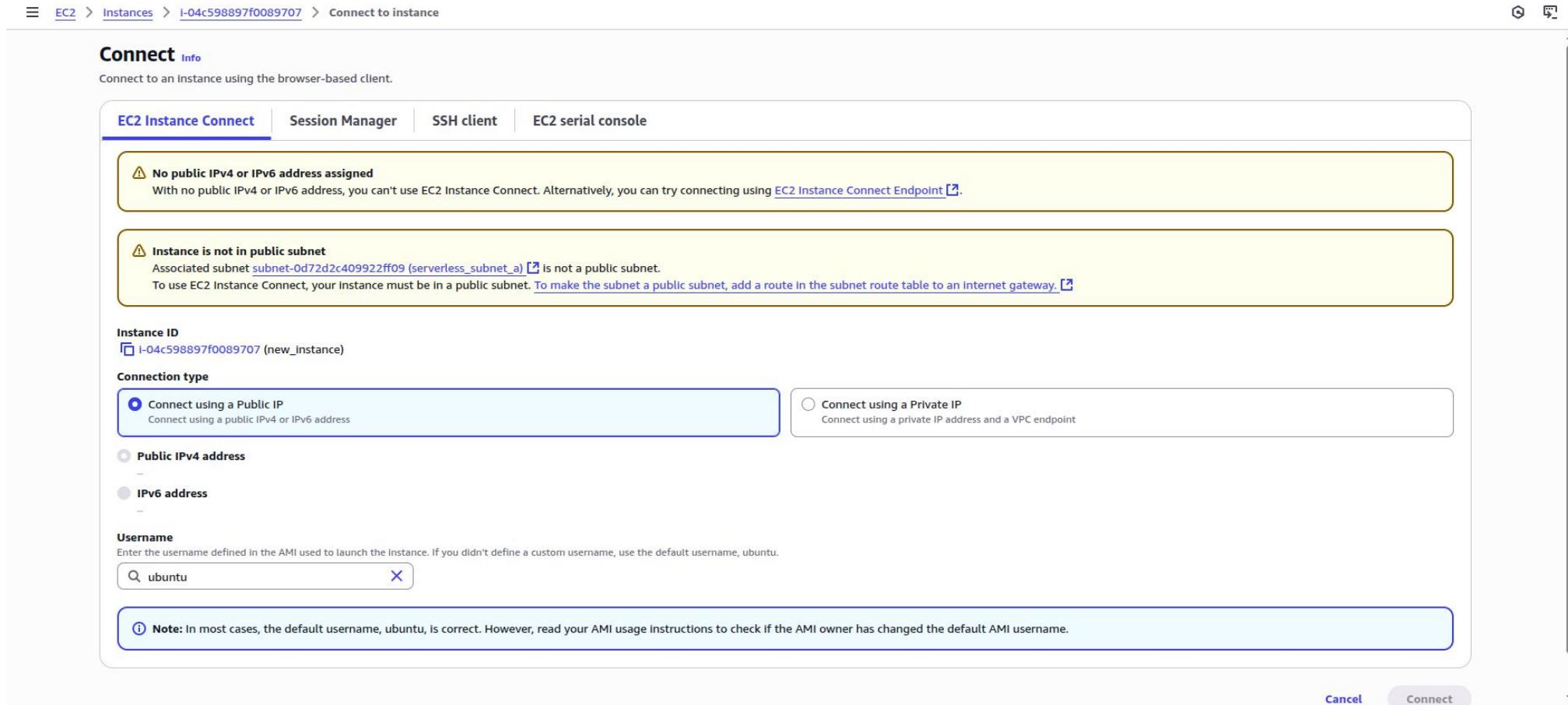
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
              inet6 ::1 prefixlen 128 scopeid 0x10<host>
                loop txqueuelen 1000 (Local Loopback)
                RX packets 204715 bytes 103065625 (103.0 MB)
                RX errors 0 dropped 0 overruns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

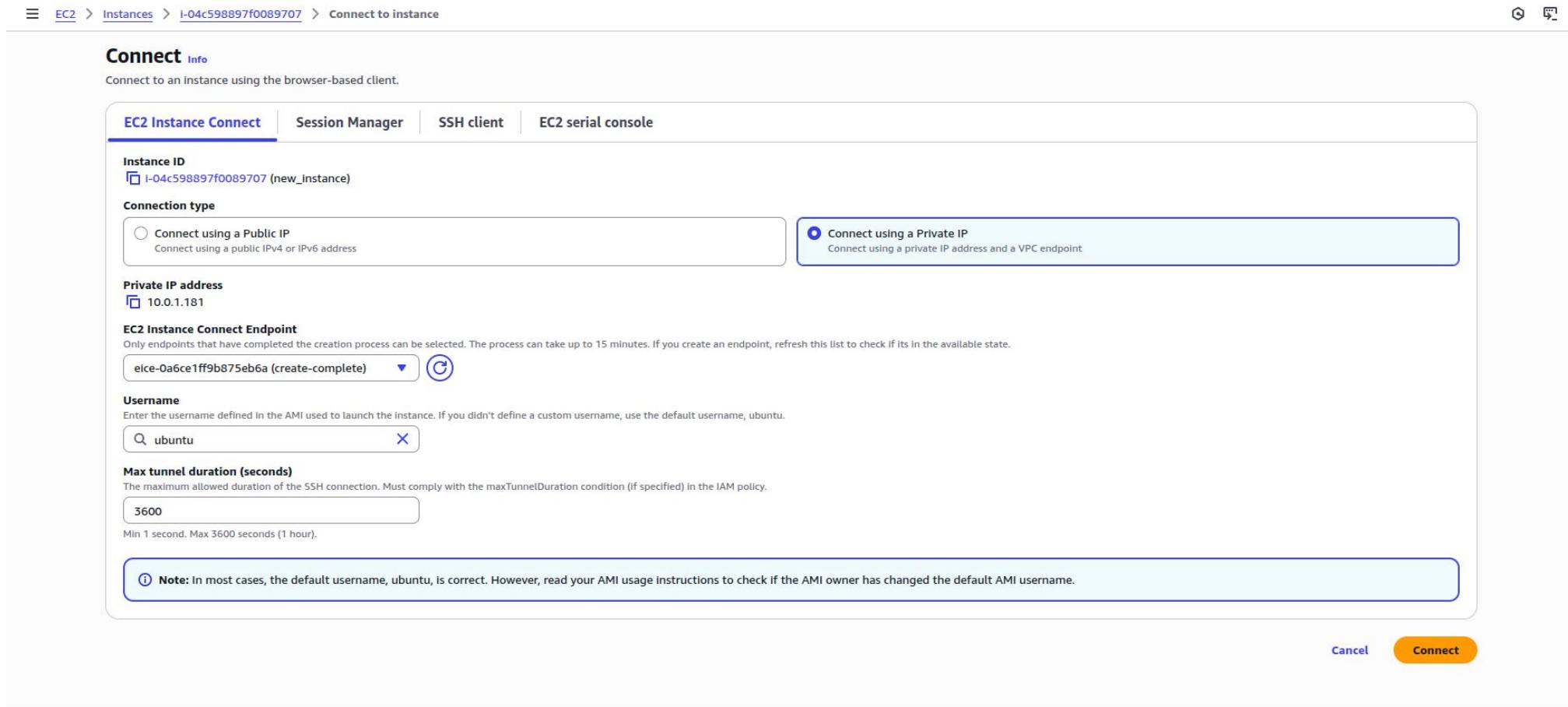
(e). Ping your laptop private IP address **192.168.10.18** from the AWS EC2 private instance which is **10.0.1.181**

```
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

The screenshot shows the 'Create endpoint' wizard in the AWS Management Console. The current step is 'Endpoint settings'. The 'Name tag - optional' field contains 'ec2_instance_connect_endpoint'. The 'Type' section is expanded, showing several options: 'AWS services', 'PrivateLink Ready partner services', 'AWS Marketplace services', 'EC2 Instance Connect Endpoint' (which is selected), 'Resources', and 'Service networks'. The 'Network settings' section is also visible, showing the VPC selection dropdown set to 'vpc-06d82d27152d9753e (serverless_vpc)'. A large blue arrow points from the text above to the 'EC2 Instance Connect Endpoint' option.

VPC > Endpoints > Create endpoint

Create endpoint Info

Create the type of VPC endpoint that supports the service, service network or resource to which you want to connect.

Endpoint settings

Specify a name and select the type of endpoint.

Name tag - optional

Creates a tag with a key of 'Name' and a value that you specify. Tags help you find and manage your endpoint.

`ec2_instance_connect_endpoint`

Type Info

Select a category

- AWS services
Connect to services provided by Amazon with an Interface endpoint, or a Gateway endpoint
- EC2 Instance Connect Endpoint
An elastic network interface that allows you to connect to resources in a private subnet
- PrivateLink Ready partner services
Connect to SaaS services which have AWS Service Ready designation with an Interface endpoint. Uses AWS PrivateLink
- AWS Marketplace services
Connect to SaaS services that you have purchased through AWS Marketplace with an Interface Endpoint
- Resources
Connect to resources like Amazon Relational Database Services (RDS) with a Resource endpoint. Uses AWS PrivateLink
- Service networks
Connect to VPC Lattice service networks with a Service network endpoint. Uses AWS PrivateLink

Network settings

Select the VPC in which to create the endpoint

VPC

Create the VPC endpoint in the VPC in the same AWS Region from which you will access a resource.

vpc-06d82d27152d9753e (serverless_vpc)

▶ Additional 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.

The screenshot shows the AWS VPC Endpoints 'Create endpoint' wizard at the 'Additional settings' step. The top navigation bar includes the AWS logo, search bar, and various services like IAM, VPC, EC2, Aurora and RDS, S3, EFS, Route 53, CloudFront, CloudWatch, Elastic Container Service, Elastic Kubernetes Service, Elastic Container Registry, Lambda, and API Gateway. The account ID is 1823-9972-4378, and the region is Asia Pacific (Mumbai).

Security groups (1/4)

VPC ID: vpc-06d82d27152d9753e

| Group ID | Group name | VPC ID | Description |
|---|---------------------------|-----------------------|---|
| sg-04e2f4aed24506648 | default | vpc-06d82d27152d9753e | default VPC security group |
| sg-0f9beb20f6d8232ef | lambda_security_group | vpc-06d82d27152d9753e | security group to connect lambda with vpc endpoints |
| sg-03b7a2e9991571173 | vpc_interface_endpoint_SG | vpc-06d82d27152d9753e | allows the inbound connections from lambda security group through ENI |
| <input checked="" type="checkbox"/> sg-0bf459ae75dd19fc | EC2-SG | vpc-06d82d27152d9753e | launch-wizard-1 created 2025-09-20T15:27:54.817Z |

Subnet
Select the Subnet in which to create the endpoint
Subnet
Select the subnets in which to create the endpoint
subnet-0d72d2c409922ff09 (serverless_subnet_a)

IP address type
 IPv4
 IPv6
 Dualstack

Tags
No tags associated with the resource.
[Add new tag](#)

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

(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

```
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 04:38:26 UTC 2025

System load: 0.0      Processes:           110
Usage of /: 25.9% of 6.71GB  Users logged in: 1
Memory usage: 21%
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
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings

Last login: Mon Sep 22 04:34:59 2025 from 192.168.10.18
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

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=103 ms
64 bytes from 192.168.10.18: icmp_seq=2 ttl=63 time=32.7 ms
64 bytes from 192.168.10.18: icmp_seq=3 ttl=63 time=34.5 ms
64 bytes from 192.168.10.18: icmp_seq=4 ttl=63 time=39.5 ms
64 bytes from 192.168.10.18: icmp_seq=5 ttl=63 time=49.1 ms
```

i-04c598897f0089707 (new_instance)

PrivateIP: 10.0.1.181

(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.

