IAM & account hardening (do this *before* creating resources)

Best practice: don’t use root account for daily tasks.

Console steps

1. Sign in as root → IAM Console → Users → Create user:
    - Username: `cloudops-admin`
    - Access type: **Console access** (enable password)
2. Attach permission: **AdministratorAccess** (ok for learning; in production you'd restrict).
3. Create an IAM group `CloudOps-Limited-Access` with a custom policy later (we’ll use it in scenarios).

Authenticating using IAM user credentials for the AWS CLI

Get your access keys

1. Sign in to the AWS Management Console and open the IAM console.
2. In the IAM console, select **Users**, then select the desired **User name**.
3. On the user's page, go to **Security credentials**.
4. Under **Access keys**, select **Create access key**.
5. In Step 1, choose **Command Line Interface (CLI)**.
6. In Step 2, optionally enter a tag and select **Next**.
7. In Step 3, select **Download .csv file** to save the access key and secret access key. This information is needed later.
8. Select **Done**.

Minimal VPC / Networking (use default VPC to keep it simple & free)

For free-tier labs use the **default VPC** that AWS provides. It already has public subnets and an Internet Gateway. Saves cost and complexity.

Check default VPC

- Console → VPC → Your VPCs → ensure there is a default VPC in `ap-south-1`.

(not creating as there is already 1 given for the console)

Create a t2.micro EC2 (web server + lab machine)

This will be your main lab VM for scenarios.

Console steps

1. EC2 Console → Launch Instances → Launch Instance.
2. Name: `cloudops-lab-ec2`
3. Amazon Machine Image: **Amazon Linux 2023 / Amazon Linux 2**
4. Instance type: **t2.micro** (free tier)
5. Key pair: Create or select an existing key pair (download `.pem`).
6. Network: Default VPC, auto-assign public IPv4 = enabled.
7. Security group:
    - Inbound: SSH (22) from your IP only, HTTP (80) from anywhere (0.0.0.0/0) for the static site.
8. Storage: 8 GiB (default) — free tier.

User data script (installs nginx, git, stress):
Paste this into the User Data field (bash):

```
#!/bin/bash
yum update -y
yum install -y httpd git
systemctl enable httpd
systemctl start httpd
echo "<h1>CloudOps Lab - $(hostname)</h1>" > /var/www/html/index.html
# optionally install stress for CPU load tests
amazon-linux-extras install -y epel
yum install -y stress
```

Connect via SSH

```
ssh -i ~/keys/mykey.pem ec2-user@<PublicIPv4>
```

Create S3 bucket (test uploads + permission scenarios)

Console steps

1. S3 → Create bucket
    - Bucket name: `cloudopslab-2025` (must be globally unique)

    - Region: `ap-south-1`
    - Uncheck "Block all public access" only if you need public access for testing; otherwise leave default (recommended to keep blocked).

Create SNS topic for alerts & subscribe email

Console steps

1. SNS Console → Topics → Create topic → Standard
    - Name: `cloudops-alerts`
2. SNS Console → subscription → Protocol: Email → Endpoint: <your email> → Confirm subscription from your email.(check spams too)

CloudWatch: dashboard & alarm (EC2 CPU)

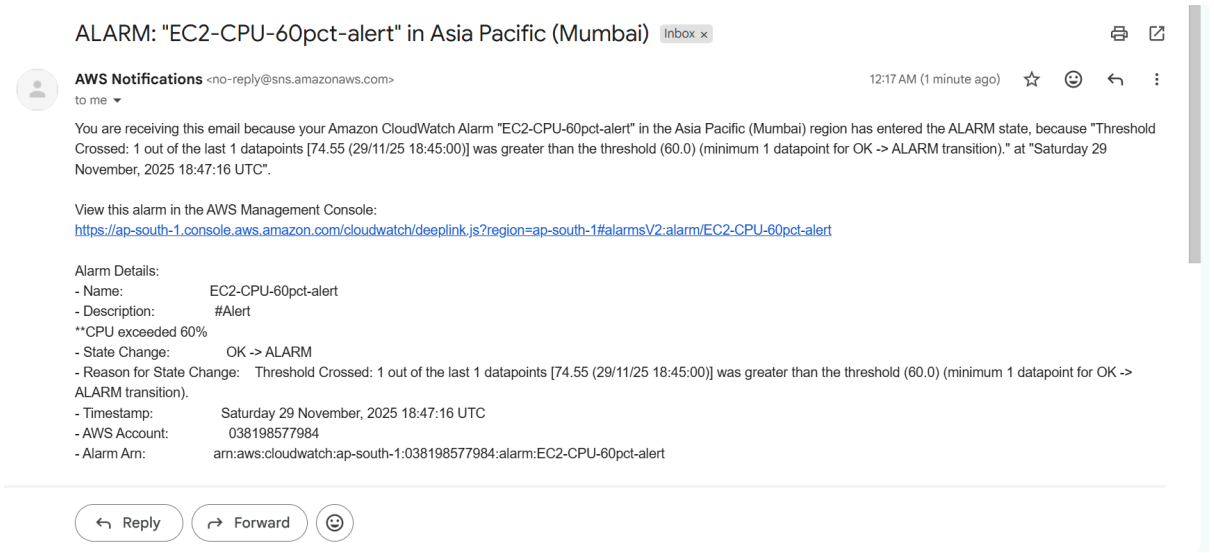
Goal: trigger email when CPU > 60% for 1 min (for simulation with `stress`).

