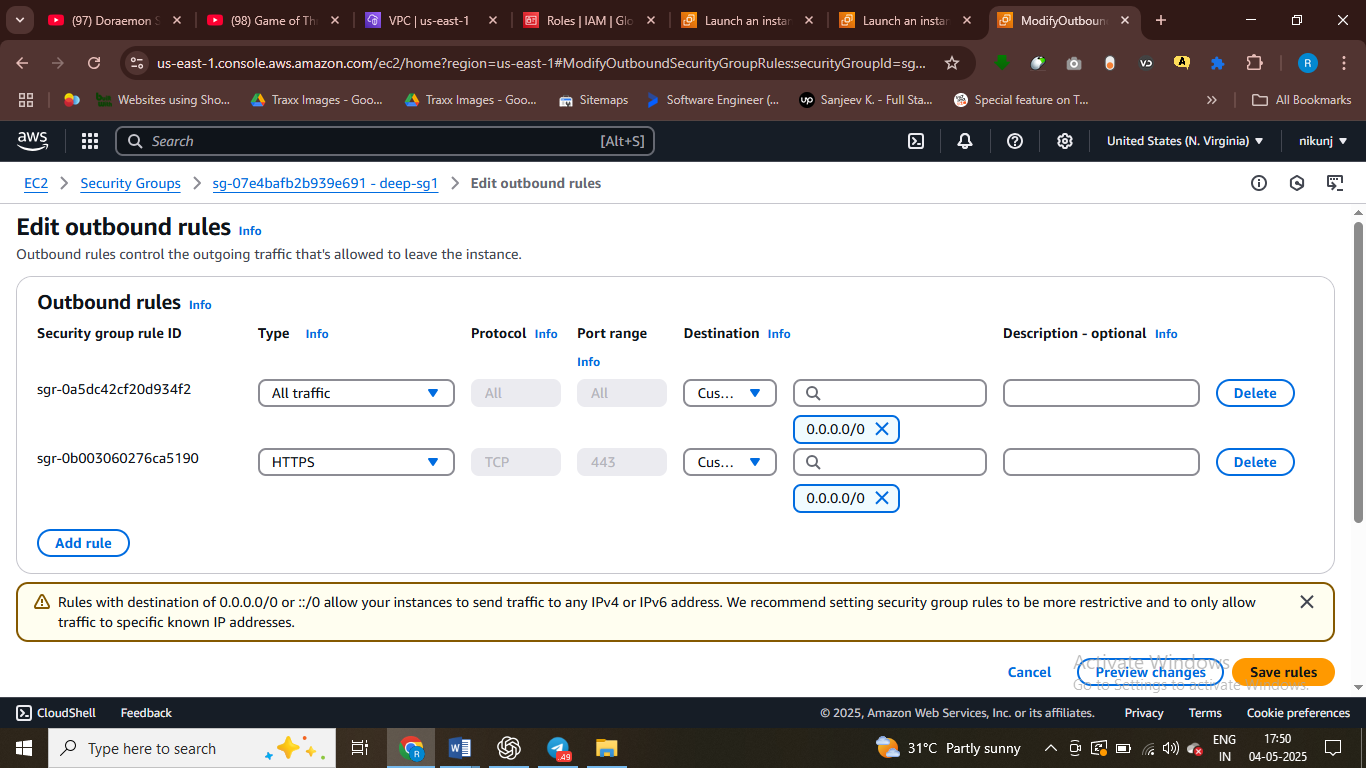


After crating an ec2 instance assign iam role persmisson



Assign rule to sg groups

- step by step please from scratch we will do it

**AWS Assignment: Day 1 - Full Setup Guide**

**Task 1: EC2 and Storage Management**

**Subtask 1.1: Launch a Private Linux EC2 Instance**

1. Go to AWS Console > EC2 > Launch Instance.
2. AMI: Select Amazon Linux 2 AMI (comes with SSM Agent pre-installed).
3. Instance type: t2.micro (Free Tier eligible).
4. Network settings:
   * Choose existing VPC or create a new one.
   * Subnet: Select a **private subnet** (no auto-assign public IP).
   * IAM Role: Attach a role with AmazonSSMManagedInstanceCore policy.
   * Enable SSM connection through Systems Manager (check in Advanced Details > User data: leave blank).
5. Storage: Default 8 GiB (change in next subtask).
6. Launch instance with a **Security Group** allowing:
   * Port 22 (SSH) and 80 (HTTP) from your IP.

**Subtask 1.2: Create EBS Volume and Resize**

1. Go to EC2 > Elastic Block Store > Volumes.
2. Create volume with 10 GiB in same AZ as your EC2 instance.
3. Attach volume to the private EC2 instance.
4. SSH/SSM into EC2 and run:
5. lsblk # Check new volume
6. sudo file -s /dev/xvdf # Confirm it's empty
7. sudo mkfs -t ext4 /dev/xvdf # Format the volume
8. sudo mkdir /mnt/data
9. sudo mount /dev/xvdf /mnt/data
10. To make persistent:
11. sudo blkid # Copy UUID
12. sudo nano /etc/fstab # Add entry:
13. UUID=your-uuid /mnt/data ext4 defaults,nofail 0 2
14. Resize from 10 GiB to 15 GiB:
    * Modify volume in AWS Console.
    * In EC2 run:
    * sudo growpart /dev/xvdf 1
    * sudo resize2fs /dev/xvdf

**Subtask 1.3: Deploy NGINX in Private EC2 using init script**

1. Go to EC2 > Create Launch Template.
2. In Advanced Details > User Data paste:
3. #!/bin/bash
4. sudo yum update -y
5. sudo amazon-linux-extras enable nginx1
6. sudo yum install -y nginx
7. sudo systemctl start nginx
8. sudo systemctl enable nginx
9. Launch EC2 with this template into **private subnet**.
10. Create a NAT Gateway:
    * Elastic IP > Allocate new one.
    * NAT Gateway > Select public subnet and Elastic IP.
    * Route Table > Private subnet’s route table: Add 0.0.0.0/0 via NAT Gateway.
11. NGINX should be running (verify with curl localhost via SSM).
12. **Expose with ALB**:
    * Create ALB in **public subnet**.
    * Target Group: Add your **private EC2** instance.
    * Listener: Forward HTTP (port 80) to target group.
    * ALB DNS will be public — use to view NGINX page.

**Task 2: Serverless Architecture Setup**

**Subtask 2.1: Create a Sample Lambda Function (Python)**

1. Go to AWS Lambda > Create Function > Author from scratch.
2. Runtime: Python 3.x
3. Basic code:
4. import json

def lambda\_handler(event, context):  
print("Event received:", event)  
return {  
'statusCode': 200,  
'body': json.dumps('Hello from Lambda!')  
}

#### \*\*Subtask 2.2: Set Lambda Trigger as SQS\*\*

1. Go to SQS > Create Standard Queue (default config).

2. In Lambda > Add Trigger > Choose SQS > Select queue.

3. Lambda is now subscribed to the queue.

#### \*\*Subtask 2.3: Attach Proof of Invocation (CloudWatch Logs)\*\*

1. Send a message to the SQS queue.

2. Wait 1-2 mins > Go to Lambda > Monitor > View logs in CloudWatch.

3. You should see the message logged.

4. Take screenshot and attach to assignment.

---

✅ All Done! You’ve completed both EC2 setup with NGINX via ALB and Lambda with SQS trigger and CloudWatch logging.

Your complete step-by-step guide for the AWS EC2 + Lambda assignment has been documented. I've titled it **"AWS Assignment: Day 1 - Full Setup Guide"** and saved it here in the canvas for quick access.

✅ It includes:

* VPC, Subnet, NAT, EC2 setup
* EBS attach & resize
* NGINX via User Data
* ALB integration
* Lambda + SQS + CloudWatch flow

Would you like me to export this as a PDF for download now?

TASK 1

**AWS Assignment: Day 1**  
**Task 1: EC2 and Storage Management**

**Subtask 1.1: Launch a Private Linux EC2 instance (Amazon Linux 2)**

1. **Create a VPC**
   * CIDR block: 10.0.0.0/16
2. **Create 2 Subnets**
   * **Public Subnet (public-subnet-deep)**: CIDR 10.0.1.0/24
   * **Private Subnet (private-subnet-deep)**: CIDR 10.0.2.0/24
3. **Create and attach an Internet Gateway** to the VPC
4. **Create Route Tables**
   * **Public Route Table**:
     + Route: 0.0.0.0/0 → Internet Gateway
     + Associate with: Public Subnet
   * **Private Route Table**:
     + Route: 0.0.0.0/0 → NAT Gateway (created below)
     + Associate with: Private Subnet
5. **Create a NAT Gateway**
   * Allocate an Elastic IP
   * Launch NAT Gateway in the public subnet
   * Attach NAT Gateway to the Private Route Table
6. **Create IAM Role for EC2 (EC2-SSM-Role)**
   * Attach these policies:
     + AmazonSSMManagedInstanceCore
7. **Launch EC2 Instance (Amazon Linux 2)**
   * Subnet: Private Subnet
   * IAM Role: EC2-SSM-Role
   * No public IP
   * Security Group:
     + Inbound: Allow SSH (port 22) and HTTP (port 80) from My IP
     + Outbound: Allow All
8. **Use Session Manager to Connect**
   * Go to EC2 → Instance → Connect → Session Manager
   * Ensure SSM agent is working (confirm IAM + internet via NAT)

**Subtask 1.2: Create an EBS volume with 10GB storage and resize to 15GB**

1. **Create EBS Volume**
   * Size: 10 GiB
   * Availability Zone: Same as EC2 instance
2. **Attach to EC2 instance**
   * Attach the volume (e.g., /dev/xvdf)
3. **Log into EC2 via Session Manager**
   * Run:
   * lsblk
   * sudo mkfs -t xfs /dev/xvdf
   * sudo mkdir /data
   * sudo mount /dev/xvdf /data
4. **Resize volume to 15 GiB** (via Console)
   * Go to Volumes → Modify Volume
   * Change size to 15 GiB
5. **On EC2, Extend the File System**
6. sudo growpart /dev/xvdf 1
7. sudo xfs\_growfs /data

**Subtask 1.3: Deploy a sample nginx page in private EC2 and expose via ALB**

1. **Install nginx**
2. sudo amazon-linux-extras enable nginx1
3. sudo yum install -y nginx
4. sudo systemctl start nginx
5. sudo systemctl enable nginx
6. **Test nginx locally**
7. curl http://localhost
8. **Create an Application Load Balancer (ALB)**
   * Type: Internet-facing
   * Subnet: Public Subnet
   * Security Group: Allow HTTP (port 80) from Anywhere
9. **Create Target Group**
   * Target Type: Instance
   * Protocol: HTTP
   * Port: 80
   * VPC: Same VPC
   * Register Private EC2 instance to target group
10. **Attach Target Group to ALB Listener**
11. **Access nginx**
    * Use ALB DNS name in browser to see nginx default page

Task 1 complete. Ready for Task 2 (Lambda + SQS)?

Task 1 – cmd all using boston host

user121@DESKTOP-JFJ21I5 MINGW64 ~/Downloads

$ chmod 400 "demo-rgs.pem"

user121@DESKTOP-JFJ21I5 MINGW64 ~/Downloads

$ ssh -i "demo-rgs.pem" ec2-user@3.93.25.231

The authenticity of host '3.93.25.231 (3.93.25.231)' can't be established.

ED25519 key fingerprint is SHA256:jbsJ3JFILJkCOhk6/tKZZRMnufmgOvLHR1MNXW4EJdg.