```
ubuntu@ip-10-0-1-181:~$ ping 192.168.10.1
PING 192.168.10.1 (192.168.10.1) 56(84) bytes of data.
64 bytes from 192.168.10.1: icmp_seq=1 ttl=254 time=30.5 ms
64 bytes from 192.168.10.1: icmp_seq=2 ttl=254 time=30.0 ms
64 bytes from 192.168.10.1: icmp_seq=3 ttl=254 time=30.0 ms
64 bytes from 192.168.10.1: icmp_seq=4 ttl=254 time=30.1 ms
64 bytes from 192.168.10.1: icmp_seq=5 ttl=254 time=30.1 ms
64 bytes from 192.168.10.1: icmp_seq=6 ttl=254 time=29.9 ms
64 bytes from 192.168.10.1: icmp_seq=7 ttl=254 time=30.1 ms
64 bytes from 192.168.10.1: icmp_seq=8 ttl=254 time=29.9 ms
64 bytes from 192.168.10.1: icmp_seq=9 ttl=254 time=30.1 ms
64 bytes from 192.168.10.1: icmp_seq=10 ttl=254 time=30.2 ms
^C
--- 192.168.10.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9013ms
rtt min/avg/max/mdev = 29.912/30.091/30.593/0.159 ms
ubuntu@ip-10-0-1-181:~$ ping 192.168.10.2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=63 time=36.5 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=63 time=30.5 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=63 time=36.4 ms
64 bytes from 192.168.10.2: icmp_seq=4 ttl=63 time=32.3 ms
^C
--- 192.168.10.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 30.414/30.952/32.305/0.782 ms
ubuntu@ip-10-0-1-181:~$ ping 192.168.10.3
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data.
64 bytes from 192.168.10.3: icmp_seq=1 ttl=63 time=31.7 ms
64 bytes from 192.168.10.3: icmp_seq=2 ttl=63 time=30.3 ms
^C
--- 192.168.10.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 30.344/31.034/31.724/0.690 ms
ubuntu@ip-10-0-1-181:~$ ping 192.168.10.11
PING 192.168.10.11 (192.168.10.11) 56(84) bytes of data.
64 bytes from 192.168.10.11: icmp_seq=1 ttl=63 time=30.1 ms
64 bytes from 192.168.10.11: icmp_seq=2 ttl=63 time=30.1 ms
64 bytes from 192.168.10.11: icmp_seq=3 ttl=63 time=30.2 ms
64 bytes from 192.168.10.11: icmp_seq=4 ttl=63 time=30.5 ms
64 bytes from 192.168.10.11: icmp_seq=5 ttl=63 time=30.2 ms
```

i-04c598897f0089707 (new_instance)

PrivateIPs: 10.0.1.181

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

Naveen Raaj

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 Lambda doesn't need interface endpoint for accessing Cloudwatch?

Naveen Raaj

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 Lambda doesn't need interface endpoint for accessing API gateway?

Naveen Raaj

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:

Naveen Raaj

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:

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

| 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 |

Where to See These ENIs

Naveen Raaj

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:

Naveen Raaj

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

Naveen Raaj

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

The screenshot shows the 'Edit VPC settings' page for a VPC with ID 'vpc-06d82d27152d9753e'. The 'DNS settings' section is expanded, displaying two checked options: 'Enable DNS resolution' and 'Enable DNS hostnames'. The 'Save' button at the bottom right is highlighted.

VPC ID: vpc-06d82d27152d9753e
Name: serverless_vpc

DNS settings

- Enable DNS resolution
- Enable DNS hostnames

Network Address Usage metrics settings

- Enable Network Address Usage metrics

Cancel Save

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 establish 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

The screenshot shows the 'Create endpoint' wizard in the AWS VPC console. The first step, 'Endpoint settings', is completed with the name 's3_endpoint' and the type 'AWS services' selected. The second step, 'Services (1/3)', lists available gateway services. The 'com.amazonaws.ap-south-1.s3' service is selected, highlighted with a blue border.

Endpoint settings
Specify a name and select the type of endpoint.
Name tag - optional
Creates a tag with a key of 'Name' and a value that you specify. Tags help you find and manage your endpoint.
s3_endpoint

Type Info
Select a category

- AWS services**
Connect to services provided by Amazon with an Interface endpoint, or a Gateway endpoint
- EC2 Instance Connect Endpoint**
An elastic network interface that allows you to connect to resources in a private subnet
- Endpoint services that use NLBs and GWLBs**
Find services shared with you by service name. Connect to a Network LoadBalancer (NLB) service with an Interface endpoint or to a Gateway LoadBalancer (GWLB) service with a Gateway Load Balancer endpoint
- PrivateLink Ready partner services**
Connect to SaaS services which have AWS Service Ready designation with an Interface endpoint. Uses AWS PrivateLink
- AWS Marketplace services**
Connect to SaaS services that you have purchased through AWS Marketplace with an Interface Endpoint
- Resources**
Connect to resources like Amazon Relational Database Services (RDS) with a Resource endpoint. Uses AWS PrivateLink
- Service networks**
Connect to VPC Lattice service networks with a Service network endpoint. Uses AWS PrivateLink

Services (1/3)

| <input type="text"/> Search | | | | |
|--|----------------------------------|--|----------------|--|
| <input type="button" value="Type = Gateway"/> | <input type="button" value="X"/> | <input type="button" value="Clear filters"/> | | |
| Service Name | Owner | Type | Service Region | |
| com.amazonaws.ap-south-1.dynamodb | amazon | Gateway | ap-south-1 | |
| <input checked="" type="radio"/> com.amazonaws.ap-south-1.s3 | amazon | Gateway | ap-south-1 | |
| com.amazonaws.ap-south-1.s3express | amazon | Gateway | ap-south-1 | |

(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.

The screenshot shows the 'Create endpoint' wizard in the AWS VPC service. The steps are:

- Network settings:** Select the VPC in which to create the endpoint. A dropdown menu shows 'vpc-06d82d27152d9753e (serverless_vpc)'.

| Name | Route Table ID | Main | Associated Id |
|------------------------|--|------|---------------|
| serverless_route_table | rtb-00e7ce38d80888a21 (serverless_ro...) | Yes | 2 subnets |
- Route tables (1/1):** Shows the selected route table 'serverless_route_table' associated with 'rtb-00e7ce38d80888a21 (serverless_ro...)'. It has 2 subnets associated with it.
- Policy:** Info: VPC endpoint policy controls access to the service.
 - Full access**: Allows access by any user or service within the VPC using credentials from any Amazon Web Services accounts to any resources in this Amazon Web Services service. All policies — IAM user policies, VPC endpoint policies, and Amazon Web Services service-specific policies (e.g. Amazon S3 bucket policies, any S3 ACL policies) — must grant the necessary permissions for access to succeed.
 - Custom**: Use the [policy creation tool](#) to generate a policy, then paste the generated policy below.A text input field contains the number '1'.

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)

The screenshot shows the AWS VPC console with the path: VPC > Endpoints > Create endpoint. The main section is titled "Create endpoint" with a "Info" link. It asks to "Create the type of VPC endpoint that supports the service, service network or resource to which you want to connect." Below this is the "Endpoint settings" section, which asks to "Specify a name and select the type of endpoint." A "Name tag - optional" field contains "secrets_manager_endpoint". The "Type" section has a "Info" link and asks to "Select a category". It lists several options: "AWS services" (selected), "PrivateLink Ready partner services", "AWS Marketplace services", "EC2 Instance Connect Endpoint", "Resources", "Service networks", and "Endpoint services that use NLBs and GWLBs". The "Services (1/1)" section shows a table with one row: Service Name = com.amazonaws.ap-south-1.secretsmanager, Owner: amazon, Type: Interface, and Service Region: ap-south-1.

| Service Name | Owner | Type | Service Region |
|---|--------|-----------|----------------|
| com.amazonaws.ap-south-1.secretsmanager | amazon | Interface | ap-south-1 |

(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**

The screenshot shows the 'Create endpoint' wizard in the AWS VPC console. The first step, 'Network settings', is displayed. Under 'VPC', the 'serverless_vpc' is selected. In the 'Additional settings' section, the 'Enable DNS name' checkbox is checked. Under 'DNS record IP type', 'IPv4' is selected. The 'Subnets (2/3)' section lists three subnets: 'aps1-az1 (ap-south-1a)', 'aps1-az3 (ap-south-1b)', and 'aps1-az2 (ap-south-1c)'. The first two are checked and have their 'Designate IP addresses' boxes checked. The third one has its checkbox checked but the 'Designate IP addresses' box is empty. The 'IP address type' section shows 'IPv4' selected.

(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.

The screenshot shows the AWS VPC console with the path: VPC > Endpoints > Create endpoint. The current step is "Security groups (1/3)". A table lists three security groups:

| Group ID | Group name | VPC ID | Description |
|--------------------------------------|---------------------------|---------------------------------------|---|
| sg-04e2f4aed24506648 | default | vpc-06d82d27152d9753e | default VPC security group |
| sg-0f9beb20f6d8232ef | lambda_security_group | vpc-06d82d27152d9753e | security group to connect lambda with vpc endpoints |
| sg-03b7a2e9991571173 | vpc_interface_endpoint_SG | vpc-06d82d27152d9753e | allows the inbound connections from lambda security group through ENI |

The third row, "vpc_interface_endpoint_SG", is selected and highlighted with a blue border. Below the table, there is a section titled "Policy" with a "Full access" radio button selected. The "Custom" option is also present with a note about using the policy creation tool.

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.

