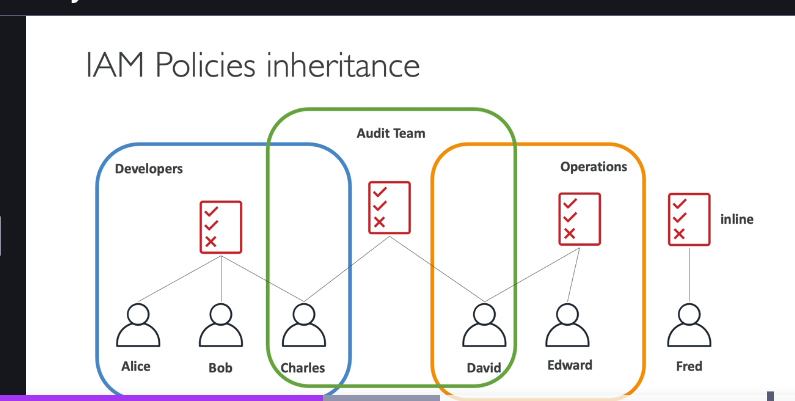
IAM

Policies:  


{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"apigateway:\*"

],

"Resource": "arn:aws:apigateway:\*::/\*"

}

]

}

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Deny",

"Action": [

"secretsmanager:GetSecretValue",

"dynamodb:PutItem",

"dynamodb:GetItem"

],

"Resource": "\*"

}

]

}

First we need to download aws cli and install on windows

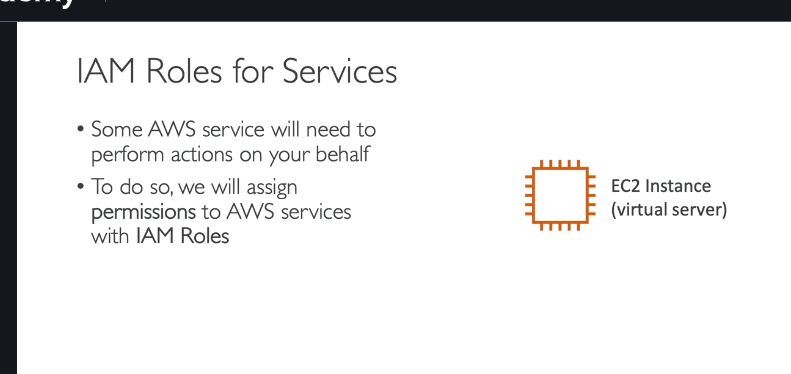
C:\Users\RAMU>aws configure

AWS Access Key ID [\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Q56A]: AKIAVK2WJ7L

AWS Secret Access Key [\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MRvu]: iny8fhnV/MIif

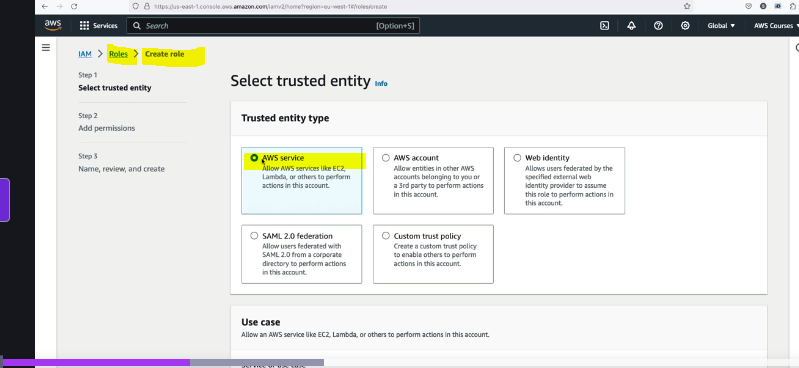
Default region name [us-east-1]:

Default output format [json]:

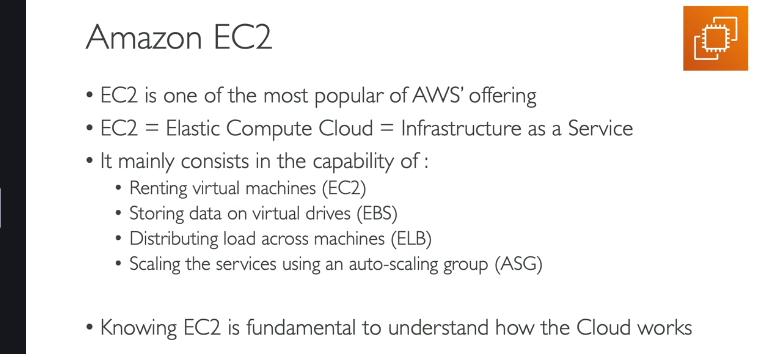


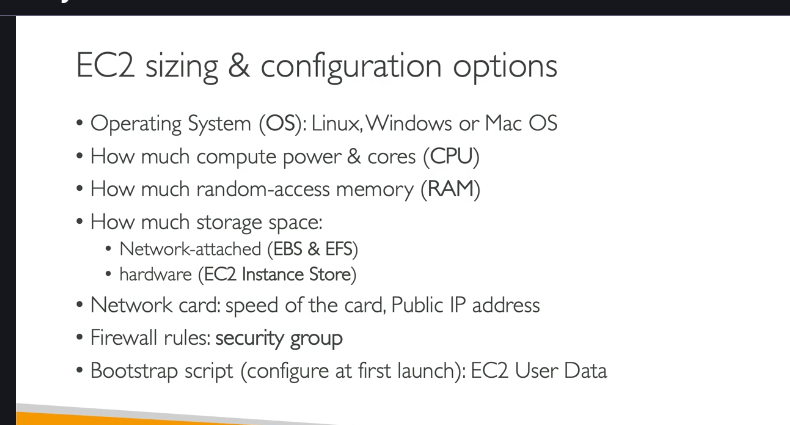
How to create a role for ec2 instance to access. After creating this role we attach it to ec2.

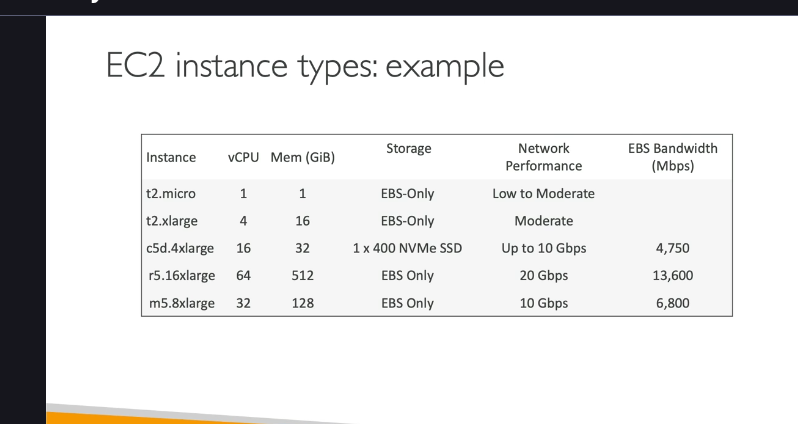
For a role we attach a policy.



**EC2: Elastic compute cloud : Infrastructure as service**







**While creating Ec2 we can provide user data, what is the use of this?**

while creating a ec2 instance we can configure user data what is the use waht data we can provide?

**ChatGPT said:**

Great question!

**✅ What is EC2 User Data?**

**User data** is a feature in AWS EC2 that allows you to **automatically run scripts** or pass configuration commands **when an EC2 instance is launched for the first time**.

It's like bootstrapping or auto-configuring your instance.

**🎯 Use Cases**

You can use **user data** to:

| **Purpose** | **Example** |
| --- | --- |
| 🛠 Install software | Apache, NGINX, Docker, Java, etc. |
| ⚙️ Configure settings | Set environment variables, create files |
| 🔐 Set up SSH keys | Add SSH public keys |
| 💻 Create users or cron jobs | Add a new user, schedule tasks |
| 📂 Mount volumes | Auto-mount EBS volumes |
| 🧪 Run startup logic | Start apps, send logs, etc. |

**📝 Types of Data You Can Provide**

You typically provide **shell scripts** or **cloud-init directives**.