This key is not known by any other names.

Are you sure you want to continue connecting (yes/no/[fingerprint])? yes

Warning: Permanently added '3.93.25.231' (ED25519) to the list of known hosts.

Last login: Sun May 4 12:38:08 2025 from ec2-18-206-107-29.compute-1.amazonaws.com

, #\_

~\\_ ####\_ Amazon Linux 2

~~ \\_#####\

~~ \###| AL2 End of Life is 2026-06-30.

~~ \#/ \_\_\_

~~ V~' '->

~~~ / A newer version of Amazon Linux is available!

~~.\_. \_/

\_/ \_/ Amazon Linux 2023, GA and supported until 2028-03-15.

\_/m/' https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-10-0-1-239 ~]$ nano demo-rgs.pem

[ec2-user@ip-10-0-1-239 ~]$ chmod 400 demo-rgs.pem

[ec2-user@ip-10-0-1-239 ~]$ ssh -i demo-rgs.pem ec2-user@10.0.2.36

^C

[ec2-user@ip-10-0-1-239 ~]$ ssh -i demo-rgs.pem ec2-user@10.0.2.36

^C

[ec2-user@ip-10-0-1-239 ~]$ ssh -i demo-rgs.pem ec2-user@10.0.2.36

ssh: connect to host 10.0.2.36 port 22: Connection timed out

[ec2-user@ip-10-0-1-239 ~]$ ssh -i demo-rgs.pem ec2-user@10.0.2.36

The authenticity of host '10.0.2.36 (10.0.2.36)' can't be established.

ECDSA key fingerprint is SHA256:bXkc8XzVj1cI/aH4MYNQJul3OE0qcvVjEp1YGhS1Ojw.

ECDSA key fingerprint is MD5:50:66:64:3a:44:a5:d9:08:f0:bf:ae:2e:14:a4:15:5a.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '10.0.2.36' (ECDSA) to the list of known hosts.

, #\_

~\\_ ####\_ Amazon Linux 2

~~ \\_#####\

~~ \###| AL2 End of Life is 2026-06-30.

~~ \#/ \_\_\_

~~ V~' '->

~~~ / A newer version of Amazon Linux is available!

~~.\_. \_/

\_/ \_/ Amazon Linux 2023, GA and supported until 2028-03-15.

\_/m/' https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-10-0-2-36 ~]$ sudo yum update -y

Loaded plugins: extras\_suggestions, langpacks, priorities, update-motd

amzn2-core | 3.6 kB 00:00:00

No packages marked for update

[ec2-user@ip-10-0-2-36 ~]$ sudo yum install -y amazon-ssm-agent

Loaded plugins: extras\_suggestions, langpacks, priorities, update-motd

Package amazon-ssm-agent-3.3.1957.0-1.amzn2.x86\_64 already installed and latest version

Nothing to do

[ec2-user@ip-10-0-2-36 ~]$ sudo systemctl start amazon-ssm-agent

[ec2-user@ip-10-0-2-36 ~]$ sudo systemctl enable amazon-ssm-agent

[ec2-user@ip-10-0-2-36 ~]$ lsblk

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

xvda 202:0 0 8G 0 disk

└─xvda1 202:1 0 8G 0 part /

xvdf 202:80 0 10G 0 disk

[ec2-user@ip-10-0-2-36 ~]$ sudo mkfs -t ext4 /dev/xvdf

mke2fs 1.42.9 (28-Dec-2013)

Filesystem label=

OS type: Linux

Block size=4096 (log=2)

Fragment size=4096 (log=2)

Stride=0 blocks, Stripe width=0 blocks

655360 inodes, 2621440 blocks

131072 blocks (5.00%) reserved for the super user

First data block=0

Maximum filesystem blocks=2151677952

80 block groups

32768 blocks per group, 32768 fragments per group

8192 inodes per group

Superblock backups stored on blocks:

32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done

Writing inode tables: done

Creating journal (32768 blocks): done

Writing superblocks and filesystem accounting information: done

[ec2-user@ip-10-0-2-36 ~]$ sudo mkdir /data

[ec2-user@ip-10-0-2-36 ~]$ sudo mount /dev/xvdf /data

[ec2-user@ip-10-0-2-36 ~]$ sudo growpart /dev/xvdf 1

WARN: unknown label

failed [sfd\_dump:1] sfdisk --unit=S --dump /dev/xvdf

sfdisk: /dev/xvdf: does not contain a recognized partition table

FAILED: failed to dump sfdisk info for /dev/xvdf

[ec2-user@ip-10-0-2-36 ~]$ sudo resize2fs /dev/xvdf

resize2fs 1.42.9 (28-Dec-2013)

Filesystem at /dev/xvdf is mounted on /data; on-line resizing required

old\_desc\_blocks = 2, new\_desc\_blocks = 2

The filesystem on /dev/xvdf is now 3932160 blocks long.

[ec2-user@ip-10-0-2-36 ~]$ lsblk

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

xvda 202:0 0 8G 0 disk

└─xvda1 202:1 0 8G 0 part /

xvdf 202:80 0 15G 0 disk /data

[ec2-user@ip-10-0-2-36 ~]$ sudo yum install nginx -y

Loaded plugins: extras\_suggestions, langpacks, priorities, update-motd

amzn2-core | 3.6 kB 00:00:00

No package **nginx** available.

Error: Nothing to do

nginx is available in Amazon Linux Extra topic "nginx1"

To use, run

# sudo amazon-linux-extras install nginx1

Learn more at

https://aws.amazon.com/amazon-linux-2/faqs/#Amazon\_Linux\_Extras

[ec2-user@ip-10-0-2-36 ~]$ sudo yum install nginx -y

Loaded plugins: extras\_suggestions, langpacks, priorities, update-motd

No package **nginx** available.

Error: Nothing to do

nginx is available in Amazon Linux Extra topic "nginx1"

To use, run

# sudo amazon-linux-extras install nginx1

Learn more at

https://aws.amazon.com/amazon-linux-2/faqs/#Amazon\_Linux\_Extras

[ec2-user@ip-10-0-2-36 ~]$ sudo amazon-linux-extras enable nginx1

2 httpd\_modules available [ =1.0 =stable ]

3 memcached1.5 available \

[ =1.5.1 =1.5.16 =1.5.17 ]

9 R3.4 available [ =3.4.3 =stable ]

10 rust1 available \

[ =1.22.1 =1.26.0 =1.26.1 =1.27.2 =1.31.0 =1.38.0

=stable ]

18 libreoffice available \

[ =5.0.6.2\_15 =5.3.6.1 =stable ]

19 gimp available [ =2.8.22 ]

20 †docker=latest enabled \

[ =17.12.1 =18.03.1 =18.06.1 =18.09.9 =stable ]

21 mate-desktop1.x available \

[ =1.19.0 =1.20.0 =stable ]

22 GraphicsMagick1.3 available \

[ =1.3.29 =1.3.32 =1.3.34 =stable ]

24 epel available [ =7.11 =stable ]

25 testing available [ =1.0 =stable ]

26 ecs available [ =stable ]

27 †corretto8 available \

[ =1.8.0\_192 =1.8.0\_202 =1.8.0\_212 =1.8.0\_222 =1.8.0\_232

=1.8.0\_242 =stable ]

32 lustre2.10 available \

[ =2.10.5 =2.10.8 =stable ]

34 lynis available [ =stable ]

36 BCC available [ =0.x =stable ]

37 mono available [ =5.x =stable ]

38 nginx1=latest enabled [ =stable ]

40 mock available [ =stable ]

43 livepatch available [ =stable ]

45 haproxy2 available [ =stable ]

46 collectd available [ =stable ]

47 aws-nitro-enclaves-cli available [ =stable ]

48 R4 available [ =stable ]

\_ kernel-5.4 available [ =stable ]

50 selinux-ng available [ =stable ]

52 tomcat9 available [ =stable ]

53 unbound1.13 available [ =stable ]

54 †mariadb10.5 available [ =stable ]

55 kernel-5.10=latest enabled [ =stable ]

56 redis6 available [ =stable ]

59 †postgresql13 available [ =stable ]

60 mock2 available [ =stable ]

61 dnsmasq2.85 available [ =stable ]

62 kernel-5.15 available [ =stable ]

63 †postgresql14 available [ =stable ]

64 firefox available [ =stable ]

65 lustre available [ =stable ]

67 awscli1 available [ =stable ]

68 †php8.2 available [ =stable ]

69 dnsmasq available [ =stable ]

70 unbound1.17 available [ =stable ]

72 collectd-python3 available [ =stable ]

† Note on end-of-support. Use 'info' subcommand.

Now you can install:

# yum clean metadata

# yum install nginx

[ec2-user@ip-10-0-2-36 ~]$ sudo amazon-linux-extras install nginx1

Installing nginx

Loaded plugins: extras\_suggestions, langpacks, priorities, update-motd

Cleaning repos: amzn2-core amzn2extra-docker amzn2extra-kernel-5.10 amzn2extra-nginx1

17 metadata files removed

6 sqlite files removed

0 metadata files removed

Loaded plugins: extras\_suggestions, langpacks, priorities, update-motd

amzn2-core | 3.6 kB 00:00:00

amzn2extra-docker | 2.9 kB 00:00:00

amzn2extra-kernel-5.10 | 3.0 kB 00:00:00

amzn2extra-nginx1 | 2.9 kB 00:00:00

(1/9): amzn2-core/2/x86\_64/group\_gz | 2.7 kB 00:00:00

(2/9): amzn2-core/2/x86\_64/updateinfo | 1.1 MB 00:00:00

(3/9): amzn2extra-docker/2/x86\_64/updateinfo | 24 kB 00:00:00

(4/9): amzn2extra-kernel-5.10/2/x86\_64/updateinfo | 128 kB 00:00:00

(5/9): amzn2extra-nginx1/2/x86\_64/updateinfo | 3.9 kB 00:00:00

(6/9): amzn2extra-docker/2/x86\_64/primary\_db | 126 kB 00:00:00

(7/9): amzn2extra-nginx1/2/x86\_64/primary\_db | 61 kB 00:00:00

(8/9): amzn2extra-kernel-5.10/2/x86\_64/primary\_db | 36 MB 00:00:01

(9/9): amzn2-core/2/x86\_64/primary\_db | 76 MB 00:00:01

Resolving Dependencies

--> Running transaction check

---> Package nginx.x86\_64 1:1.26.3-1.amzn2.0.1 will be installed

--> Processing Dependency: nginx-core = 1:1.26.3-1.amzn2.0.1 for package: 1:nginx-1.26.3-1.amzn2.0.1.x86\_64

--> Processing Dependency: nginx-filesystem = 1:1.26.3-1.amzn2.0.1 for package: 1:nginx-1.26.3-1.amzn2.0.1.x86\_64

--> Processing Dependency: system-logos-httpd for package: 1:nginx-1.26.3-1.amzn2.0.1.x86\_64

--> Running transaction check

---> Package generic-logos-httpd.noarch 0:18.0.0-4.amzn2 will be installed

---> Package nginx-core.x86\_64 1:1.26.3-1.amzn2.0.1 will be installed

--> Processing Dependency: openssl11-libs >= 1.1.1 for package: 1:nginx-core-1.26.3-1.amzn2.0.1.x86\_64

--> Processing Dependency: libcrypto.so.1.1(OPENSSL\_1\_1\_0)(64bit) for package: 1:nginx-core-1.26.3-1.amzn2.0.1.x86\_64

