**Task 1: Write a blog on the AWS user case studies**

* **Objective:**  
  Understand how top companies are using AWS services like EC2, S3, Lambda, and RDS to scale applications and reduce infrastructure costs.
* **Tools:**  
  AWS case studies (official AWS site), Medium/WordPress or PDF for writing.
* **Flow:**
  1. Visit AWS Case Studies.
  2. Choose examples from companies like Netflix, Airbnb, Samsung, or NASA.
  3. Focus on what services they used and what benefits they got (cost, scalability, security).
* **Output:**  
  A blog or report summarizing real-world AWS use cases.

**✅ Task 2: With the help of Boto3, launch and terminate EC2 instance**

* **Objective:**  
  Use Python (Boto3) to programmatically launch and stop EC2 instances.
* **Tools:**  
  Boto3, Python, IAM User with EC2 permissions, AWS CLI configured.
* **Code Flow:**

python

CopyEdit

import boto3

ec2 = boto3.resource('ec2')

# Launch Instance

instance = ec2.create\_instances(

ImageId='ami-0abcdef1234567890', # Use a real AMI

MinCount=1,

MaxCount=1,

InstanceType='t2.micro',

KeyName='your-key-name'

)

print(f'Launched EC2 instance with ID: {instance[0].id}')

# Terminate Instance

instance[0].terminate()

print("Terminated instance.")

* **Output:**  
  Instance created and then terminated using Python code.

**✅ Task 3: Create a Boto3 code to access logs from CloudWatch**

* **Objective:**  
  Retrieve logs from CloudWatch log groups for monitoring and debugging.
* **Tools:**  
  Boto3, CloudWatch log groups, Python.
* **Code Flow:**

python

CopyEdit

import boto3

client = boto3.client('logs')

response = client.get\_log\_events(

logGroupName='your-log-group',

logStreamName='your-log-stream',

startFromHead=True

)

for event in response['events']:

print(event['message'])

* **Output:**  
  You’ll see printed logs from a specific log stream in your terminal.

**✅ Task 4: Find an AMI with RHEL GUI, install VirtualBox and create a VM**

* **Objective:**  
  Create a virtualized environment inside an EC2 instance.
* **Tools:**  
  AWS EC2, RHEL AMI with GUI, VirtualBox, yum installer.
* **Flow:**
  1. Launch EC2 using RHEL 9 AMI with GUI (search in AWS marketplace).
  2. SSH into the instance.
  3. Install dependencies and VirtualBox:

sh

CopyEdit

sudo yum update -y

sudo yum install -y kernel-devel gcc make perl

sudo yum install -y VirtualBox

* 1. Create a new VM inside VirtualBox.
* **Output:**  
  EC2 runs GUI and hosts VirtualBox with a VM inside.

**✅ Task 5: Study different storage classes of S3 and create a blog**

* **Objective:**  
  Understand cost vs. availability tradeoffs across S3 storage classes.
* **Storage Classes to Study:**
  + S3 Standard
  + S3 Intelligent-Tiering
  + S3 Standard-IA (Infrequent Access)
  + S3 One Zone-IA
  + S3 Glacier / Glacier Deep Archive
* **Flow:**
  + Go to AWS documentation on S3 storage classes.
  + Compare cost, availability, retrieval time.
  + Add diagrams or tables for clarity.
* **Output:**  
  A blog comparing storage classes with real use cases.

**✅ Task 6: Explore AWS Q service and create a blog on it**

* **Objective:**  
  Learn about AWS Q – a generative AI-powered assistant for developers and businesses.
* **Tools:**  
  AWS Q console, blog tools (Medium, Notion, Markdown)
* **Flow:**
  1. Explore AWS Q features on the official AWS Q page.
  2. Understand how it helps in code generation, documentation, and chat-like support.
  3. Try hands-on in AWS Console (if available to you).
* **Output:**  
  Blog explaining what AWS Q is, use cases, and potential impact.

**✅ Task 7: Create event-driven architecture for audio-to-text using AWS Transcribe**

* **Objective:**  
  Automate transcription using AWS Transcribe when a file is uploaded to S3.
* **Tools:**  
  AWS S3, Lambda, AWS Transcribe, Python, IAM Roles, EventBridge.
* **Code Flow:**
  1. **S3 Bucket:** Upload audio file (WAV/MP3).
  2. **Event Trigger:** Lambda function triggered on S3 upload.
  3. **Lambda Code:**

python

CopyEdit

import boto3

def lambda\_handler(event, context):

transcribe = boto3.client('transcribe')

job\_name = 'audio-job'

file\_uri = 's3://your-bucket/audio.mp3'

transcribe.start\_transcription\_job(

TranscriptionJobName=job\_name,

Media={'MediaFileUri': file\_uri},

MediaFormat='mp3',

LanguageCode='en-US'

)

* **Output:**  
  Auto-transcribed text file in S3 after upload.

**✅ Task 8: Connect Python to MongoDB service of AWS using Lambda**

* **Objective:**  
  Integrate AWS Lambda with MongoDB (e.g., MongoDB Atlas) for serverless access.
* **Tools:**  
  AWS Lambda, Python, pymongo, MongoDB Atlas.
* **Code Flow:**
  1. Create a MongoDB Atlas cluster and whitelist IPs.
  2. Use a Python Lambda Layer with pymongo.
  3. Sample Lambda Code:

python

CopyEdit

from pymongo import MongoClient

def lambda\_handler(event, context):

client = MongoClient("your-mongo-uri")

db = client['test']

result = db.collection.find\_one()

print(result)

* **Output:**  
  Data fetched from MongoDB within Lambda execution.

**✅ Task 9: Upload object to S3 without logging into AWS Console**

* **Objective:**  
  Allow uploads to S3 without using the AWS console using pre-signed URLs or CLI.
* **Methods:**
  1. **Using Pre-Signed URL (most common):**
     + Generate a pre-signed URL:

python

CopyEdit

import boto3

s3 = boto3.client('s3')

url = s3.generate\_presigned\_url('put\_object',

Params={'Bucket': 'your-bucket', 'Key': 'yourfile.txt'},

ExpiresIn=3600)

print(url)

* + - Use curl to upload:

sh

CopyEdit

curl -X PUT -T file.txt "https://signed-url"

* 1. **Using AWS CLI:**

sh

CopyEdit

aws s3 cp file.txt s3://your-bucket-name/ --profile myprofile

* **Output:**  
  File uploaded to S3 without manual AWS console login.