**🟢 Example 1: Shell Script (Amazon Linux/Ubuntu)**

bash

CopyEdit

#!/bin/bash

yum update -y

yum install -y httpd

systemctl start httpd

systemctl enable httpd

echo "Welcome to EC2 instance!" > /var/www/html/index.html

**🟣 Example 2: Cloud-Init YAML (more structured)**

yaml

CopyEdit

#cloud-config

packages:

- httpd

runcmd:

- systemctl start httpd

- systemctl enable httpd

- echo "Deployed using cloud-init" > /var/www/html/index.html

**📌 Where Do You Enter User Data?**

You can provide user data:

* In the EC2 **Launch wizard** under “Advanced Details”
* Via **CLI** using --user-data file://my-script.sh
* Through **CloudFormation** using UserData section in your template

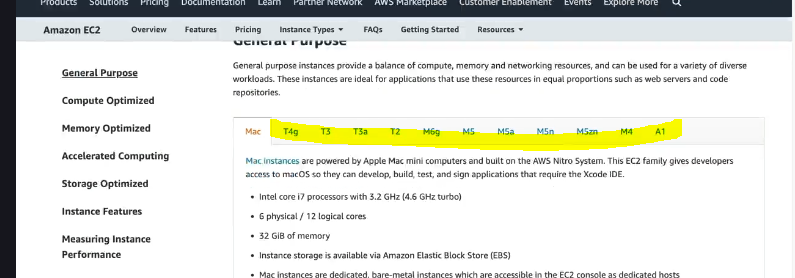
**🔄 One-Time Use Only**

* User data scripts run **only on first boot** by default.
* To make it re-run on every reboot, you must customize it (e.g., modify cloud-init settings or use cron).

**🧪 How to Test?**

1. Launch an EC2 instance and add the user data.
2. After boot, check the results:

sudo cat /var/log/cloud-init-output.log



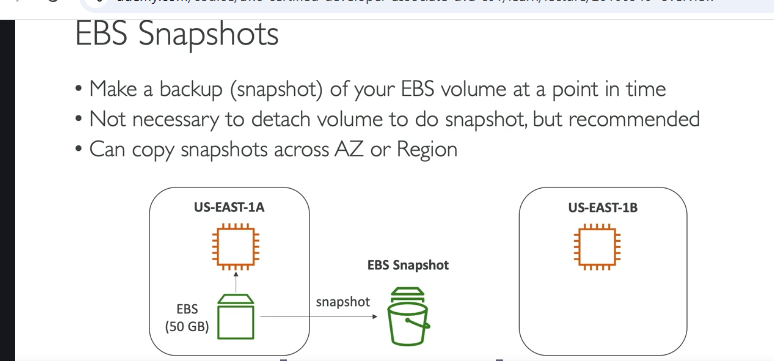
**Can we assign role ec2 ?**

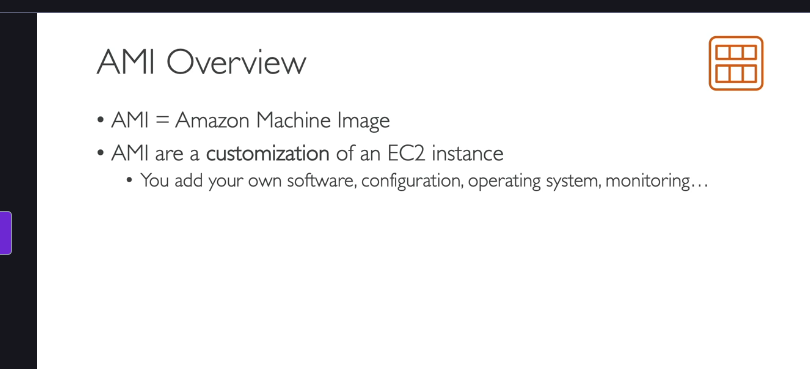
**Yes**

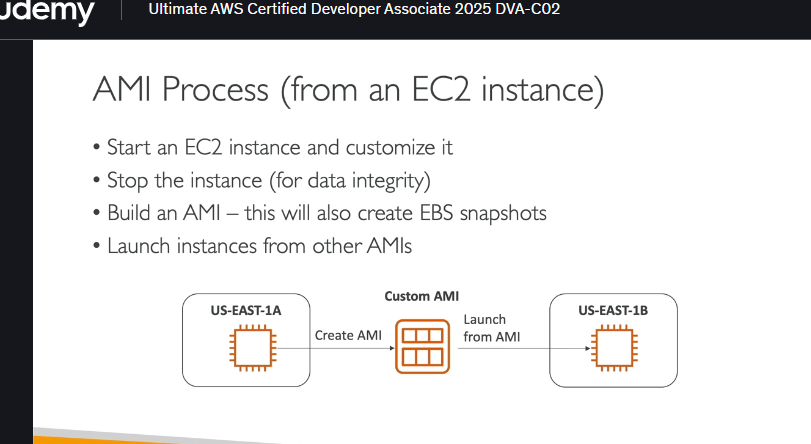
**…………………………………………………………………………………………………**