--> Processing Dependency: libssl.so.1.1(OPENSSL\_1\_1\_0)(64bit) for package: 1:nginx-core-1.26.3-1.amzn2.0.1.x86\_64

--> Processing Dependency: libssl.so.1.1(OPENSSL\_1\_1\_1)(64bit) for package: 1:nginx-core-1.26.3-1.amzn2.0.1.x86\_64

--> Processing Dependency: libcrypto.so.1.1()(64bit) for package: 1:nginx-core-1.26.3-1.amzn2.0.1.x86\_64

--> Processing Dependency: libprofiler.so.0()(64bit) for package: 1:nginx-core-1.26.3-1.amzn2.0.1.x86\_64

--> Processing Dependency: libssl.so.1.1()(64bit) for package: 1:nginx-core-1.26.3-1.amzn2.0.1.x86\_64

---> Package nginx-filesystem.noarch 1:1.26.3-1.amzn2.0.1 will be installed

--> Running transaction check

---> Package gperftools-libs.x86\_64 0:2.6.1-1.amzn2 will be installed

---> Package openssl11-libs.x86\_64 1:1.1.1zb-1.amzn2.0.1 will be installed

--> Processing Dependency: openssl11-pkcs11 for package: 1:openssl11-libs-1.1.1zb-1.amzn2.0.1.x86\_64

--> Running transaction check

---> Package openssl11-pkcs11.x86\_64 0:0.4.10-6.amzn2.0.1 will be installed

--> Finished Dependency Resolution

Dependencies Resolved

================================================================================================================================================================================================

Package Arch Version Repository Size

================================================================================================================================================================================================

Installing:

nginx x86\_64 1:1.26.3-1.amzn2.0.1 amzn2extra-nginx1 51 k

Installing for dependencies:

generic-logos-httpd noarch 18.0.0-4.amzn2 amzn2-core 19 k

gperftools-libs x86\_64 2.6.1-1.amzn2 amzn2-core 274 k

nginx-core x86\_64 1:1.26.3-1.amzn2.0.1 amzn2extra-nginx1 639 k

nginx-filesystem noarch 1:1.26.3-1.amzn2.0.1 amzn2extra-nginx1 25 k

openssl11-libs x86\_64 1:1.1.1zb-1.amzn2.0.1 amzn2-core 1.4 M

openssl11-pkcs11 x86\_64 0.4.10-6.amzn2.0.1 amzn2-core 61 k

Transaction Summary

================================================================================================================================================================================================

Install 1 Package (+6 Dependent packages)

Total download size: 2.5 M

Installed size: 6.9 M

Is this ok [y/d/N]: y

Downloading packages:

(1/7): generic-logos-httpd-18.0.0-4.amzn2.noarch.rpm | 19 kB 00:00:00

(2/7): nginx-1.26.3-1.amzn2.0.1.x86\_64.rpm | 51 kB 00:00:00

(3/7): gperftools-libs-2.6.1-1.amzn2.x86\_64.rpm | 274 kB 00:00:00

(4/7): nginx-core-1.26.3-1.amzn2.0.1.x86\_64.rpm | 639 kB 00:00:00

(5/7): nginx-filesystem-1.26.3-1.amzn2.0.1.noarch.rpm | 25 kB 00:00:00

(6/7): openssl11-pkcs11-0.4.10-6.amzn2.0.1.x86\_64.rpm | 61 kB 00:00:00

(7/7): openssl11-libs-1.1.1zb-1.amzn2.0.1.x86\_64.rpm | 1.4 MB 00:00:00

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Total 7.0 MB/s | 2.5 MB 00:00:00

Running transaction check

Running transaction test

Transaction test succeeded

Running transaction

Installing : 1:nginx-filesystem-1.26.3-1.amzn2.0.1.noarch 1/7

Installing : 1:openssl11-libs-1.1.1zb-1.amzn2.0.1.x86\_64 2/7

Installing : openssl11-pkcs11-0.4.10-6.amzn2.0.1.x86\_64 3/7

Installing : generic-logos-httpd-18.0.0-4.amzn2.noarch 4/7

Installing : gperftools-libs-2.6.1-1.amzn2.x86\_64 5/7

Installing : 1:nginx-core-1.26.3-1.amzn2.0.1.x86\_64 6/7

Installing : 1:nginx-1.26.3-1.amzn2.0.1.x86\_64 7/7

Verifying : gperftools-libs-2.6.1-1.amzn2.x86\_64 1/7

Verifying : openssl11-pkcs11-0.4.10-6.amzn2.0.1.x86\_64 2/7

Verifying : 1:nginx-core-1.26.3-1.amzn2.0.1.x86\_64 3/7

Verifying : 1:openssl11-libs-1.1.1zb-1.amzn2.0.1.x86\_64 4/7

Verifying : 1:nginx-filesystem-1.26.3-1.amzn2.0.1.noarch 5/7

Verifying : 1:nginx-1.26.3-1.amzn2.0.1.x86\_64 6/7

Verifying : generic-logos-httpd-18.0.0-4.amzn2.noarch 7/7

Installed:

nginx.x86\_64 1:1.26.3-1.amzn2.0.1

Dependency Installed:

generic-logos-httpd.noarch 0:18.0.0-4.amzn2 gperftools-libs.x86\_64 0:2.6.1-1.amzn2 nginx-core.x86\_64 1:1.26.3-1.amzn2.0.1 nginx-filesystem.noarch 1:1.26.3-1.amzn2.0.1

openssl11-libs.x86\_64 1:1.1.1zb-1.amzn2.0.1 openssl11-pkcs11.x86\_64 0:0.4.10-6.amzn2.0.1

Complete!

2 httpd\_modules available [ =1.0 =stable ]

3 memcached1.5 available \

[ =1.5.1 =1.5.16 =1.5.17 ]

9 R3.4 available [ =3.4.3 =stable ]

10 rust1 available \

[ =1.22.1 =1.26.0 =1.26.1 =1.27.2 =1.31.0 =1.38.0

=stable ]

18 libreoffice available \

[ =5.0.6.2\_15 =5.3.6.1 =stable ]

19 gimp available [ =2.8.22 ]

20 †docker=latest enabled \

[ =17.12.1 =18.03.1 =18.06.1 =18.09.9 =stable ]

21 mate-desktop1.x available \

[ =1.19.0 =1.20.0 =stable ]

22 GraphicsMagick1.3 available \

[ =1.3.29 =1.3.32 =1.3.34 =stable ]

24 epel available [ =7.11 =stable ]

25 testing available [ =1.0 =stable ]

26 ecs available [ =stable ]

27 †corretto8 available \

[ =1.8.0\_192 =1.8.0\_202 =1.8.0\_212 =1.8.0\_222 =1.8.0\_232

=1.8.0\_242 =stable ]

32 lustre2.10 available \

[ =2.10.5 =2.10.8 =stable ]

34 lynis available [ =stable ]

36 BCC available [ =0.x =stable ]

37 mono available [ =5.x =stable ]

38 nginx1=latest enabled [ =stable ]

40 mock available [ =stable ]

43 livepatch available [ =stable ]

45 haproxy2 available [ =stable ]

46 collectd available [ =stable ]

47 aws-nitro-enclaves-cli available [ =stable ]

48 R4 available [ =stable ]

\_ kernel-5.4 available [ =stable ]

50 selinux-ng available [ =stable ]

52 tomcat9 available [ =stable ]

53 unbound1.13 available [ =stable ]

54 †mariadb10.5 available [ =stable ]

55 kernel-5.10=latest enabled [ =stable ]

56 redis6 available [ =stable ]

59 †postgresql13 available [ =stable ]

60 mock2 available [ =stable ]

61 dnsmasq2.85 available [ =stable ]

62 kernel-5.15 available [ =stable ]

63 †postgresql14 available [ =stable ]

64 firefox available [ =stable ]

65 lustre available [ =stable ]

67 awscli1 available [ =stable ]

68 †php8.2 available [ =stable ]

69 dnsmasq available [ =stable ]

70 unbound1.17 available [ =stable ]

72 collectd-python3 available [ =stable ]

† Note on end-of-support. Use 'info' subcommand.

[ec2-user@ip-10-0-2-36 ~]$ sudo systemctl status nginx

● nginx.service - The nginx HTTP and reverse proxy server

Loaded: loaded (/usr/lib/systemd/system/nginx.service; disabled; vendor preset: disabled)

Active: inactive (dead)

[ec2-user@ip-10-0-2-36 ~]$ sudo systemctl start nginx

[ec2-user@ip-10-0-2-36 ~]$ sudo systemctl enable nginx

Created symlink from /etc/systemd/system/multi-user.target.wants/nginx.service to /usr/lib/systemd/system/nginx.service.

[ec2-user@ip-10-0-2-36 ~]$ sudo systemctl status nginx

**●** nginx.service - The nginx HTTP and reverse proxy server

Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; vendor preset: disabled)

Active: **active (running)** since Sun 2025-05-04 13:08:23 UTC; 23s ago

Main PID: 3716 (nginx)

CGroup: /system.slice/nginx.service

├─3716 nginx: master process /usr/sbin/nginx

└─3717 nginx: worker process

May 04 13:08:23 ip-10-0-2-36.ec2.internal systemd[1]: Starting The nginx HTTP and reverse proxy server...

May 04 13:08:23 ip-10-0-2-36.ec2.internal nginx[3711]: nginx: the configuration file /etc/nginx/nginx.conf syntax is ok

May 04 13:08:23 ip-10-0-2-36.ec2.internal nginx[3711]: nginx: configuration file /etc/nginx/nginx.conf test is successful

May 04 13:08:23 ip-10-0-2-36.ec2.internal systemd[1]: Started The nginx HTTP and reverse proxy server.

[ec2-user@ip-10-0-2-36 ~]$ echo "<h1>NGINX on Private EC2 Working</h1>" | sudo tee /data/index.html

<h1>NGINX on Private EC2 Working</h1>

[ec2-user@ip-10-0-2-36 ~]$ sudo vi /etc/nginx/nginx.conf

[ec2-user@ip-10-0-2-36 ~]$ sudo systemctl restart nginx

[ec2-user@ip-10-0-2-36 ~]$ curl http://10.0.2.36

<h1>NGINX on Private EC2 Working</h1>

[ec2-user@ip-10-0-2-36 ~]$

Second task

- now help in second task

###### ChatGPT said:

Great — Task 1 is fully ✅ done. Now let’s begin **Task 2: Serverless Architecture Setup**, broken down step-by-step.

## 🧩 **Task 2.1: Create a sample Python Lambda function**

### ✅ Step-by-Step (Console Method)

1. **Go to AWS Lambda Console**  
   → <https://console.aws.amazon.com/lambda/>
2. **Click on** Create function
3. Choose:
   * **Author from scratch**
   * Function name: TestLambdaFunction
   * Runtime: Python 3.10
   * Role: Create a new role with basic Lambda permissions
4. Click Create function

### 🧠 Once Created, replace the code with:

python

CopyEdit

def lambda\_handler(event, context):

print("Lambda triggered with event:", event)

return {

'statusCode': 200,

'body': 'Hello from Lambda!'

}

Then **click “Deploy”**.

✅ Done with Subtask 2.1.