The screenshot shows the AWS VPC Endpoint creation interface. At the top, there is an error message: "There was an error creating VPC endpoint" followed by the subtext "Enabling private DNS requires both enableDnsSupport and enableDnsHostnames VPC attributes set to true for vpc-06d82d27152d9753e". Below this, there are four columns: Service Name, Owner, Type, and Service Region. Under Network settings, it says "Select the VPC in which to create the endpoint" and shows "vpc-06d82d27152d9753e (serverless_vpc)" selected. In the Additional settings section, under DNS name, the "Enable DNS name" checkbox is checked. Under DNS record IP type, "IPv4" is selected. In the Subnets section, three subnets are listed: "aps1-az1 (ap-south-1a)" with subnet ID "subnet-0d72d2c409922ff09", "aps1-az3 (ap-south-1b)" with subnet ID "subnet-055b9a2e51f7d2178", and "aps1-az2 (ap-south-1c)" with the note "(i) No subnet available".

(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

The screenshot shows the 'Edit VPC settings' page for a VPC with ID 'vpc-06d82d27152d9753e'. The 'DNS settings' section is expanded, displaying two checked options: 'Enable DNS resolution' and 'Enable DNS hostnames'. The 'Save' button at the bottom right is highlighted.

VPC details

VPC ID: vpc-06d82d27152d9753e
Name: serverless_vpc

DHCP settings

DHCP option set: dopt-0c044edcb1e622a27

DNS settings

Enable DNS resolution

Enable DNS hostnames

Network Address Usage metrics settings

Enable Network Address Usage metrics

Cancel Save

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

| Endpoints (2) <small>Info</small> | | | | | |
|-----------------------------------|--------------------------|-------------------------|---------------|------------------------|---|
| <input type="checkbox"/> | Name | VPC endpoint ID | Endpoint type | Status | Service name |
| <input type="checkbox"/> | s3_endpoint | vpc-e-0b80e78a6f1a698bd | Gateway | Available | com.amazonaws.ap-south-1.s3 |
| <input type="checkbox"/> | secrets_manager_endpoint | vpc-e-0b9a2202ea9d35336 | Interface | Available | com.amazonaws.ap-south-1.secretsmanager |

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

| Network interfaces (1/4) <small>Info</small> | | | | | | | | | |
|---|----------------------------|-----------------------|--------------------------|-----------------------|-------------------|------------------------|----------------------|-------------|--|
| Last updated less than a minute ago <input type="button" value="Actions"/> <input type="button" value="Create network interface"/> | | | | | | | | | |
| <input type="checkbox"/> | Name <small>Filter</small> | Network interface ID | Subnet ID | VPC ID | Availability Zone | Security group n... | Security group IDs | Interfac... | |
| <input checked="" type="checkbox"/> | ENI_Cloudwatch_A | eni-0df35b77dff2e3425 | subnet-0d72d2c409922ff09 | vpc-06d82d27152d9753e | ap-south-1a | lambda_security_gro... | sg-0f9beb20f6d823... | lambda | |
| <input type="checkbox"/> | ENI_Secret_B | eni-0ee15a0b024d42504 | subnet-0d72d2c409922ff09 | vpc-06d82d27152d9753e | ap-south-1a | vpc_interface_endpo... | sg-03b7a2e9991571... | vpc_end | |
| <input type="checkbox"/> | ENI_Cloudwatch_B | eni-0b2097ba973aeef02 | subnet-055b9a2e51f7d2178 | vpc-06d82d27152d9753e | ap-south-1b | lambda_security_gro... | sg-0f9beb20f6d823... | lambda | |
| <input type="checkbox"/> | ENI_Secret_B | eni-0d4e7111987f99a4d | subnet-055b9a2e51f7d2178 | vpc-06d82d27152d9753e | ap-south-1b | vpc_interface_endpo... | sg-03b7a2e9991571... | vpc_end | |

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

The screenshot shows the 'Choose secret type' step of the AWS Secrets Manager wizard. On the left, a vertical navigation bar lists four steps: Step 1 (selected), Step 2, Step 3 - optional, and Step 4. The selected step, 'Step 1 Choose secret type', is highlighted with a blue circle. The main content area is titled 'Choose secret type'. It contains a section for 'Secret type' with five options: 'Credentials for Amazon RDS database', 'Credentials for Amazon DocumentDB database', 'Credentials for Amazon Redshift data warehouse', 'Credentials for other database' (which is selected and highlighted with a blue border), and 'Other type of secret API key, OAuth token, other.' Below this, there are three sections: 'Credentials' (User name: 'sa', Password: '*****', Show password checkbox), 'Encryption key' (aws/secretsmanager dropdown, Add new key button), and 'Tags' (empty field).

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

Encryption key Info
You can encrypt using the KMS key that Secrets Manager creates or a customer managed KMS key that you create.

▼ C
[Add new key](#)

Database Info

MariaDB 
 MySQL 
 PostgreSQL 

Oracle Database 
 SQL Server 

Server address
Database name
Port

Cancel Next

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

The screenshot shows the 'Configure secret' step in the AWS Secrets Manager wizard. The left sidebar lists four steps: Step 1 (Choose secret type), Step 2 (Configure secret, which is selected and highlighted in blue), Step 3 - optional (Configure rotation), and Step 4 (Review). The main area is titled 'Configure secret' and contains the following fields:

- Secret name and description**:
 - Secret name**: A descriptive name that helps you find your secret later. The value is 'my-sqlserver-credentials'.
 - Description - optional**: Access to on-prem SQL server database from AWS.
- Tags - optional**: No tags associated with the secret. There is an 'Add' button.
- Resource permissions - optional**: Add or edit a resource policy to access secrets across AWS accounts. There is an 'Edit permissions' button.
- Replicate secret - optional**: Create read-only replicas of your secret in other Regions. Replica secrets incur a charge.

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

The screenshot shows the AWS Secrets Manager interface for creating a new secret. The navigation bar at the top indicates the user is in the 'Secrets' section under 'Store a new secret'. On the left, a vertical navigation menu lists four steps: 'Step 1 Choose secret type', 'Step 2 Configure secret', 'Step 3 - optional Configure rotation' (which is selected and highlighted in blue), and 'Step 4 Review'. The main content area is titled 'Configure rotation - optional'. It contains three sections: 'Configure automatic rotation' (with a note to 'Configure AWS Secrets Manager to rotate this secret automatically.' and a toggle switch for 'Automatic rotation'), 'Rotation schedule' (with tabs for 'Schedule expression builder' (selected) and 'Schedule expression', and fields for 'Time unit' set to 'Hours' and 'Hours' set to '23'; also includes a 'Window duration - optional' field with '4h' and a note to 'Enter the time in hours.', and a checkbox for 'Rotate immediately when the secret is stored. The next rotation will begin on your schedule.'), and 'Rotation function' (with a 'Lambda rotation function' dropdown containing 'serverless_function' and a 'Create function' button). At the bottom right are 'Cancel', 'Previous', and 'Next' buttons.

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

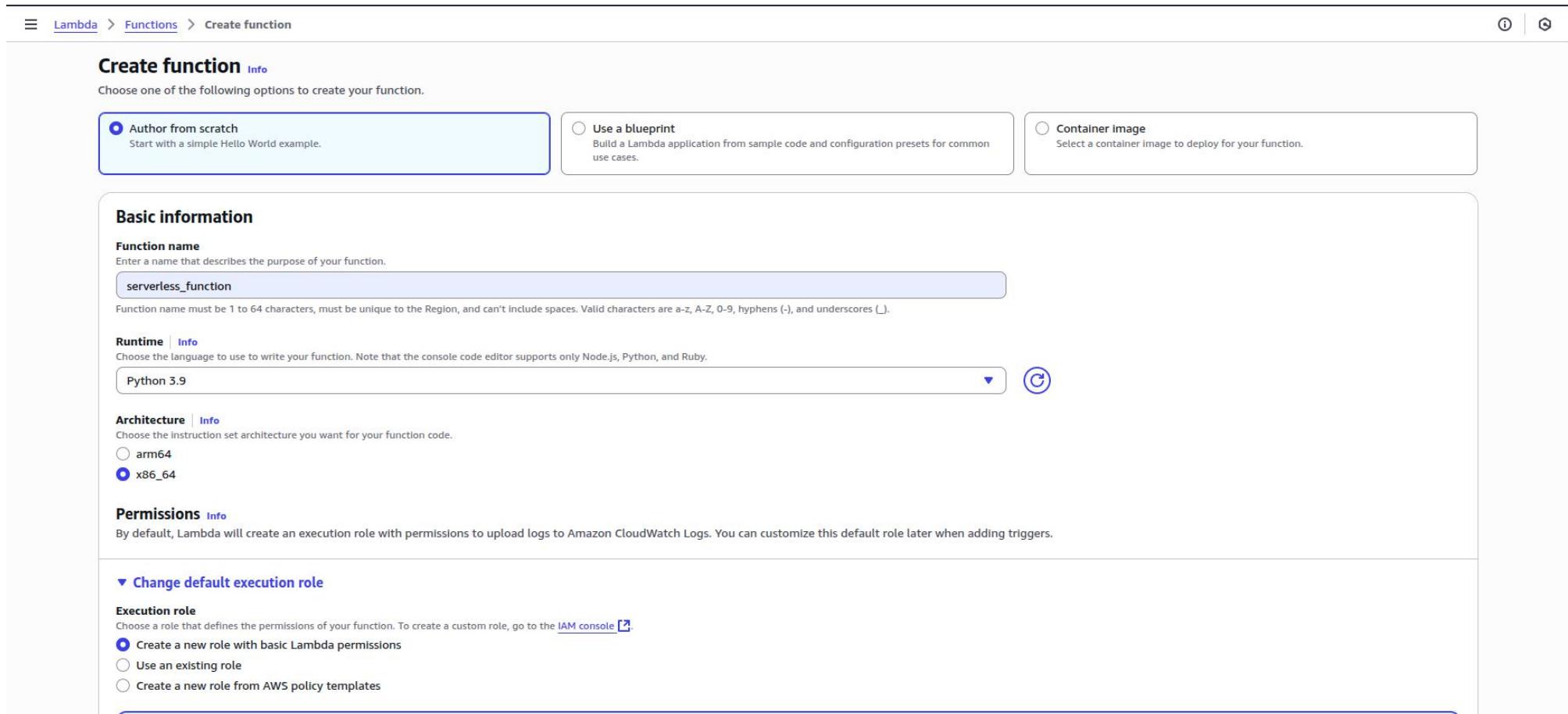
The screenshot shows the AWS Secrets Manager console. At the top, there's a breadcrumb navigation: AWS Secrets Manager > Secrets. Below it, a search bar with placeholder text "Filter secrets by name, description, tag key, tag value, owning service or primary Region". To the right is a blue "Store a new secret" button with a circular arrow icon. The main area is titled "Secrets" and contains a table with one row. The table has three columns: "Secret name", "Description", and "Last retrieved (UTC)". The secret name is "my-sqlserver-credentials", the description is "Access to onprem SQL server database from AWS", and the last retrieved time is "-".

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

The screenshot shows the AWS IAM Roles page. The left sidebar has a "Identity and Access Management (IAM)" section with "Roles" selected, and a "Permissions" section. The main area shows the "serverless_function-role-121m310g" role. It has a "Summary" section with creation date (September 20, 2025), last activity (8 days ago), ARN (arn:aws:iam::182399724378:role/service-role/serverless_function-role-121m310g), and maximum session duration (1 hour). Below this is a "Permissions" tab showing "Permissions policies (4)". There are four policies listed: "AmazonS3FullAccess" (AWS managed, 6 entities), "AWSLambdaBasicExecutionRole-5c27530c-36f1-4c82-949a-b7831d424300" (Customer managed, 1 entity), "AWSLambdaBasicExecutionRole-5c27538c-36f1-4c82-949a-b7831d424300" (Customer managed, 1 entity), and "lambda_secretmanager_access" (Customer inline, 0 entities). There are buttons for "Simulate", "Remove", and "Add permissions".

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.

The screenshot shows the 'Create function' wizard in the AWS Lambda console. The 'Additional configurations' section is expanded, specifically the 'Networking' tab. Under 'Function URL' and 'Info', there is an 'Enable' checkbox. Under 'VPC' and 'Info', there is also an 'Enable' checkbox which is checked. A note below states: 'When you connect a function to a VPC in your account, it does not have access to the internet unless your VPC provides access. To give your function access to the internet, route outbound traffic to a NAT gateway in a public subnet.' Below this note is a 'Learn more' link. In the 'Subnets' section, a dropdown menu shows 'vpc-06d82d27152d9753e (10.0.0.0/16)' with a refresh icon. An unchecked checkbox for 'Allow IPv6 traffic for dual-stack subnets' has a note: 'You can allow outbound IPv6 traffic to subnets that have both IPv4 and IPv6 CIDR blocks.' In the 'Security groups' section, a dropdown menu shows 'sg-0f9beb20f6d8232ef (lambda_security_group)' with a refresh icon. This security group is described as 'security group to connect lambda with vpc endpoints' and its name is listed as 'lambda_security_group'.