**EBS:**

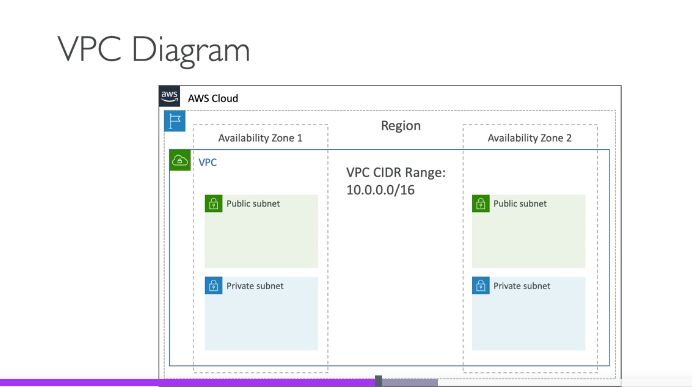
**EBS snapcahat:**

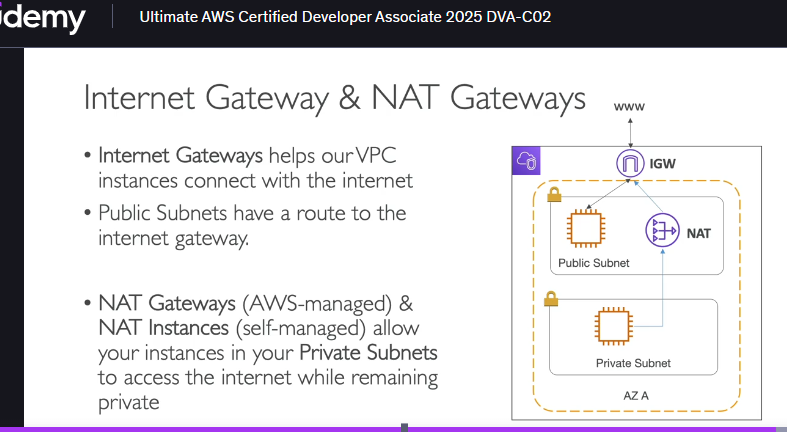


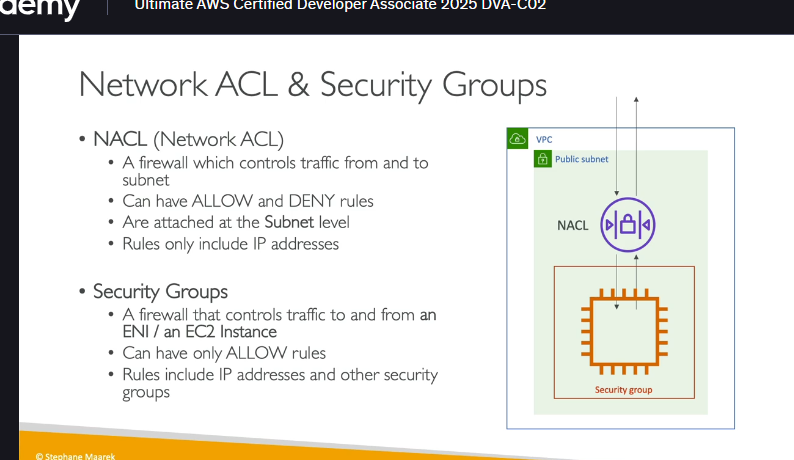


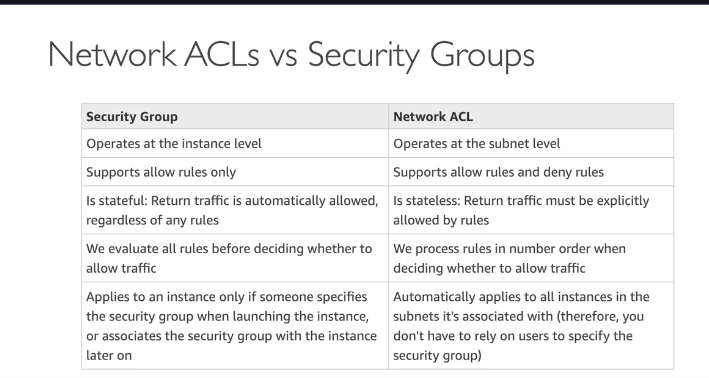


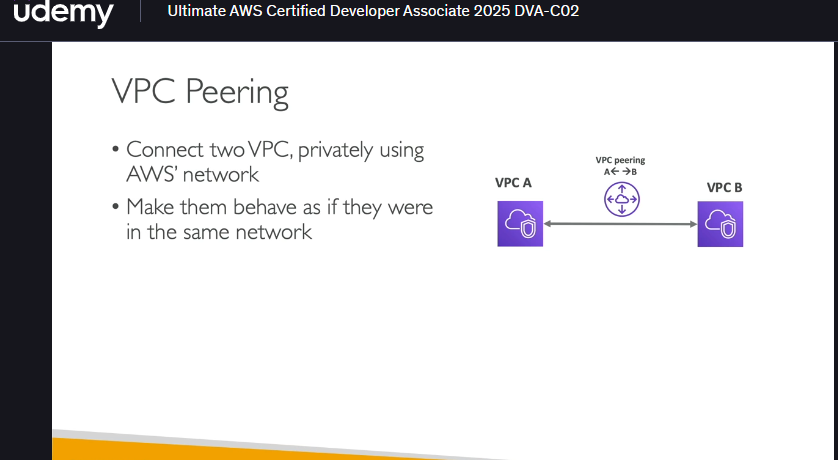
**VPC:**

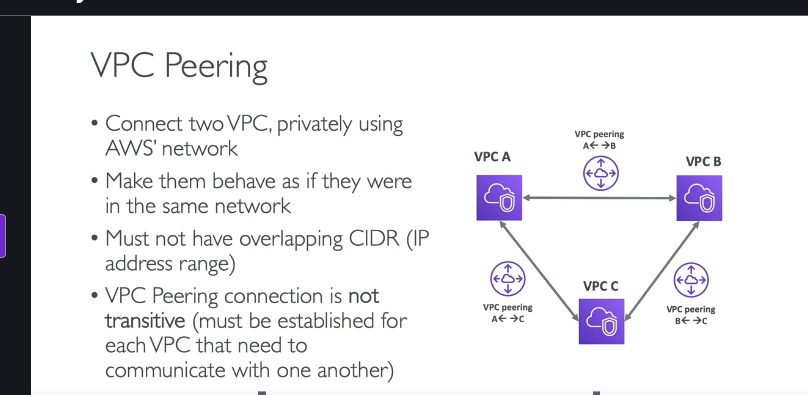


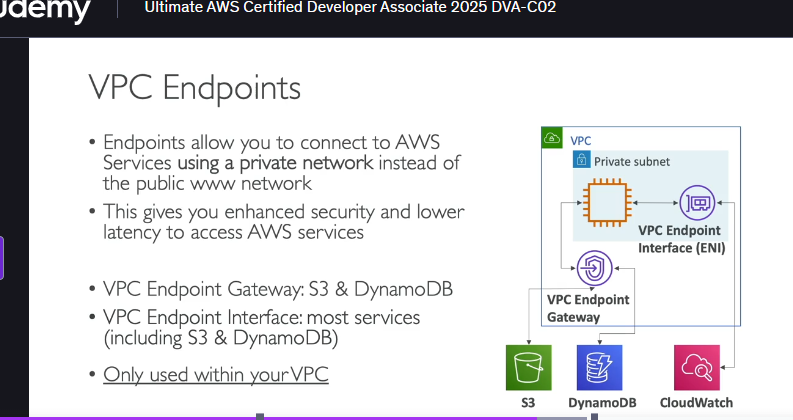


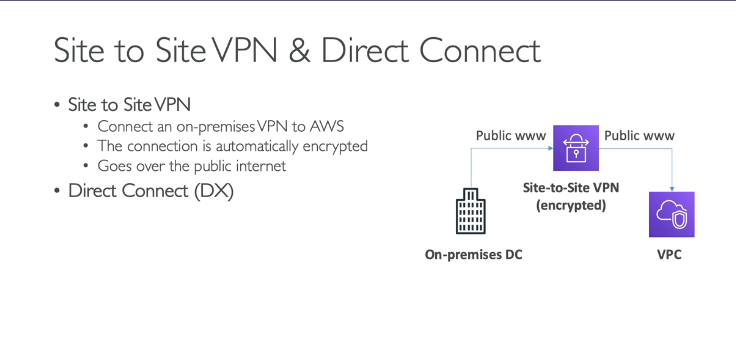