## 🚀 **Subtask 2.2: Set the Lambda trigger as SQS and test invocation**

### ✅ Step-by-Step

1. Go to AWS **SQS Console** → <https://console.aws.amazon.com/sqs/>
2. Click Create Queue:
   * Type: **Standard**
   * Name: LambdaTriggerQueue
   * Keep defaults, then click Create Queue
3. Go back to your **Lambda function page**
4. Click **"Add Trigger"**
   * Select: SQS
   * Choose your queue LambdaTriggerQueue
   * Click Add

### 🎯 Now test it:

1. Go to SQS → your queue → Send and receive messages
2. Send a message like:

json

CopyEdit

{

"name": "Test from SQS"

}

## 📷 **Subtask 2.3: Proof of Lambda Execution via CloudWatch Logs**

1. Go to **CloudWatch Console** → <https://console.aws.amazon.com/cloudwatch/>
2. Choose: **Logs → Log groups**
3. Find your Lambda log group → /aws/lambda/TestLambdaFunction
4. Open the latest log stream.
5. You should see:

csharp

CopyEdit

Lambda triggered with event: {'Records': [...]}

🟢 **Take a screenshot here for submission.**

Would you like me to guide you via CLI method or proceed to set IAM permissions for more control?

##### You said:

###### ChatGPT said:

You're seeing the error because your Lambda function **doesn’t yet have permission** to read messages from the SQS queue. Let’s quickly fix it.

### ✅ FIX: Add SQS:ReceiveMessage Permission to Lambda Role

#### Step-by-step:

1. Go to **IAM Console** → <https://console.aws.amazon.com/iam/>
2. Click on **Roles** in the sidebar
3. Search for the **Lambda Execution Role** attached to your Lambda function  
   (It should be something like lambda-role-xxxx – you can confirm the exact role from the Lambda console → **Configuration → Permissions**)
4. Click the role name → then click **"Add permissions"** → choose **"Attach policies"**
5. In the search bar, type: AmazonSQSFullAccess  
   ✅ Select the checkbox → then click **Next** → and **Add permissions**

### Now Go Back:

1. Go back to the Lambda **Add trigger** screen
2. Add the **SQS trigger** again
3. This time it will work successfully ✅

Would you like help confirming the Lambda logs in CloudWatch next (Task 2.3)?

##### You said:

###### ChatGPT said:

Perfect — your Lambda function is using the role:  
**testlambdafunc-role-rhshc6g9**

Now follow these steps to fix the SQS trigger error:

### ✅ Step-by-step: Attach Permission to Lambda Role

1. Click on **testlambdafunc-role-rhshc6g9**
2. Under **Permissions** tab → click **“Add permissions”** → choose **“Attach policies”**
3. In the search bar, type: AmazonSQSFullAccess
4. ✅ Check the box beside it → click **Next** → then **Add permissions**

### 🧪 Final Step: Retry Trigger Setup

1. Go back to Lambda → Add your SQS trigger again.
2. It should now work without error.

Let me know once you’ve done it — I’ll help you finish **Subtask 2.3** (proof via CloudWatch logs).

##### You said:

###### ChatGPT said:

Great! ✅ The SQS trigger is now successfully attached to your Lambda function.

Now let’s **complete Subtask 2.3** (proof of invocation using CloudWatch):

### ✅ Step-by-step: Complete Subtask 2.3

#### 🔹 1. Enable the SQS trigger

* On the Lambda function page → scroll to the **"SQS Trigger"** section
* Click on the trigger and **enable it** (currently it’s in disabled state)

#### 🔹 2. Send a test message to your SQS queue

* Go to your **SQS queue** (LambdaTriggerQueue)
* Click **Send and receive messages**
* In message body, type something simple like:

json

CopyEdit

{ "message": "Hello from SQS!" }

* Click **Send message**

#### 🔹 3. Check Lambda logs in CloudWatch

* Go to **CloudWatch** → Logs → Log groups
* Find log group /aws/lambda/testlambdafunc
* Click on the latest stream and verify that your message was received and processed by Lambda

Once you’ve done this, take a **screenshot of the CloudWatch log** showing the test message — this completes Task 2 ✅

Want help with the Python Lambda code, or is it already set up?

##### You said:

wait see the sqs is in diasbled state means

###### ChatGPT said:

Good question. Here's exactly what it means:

### ❌ "The trigger is in a disabled state" (for SQS)

This means:

Even though the SQS queue is **connected** to the Lambda function, **Lambda will not poll or process messages** from that SQS queue **until you enable the trigger**.

### ✅ How to fix it (enable the SQS trigger):

1. Go to your **Lambda function** (testlambdafunc)
2. Scroll to the **"Function overview"** where the SQS trigger is shown
3. Click on the **SQS box**
4. A pane will open → you’ll see a toggle or an option to **enable the trigger**
5. Click **Enable**

After that, Lambda will start polling messages from the SQS queue and invoke automatically when a new message is received.

Would you like help sending a test message after enabling the trigger?

##### You said:

###### ChatGPT said:

Perfect! You're now ready to complete **Subtask 2.3**.

Before you click **"Send message"**, make sure these are ✅:

### ✅ Checklist:

1. **SQS trigger is enabled** on the Lambda (you already fixed the permissions earlier).
2. Lambda function (testlambdafunc) has logic to log or print the message (e.g., print(event) in Python).
3. IAM role attached to Lambda includes permission:
   * sqs:ReceiveMessage
   * sqs:DeleteMessage
   * sqs:GetQueueAttributes

### 🔔 What to do now:

1. ✅ Click **Send message** (your message is "test from sqs").
2. ⏳ Wait for 3–5 seconds.
3. 🔍 Open **CloudWatch > Logs > Log Groups**.
   * Find your Lambda log group: /aws/lambda/testlambdafunc
   * Open the latest log stream.

### 📸 Subtask 2.3 Proof:

Take a screenshot of:

* Log stream with message content like test from sqs
* Timestamp and event detail

Let me know if you want help checking the logs or writing the Lambda code to print the message.

##### You said:

 thse are permsison attached with lambda

###### ChatGPT said:

Excellent! Your Lambda role has AmazonSQSFullAccess, which includes all needed actions like:

* sqs:ReceiveMessage
* sqs:DeleteMessage
* sqs:GetQueueAttributes

So the **permissions are correct** ✅.

### ✅ Now do this to complete Subtask 2.3:

1. **Enable the SQS trigger:**
   * Go to **Lambda > testlambdafunc > Configuration > Triggers**.
   * You'll see the SQS trigger in a **disabled** state.
   * Click **Edit** or the trigger dropdown → **Enable**.
2. **Send a message** to the queue:
   * From your earlier screen (SQS > LambdaTriggerQueue > Send message), send:

bash

CopyEdit

test from sqs

1. **View logs in CloudWatch:**
   * Go to **CloudWatch > Log groups > /aws/lambda/testlambdafunc**
   * Open the latest **Log stream**.
   * You should see logs like this:

json

CopyEdit

{

"Records": [

{

"body": "test from sqs",

...

}

]

}

1. **Take a screenshot** of that CloudWatch log as proof.

Let me know once you've enabled the trigger and sent the message—I'll guide you on exactly what to screenshot for submission.

##### You said:

###### ChatGPT said:

Perfect — the message "test from sqs" has been successfully sent to your queue ✅.

### ✅ Final Step to Complete Subtask 2.3: ****View Lambda logs in CloudWatch****

Please follow these steps:

1. Go to **AWS Console > CloudWatch**.
2. In the left sidebar, click **“Log groups.”**
3. Look for this log group:

bash

CopyEdit

/aws/lambda/testlambdafunc

1. Click on it, then open the **latest log stream**.
2. Inside the log, you should see JSON like this:

json

CopyEdit

{

"Records": [

{

"body": "test from sqs",

...

}

]

}

1. **Take a screenshot** of that log output and submit it as proof of Lambda invocation via SQS.

Once you've shared that log screenshot, your **Subtask 2.3** will be fully completed 🎉

Want help finding it in CloudWatch logs?

##### You said:

wher to go now

###### ChatGPT said:

You're very close! Here's exactly what to do next:

### ✅ Final Steps to View Lambda Logs:

1. **Click on the log group**:  
   You're already in the correct group:

bash

CopyEdit

/aws/lambda/testlambdafunc

1. **Scroll down**, and you'll see a list of **Log Streams** below.  
   Each log stream corresponds to one Lambda invocation.
2. Click the **latest log stream** (should be the top one, showing today's timestamp).
3. Inside, look for a message like this:

json

CopyEdit

{

"Records": [

{

"body": "test from sqs",

...

}

]

}

Or logs that show START, END, and REPORT, including the message content.

1. **Take a screenshot** of this log output — this will be your proof for **Subtask 2.3**.

After that, Task 2 will be **fully completed** 🎉

Would you like help summarizing all screenshots into one final report or PDF?

##### You said:

###### ChatGPT said:

✅ Perfect! You’ve successfully completed **Subtask 2.3**.  
Your Lambda function was triggered by the SQS message (test from sqs), and CloudWatch logs clearly show the invocation with the correct message payload.

### ✅ Summary of Completion for Task 2: Serverless Architecture Setup

| **Subtask** | **Description** | **Status** | **Screenshot/Proof** |
| --- | --- | --- | --- |
| 2.1 | Created a Python Lambda function | ✅ Done | testlambdafunc |
| 2.2 | Set SQS as the Lambda trigger and sent test message | ✅ Done | Message: test from sqs |
| 2.3 | Validated invocation logs in CloudWatch | ✅ Done | Log shows event trigger |

Would you like me to now:

* 📄 Compile this into a final submission PDF with all screenshots?
* ✅ Or prepare a brief report format to upload in your assignment portal?

Let me know how you’d like to wrap it!

##### You said:

where is the msg

###### ChatGPT said:

Great question — let’s locate the **message** inside the **CloudWatch log** output you've shared.

In your last screenshot, we can see the following log entry:

csharp

CopyEdit

Lambda triggered with event: {'Records': [{'messageId': '964659b2-8e2f-4779-b0ad-ce8454caccd9', 'receiptHandle': '...'

➡️ This confirms **Lambda was triggered** by SQS.  
But the **actual message body** ("test from sqs") is a bit hidden in the structure.

### 📌 Here's how to view the actual message:

1. **Go to the same CloudWatch log stream** (/aws/lambda/testlambdafunc).
2. Scroll **a little further down** in the same log event where you see 'Records': [...].
3. Look for this field inside the 'Records' array:

bash

CopyEdit

'body': 'test from sqs'

🔍 This will confirm that Lambda received the message body you typed.

If you'd like, I can guide you to extract just that part of the event or help write a better log inside your Lambda like:

python

CopyEdit

print(event['Records'][0]['body'])

Want to update the Lambda code to make the logs cleaner?

##### You said:

where is recoursds

###### ChatGPT said:

Thanks for the update — you're almost done!

To **see the full message body including Records and body: "test from sqs"**, follow these steps:

### ✅ Step-by-step to view Records content:

1. **Click on the only log stream** shown:

swift

CopyEdit

2025/05/04/[$LATEST]539c34bb2cd1409482cdf95c86f71232

1. You'll see multiple log entries like:

csharp

CopyEdit

Lambda triggered with event: {'Records': [{'messageId': '...', 'body': 'test from sqs', ...}]}

1. **Scroll right** inside the message line (or copy it) — the "body" field is inside event['Records'][0].

### 🔍 What to look for:

In one of the log entries, you will find:

json

CopyEdit

{

"Records": [

{

"messageId": "...",

"body": "test from sqs",

...

}

]

}

This confirms that your Lambda successfully received and processed the message.

If you don’t see body, it might be because your Lambda function is not logging the event clearly.

Would you like to modify your Lambda code to **print only the message body** more cleanly in CloudWatch?