Create alarm (console)

- CloudWatch → Alarms → Create alarm → Select metric → EC2 → Per-Instance Metrics → CPUUtilization for your instance → Next.
- Conditions: Threshold type: Static → Whenever `CPUUtilization` > 60% for 1 consecutive period (1 min).
- Notification: Select SNS topic `cloudops-alerts`.
- Name: `EC2-CPU-60pct-alert`.

Test (to trigger)
SSH into the EC2 and run:

```
# run for 120 seconds
stress --cpu 2 --timeout 120
```



Systems Manager (SSM) – enable for remote run commands (no SSH needed)

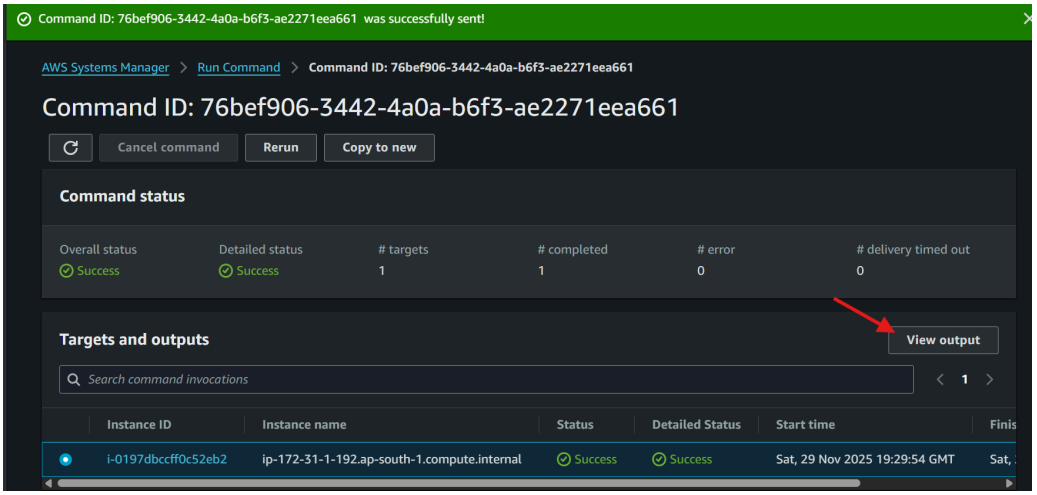
SSM Agent is preinstalled on Amazon Linux. Attach an IAM role to the EC2 with `AmazonSSMManagedInstanceCore`.

Console steps

1. IAM → Roles → Create role → EC2 → Attach `AmazonSSMManagedInstanceCore`.
2. EC2 → Instances → Actions → Security → Modify IAM role → Attach the new role.
3. Open Systems Manager → Fleet Manager or Managed Instances (left menu).

You should see your instance appear (may take ~30–60 seconds).
If it shows, the instance is registered to SSM and ready.

4. Run a simple command via Run Command (Console & CLI)
    - **Console (quick):**
    - Systems Manager → Run Command → Run command.
    - Select `AWS-RunShellScript`.
    - Target your instance.
    - In Commands, paste: `whoami`, `uptime`, `df -h /var`.
    - Click Run.
    - After completion, view results via command ID → Outputs.



5. Open an interactive shell using Session Manager (no SSH)
    - Systems Manager → **Session Manager** → **Start session**.
    - Select the instance → **Start session**.
    - A browser-based terminal opens and you're in a shell on the instance.
6. Open an interactive shell using Session Manager (no SSH)

Create Lambda Function for Daily Health Check (Short Version)

Objective:

Create a Lambda function that can send a basic health-check message to an SNS topic. This will later be expanded to full CloudOps reporting.