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

Lambda --> Functions --> Configurations

The screenshot shows the AWS Lambda Functions configuration page for a function named 'serverless_function'. The top navigation bar includes 'Lambda > Functions > serverless_function'. On the right, there are buttons for 'Throttle', 'Copy ARN', 'Actions', 'Export to Infrastructure Composer', and 'Download'. The main area displays the 'Function overview' section with a 'Diagram' tab selected, showing a single function icon labeled 'serverless_function' with '(0)' layers. Below this are buttons for '+ Add trigger' and '+ Add destination'. To the right, there's a detailed sidebar with fields for 'Description' (empty), 'Last modified' (15 hours ago), 'Function ARN' (arn:aws:lambda:ap-south-1:182399724378:function:serverless_function), and 'Function URL' (empty). At the bottom, tabs for 'Code', 'Test', 'Monitor', 'Configuration' (selected), 'Aliases', and 'Versions' are visible. The 'General configuration' section on the left lists 'Triggers', 'Permissions', 'Destinations', 'Function URL', 'Environment variables', and 'Tags'. The 'General configuration' details on the right show 'Description' (empty), 'Memory' (128 MB), 'Ephemeral storage' (512 MB), 'Timeout' (0 min 3 sec), and 'SnapStart' (None).

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.

The screenshot shows the 'Edit basic settings' page for a Lambda function. The 'Memory' field is set to 128 MB, and the 'Timeout' field is set to 30 seconds. The 'Execution role' is set to 'Use an existing role' with the role name 'service-role/serverless_function-role-121m310g' selected. Other settings like 'Description', 'Ephemeral storage', 'SnapStart', and 'Supported runtimes' are also visible.

Basic settings

Description - optional

Memory Info
Your function is allocated CPU proportional to the memory configured.
128 MB
Set memory to between 128 MB and 10240 MB

Ephemeral storage Info
You can configure up to 10 GB of ephemeral storage (/tmp) for your function. [View pricing](#)
512 MB
Set ephemeral storage (/tmp) to between 512 MB and 10240 MB

SnapStart Info
Reduce startup time by having Lambda cache a snapshot of your function after the function has initialized. To evaluate whether your function code is resilient to snapshot operations, review the [SnapStart compatibility considerations](#). For Python and .NET runtimes, [view pricing](#).
None

Supported runtimes: .NET 8 (C#/F#/PowerShell), Java 11, Java 17, Java 21, Python 3.12, Python 3.13.

Timeout
0 min 30 sec

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).
 Use an existing role
 Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
service-role/serverless_function-role-121m310g

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.

The screenshot shows the AWS Lambda console interface. The top navigation bar includes links for IAM, VPC, EC2, Aurora and RDS, S3, EFS, Route 53, CloudFront, CloudWatch, Elastic Container Service, Elastic Kubernetes Service, Elastic Container Registry, Lambda, and API Gateway. The account ID is 1823-9972-4378, and the region is Asia Pacific (Mumbai). The user Naveen is logged in.

The left sidebar lists configuration tabs: General configuration, Triggers, **Permissions**, Destinations, Function URL, Environment variables, Tags, VPC, RDS databases, Monitoring and operations tools, Concurrency and recursion detection, Asynchronous invocation, Code signing, File systems, and State machines. The **Permissions** tab is currently selected.

The main content area is titled "Execution role". It shows the "Role name" as "serverless_function-role-121m310g". There are three buttons: "Edit", "View role document", and a refresh icon. Below this is a "Resource summary" section with a note: "To view the resources and actions that your function has permission to access, choose a service." A dropdown menu is open, showing "Amazon CloudWatch Logs" with "3 actions, 2 resources".

Below the dropdown, there are two tabs: "By action" and "By resource", with "By resource" being the active tab. It displays a table with columns "Resource" and "Actions". Two entries are listed:

| Resource | Actions |
|--|---|
| arn:aws:logs:ap-south-1:182399724378:* | Allow: logs>CreateLogGroup |
| arn:aws:logs:ap-south-1:182399724378:log-group:/aws/lambda/serverless_function:* | Allow: logs>CreateLogStream Allow: logs>PutLogEvents |

A callout box at the bottom states: "Lambda obtained this information from the following policy statements:" followed by a bulleted list:

- Managed policy AWSLambdaBasicExecutionRole-5c27538c-36f1-4c82-949a-b7831d424300, statement 0
- Managed policy AWSLambdaBasicExecutionRole-5c27538c-36f1-4c82-949a-b7831d424300, statement 1

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.

The screenshot shows the AWS Lambda Execution role configuration page. The top navigation bar includes tabs for Code, Test, Monitor, Configuration (which is selected), Aliases, and Versions. On the left, a sidebar lists various configuration options: General configuration, Triggers, Permissions (selected), Destinations, Function URL, Environment variables, Tags, VPC, RDS databases, Monitoring and operations tools, Concurrency and recursion detection, Asynchronous invocation, Code signing, and File systems. The main content area is titled 'Execution role' and shows the 'Role name' as 'serverless_function-role-121m310g'. It features a 'Resource summary' section with a dropdown menu showing 'AWS Secrets Manager' (1 action, 1 resource). Below this, there are two tabs: 'By action' and 'By resource' (which is selected). A table lists a single resource: 'arn:aws:secretsmanager:ap-south-1:182399724378:secret:my-sqlserver-credentials-*' with the action 'Allow: secretsmanager:GetSecretValue'. At the bottom, a note states: 'Lambda obtained this information from the following policy statements:' followed by a bullet point: 'Inline policy lambda_secretmanager_access, statement 0'.

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

The screenshot shows the AWS CloudWatch Log groups interface. On the left, there's a navigation sidebar with options like CloudWatch, Favorites and recents, Dashboards, AI Operations, Alarms, and Logs. Under Logs, Log groups is selected. The main area displays a table titled "Log groups (1/1)". The table has columns for Log group, Log class, Anomaly detection, Data protection, Sensitive data controls, Retention, Metric filters, and Contributor Insights. One row is visible, representing a log group named "/aws/lambda/serverless_function". This row is highlighted with a blue border. The "Log group" column shows a checkbox followed by the path. The "Log class" column shows "Standard". The "Retention" column shows "Never expire". At the top right of the main area, there are buttons for Actions, View in Logs Insights, Start tailing, and Create log group. There's also a search bar and some pagination controls.

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

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

The screenshot shows the AWS Lambda function configuration interface. The top navigation bar includes the AWS logo, search bar, and account information (Account ID: 1823-9972-4378, Region: Asia Pacific (Mumbai), User: Naveen). Below the navigation is a breadcrumb trail: Lambda > Functions > serverless_function. The main menu tabs are Code, Test, Monitor, Configuration (which is selected and highlighted in blue), Aliases, and Versions. On the left, a sidebar lists various configuration sections: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables (which is also selected and highlighted in blue), Tags, VPC, RDS databases, Monitoring and operations tools, Concurrency and recursion detection, Asynchronous invocation, Code signing, File systems, and State machines. The central content area is titled "Environment variables (0)" and contains a search bar with placeholder text "Find environment variables". A table with columns "Key" and "Value" is shown, with the message "No environment variables" and "No environment variables associated with this function." An "Edit" button is located at the bottom right of this section.

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

The screenshot shows the AWS Lambda 'Edit environment variables' interface. At the top, there's a navigation bar with links for IAM, VPC, EC2, Aurora and RDS, S3, EFS, Route 53, CloudFront, CloudWatch, Elastic Container Service, Elastic Kubernetes Service, Elastic Container Registry, Lambda, and API Gateway. Below the navigation bar, the path 'Lambda > Functions > serverless_function > Edit environment variables' is visible. The main section is titled 'Edit environment variables' and contains a table with one row. The table has two columns: 'Key' and 'Value'. The 'Key' column contains the value 'DB_SECRET_NAME' and the 'Value' column contains the value 'my-sqlserver-credentials'. There are 'Add environment variable' and 'Remove' buttons next to the table. At the bottom right, there are 'Cancel' and 'Save' buttons.

| Key | Value |
|----------------|--------------------------|
| DB_SECRET_NAME | my-sqlserver-credentials |

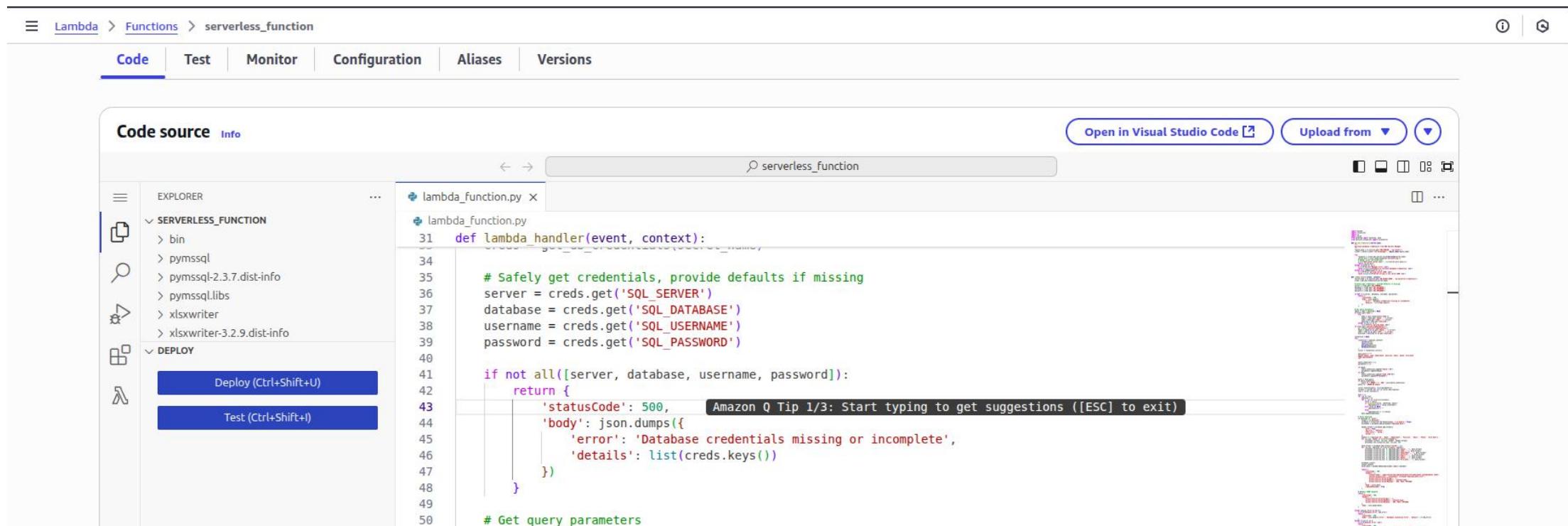
Add environment variable Remove

Encryption configuration

Cancel Save

10. Create your backend script **.py** file with the dependency 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.

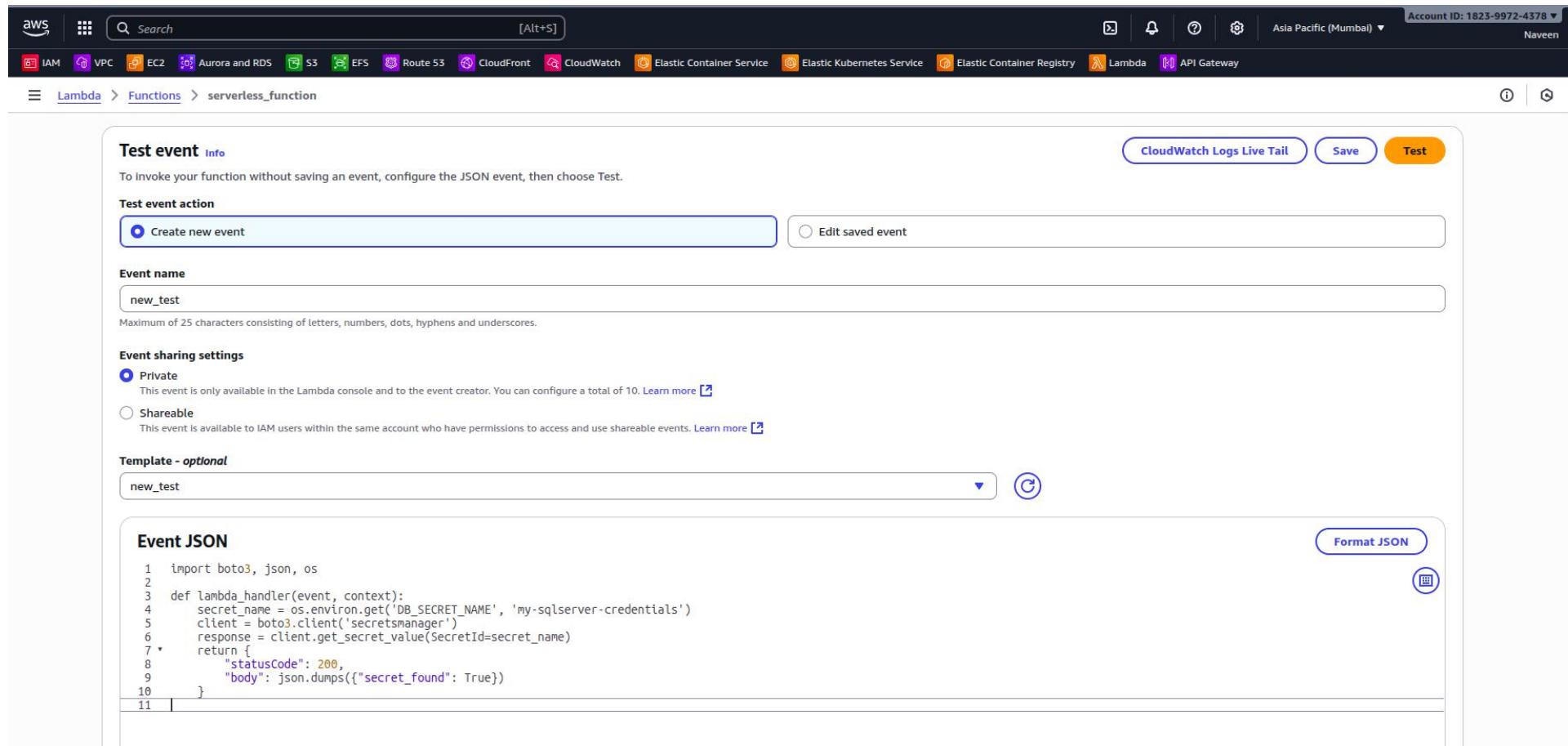


The screenshot shows the AWS Lambda function editor for a function named 'serverless_function'. The 'Code' tab is selected. On the left, the 'EXPLORER' sidebar shows the project structure with 'lambda_function.py' as the active file. The main area displays the Python code for the Lambda function:

```
lambda_function.py
31 def lambda_handler(event, context):
32     # Safely get credentials, provide defaults if missing
33     server = creds.get('SQL_SERVER')
34     database = creds.get('SQL_DATABASE')
35     username = creds.get('SQL_USERNAME')
36     password = creds.get('SQL_PASSWORD')
37
38     if not all([server, database, username, password]):
39         return {
40             'statusCode': 500,
41             'body': json.dumps({
42                 'error': 'Database credentials missing or incomplete',
43                 'details': list(creds.keys())
44             })
45         }
46
47     # Get query parameters
```

At the bottom of the code editor, there is an 'Amazon Q Tip 1/3: Start typing to get suggestions ([ESC] to exit)' message. The 'DEPLOY' section on the left sidebar has two buttons: 'Deploy (Ctrl+Shift+U)' and 'Test (Ctrl+Shift+T)', both of which are highlighted in blue.

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!

The screenshot shows the AWS Lambda Functions console. The top navigation bar includes 'Lambda', 'Functions', and the specific function name 'serverless_function'. Below the navigation is a tab bar with 'Code' (selected), 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The main area is titled 'Code source' with an 'Info' link. On the left is an 'EXPLORER' sidebar showing the project structure: 'SERVERLESS_FUNCTION' contains 'pymssql-2.3.7.dist-info', 'pymssql.libs', 'xlsxwriter', 'xlsxwriter-3.2.9.dist-info', and 'lambda_function.py'. The 'lambda_function.py' file is selected. The 'DEPLOY' section has 'Deploy (Ctrl+Shift+U)' and 'Test (Ctrl+Shift+I)' buttons. The right side displays the 'PROBLEMS', 'OUTPUT', 'CODE REFERENCE LOG', and 'TERMINAL' tabs. The 'OUTPUT' tab is active, showing the status 'Succeeded' and 'Test Event Name: new_test'. A large code block shows the response body of the test event:

```
{
    "statusCode": 200,
    "headers": {
        "Access-Control-Allow-Origin": "*",
        "Access-Control-Allow-Headers": "Content-Type",
        "Access-Control-Allow-Methods": "GET, POST, OPTIONS"
    },
    "body": "[{"empid": "101", "name": "John Doe", "department": "IT", "position": "Developer", "email": "john.doe@example.com", "phone": "123456789", "hire_date": "2025-09-22"}, {"empid": "430363", "name": "Naveen Raaj", "department": "Information Technology", "position": "System Admin", "email": "naveenraaj@company.com", "phone": "22254111", "hire_date": "2024-09-16"}]"
}
```

Below the code block, 'Function Logs:' are displayed, showing the request ID, version, retrieved secret keys, end request ID, report request ID, duration, billed duration, memory size, used memory, and init duration. The 'TEST EVENTS [SELECTED: NEW_TEST]' section shows 'Create new test event' and 'Private saved events' with one entry 'new_test'.

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

The screenshot shows the AWS CloudWatch Log Groups interface. The left sidebar navigation includes CloudWatch, Favorites and recents, Dashboards, AI Operations, Alarms, Logs (with Log groups selected), Metrics, Application Signals (APM), Network Monitoring, and Insights. The main content area displays the log group details for '/aws/lambda/serverless_function'. The details include Log class (Standard), ARN (arn:aws:logs:ap-south-1:182399724378:log-group:/aws/lambda/serverless_function), Creation time (1 day ago), Retention (Never expire), and Stored bytes (20.97 KB). Metrics filters, Subscription filters, Contributor Insights rules, KMS key ID, and Anomaly detection are listed as 0. Data protection, Sensitive data count, and Custom field indexes sections are present with 'Configure' links. Below the details, a tab bar offers Log streams, Tags, Anomaly detection, Metric filters, Subscription filters, Contributor Insights, Data protection, Field indexes, and Transformer. The Log streams section shows 12 entries, each with a checkbox, a log stream name, and its last event time. A search bar at the top of the log streams table allows filtering by prefix.

| Log Stream | Last event time |
|---|---------------------------|
| 2025/09/22/[\$LATEST]3d1920cdc44642c98bbac24a23cd301d | 2025-09-22 13:58:33 (UTC) |
| 2025/09/22/[\$LATEST]bc293d5db7694d6b977c33a0990f4551 | 2025-09-22 13:46:00 (UTC) |
| 2025/09/22/[\$LATEST]b964558b801d4f4fb676ea4e12cc6b7 | 2025-09-22 13:32:47 (UTC) |
| 2025/09/22/[\$LATEST]4a40bb54eb9d4d0aa7dd5c2465af3faa | 2025-09-22 11:13:10 (UTC) |

9.API Gateway (Regional)

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) |

| 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)

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.

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)

Naveen Raaj

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.

The screenshot shows the AWS API Gateway 'Create HTTP API' configuration interface. The left sidebar lists steps: Step 1 (Configure API, currently selected), Step 2 - optional (Configure routes), Step 3 - optional (Define stages), and Step 4 (Review and create). The main area is titled 'Configure API' and contains the 'API details' section. In the 'API name' field, 'GetEmployee_LambdaFunction' is entered. Under 'IP address type', 'IPv4' is selected. The 'Integrations' section shows one entry for 'Lambda'. The integration table has columns for 'AWS Region' (ap-south-1), 'Lambda function' (arn:aws:lambda:ap-south-1:182399724378:function:serverless_function), and 'Version' (2.0). Buttons at the bottom include 'Cancel', 'Review and create', and 'Next'.

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.

The screenshot shows the 'Configure routes - optional' step of creating an API. On the left, a vertical navigation bar lists steps: Step 1 (Configure API), Step 2 - optional (Configure routes, which is selected and highlighted in blue), Step 3 - optional (Define stages), and Step 4 (Review and create). The main area contains a 'Configure routes' section with an 'Info' link. It explains that API Gateway uses routes to expose integrations to consumers. Routes consist of an HTTP method and a resource path. A table shows a single route: Method (GET), Resource path (/employee), and Integration target (serverless_function). Buttons for 'Add route' and 'Remove' are present. At the bottom are 'Cancel', 'Review and create', 'Previous', and a yellow 'Next' button.

The screenshot shows the 'Define stages - optional' step of creating an API. The vertical navigation bar on the left shows Step 1 (Configure API), Step 2 - optional (Configure routes), Step 3 - optional (Define stages, which is selected and highlighted in blue), and Step 4 (Review and create). The main area contains a 'Configure stages' section with an 'Info' link. It explains that stages are independently configurable environments for deployment. A table shows one stage: Stage name (\$default) and Auto-deploy (which is turned on). Buttons for 'Add stage' and 'Remove' are present. At the bottom are 'Cancel', 'Previous', and a yellow 'Next' button.

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

Naveen Raaj

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

The screenshot shows the AWS API Gateway CORS configuration page for an API named 'GetEmployee_LambdaFunction'. The left sidebar has sections for API Gateway, APIs, and Develop (with CORS selected). The main area is titled 'Cross-Origin Resource Sharing' and contains fields for configuring CORS. The 'Access-Control-Allow-Origin' field contains 'https://d3pqw007u4mrql.cloudfront.net/'. The 'Access-Control-Allow-Headers' field contains 'content-type', 'x-amz-date', 'authorization', 'x-api-key', and 'x-amz-security-token'. The 'Access-Control-Allow-Methods' dropdown shows 'GET', 'POST', and 'OPTIONS'. The 'Access-Control-Expose-Headers' field contains 'content-type', 'x-amz-date', 'authorization', 'x-api-key', and 'x-amz-security-token'. The 'Access-Control-Max-Age' field is set to '3600'. The 'Access-Control-Allow-Credentials' toggle is set to 'NO'.

API Gateway > APIs > GetEmployee_LambdaFunction (kju22b30m0) > CORS

Cross-Origin Resource Sharing

Configure CORS Info

CORS allows resources from different domains to be loaded by browsers. If you configure CORS for an API, API Gateway ignores CORS headers returned from your backend integration. See our [CORS documentation](#) for more details.

Access-Control-Allow-Origin

Enter a value for Allowed Origins

<https://d3pqw007u4mrql.cloudfront.net/> [X](#)

[Add](#)

Access-Control-Allow-Headers

Enter a value for Allowed Headers

content-type [X](#) x-amz-date [X](#) authorization [X](#)
x-api-key [X](#) x-amz-security-token [X](#)

[Add](#)

Access-Control-Allow-Methods

Choose Allowed Methods

GET [X](#) POST [X](#) OPTIONS [X](#)

Access-Control-Expose-Headers

Enter a value for Exposed Headers

content-type [X](#) x-amz-date [X](#) authorization [X](#)
x-api-key [X](#) x-amz-security-token [X](#)

[Add](#)

Access-Control-Max-Age

3600

Access-Control-Allow-Credentials

NO

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.

The screenshot shows the AWS Lambda Functions overview for a function named "serverless_function".

Function overview:

- Diagram:** Shows the function structure with "serverless_function" at the top and "Layers" below it.
- Destinations:** Shows an "API Gateway" destination with a "Add destination" button.
- Triggers:** Shows a "+ Add trigger" button.
- Actions:** Buttons for "Throttle", "Copy ARN", and "Actions".
- Export:** Buttons for "Export to Infrastructure Composer" and "Download".

Description:

- Last modified: 2 hours ago
- Function ARN: arn:aws:lambda:ap-south-1:182399724378:function:serverless_function
- Function URL: [Info]

Resource-based policy statements (1)

Resource-based policies grant other AWS accounts and services permissions to access your Lambda resources.

| Statement ID | Principal | PrincipalOrgID | Conditions | Action |
|------------------------------|--------------------------|----------------|------------|-----------------------|
| 952b7136-708a-5b54-bd37-0... | apigateway.amazonaws.com | - | ArnLike | lambda:InvokeFunction |

Auditing and compliance:

AWS CloudTrail can log this function's invocations for operational and risk auditing, governance, and compliance. [Get started](#) on the CloudTrail console.

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

The screenshot shows the AWS API Gateway interface. The top navigation bar includes links for IAM, VPC, EC2, Aurora and RDS, S3, EFS, Route 53, CloudFront, CloudWatch, Elastic Container Service, Elastic Kubernetes Service, Elastic Container Registry, Lambda, and API Gateway. The main navigation path is API Gateway > APIs > GetEmployee_LambdaFunction (kju22b30m0) > Stages. On the left, a sidebar menu is open under the Deploy section, specifically the Stages subsection. The main content area displays the 'Stages' section for the 'GetEmployee_LambdaFunction' API. A green success message at the top states 'Successfully created API GetEmployee_LambdaFunction (kju22b30m0.)'. The 'Stages for GetEmployee_LambdaFunction' section contains a 'Create' button and a search bar. Below it, the '\$default' stage is selected. The 'Stage details' section provides information about the stage: Name (\$default), Created (September 23, 2025 12:32 PM), Last updated (September 23, 2025 12:34 PM), and Invoke URL (<https://kju22b30m0.execute-api.ap-south-1.amazonaws.com>). It also shows the Attached deployment status (Automatic Deployment, Enabled), Deployment ID (5s29f9), Deployment created (September 23, 2025 12:34 PM), and Deployment description (Automatic deployment triggered by changes to the Api configuration). The Stage variables section is currently empty, showing 'No Stage Variables'.

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