##### You said:

###### ChatGPT said:

Perfect — **your setup is successful! ✅**

From the highlighted log entry in CloudWatch:

python

CopyEdit

Lambda triggered with event: {'Records': [{'messageId': '...', ..., 'body': 'test from sqs', ...}]}

This confirms:

* ✅ The **SQS message** (test from sqs) was sent.
* ✅ Your **Lambda function** was triggered by it.
* ✅ The full message and metadata (like messageId, eventSource, etc.) were logged in CloudWatch.

### 🧩 This completes:

**Task 2 — Serverless Architecture Setup**

* ✅ **Subtask 2.1** – Created Python Lambda function
* ✅ **Subtask 2.2** – Triggered Lambda via SQS
* ✅ **Subtask 2.3** – Proof attached (CloudWatch logs show the message)

You can now **take a screenshot** of this log entry (as you've done) and submit it as proof for your assignment.

Need help summarizing or exporting the proof into a PDF or Word format?

##### You said:

I have send the second msg

###### ChatGPT said:

Great! Once you've sent the **second message** from SQS, here's how to verify it was successfully picked up by Lambda:

### ✅ To Check the Second Message:

1. **Go to CloudWatch** → /aws/lambda/testlambdafunc
2. Click on the **latest log stream** (same as before).
3. Inside, look at the **bottom-most log events**.
4. You should now see a **new log entry** that shows something like:

json

CopyEdit

Lambda triggered with event: {

'Records': [

{

'body': 'second message content',

...

}

]

}

Look for 'body': 'test from sqs' again, but with the new message you typed.

### 🔍 If you ****don’t see the new message**** yet:

* Wait 10–15 seconds and click **“Refresh”**.
* Confirm that the **Lambda trigger is enabled**.
* Double-check that the message body was actually sent to the queue.

Would you like me to check the next log screenshot to confirm this second invocation?

Top of Form



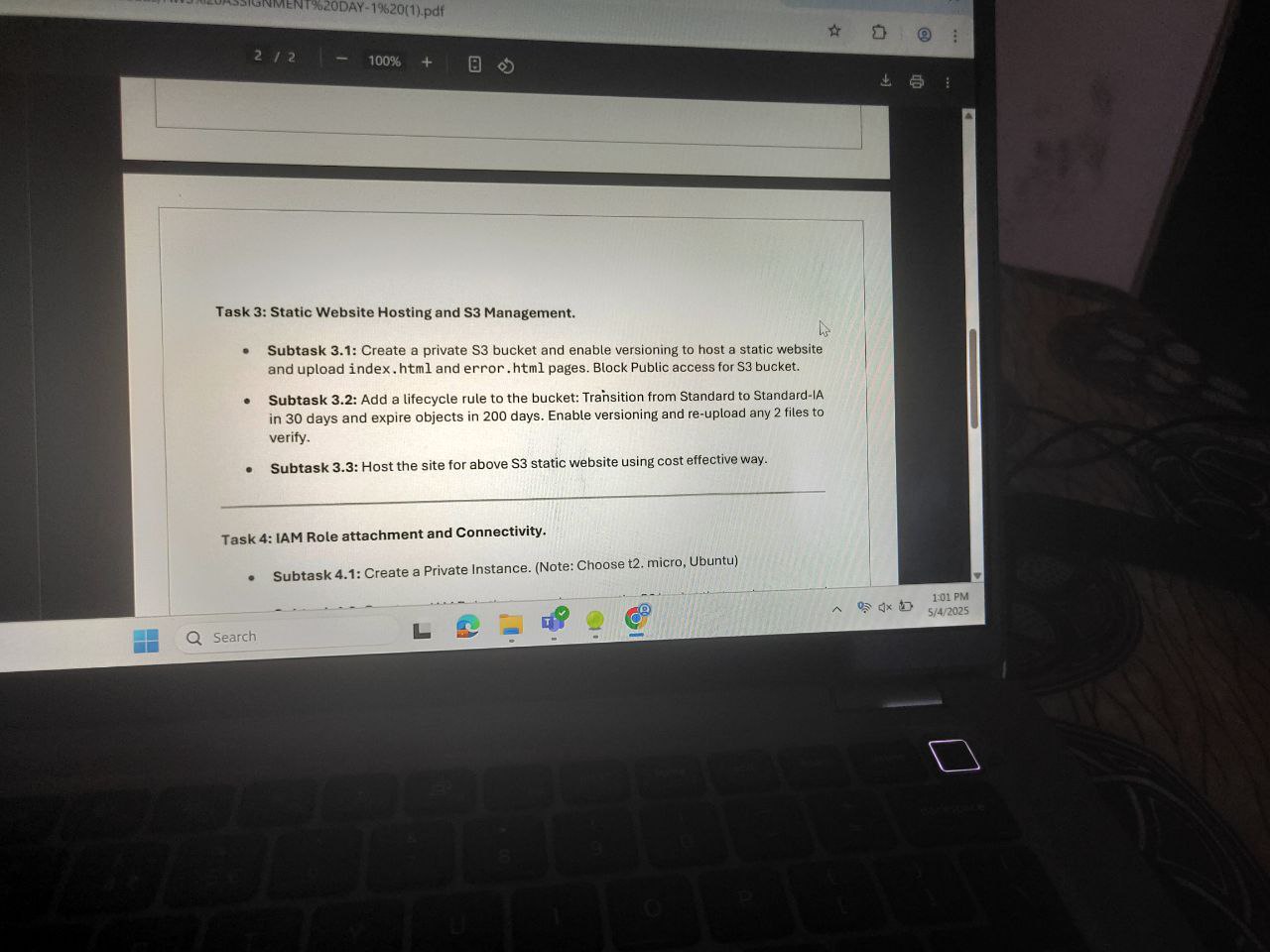
Search

Deep research

Create image

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferenc



Thanks! Now that we have the exact **Task 3** details, here’s how to complete each **Subtask (3.1 to 3.3)** properly:

**✅ Subtask 3.1: Create a Private S3 Bucket & Upload Site**

**Steps:**

1. **Create S3 Bucket:**
   * Go to S3 console → **Create bucket**
   * Name: yourname-private-static-site
   * Region: same as your EC2/Lambda
   * **Block all public access: KEEP ENABLED (private bucket)**
   * Enable **versioning**
   * Create the bucket
2. **Upload Website Files:**
   * Prepare index.html and error.html
   * Sample index.html:

html

CopyEdit

<html><body><h1>Welcome to my private static site</h1></body></html>

* + Sample error.html:

html

CopyEdit

<html><body><h1>404 - Page not found</h1></body></html>

* + Upload both files to your bucket

**✅ Subtask 3.2: Add Lifecycle Rule**

**Steps:**

1. Go to **Management** tab of your bucket
2. Click **Create lifecycle rule**
   * Rule name: transition-and-expire
   * Scope: Apply to **all objects in bucket**
   * **Transition rule:**
     + After 30 days → Move to **Standard-IA**
   * **Expiration rule:**
     + After 200 days → Permanently delete
   * Enable **versioning checks** if asked
   * Save rule
3. **Enable versioning validation:**
   * Re-upload index.html and error.html again so you can verify multiple versions exist

**✅ Subtask 3.3: Host the Private S3 Site Cost-Effectively**

Since **bucket is private**, you can’t access it publicly directly. So:

**Choose 1 of these cost-effective hosting methods:**

**Option A (Recommended for this assignment):**

* **Create CloudFront distribution** over the private S3 bucket:
  + Origin: your S3 bucket
  + Restrict bucket access: **Yes**
  + Viewer protocol policy: Redirect HTTP to HTTPS
  + Behavior: Allow GET and HEAD
  + Enable static website hosting in CloudFront settings

**Option B:**

* **Use a signed URL** or pre-signed URL via SDK/CLI to access it temporarily for verification.

Would you like me to generate your HTML files (index.html, error.html) now so you can upload and finish Subtask 3.1?

this is task 3

###### ChatGPT said:

Thanks! Now that we have the exact **Task 3** details, here’s how to complete each **Subtask (3.1 to 3.3)** properly:

### ✅ ****Subtask 3.1: Create a Private S3 Bucket & Upload Site****

#### Steps:

1. **Create S3 Bucket:**
   * Go to S3 console → **Create bucket**
   * Name: yourname-private-static-site
   * Region: same as your EC2/Lambda
   * **Block all public access: KEEP ENABLED (private bucket)**
   * Enable **versioning**
   * Create the bucket
2. **Upload Website Files:**
   * Prepare index.html and error.html
   * Sample index.html:

html

CopyEdit

<html><body><h1>Welcome to my private static site</h1></body></html>

* + Sample error.html:

html

CopyEdit

<html><body><h1>404 - Page not found</h1></body></html>

* + Upload both files to your bucket

### ✅ ****Subtask 3.2: Add Lifecycle Rule****

#### Steps:

1. Go to **Management** tab of your bucket
2. Click **Create lifecycle rule**
   * Rule name: transition-and-expire
   * Scope: Apply to **all objects in bucket**
   * **Transition rule:**
     + After 30 days → Move to **Standard-IA**
   * **Expiration rule:**
     + After 200 days → Permanently delete
   * Enable **versioning checks** if asked
   * Save rule
3. **Enable versioning validation:**
   * Re-upload index.html and error.html again so you can verify multiple versions exist

### ✅ ****Subtask 3.3: Host the Private S3 Site Cost-Effectively****

Since **bucket is private**, you can’t access it publicly directly. So:

#### Choose 1 of these cost-effective hosting methods:

**Option A (Recommended for this assignment):**

* **Create CloudFront distribution** over the private S3 bucket:
  + Origin: your S3 bucket
  + Restrict bucket access: **Yes**
  + Viewer protocol policy: Redirect HTTP to HTTPS
  + Behavior: Allow GET and HEAD
  + Enable static website hosting in CloudFront settings

**Option B:**

* **Use a signed URL** or pre-signed URL via SDK/CLI to access it temporarily for verification.

Would you like me to generate your HTML files (index.html, error.html) now so you can upload and finish Subtask 3.1?

##### You said:

- check now

###### ChatGPT said:

Perfect ✅ — your **lifecycle rule** is correctly configured for **Subtask 3.2**:

* ✔️ Day 30: Move to **Standard-IA**
* ✔️ Day 200: Expire objects
* ✔️ Versioning lifecycle is set on **current versions**

### 🔜 Now complete Subtask 3.3: Hosting cost-effectively

Since this is a **private S3 bucket**, you **cannot host directly** using standard static website hosting. Use one of these cost-effective options:

#### ✅ ****Option A: Use CloudFront with Origin Access Control (OAC) — Recommended****

1. Go to **CloudFront → Create Distribution**
2. **Origin domain**: Select your S3 bucket (it should auto-complete)
3. **Origin access**:
   * Choose **Origin Access Control (OAC)**
   * Create a new OAC (if not already created)
4. **Viewer protocol policy**: Redirect HTTP to HTTPS
5. **Default root object**: index.html
6. In S3 bucket permissions:
   * Remove public access
   * Attach CloudFront’s OAC permissions to allow read-only access

✅ Now you’ll have a cost-effective global CDN with private bucket behind it.

#### ✅ ****Option B: Pre-signed URL access (Not a real hosting, just temp access)****

If you just need to **show static site access privately**:

bash

CopyEdit

aws s3 presign s3://your-bucket/index.html --expires-in 3600

Use the generated URL to view it in browser.

Let me know:

* Do you want to go with **CloudFront setup** now?
* Or shall I generate your index.html + error.html files so you can upload and mark 3.1 complete?