— Create IAM Role for Lambda

1. Go to **IAM Console → Roles → Create role**.
2. Select:
    - **Trusted entity:** AWS Service
    - **Use case:** Lambda
3. Attach these AWS managed policies:
    - `AmazonEC2ReadOnlyAccess`
    - `IAMReadOnlyAccess`
    - `CloudWatchReadOnlyAccess`
    - `AmazonSNSFullAccess`
    - `AWSLambdaBasicExecutionRole`
4. Name the role: **lambda-cloudops-health-role**.
5. Click **Create role**.

— Create Lambda Function

1. Go to **Lambda Console → Create function**.
2. Choose:
    - **Author from scratch**
    - Name: `cloudops-health-check`
    - Runtime: **Python 3.11**
    - Architecture: **x86_64**
3. Under **Permissions**, select:
    - **Use existing role**
    - Choose: `lambda-cloudops-health-role`
4. Click **Create function**.

— Add Environment Variable

1. Open your Lambda function → **Configuration tab**.
2. Select **Environment variables → Edit → Add variable**.
3. Add:
    - **Key:** `SNS_TOPIC_ARN`
    - **Value:** *your SNS topic ARN*
4. Click **Save**.

— Add Lambda Code

Navigate to the **Code** tab → replace default code with:

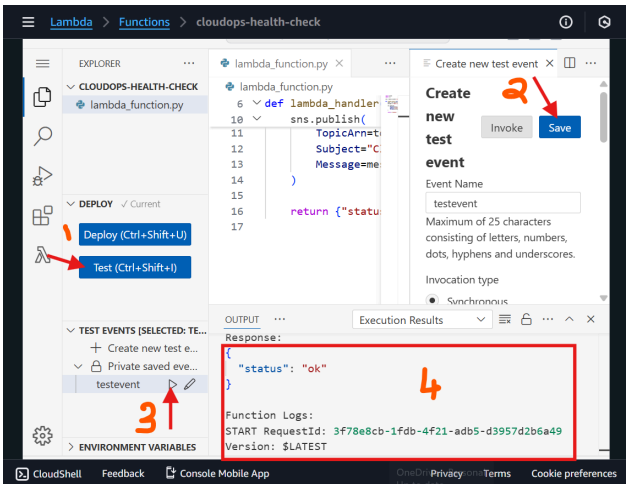
```
import boto3
import os
```

```
sns = boto3.client('sns')

def lambda_handler(event, context):
    topic = os.environ.get("SNS_TOPIC_ARN")
    message = "Daily CloudOps Health Check: Lambda is working successfully."

    sns.publish(
        TopicArn=topic,
        Subject="CloudOps Health Report",
        Message=message
    )

    return {"status": "ok"}
```

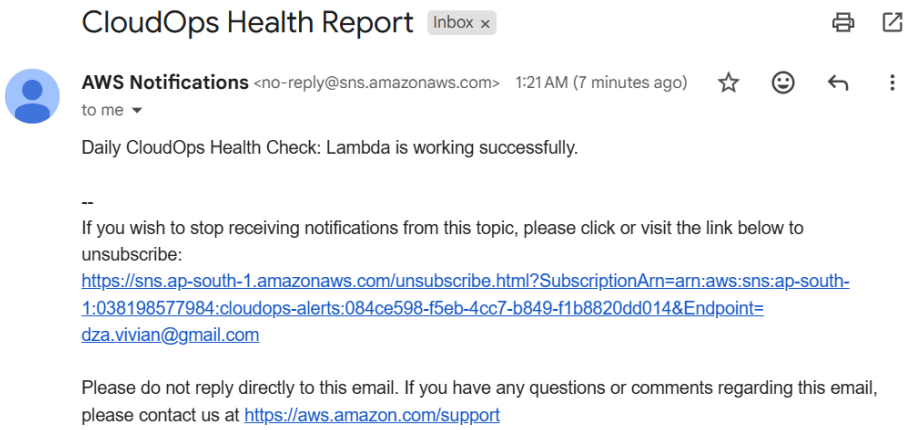
Click **Deploy**.

— Test Lambda

1. Click **Test** → Create test event (use default JSON).
2. Run test.
3. You should receive an email from SNS confirming the health check.



Test Result:



CI/CD (GitHub Actions) — deploy a static site to EC2 via SSH

This is a minimal pipeline to show DevOps basics.

Prereqs

- GitHub repo with website files.
- In EC2 create a deploy user `deployer` and add your public key to `~deployer/.ssh/authorized_keys`.
- On GitHub repo → Settings → Secrets → Add `EC2_HOST`, `EC2_USER` (deployer), `EC2_SSH_KEY` (private key as secret).