```
[{"empid": "101", "name": "John Doe", "department": "IT", "position": "Developer", "email": "john.doe@example.com", "phone": "123456789", "hire_date": "2025-09-22"}]
```

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

```
{"message": "Not Found"}naveen@[REDACTED]:~$ curl -X GET https://kju22b30m0.execute-api.ap-south-1.amazonaws.com/employee
[{"empid": "101", "name": "John Doe", "department": "IT", "position": "Developer", "email": "john.doe@example.com", "phone": "123456789", "hire_date": "2025-09-22"}, {"empid": "430", "name": "Naveen Raaj", "department": "Information Technology", "position": "System Admin", "email": "naveenraaj@company.com", "phone": "22254111", "hire_date": "2024-09-16"}]naveen@[REDACTED]:~$
```

(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.

```
naveen@[REDACTED]:~$ curl -X GET "https://kju22b30m0.execute-api.ap-south-1.amazonaws.com/employee?download=excel" -o employee_data.xlsx
% Total    % Received % Xferd  Average Speed   Time     Time      Current
          Dload  Upload   Total Spent  Left  Speed
100  5744  100  5744    0     0  1217      0  0:00:04  0:00:04  --:--:--  1798
naveen@[REDACTED]:~$ ls
aws      Documents  employee_data.xlsx  lambda_build  my-aws-portfolio  new-project  nginx-php.zip  Public  SQLserver  Videos
Desktop  Downloads  jenkins           Music        my-nginx-app    nginx-php    Pictures    snap    Templates
```

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.

The screenshot shows the 'Create bucket' wizard in the AWS S3 console. The 'General configuration' section is active, displaying the following details:

- AWS Region:** Asia Pacific (Mumbai) ap-south-1
- Bucket type:** General purpose (selected)
- Bucket name:** cbe-serverless-bucket
- Copy settings from existing bucket - optional:** Choose bucket
- Object Ownership:** ACLs disabled (recommended)
- Block Public Access settings for this bucket:** (checkbox)

The 'Object Ownership' section includes a note: "Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects."

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.)

The screenshot shows the 'Create bucket' wizard in the Amazon S3 console. The current step is 'Block Public Access settings for this bucket'. The 'Block all public access' checkbox is checked. Below it, four sub-options are listed, each with a checkbox and a description:

- Block public access to buckets and objects granted through new access control lists (ACLs)**
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- Block public access to buckets and objects granted through any access control lists (ACLs)**
S3 will ignore all ACLs that grant public access to buckets and objects.
- Block public access to buckets and objects granted through new public bucket or access point policies**
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- Block public and cross-account access to buckets and objects through any public bucket or access point policies**
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Below this section is the 'Bucket Versioning' section, which includes a sub-section for 'Bucket Versioning' with a radio button set to 'Disable'. The 'Tags - optional (0)' section shows no tags associated with the bucket and includes an 'Add new tag' button and a note about adding up to 50 tags.

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

Default encryption [Info](#)

Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption type [Info](#)

Secure your objects with two separate layers of encryption. For details on pricing, see DSSE-KMS pricing on the Storage tab of the [Amazon S3 pricing page](#).

Server-side encryption with Amazon S3 managed keys (SSE-S3)

Server-side encryption with AWS Key Management Service keys (SSE-KMS)

Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)

Bucket Key

Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. [Learn more](#)

Disable

Enable

▼ Advanced settings

Object Lock

Store objects using a write-once-read-many (WORM) model to help you prevent objects from being deleted or overwritten for a fixed amount of time or indefinitely. Object Lock works only in versioned buckets. [Learn more](#)

Disable

Enable

Permanently allows objects in this bucket to be locked. Additional Object Lock configuration is required in bucket details after bucket creation to protect objects in this bucket from being deleted or overwritten.

ⓘ Object Lock works only in versioned buckets. Enabling Object Lock automatically enables Versioning.

ⓘ After creating the bucket, you can upload files and folders to the bucket, and configure additional bucket settings.

[Cancel](#) [Create bucket](#)

4. Now upload the frontend web files **index.html** and **script.js file** and click **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 SDKs 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 (2 total, 13.6 KB)

All files and folders in this table will be uploaded.

| <input type="checkbox"/> Name | Folder | Type | Size |
|-------------------------------------|--------|-----------------|--------|
| <input type="checkbox"/> index.html | - | text/html | 6.8 KB |
| <input type="checkbox"/> script.js | - | text/javascript | 6.9 KB |

Destination Info

Destination
<s3://serverless-bucket-cbe>

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

▶ **Permissions**
Grant public access and access to other AWS accounts.

▶ **Properties**
Specify storage class, encryption settings, tags, and more.

Cancel **Upload**

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

CloudFront > Distributions > Create distribution

Get started

Step 1 Get started

Step 2 Specify origin

Step 3 Enable security

Step 4 Get TLS certificate

Step 5 Review and create

Distribution options Info

Distribution name
Name will be stored as a tag on the resource. You can add a name, or more tags, later.

Description - optional

Distribution type

Single website or app
Choose if each website or application will have a unique configuration.

Multi-tenant architecture - New
Choose when you have multiple domains that need to share configurations. This is a common architecture for SaaS providers.

Custom domain Info

Domain - optional
Use your own custom domain with free HTTPS to provide a secure, friendly URL for your app. You can add a custom domain later if you do not have a Route 53 zone in this account.

▶ Tags - optional

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

The screenshot shows the 'Specify origin' step in the CloudFront distribution creation wizard. The left sidebar lists steps: 'Get started' (selected), 'Specify origin' (highlighted with a blue circle), 'Enable security', 'Step 4', and 'Review and create'. The main area is titled 'Specify origin' and 'Origin type'. It lists six options: 'Amazon S3' (selected), 'Elastic Load Balancer', 'API Gateway', 'Elemental MediaPackage', 'VPC origin', and 'Other'. Below this is the 'Origin' section for 'S3 origin', where the URL 'cbe-serverless-bucket.s3.ap-south-1.amazonaws.com' is entered. A warning message says: '⚠ This S3 bucket has static web hosting enabled. If you plan to use this distribution as a website, we recommend using the S3 website endpoint rather than the bucket endpoint.' A 'Use website endpoint' button is shown. The 'Origin path - optional' field contains '/path'. The 'Settings' section notes: 'CloudFront provides default origin and cache settings based on what origin you selected. View default settings for S3'.

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

The screenshot shows the AWS CloudFront configuration wizard. In the first section, 'S3 origin', a private S3 bucket named 'cbe-serverless-buckets.s3.ap-south-1.amazonaws.com' is selected as the origin. A warning message indicates that static web hosting is enabled on the S3 bucket and recommends using the S3 website endpoint instead of the bucket endpoint. In the 'Origin path - optional' field, '/path' is specified. The second section, 'Settings', includes 'Allow private S3 bucket access to CloudFront' (which is checked), 'Origin settings' (using recommended settings), and 'Cache settings' (using recommended settings). Navigation buttons at the bottom include 'Cancel', 'Previous', and 'Next'.

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

Step 1
Get started

Step 2
Specify origin

Step 3
Enable security

Step 4
Review and create

Enable security

Web Application Firewall (WAF) Info

Enable security protections
Keep your application secure from the most common web threats and security vulnerabilities using AWS WAF. Blocked requests are stopped before they reach your web servers.

Do not enable security protections
Select this option if your application does not need security protections from AWS WAF.

Use monitor mode
Count how many of your requests would be blocked by this WAF configuration. When ready, you can disable monitor mode to begin blocking requests.

Included security protections

- Protect against the most common vulnerabilities found in web applications.
- Protect against malicious actors discovering application vulnerabilities.
- Block IP addresses from potential threats based on Amazon internal threat intelligence

Price estimate

This AWS WAF configuration is estimated to cost \$14 for 10 million requests/month

| Number of requests/month | Estimated cost |
|--------------------------|----------------|
| 100000 | \$8.06/month |

View AWS WAF pricing [\[?\]](#) for more details.

[Cancel](#) [Previous](#) [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**

The screenshot shows the AWS CloudFront distribution settings for a distribution named 'serverless_cdn'. The 'General' tab is selected. A green success message at the top states 'Successfully updated distribution settings.' The distribution domain name is listed as 'd3pqw007u4mrqi.cloudfront.net'. The ARN is 'arn:aws:cloudfront::182399724378:distribution/E3PJU556CRL6CT'. The last modified date is 'September 20, 2025 at 8:42:03 AM UTC'. In the 'Settings' section, the 'Default root object' is set to 'index.html'. The 'Edit' button is visible in the top right corner of the settings area.

Successfully updated distribution settings.

serverless_cdn Standard

View metrics

General Security Origins Behaviors Error pages Invalidations Tags Logging

Details

Name: serverless_cdn

Distribution domain name: d3pqw007u4mrqi.cloudfront.net

ARN: arn:aws:cloudfront::182399724378:distribution/E3PJU556CRL6CT

Last modified: September 20, 2025 at 8:42:03 AM UTC

Settings

Description: -

Alternate domain names: Add domain

Standard logging: Off

Cookie logging: Off

Default root object: index.html

Continuous deployment Info

Create staging distribution

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)

The screenshot shows the 'Edit origin' page in the AWS CloudFront console. The left sidebar includes sections for CloudFront, Distributions, Policies, Functions, Static IPs, VPC origins, SaaS, Telemetry, Reports & analytics, Security, and Key management. The main 'Edit origin' page has a 'Settings' section with an 'Origin domain' input field containing 'cbe-serverless-bucket.s3.ap-south-1.amazonaws.com'. A note below it says, 'This S3 bucket has static web hosting enabled. If you plan to use this distribution as a website, we recommend using the S3 website endpoint rather than the bucket endpoint.' There is a 'Use website endpoint' button. The 'Origin path - optional' section has a placeholder 'Enter the origin path'. The 'Name' section contains the value 'cbe-serverless-bucket.s3.ap-south-1.amazonaws.com-mfs0a2l72oi'. The 'Origin access' section shows 'Origin access control settings (recommended)' selected, with a note that 'Bucket can restrict access to only CloudFront.' It also lists 'Public' and 'Legacy access identities' options. The 'Origin access control' section shows an existing policy 'oac-cbe-serverless-bucket.s3.ap-south-1.amazonaws.co-mfs0bymg6xn' and a 'Create new OAC' button. A note at the bottom says, 'You must allow access to CloudFront using this policy statement. Learn more about giving CloudFront permission to access the S3 bucket.' A 'Copy policy' button is also present.

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

Bucket policy

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. [Learn more](#)

ⓘ Public access is blocked because Block Public Access settings are turned on for this bucket
To determine which settings are turned on, check your Block Public Access settings for this bucket. Learn more about [using Amazon S3 Block Public Access](#)

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "Statement1",  
      "Effect": "Allow",  
      "Principal": "*",  
      "Action": "s3:GetObject",  
      "Resource": "arn:aws:s3:::cbe-serverless-bucket/*"  
    },  
    {  
      "Sid": "AllowCloudFrontServicePrincipal",  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "cloudfront.amazonaws.com"  
      },  
      "Action": "s3:GetObject",  
      "Resource": "arn:aws:s3:::cbe-serverless-bucket/*",  
      "Condition": {  
        "ArnLike": {  
          "AWSRegion": "us-east-1"  
        }  
      }  
    }  
  ]  
}
```

[Copy](#)

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**

☰ CloudFront > Distributions > E2CZL8RTGS227E > Edit origin

Edit origin

Settings

Origin domain
Choose an AWS origin, or enter your origin's domain name. [Learn more](#)

 X

Enter a valid DNS domain name, such as an S3 bucket, HTTP server, or VPC origin ID.

Protocol | [Info](#)

HTTP only
 HTTPS only
 Match viewer

HTTPS port
Enter your origin's HTTPS port. The default is port 443.
443

Minimum Origin SSL protocol
The minimum SSL protocol that CloudFront uses with the origin.

TLSv1.2
 TLSv1.1
 TLSv1
 SSLv3

Origin path - optional
Enter a URL path to append to the origin domain name for origin requests.
/employee

Name
Enter a name for this origin.
idrdtg8j85.execute-api.ap-south-1.amazonaws.com

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

The screenshot displays two instances of the 'Create behavior' configuration interface from the AWS CloudFront console. Both instances have identical settings, except for the 'Path pattern' field which is highlighted in red.

Path pattern: /api/*

Origin: idrdtg8j85.execute-api.ap-south-1.amazonaws.com

Compress objects automatically: Yes

Viewer:

- Viewer protocol policy:** Redirect HTTP to HTTPS
- Allowed HTTP methods:** GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE
- Cache HTTP methods:** GET and HEAD methods are cached by default. Options: OPTIONS (unchecked)
- Allow gRPC requests over HTTP/2:** Enable (unchecked)

Restrict viewer access: No

Cache key and origin requests:

- Cache policy and origin request policy (recommended):** Cache policy: CachingDisabled (selected), Origin request policy: AllViewerExceptHostHeader (selected)
- Legacy cache settings:** None selected

Cache policy: CachingDisabled (selected). Recommended for API Gateway ▾

Origin request policy - optional: AllViewerExceptHostHeader (selected). Recommended for API Gateway ▾

Response headers policy - optional: Select response headers (dropdown menu open). ▼

Additional settings: None visible.

14. Now the cloudfont 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)

The screenshot shows the AWS CloudFront Origins configuration page for a distribution named "serverless_cdn". The "Origins" tab is selected. There are two origins listed:

| Origin name | Origin path | Origin type | Origin Shield region | Origin access |
|---------------------------------------|---------------|-------------|----------------------|----------------|
| idrdtg8j85.execute-api.ap-south-1.am | /employee | API Gateway | - | - |
| serverless-bucket-cbe.s3.ap-south-1.a | serverless... | S3 | - | E2YIT7LQ5YNE2Z |

The screenshot shows the AWS CloudFront Behaviors configuration page for the same distribution. The "Behaviors" tab is selected. There are two behaviors listed:

| Preced... | Path pattern | Origin or origin group | Viewer protocol policy | Cache policy name | Origin request policy name | Response headers policy na... |
|-----------|--------------|---------------------------------|------------------------|--------------------------|--------------------------------|-------------------------------|
| 0 | /api/* | idrdtg8j85.execute-api.ap-so... | Redirect HTTP to HTTPS | Managed-CachingDisabled | Managed-AllViewerExceptHostHea | - |
| 1 | Default (*) | serverless-bucket-cbe.s3.ap... | Redirect HTTP to HTTPS | Managed-CachingOptimized | - | - |

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.

The screenshot shows the 'Create invalidation' page in the AWS CloudFront console. The URL in the browser is: CloudFront > Distributions > EMCCFTI9GN4KR > Create invalidation. The main section is titled 'Create invalidation' and contains a 'Object paths' input field. Inside the field, the character '/' is typed. Below the input field is a note: 'To add object paths individually, use the standard editor.' At the bottom right are 'Cancel' and 'Create invalidation' buttons.