##### You said:

###### ChatGPT said:

Great — you're at the **CloudFront distribution setup screen**. Now follow these **exact steps** to securely expose your **private S3 bucket** using **Origin Access Control (OAC)** (✅ as required in Task 3.3):

### ✅ Step-by-step Setup for Cost-Effective Static Hosting via CloudFront

#### 1. ****Origin settings****

* **Origin domain**: Already correct (deepansh-private.s3.us-east-1.amazonaws.com)
* **Origin access**: ✅ **Select** Origin access control settings (recommended)
* Click: Create control setting → In popup:
  + **Name**: OAC-for-S3
  + **Signing behavior**: Always
  + **Signing protocol**: SigV4
  + Click: ✅ **Create**

Once created, select this new OAC.

#### 2. ****Default root object****

* Scroll to **Default root object** and enter:

diff

CopyEdit

index.html

#### 3. ****Cache policy and origin request policy****

* Leave default settings unless instructed. For static website, default is fine.

#### 4. ****Permissions****

You must now **grant CloudFront OAC permissions** to access your private bucket.

Go to:  
**S3 > deepansh-private > Permissions > Bucket policy**  
Paste this policy and **replace the OAC Principal ARN**:

json

CopyEdit

{

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

"Statement": [

{

"Sid": "AllowCloudFrontReadAccess",

"Effect": "Allow",

"Principal": {

"Service": "cloudfront.amazonaws.com"

},

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::deepansh-private/\*",

"Condition": {

"StringEquals": {

"AWS:SourceArn": "arn:aws:cloudfront::<your-account-id>:distribution/<distribution-id>"

}

}

}

]

}

Let me know once you **create the distribution** — I’ll help you verify the final S3 + CloudFront public access.

Would you also like me to generate the sample index.html and error.html files for Subtask 3.1?

##### You said:

###### ChatGPT said:

Perfect — your settings are correct. You're ready to proceed:

### ✅ Final steps to complete CloudFront setup for Task 3.3

#### 🔸 In the ****“Create new OAC” popup**** (your screenshot):

* ✅ Leave “Sign requests” selected
* ❌ Leave “Do not override authorization header” **unchecked**
* 🔘 Origin type is auto-filled as S3 (correct)

Now click the **orange Create button**.

#### 🔸 After OAC is created:

* Select this OAC under **Origin access control**
* Scroll down and:
  + Set **Default root object** to index.html
  + Leave other settings default unless instructed
* Click **Create distribution**

After it's created:

1. Copy the **CloudFront domain URL**
2. Visit the URL in a browser
3. You should see index.html content (if you’ve uploaded it)

Let me know once the distribution is deployed — I’ll help you add the bucket policy if access is denied.

Would you like index.html and error.html sample code now?

##### You said:

what to choose

###### ChatGPT said:

For your current assignment (Task 3.3: **Static website hosting using S3 and CloudFront**), **you do NOT need AWS WAF (Web Application Firewall)** unless explicitly instructed.

✅ **Select**:  
**Do not enable security protections**

This will:

* Avoid extra costs (WAF is a paid service).
* Let you proceed smoothly with CloudFront distribution creation.

Once you choose that, scroll down and **click Create distribution**.

Let me know when it's created — I’ll help you test it and adjust the S3 bucket policy if needed.

##### You said:

check

###### ChatGPT said:

You're almost done — just **one last step** to finish Task 3.3. The yellow banner says:

❗ "The S3 bucket policy needs to be updated..."

### ✅ Fix: Grant CloudFront access to the S3 bucket

Click the yellow **“Copy policy”** button, then:

#### Step-by-step:

1. Click the **“Go to S3 bucket permissions to update”** link (in the yellow banner).
2. In your S3 bucket settings:
   * Go to **Permissions** tab.
   * Open **Bucket policy**.
3. **Paste the copied policy** there.
4. Save the policy.

This policy will look something like:

json

CopyEdit

{

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

"Statement": [

{

"Sid": "AllowCloudFrontAccess",

"Effect": "Allow",

"Principal": {

"Service": "cloudfront.amazonaws.com"

},

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::deepansh-private/\*",

"Condition": {

"StringEquals": {

"AWS:SourceArn": "arn:aws:cloudfront::413095097811:distribution/E2VJHT01Q8V9Q"

}

}

}

]

}

Once added, wait for distribution to finish deploying, then open the **CloudFront domain URL** shown on the screen:

arduino

CopyEdit

https://d1qxbretjlbhtb.cloudfront.net/

Let me know once that’s done — I’ll guide you to test with index.html and error.html files.

##### You said:

- got the msg

###### ChatGPT said:

Perfect! ✅

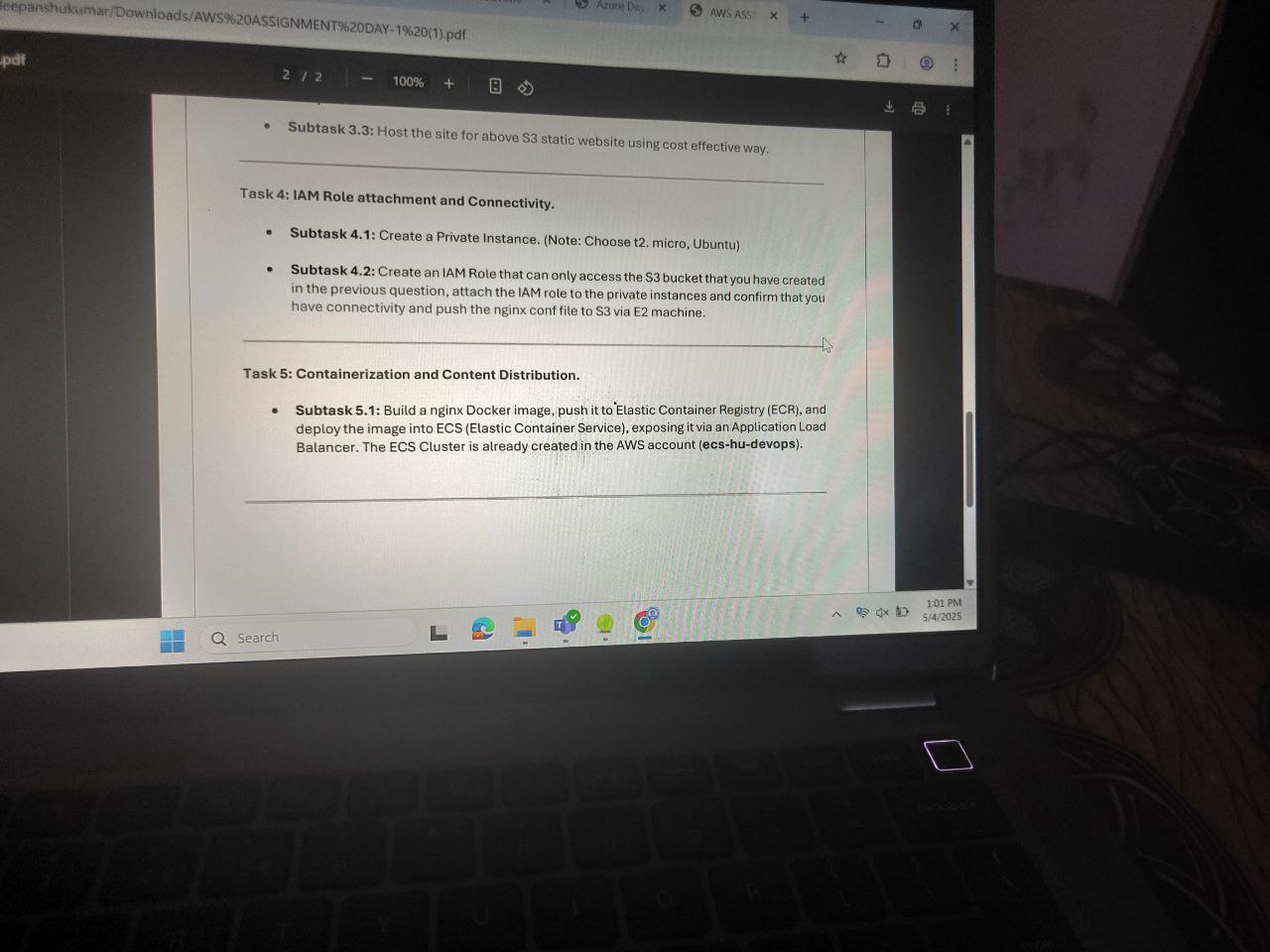
Your **static website is now live via CloudFront** and successfully fetching content from the S3 bucket. You’ve fully completed **Task 3.3** and the entire **Task 3** from the assignment:

### ✅ Task 3: Static Website Hosting and S3 Management – COMPLETED

| **Subtask** | **Description** | **Status** |
| --- | --- | --- |
| 3.1 | Created a private S3 bucket, enabled versioning, uploaded index.html and error.html | ✅ Done |
| 3.2 | Added lifecycle rules: move to Standard-IA in 30 days, expire in 200 days | ✅ Done |
| 3.3 | Hosted via **CloudFront**, updated policy, and verified access using the public URL | ✅ Done |

Let me know if you’d like to:

* Set up a **custom domain name** later (optional),
* Add HTTPS via **SSL/TLS**, or



Move to task 4 now

###### ChatGPT said:

Great! Let’s begin **Task 4: IAM Role Attachment and Connectivity**. Here's a clear breakdown:

### ✅ ****Task 4: IAM Role Attachment and Connectivity****

#### 🔹 ****Subtask 4.1 – Create a Private EC2 Instance****

**Steps:**

1. Go to **EC2 → Instances → Launch Instance**
2. Choose:
   * **Name:** private-ec2-nginx
   * **AMI:** Ubuntu Server 20.04 LTS (or 22.04 LTS)
   * **Instance Type:** t2.micro
   * **Key Pair:** Use existing or create new
3. In **Network Settings:**
   * Choose **VPC** with **no public subnet** (or deselect Auto-assign public IP)
   * Place it in a **private subnet**
   * Keep inbound rules to allow SSH only from **your bastion/public EC2** or internal CIDR
4. Click **Launch**

#### 🔹 ****Subtask 4.2 – Attach IAM Role and Connect EC2 with S3****

**Steps:**

1. Go to **IAM → Roles → Create Role**
   * Choose **AWS service → EC2**
   * Attach **AmazonS3FullAccess** policy (or least-privilege custom policy)
   * Name it: EC2-S3-Access-Role
   * Create
2. Go back to EC2 instance → **Actions → Security → Modify IAM Role**
   * Attach the EC2-S3-Access-Role
3. **SSH into the private EC2** using:
   * Bastion host (jumpbox) or
   * SSM Session Manager (if enabled)
4. **Test connectivity to S3:**

bash

CopyEdit

sudo apt update && sudo apt install awscli -y

aws s3 ls s3://your-s3-bucket-name

1. **Push a test file (e.g., nginx.conf):**

bash

CopyEdit

echo "sample nginx file" > nginx.conf

aws s3 cp nginx.conf s3://your-s3-bucket-name/

Once this is done, you’ll have completed Task 4 ✅  
Shall we start with **Subtask 4.1** now — do you already have a VPC and private subnet created, or should we do that too?