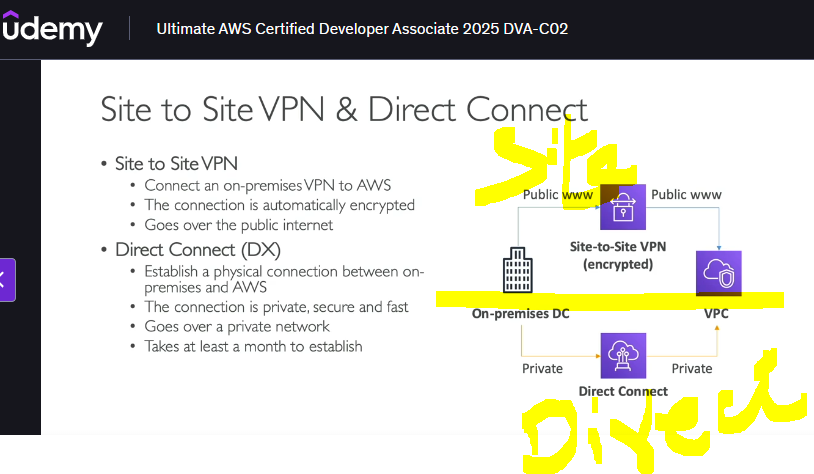


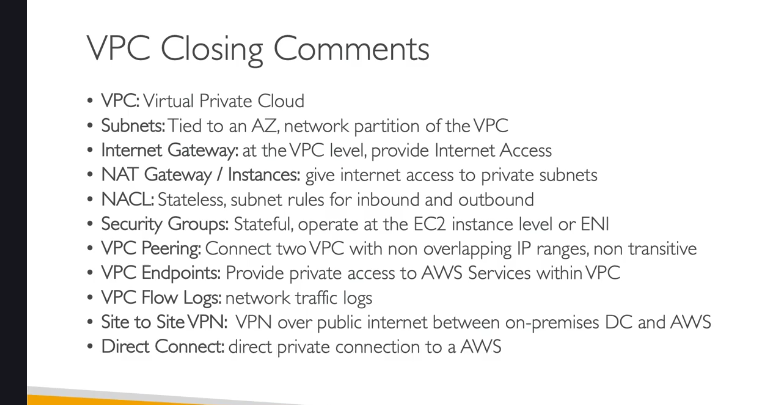




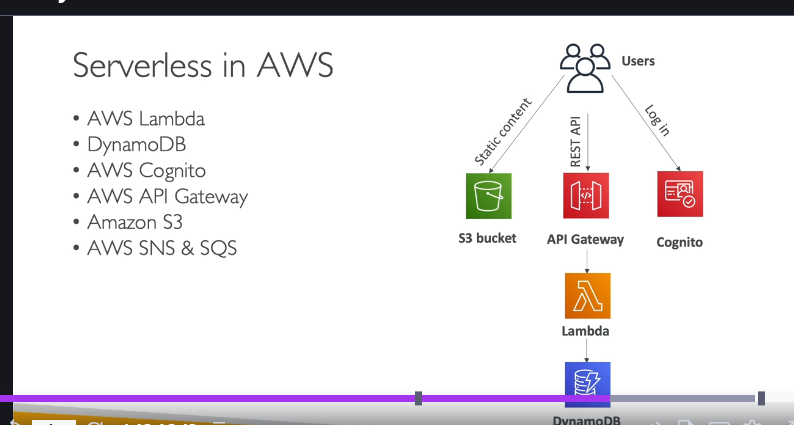


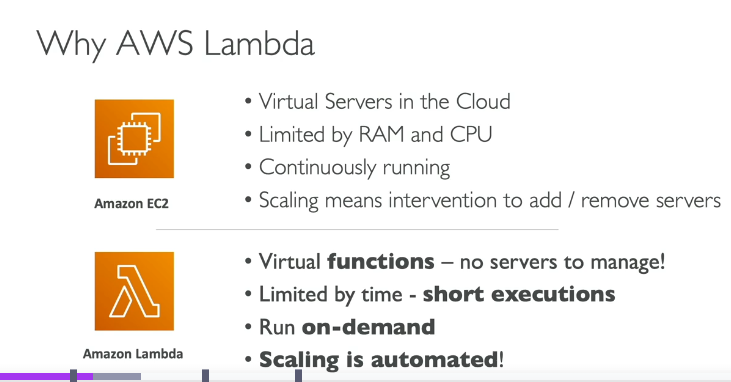


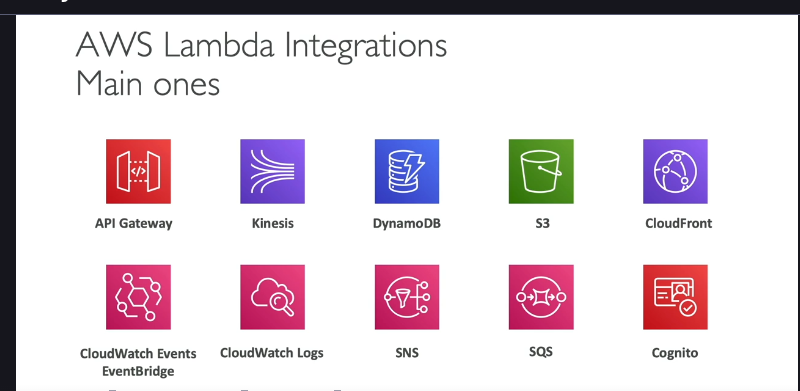




**AWS Lambda:**







**public class Hello implements RequestHandler<Object, String> {**

**public String handleRequest(final Object