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

The screenshot shows the AWS CloudFront distribution settings page for a distribution named 'serverless_cdn'. The 'General' tab is selected. A green success message at the top states 'Successfully updated distribution settings.' Below the tabs, there are two main sections: 'Details' and 'Settings'. In the 'Details' section, the distribution's ARN is listed as 'arn:aws:cloudfront::182399724378:distribution/E3PJU556CRL6CT'. In the 'Settings' section, under 'Alternate domain names', there is a blue 'Add domain' button. Other settings include 'Standard logging Off', 'Cookie logging Off', and 'Default root object index.html'. At the bottom, there is a 'Continuous deployment' section with a 'Create staging distribution' button.

Successfully updated distribution settings.

serverless_cdn Standard

View metrics

General Security Origins Behaviors Error pages Invalidations Tags Logging

Details

Name serverless_cdn

Distribution domain name d3pqw007u4mrqi.cloudfront.net

ARN arn:aws:cloudfront::182399724378:distribution/E3PJU556CRL6CT

Last modified September 20, 2025 at 8:42:03 AM UTC

Settings

Description -

Price class Use all edge locations (best performance)

Supported HTTP versions HTTP/2, HTTP/1.1, HTTP/1.0

Alternate domain names

Add domain

Standard logging Off

Cookie logging Off

Default root object index.html

Continuous deployment Info

Create staging distribution

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

Step 1

Configure domains

Step 2

Get TLS certificate

Step 3

Review changes

Configure domains

Choose the domains that will be served by this distribution.

Domains

Domains to serve

CloudFront will serve the following domains. They must all be under the same root domain.

serverless.com

Add another domain

Remove

Cancel

Next

Step 1

Configure domains

Step 2

Get TLS certificate

Step 3

Review changes

Get TLS certificate

TLS certificate Info

Transport layer security (TLS) encrypts communication to and from your domain. You must have a TLS certificate with AWS Certificate Manager (ACM) to use CloudFront.

⚠ We couldn't find an ACM certificate in the us-east-1 Region that covers the domains you specified. To continue, create a new certificate.

CloudFront can automatically create a certificate for the following domains:

- serverless.com

Create a wildcard certificate
This certificate will be valid for *.serverless.com

Create certificate

Create certificate in ACM

Cancel

Previous

Next

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.

The screenshot shows the 'Get TLS certificate' step in the CloudFront configuration process. The page title is 'TLS certificate' with an 'Info' link. A note states: 'Transport layer security (TLS) encrypts communication to and from your domain. You must have a TLS certificate with AWS Certificate Manager (ACM) to use CloudFront.' A warning message in a yellow box says: '⚠ We couldn't find an ACM certificate in the us-east-1 Region that covers the domains you specified. To continue, create a new certificate.' Below this, it says: 'CloudFront can automatically create a certificate for the following domains:' followed by a list: 'serverless.com'. There is an unchecked checkbox for 'Create a wildcard certificate' which would cover *.serverless.com. Two buttons are present: 'Create certificate' (disabled) and 'Create certificate in ACM' (highlighted). The 'Creating certificate' section shows a green checkmark for 'Provisioned certificate in AWS Certificate Manager' and a grey circle for 'Waiting for Certificate Manager to validate the certificate'. A blue box contains a note: 'ⓘ Domain update required' with the subtext: 'Before we can validate your certificate, you must update domain records with your DNS provider. When you're done, choose Validate certificate.' Below this, fields for 'Record type' (set to CNAME), 'Name' (containing a placeholder for a DNS record), and 'Value' (containing a placeholder for a DNS value) are shown. At the bottom is a 'Validate certificate' button.

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

Naveen Raaj

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

Certificates (1)

| Certificate ID | Domain name | Type | Status | In use | Renewal eligibility | Key algorithm |
|--|----------------|---------------|--------------------|--------|---------------------|---------------|
| c96709bd-c27a-4e64-978a-813f97626686 | serverless.com | Amazon Issued | Pending validation | No | Ineligible | RSA 2048 |

c96709bd-c27a-4e64-978a-813f97626686

Certificate status

| | |
|---|--|
| Identifier c96709bd-c27a-4e64-978a-813f97626686 | Status Pending validation Info |
| ARN arn:aws:acm:us-east-1:182399724378:certificate/c96709bd-c27a-4e64-978a-813f97626686 | |
| Type Amazon Issued | |

Domains (1)

| Domain | Status | Renewal status | Type | CNAME name | CNAME value |
|----------------|--------------------|----------------|-------|--|---|
| serverless.com | Pending validation | - | CNAME | _ae6f8ef13d696bc631b9f29fc2706bd8.serverless.com | _b0bce77787589bf2885f937041validations.aws. |

Details

| | | | |
|--|--|---|--|
| In use No | Serial number N/A | Requested at October 04, 2025, 19:53:47 (UTC+05:30) | Renewal eligibility Ineligible |
| Domain name serverless.com | Public key info RSA 2048 | Issued at N/A | Export option Disabled |
| Number of additional names 0 | Signature algorithm SHA-256 with RSA | Not before N/A | |

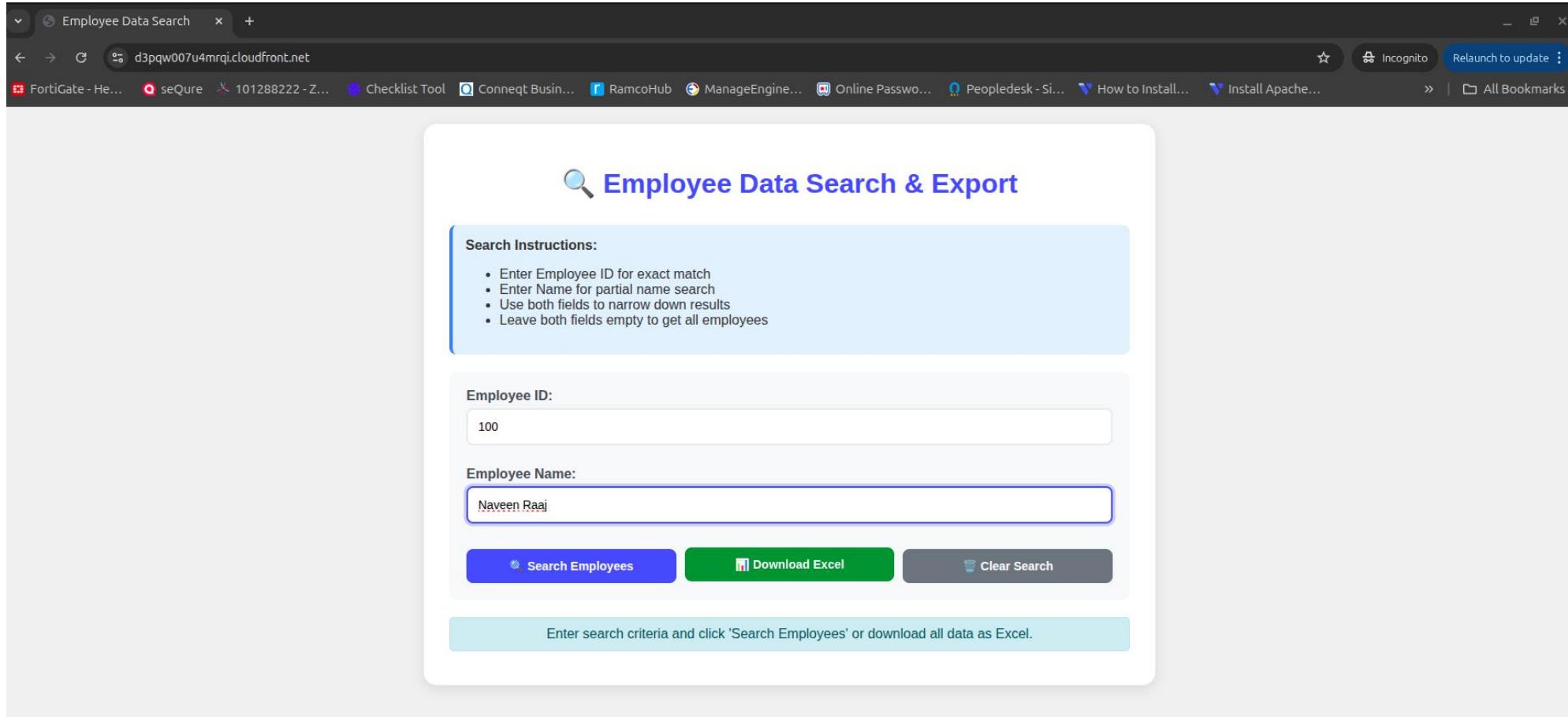
20. If you already have a SSL/TLS certificate for your domain then you can import the cert private key and cert chain over here.

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

The screenshot shows the 'Import certificate' page in the AWS Certificate Manager. The top navigation bar includes 'AWS Certificate Manager > Certificates > Import certificate'. The main section is titled 'Import certificate' and contains three input fields: 'Certificate details' (with 'Info' link), 'Certificate body' (with placeholder 'Paste the PEM-encoded certificate body below.'), 'Certificate private key' (with placeholder 'Paste the PEM-encoded certificate private key below.'), and 'Certificate chain - optional' (with 'Info' link and placeholder 'Paste the PEM-encoded certificate chain below.'). Below these fields is a 'Tags' section with 'Info' link, stating 'No tags associated with the resource.' and an 'Add new tag' button. At the bottom right are 'Cancel' and 'Import certificate' buttons, with the latter being orange.

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.

The screenshot shows a file explorer interface with a sidebar on the left containing 'Recent', 'Starred', 'Home', and 'Desktop' sections. A file named 'employee_data.xlsx' is selected, showing its icon and name. The main area displays the contents of the Excel spreadsheet. The spreadsheet has a header row with columns labeled 'Employee ID', 'Name', 'Department', 'Position', 'Email', 'Phone', and 'Hire Date'. The first data row shows values: Employee ID 100, Name Naveen Raaj, Department Information Technology, Position System Admin, Email naveenraaj@company.com, Phone 22254111, and Hire Date 2024-09-16. The rest of the rows are blank.

| | Employee ID | Name | Department | Position | Email | Phone | Hire Date |
|---|-------------|-------------|------------------------|--------------|------------------------|----------|------------|
| 1 | 100 | Naveen Raaj | Information Technology | System Admin | naveenraaj@company.com | 22254111 | 2024-09-16 |
| 2 | | | | | | | |
| 3 | | | | | | | |
| 4 | | | | | | | |
| 5 | | | | | | | |
| 6 | | | | | | | |
| 7 | | | | | | | |

Thank You