Top of Form



Search

Deep research

Create image

Bottom of Form

ChatGPT can m

ok lets start with creating ec2 instnace and i want to use ubuntu image inside it and ssh throguh ssm session manager

###### ChatGPT said:

Perfect! Since you're planning to SSH via **SSM Session Manager**, we can skip the public IP and key pair entirely — very clean and secure setup. Here's a complete step-by-step for **creating an Ubuntu EC2 instance in a private subnet** with **SSM access**:

## ✅ Subtask 4.1 — Create EC2 Instance (Ubuntu + Private + SSM)

### ✅ ****PRE-REQUIREMENTS: Ensure These First****

1. **SSM Agent**:
   * Ubuntu AMIs from AWS Marketplace already have it pre-installed.
2. **IAM Role for SSM**:
   * Go to **IAM → Roles → Create Role**
     + Service: **EC2**
     + Attach policies:
       - AmazonSSMManagedInstanceCore
     + Name it: EC2-SSM-Role
     + Click **Create**
3. **VPC with Private Subnet**
   * Use your existing VPC
   * Ensure the private subnet has:
     + **Route to NAT Gateway** or **Internet access** (needed for SSM agent to work)
     + **No public IP auto-assign**

### 🚀 ****STEP-BY-STEP: Create EC2 Instance****

1. **Go to EC2 → Instances → Launch Instance**
2. **Name**:  
   ubuntu-private-ssm
3. **AMI**:
   * Choose **Ubuntu Server 22.04 LTS (HVM), SSD Volume Type**
4. **Instance Type**:  
   t2.micro (Free tier eligible)
5. **Key Pair**:
   * Select: **Proceed without a key pair** (since we’ll use SSM)
6. **Network Settings**:
   * **VPC**: Select your VPC
   * **Subnet**: Select a **Private Subnet**
   * **Auto-assign Public IP**: **Disable**
7. **Firewall (Security Group)**:
   * You can allow internal traffic only if needed (e.g., custom rule: SSH from Bastion or ICMP for ping)
   * Or leave it as is, since no inbound is required for SSM.
8. **Advanced Details → IAM Role**:
   * Attach the IAM role: EC2-SSM-Role

### ✅ Finalize

* Click **Launch Instance**
* Wait until status shows **Running → 2/2 checks passed**

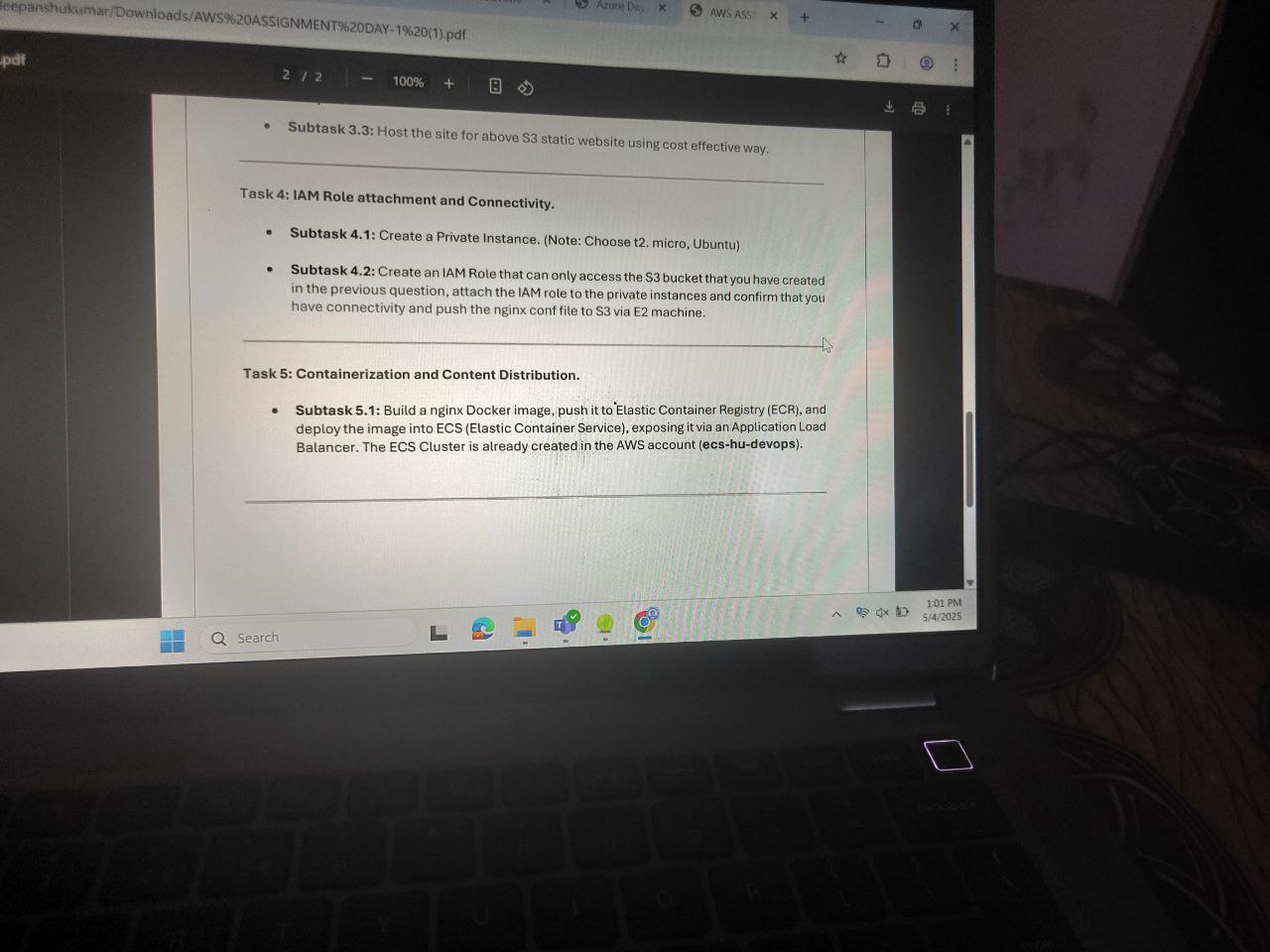
### 🧪 ****Test SSM Access****

1. Go to **EC2 → Instances**
2. Select your new instance
3. Click **Connect → Session Manager**
4. Start session

If everything’s configured right (SSM role + NAT connectivity), you’ll get shell access directly ✅

**Next Steps (EC2 via SSM with Ubuntu in Private Subnet):**

1. **Create an EC2 Instance**
   * AMI: **Ubuntu 22.04 LTS**
   * Instance type: t2.micro
   * Network: Select your **custom VPC**
   * Subnet: Select Private-subnet-deep
   * **Auto-assign Public IP**: **Disable** (since it's private)
   * IAM Role: Select or create one with AmazonSSMManagedInstanceCore
   * Storage: Default (8–10 GB EBS)
   * Tags: (optional)
   * Security Group:
     + Inbound rules: can be empty
     + Outbound: allow all
2. **IAM Role Setup** (if not done)
   * Go to **IAM > Roles > Create Role**
   * Use **EC2** as trusted entity
   * Attach policy: AmazonSSMManagedInstanceCore
   * Name: EC2-SSM-Role or similar
   * Attach it to your EC2 instance
3. **Enable SSM (Systems Manager)**
   * Go to **Systems Manager > Fleet Manager**
   * After instance boots, go to **Session Manager** and start a terminal session
   * No key pair or SSH needed



Great — let's begin **Task 5: Containerization and Content Distribution**, step-by-step.

### 📌 ****Task 5.1 Breakdown (from your image):****

Build an **nginx Docker image**, push it to **Elastic Container Registry (ECR)**, and deploy it into **ECS (Elastic Container Service)** using an **Application Load Balancer**.  
The ECS cluster is already created in AWS account (**ecs-hu-devops**) — but you said it's **not present**, so we'll create it.

## ✅ Step-by-Step Plan for Task 5:

### ****STEP 1: Create ECR Repository****

To store your Docker image (nginx):

1. Go to **Amazon ECR** → **Repositories** → Click **"Create repository"**
2. Name it: nginx-ecr
3. Visibility: **Private**
4. Leave rest as default → Create.

### ****STEP 2: Build and Push Docker Image to ECR****

On your local machine (or EC2 if using it), follow:

bash

CopyEdit

# 1. Authenticate Docker to your ECR

aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <your\_aws\_account\_id>.dkr.ecr.us-east-1.amazonaws.com

# 2. Create Dockerfile (if not already)

echo -e "FROM nginx\nCOPY . /usr/share/nginx/html" > Dockerfile

# 3. Build the image

docker build -t nginx-ecr .

# 4. Tag the image

docker tag nginx-ecr:latest <your\_aws\_account\_id>.dkr.ecr.us-east-1.amazonaws.com/nginx-ecr:latest

# 5. Push to ECR

docker push <your\_aws\_account\_id>.dkr.ecr.us-east-1.amazonaws.com/nginx-ecr:latest

✅ Replace <your\_aws\_account\_id> with your actual AWS account ID.

### ****STEP 3: Create ECS Cluster (since not pre-existing)****

1. Go to **ECS > Clusters > Create Cluster**
2. Choose **Networking only (Fargate)** → Next
3. Name: nginx-cluster
4. VPC: Select your existing VPC
5. Subnets: Choose **2 subnets** from different AZs (1 public, 1 private is OK)
6. Leave rest default → Create

### ****STEP 4: Create Task Definition****

1. Go to **ECS → Task Definitions** → Create new
2. Launch type: **Fargate**
3. Task name: nginx-task
4. Task role: Select an IAM role with ECS task execution permissions (or create one with AmazonECSTaskExecutionRolePolicy)
5. Container:
   * Name: nginx
   * Image: your\_aws\_account\_id.dkr.ecr.us-east-1.amazonaws.com/nginx-ecr:latest
   * Port mapping: 80
6. Click **Create**

### ****STEP 5: Create ECS Service (with ALB)****

1. ECS → Clusters → nginx-cluster → Services → Create
2. Launch type: **Fargate**
3. Task definition: Select nginx-task
4. Service name: nginx-service
5. Number of tasks: 1
6. Load balancer:
   * ✅ Enable
   * Choose **Application Load Balancer (ALB)**
   * Create new target group: nginx-target-group
   * Listener port: 80
   * Health check path: /
7. Security group: Allow **HTTP (port 80) from anywhere**
8. Deploy service

### ✅ FINAL OUTPUT:

You’ll get an **ALB DNS URL** → Access it in browser to see your nginx homepage.

Would you like to begin with **ECR repo creation**, or do you want help writing the full Dockerfile too?

Account ID 4130-9509-7811 - this is my account id - tell me what is need to authenicate now

###### ChatGPT said:

Perfect, thanks for sharing your **AWS Account ID**: 413095097811.  
Let’s now authenticate your **Docker CLI** to push the NGINX image to your **ECR repo**.

## ✅ Step-by-Step: ECR Authentication (for us-east-1 region)

### 🔧 1. First, log in to your ECR via Docker:

bash

CopyEdit

aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 413095097811.dkr.ecr.us-east-1.amazonaws.com

🔒 This command logs you into your private Elastic Container Registry securely using the AWS CLI.

### 🔐 Requirements:

Make sure:

* You're **running this on a system where AWS CLI is installed**.
* You're **already configured with aws configure** using IAM credentials that have **ECR permissions** like:
  + AmazonEC2ContainerRegistryFullAccess
  + or custom permissions to push/pull to ECR.

## ✅ What’s Next After Auth?

Once logged in successfully, follow these:

### 📦 2. Build your NGINX image:

bash

CopyEdit

docker build -t nginx-ecr .

### 🏷️ 3. Tag the image:

bash

CopyEdit

docker tag nginx-ecr:latest 413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-ecr:latest

Replace nginx-ecr with the actual name of your repo in ECR if it's different.