input, final Context context) {**

**System.out.println(input);**

**return "Hello from Lambda!";**

**}**

**}**

**public class HelloHandler implements RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {**

**public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent request, Context context) {**

**String name = request.getQueryStringParameters().getOrDefault("name", "Guest");**

**String response = "Hello, " + name;**

**return new APIGatewayProxyResponseEvent()**

**.withStatusCode(200)**

**.withBody(response);**

**}**

**}**

no my qustion is i want send only https call only, with out api gateway possible?

###### ChatGPT said:

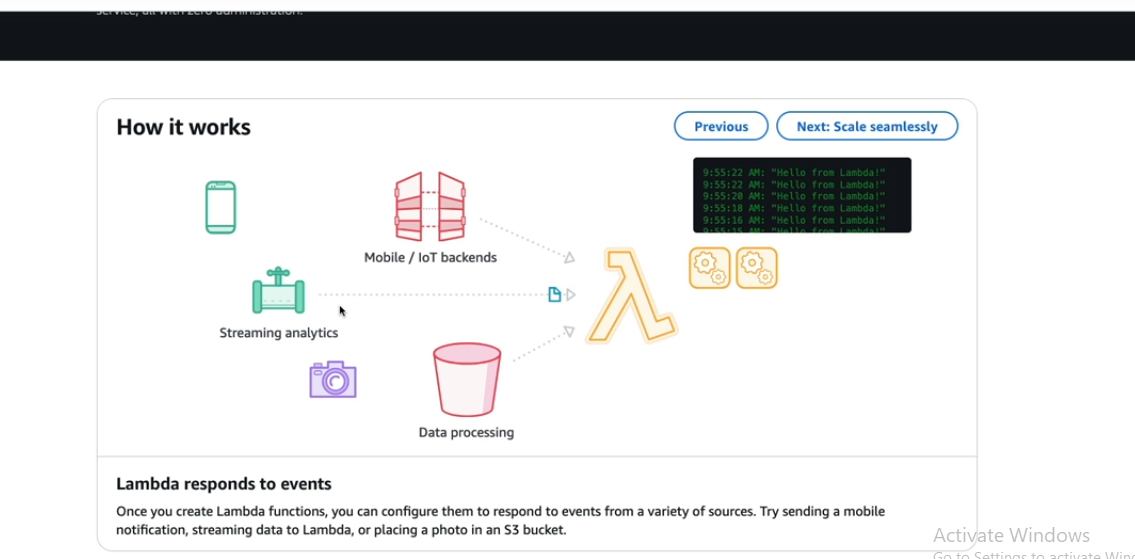
Ah! Now your question is **very clear**:

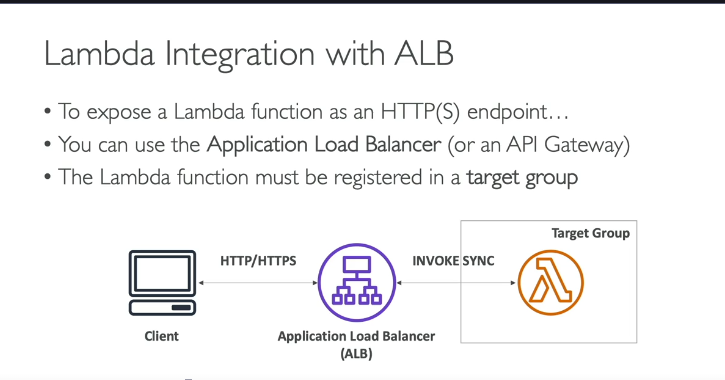
🧾 “Can I invoke a Lambda using only an **HTTPS call**, but **without using API Gateway**?”

## 🔴 Short Answer: ****No, not directly.****

### ❌ AWS Lambda does ****not expose an HTTPS endpoint by itself****.

Lambda is a **compute service**, not a web server.





**What is cold start in AWS lambda?**

**Aws lambda takes time for every time boot up, this is called cold start. We can reduce it by using** SnapStart.

### How to Reduce AWS Lambda Boot-Up Time (Cold Start)

Cold starts in AWS Lambda refer to the latency added when the Lambda service initializes a new instance of your function. Here are effective strategies to reduce cold start time:

#### 1. ****Choose the Right Runtime****

* **Lighter runtimes** like Node.js and Python have faster cold start times compared to Java or .NET.
* If using Java, prefer Java 11 or 17 over older versions, and avoid frameworks with heavy startup overhead.

#### 2. ****Reduce Deployment Package Size****

* Minimize the size of your Lambda function package.
* Avoid unnecessary dependencies.
* Use Lambda Layers for shared libraries.

#### 3. ****Enable Provisioned Concurrency****

* Keeps a specified number of Lambda instances warm and ready to serve.
* Ideal for production workloads that require low latency.

#### 4. ****Avoid VPC Attachments When Possible****

* VPC-attached Lambdas take longer to initialize network interfaces.
* Use Lambda outside of a VPC when possible, or set up efficient VPC endpoints and use Amazon provided DNS.

#### 5. ****Optimize Code Initialization****

* Move heavy computations or setup logic from the global scope to inside the handler (lazy initialization).
* Keep global scope light to reduce time spent during cold start.

#### 6. ****Keep Function Warm**** (For low-traffic apps)

* Use a scheduled CloudWatch Event (every 5–15 mins) to invoke the Lambda and prevent cold start.

#### 7. ****Use ARM64 Architecture****

* ARM-based Lambdas can start up faster and be more cost-efficient.
* Change architecture in the Lambda configuration to arm64.

#### 8. ****Optimize Memory Allocation****

* More memory = more CPU power = faster boot time.
* Test with higher memory settings and benchmark startup times.

#### 9. ****Use SnapStart**** (Java only)

* AWS Lambda SnapStart is available for Java 11.
* It reduces cold starts by caching and restoring function snapshots.

⚠️ Cold starts mainly affect **first requests** or **infrequently used Lambdas**. Frequent traffic naturally keeps Lambdas warm.

#### 9. ****Use SnapStart**** (Java only)

* AWS Lambda SnapStart is available for Java 11.
* It reduces cold starts by caching and restoring function snapshots.

What is dead letter queue?

Whenever any event failed twice or multiple times then events will be stored into dead letter queue (SQS or SNS)

This can be configured in asynchronous invocations under lambda functions