Sample `.github/workflows/deploy.yml`:

```
name: Deploy to EC2
on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Setup SSH
```

```
        run: |
          mkdir -p ~/.ssh
          echo "${{ secrets.EC2_SSH_KEY }}" > ~/.ssh/id_rsa
          chmod 600 ~/.ssh/id_rsa
          ssh-keyscan -H ${{ secrets.EC2_HOST }} >> ~/.ssh/known_hosts
      - name: Sync files to EC2
        run: |
          rsync -avz --delete ./ ec2-user@${{ secrets.EC2_HOST }}:/var/www/html/
```

(Change `ec2-user` if using Amazon Linux; adjust path and user accordingly.)

Test: Make a commit to `main`, watch Actions, see files on EC2.

Cleanup: Remove secrets, stop pipelines, terminate EC2.

STEP 1: Create GitHub Repo

1. Go to `[https://github.com](https://github.com)`
2. Click **New Repository**.
3. Name it: **cloudops-ci-cd-demo**.
4. Choose **Public**.
5. Click **Create Repository**.

STEP 2: Add `index.html`

Create `index.html` on your computer:

```
<h1>AWS CloudOps CI/CD Demo</h1>
<p>Deployment Successful!</p>
```

Then: Go to your GitHub repo → Upload → select `index.html` → Commit.

STEP 3: Create `deployer` Linux User on EC2

SSH into EC2 and run:

```
sudo adduser deployer
sudo mkdir -p /home/deployer/.ssh
sudo chmod 700 /home/deployer/.ssh
```

STEP 4: Create SSH Key Pair on YOUR PC

On your local terminal:

```
ssh-keygen -t rsa -b 4096 -C "deploy-key"
```

When prompted for filename, type: `id_rsa_github`

(This creates `id_rsa_github` (private) and `id_rsa_github.pub` (public)).

STEP 5: Copy PUBLIC KEY to EC2

On your local machine: `cat id_rsa_github.pub` (copy the output).

SSH into EC2:
`sudo nano /home/deployer/.ssh/authorized_keys` (Paste the public key).
`sudo chown -R deployer:deployer /home/deployer/.ssh`
`sudo chmod 600 /home/deployer/.ssh/authorized_keys`

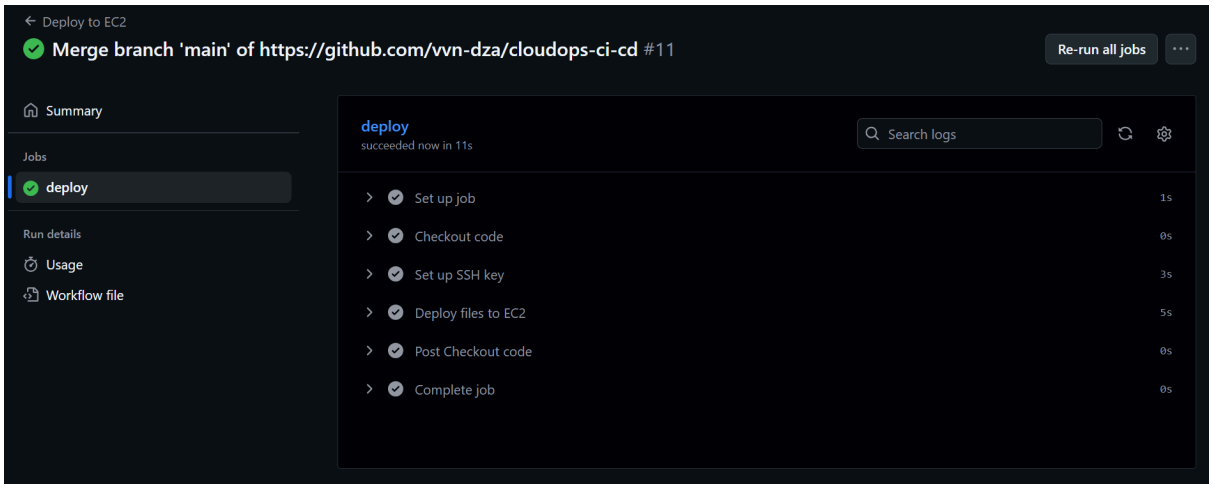
STEP 6: Give `deployer` access to Apache folder

`sudo chown -R deployer:deployer /var/www/html`**STEP 7: Add Secrets in GitHub**
Go to GitHub Repo → Settings → Secrets and Variables → **Actions** → New Repository Secret.

Create 3 secrets:

1. **EC2_HOST**: Your EC2 Public IP (e.g., 3.110.15.20)
2. **EC2_USER**: `deployer`
3. **EC2_SSH_KEY**: Content of your **private key** file (`cat id_rsa_github`).

STEP 8: Add GitHub Actions Workflow



Deployment Test

Home Services About Contact

This was deployed automatically!

Welcome to Our Modern Design

A simple, clean, and responsive layout using inline styles.