### ⬆️ 4. Push the image to ECR:

bash

CopyEdit

docker push 413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-ecr:latest

Let me know if you want a **ready-made Dockerfile** for your nginx setup or want help creating the ECR repository first.

##### You said:

###### ChatGPT said:

##### You said:

check now

###### ChatGPT said:

You're successfully authenticated with your Amazon ECR repository — the line Login Succeeded confirms that!

✅ **Next Steps for Task 5 (Subtask 5.1)**:

Now that you're logged in to Amazon ECR, follow these steps to complete Subtask 5.1:

### 🔧 Step-by-Step: Build & Push NGINX Docker Image

#### ✅ 1. ****Create a Dockerfile****

Create a Dockerfile in an empty folder with the following content:

Dockerfile

CopyEdit

FROM nginx:latest

COPY index.html /usr/share/nginx/html/index.html

Also create a simple index.html file in the same folder to test.

#### ✅ 2. ****Build the Docker Image****

In terminal, run:

bash

CopyEdit

docker build -t nginx-ecr-demo .

#### ✅ 3. ****Tag the Docker Image****

Use your account ID and region:

bash

CopyEdit

docker tag nginx-ecr-demo:latest 413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-ecr-demo

#### ✅ 4. ****Push the Docker Image to ECR****

bash

CopyEdit

docker push 413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-ecr-demo

After this, we’ll move on to deploying this image to ECS using a cluster and Application Load Balancer.

Do you want to proceed with creating the ECS cluster next?

##### You said:

 can you build light weight image just simple image to push

###### ChatGPT said:

Yes — you can use **nginx:alpine** instead of the full nginx:latest image. It’s a **much lighter version** (about 5 MB vs 133 MB+), ideal for quick testing and pushing to ECR.

### ✅ Updated Lightweight Dockerfile:

Run this:

bash

CopyEdit

echo "FROM nginx:alpine\nCOPY index.html /usr/share/nginx/html/index.html" > Dockerfile

And also create a basic HTML file:

bash

CopyEdit

echo "<h1>Lightweight NGINX from Alpine!</h1>" > index.html

### ✅ Then build and tag:

bash

CopyEdit

docker build -t nginx-light .

docker tag nginx-light:latest 413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-light

### ✅ Push to ECR:

bash

CopyEdit

docker push 413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-light

Let me know once pushed — then we’ll deploy it to ECS via the next steps.

##### You said:

echo "FROM nginx:alpine\nCOPY index.html /usr/share/nginx/html/index.html" > Dockerfile - in this just use image that is build in one minute - most simple dockerfile dont copy anything

###### ChatGPT said:

Got it. Here's the **most minimal Dockerfile** you can use — no copying, just pulling a super-light image that builds in **seconds**:

### ✅ Simplest Dockerfile (1-line, near-instant build):

bash

CopyEdit

echo "FROM nginx:alpine" > Dockerfile

### ✅ Build and tag it:

bash

CopyEdit

docker build -t nginx-light .

docker tag nginx-light:latest 413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-light

### ✅ Push to ECR:

bash

CopyEdit

docker push 413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-light

This will pull and push **nginx:alpine**, one of the smallest official NGINX images (~5MB). No files copied — just the image itself.

Ready to move on to ECS setup after this?

##### You said:

this is reportisy where you need to push - user121@DESKTOP-JFJ21I5 MINGW64 ~ $ docker tag nginx-light:latest 413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-light user121@DESKTOP-JFJ21I5 MINGW64 ~ $ docker push 413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-light Using default tag: latest The push refers to repository [413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-light] 0d853d50b128: Preparing 947e805a4ac7: Preparing 811a4dbbf4a5: Preparing b8d7d1d22634: Preparing e244aa659f61: Preparing c56f134d3805: Waiting d71eae0084c1: Waiting 08000c18d16d: Waiting name unknown: The repository with name 'nginx-light' does not exist in the registry with id '413095097811'

###### ChatGPT said:

The error you're getting:

**name unknown: The repository with name 'nginx-light' does not exist in the registry**

...is because your **actual ECR repository name is nginx-ecr**, not nginx-light.

### ✅ Fix: Tag your image using the ****correct repository name****:

Run the following two commands:

bash

CopyEdit

docker tag nginx-light:latest 413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-ecr

docker push 413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-ecr

After this, it will successfully push your nginx-light image into your nginx-ecr repository.

Want to proceed with ECS Cluster setup next?

##### You said:

now moved to next part of question as i have pushed the image

###### ChatGPT said:

Great! Since you’ve successfully pushed your image to **Amazon ECR**, you’re now ready to move to:

## ✅ **Task 5: Containerization and Content Distribution (ECS + ALB)**

Let’s break it down based on **Subtask 5.1**:

Build a nginx Docker image, push it to ECR (✅ Done), and deploy the image into **ECS**, exposing it via an **Application Load Balancer**.  
Note: ECS Cluster is already created in the AWS account (ecs-hu-devops).

### ✅ Step-by-Step Guide (Next Actions):

#### ****1. Go to ECS and confirm the cluster**** ecs-hu-devops ****exists:****

* AWS Console → ECS → Clusters
* Look for ecs-hu-devops
* If it's there, click it — we will deploy the service here.

#### ****2. Create a Task Definition (Fargate)****

1. Go to: **ECS > Task Definitions > Create new task definition**
2. **Launch type:** FARGATE
3. **Task name:** nginx-task
4. **Task Role:** Leave default (unless your app requires permissions)
5. **Container definitions:**
   * Click **Add container**
   * **Name:** nginx-container
   * **Image:** 413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-ecr
   * **Port mappings:** 80
6. Click **Add** → **Create**

#### ****3. Create a Service in ECS Cluster****

1. Go to your ECS Cluster: ecs-hu-devops
2. Click **Services → Create**
3. **Launch type:** Fargate
4. **Task Definition:** Select the nginx-task
5. **Service name:** nginx-service
6. **Number of tasks:** 1
7. Click **Next**

#### ****4. Configure Networking****

1. **Cluster VPC:** Choose default VPC (or your custom one)
2. **Subnets:** Select **2 public subnets**
3. **Security group:** Create a new one or use one that allows **port 80 (HTTP)** inbound
4. **Auto-assign public IP:** Yes

#### ****5. Attach an Application Load Balancer (ALB)****

1. Select **Application Load Balancer**
2. Create new ALB (or choose existing one)
3. Listener: HTTP:80
4. Create a new **target group** → Target type: ip
5. Register the ECS service with this target group

#### ****6. Review & Create****

* Review your settings
* Click **Create Service**
* Wait for deployment to complete

#### ✅ Final Output

Once everything is done:

* Go to **EC2 > Load Balancers**
* Copy the **DNS name** of the ALB
* Paste it in the browser — you should see the **default NGINX welcome page**.

Would you like me to give you the exact Fargate-compatible JSON for the task definition or help create the ALB step-by-step?

##### You said:

creating a cluster now

###### ChatGPT said:

Perfect — you’re on the right track!

Since you're creating a **new ECS cluster** (deepanshu-cluster) with **Fargate** enabled, here's what to do **immediately after** creating the cluster:

### ✅ Next Steps After Cluster Creation

#### ****1. Create a Task Definition (Fargate-compatible)****

Go to **ECS > Task Definitions > Create new task definition**:

* **Launch Type**: Fargate
* **Task Name**: nginx-task
* **Task Role**: None (or default if required)
* **Task memory**: 0.5 GB
* **CPU**: 0.25 vCPU

👉 **Add container**:

* **Container name**: nginx-container
* **Image URI**:

bash

CopyEdit

413095097811.dkr.ecr.us-east-1.amazonaws.com/nginx-ecr:latest

* **Port mappings**: 80 (container port)

Click **Add**, then **Create task definition**.

#### ****2. Deploy a Service to Your Cluster****

Go to:  
**ECS > Clusters > deepanshu-cluster > Services > Create**

* **Launch Type**: Fargate
* **Task Definition**: nginx-task
* **Service Name**: nginx-service
* **Number of Tasks**: 1
* **Deployment type**: Rolling update

#### ****3. Networking Configuration****

* **VPC**: Use your VPC (or default one)
* **Subnets**: Choose 2 public subnets
* **Security Group**: One that allows **inbound HTTP (port 80)**
* **Auto-assign public IP**: ENABLE it ✅

#### ****4. (Optional but Recommended) Attach Load Balancer****

* **Type**: Application Load Balancer
* If needed, create new ALB with:
  + Listener: HTTP on port 80
  + New target group (type: ip)
  + ECS will register your container to this ALB

#### ✅ Final Step: Access Your App

After ECS service is deployed:

* Go to **EC2 > Load Balancers**
* Copy the **DNS name** of the ALB
* Open it in browser — you should see the **NGINX Welcome Page**

Let me know if you want:

* Help with JSON for task definition
* Help configuring the ALB
* Or skipping ALB and using public IP of the task directly for quick test

##### You said:

check this

###### ChatGPT said:

Nice! 🎯 Your ECS service deep-task-service-xpx9jlic is **Active** and **Running 1 task**, which means the container is successfully launched.

### ✅ Next, check connectivity:

Let’s make sure you can access the app in your browser.

#### 🧩 Did you do any of the following?

1. **Enable public IP** for the task in networking settings?
2. Or attach it to an **Application Load Balancer**?

### 🔍 Option 1: Without Load Balancer (quick check)

1. Go to **ECS > Clusters > deepanshu-cluster > Tasks**
2. Click the **running task**
3. In the task detail page, scroll down to **Networking**
4. Check if it has a **public IP** assigned

If yes: copy the public IP, open in browser → should show NGINX welcome page.

### 🧩 Option 2: If you used Load Balancer:

1. Go to **EC2 > Load Balancers**
2. Click the ALB that was created
3. Copy its **DNS name**
4. Paste it in your browser → NGINX page should open.

Let me know:

* Which option you used
* If it’s not opening, I’ll help debug security group, port 80, or target group registration.

##### You said:

 i am using task public ip

###### ChatGPT said:

Perfect, thanks! Since you’re using the **public IP of the ECS task**, and seeing ERR\_CONNECTION\_TIMED\_OUT, here’s a step-by-step checklist to fix it:

### ✅ QUICK FIX CHECKLIST for Fargate Public IP Access:

#### 1. ****Public IP enabled****

* Go to: ECS > Clusters > your-task > Networking
* Check if the **ENI (elastic network interface)** has a **public IPv4 address** (you already did, so ✅)

#### 2. ****Security Group allows port 80****

* Go to: **EC2 > Network Interfaces**
* Filter by your **task ENI** (you’ll find this in ECS task networking section)
* Click the **ENI → View Security Groups**
* Edit the security group:
  + Inbound rules:
    - **Type:** HTTP
    - **Port:** 80
    - **Source:** 0.0.0.0/0

✅ If this isn’t added, the browser cannot connect.

#### 3. ****Task definition port mapping****

* Check your ECS **task definition** → container section:
  + Make sure **port 80 is exposed**
  + Example:
    - Container port: 80
    - Protocol: tcp
    - Host port: (leave default for Fargate)
  + ✅ Ensure this matches your app (nginx default = port 80)

#### 4. ****VPC/Subnet supports internet****

* Ensure the ECS task is running in a **public subnet**
  + The subnet must have:
    - **Auto-assign public IP: enabled**
    - Route to **Internet Gateway**

Once all the above are checked, it should work fine.  
Reload http://34.224.87.192/ after fixing.

Let me know if you want help:

* Finding the task’s ENI
* Editing the security group
* Or re-checking port exposure in task definition

Top of Form



Bottom of Form