



Auto-Scaling Local Virtual Machines to AWS Based on Resource Utilization

-- Prof. Sumit Kalra

Submitted By: **G24AI1009**

Name: Nitin Awasthi

Date: 21-March-2025

Contents

| | | |
|-----|---|----|
| 1. | Objective | 3 |
| 2. | Introduction | 3 |
| 3. | Architecture of VM Monitoring & Auto-Scaling to AWS | 4 |
| 4. | Installation of Virtual Box and VM | 5 |
| 4.2 | Various Steps Involved During Installation | 6 |
| 5. | Downloading Ubuntu 24.04.1 LTS File for Installation | 7 |
| 6. | Creation of Virtual Machine | 7 |
| 6.1 | Configuring Virtual Machine | 8 |
| 7. | Creation of AWS Consol | 10 |
| 7.1 | Steps to Perform in AWS for Auto-Scaling Project | 10 |
| 7.2 | Steps Create a Key Pair | 12 |
| 7.3 | Steps Create a Security Group | 12 |
| 7.4 | Find a Valid AMI ID | 13 |
| 7.5 | Configure IAM Permissions (Optional but Recommended) | 13 |
| 8. | Installation & Configuration AWS CLI in VM | 14 |
| 9. | Implementation of resource monitoring using custom script | 14 |
| 10. | Pushing the VM to Trigger AWS Scaling (Loading and testing the application) | 16 |

Figures

| | | |
|------------|--|----|
| Figure 1: | Workflow Diagram of Resource-Aware Auto-Scaling from Local VM to AWS EC2 | 4 |
| Figure 2: | Google Search Result for Oracle VirtualBox Download Page | 5 |
| Figure 3: | Steps for Downloading and Installing VirtualBox on Windows | 6 |
| Figure 4: | Downloading Ubuntu 24.04.1 LTS ISO File for Virtual Machine Installation | 7 |
| Figure 5: | Oracle VirtualBox Manager Welcome Screen and Experience Mode Selection | 7 |
| Figure 6: | Creating a New Virtual Machine – Setting Name and Operating System | 8 |
| Figure 7: | Creating a Virtual Hard Disk for the New Virtual Machine | 8 |
| Figure 8: | Virtual Machine Configuration Summary in Oracle VirtualBox Manager | 9 |
| Figure 9: | Ubuntu Desktop Environment Successfully Loaded in Virtual Machine | 9 |
| Figure 10: | Google Search Results for Accessing AWS Management Console | 10 |
| Figure 11: | IAM User Login Screen for AWS Management Console Access | 11 |
| Figure 12: | AWS Console Home Page Showing User Dashboard and Recently Visited Services | 11 |
| Figure 13: | Key Pair Management in AWS EC2 Console | 12 |
| Figure 14: | Security Group Configuration Section in AWS EC2 Console | 12 |
| Figure 15: | Viewing and Selecting AMI IDs from the AWS EC2 AMI Library | 13 |
| Figure 16: | EC2 Instances List Showing Auto-Scaled and Running Instances in AWS Console | 13 |
| Figure 17: | Verifying AWS CLI Configuration Using aws configure list Command | 14 |
| Figure 18: | Python script to monitor CPU and RAM Usage | 15 |
| Figure 19: | Running Python Monitoring Script to Track CPU and RAM Usage in Real Time | 15 |
| Figure 20: | Attempt to Simulate CPU Load Using Background Process in Terminal | 16 |
| Figure 21: | EC2 Instance Automatically Launched After CPU Threshold is Exceeded | 16 |
| Figure 22: | Newly Launched EC2 Instance Confirmed in AWS Console After Auto-Scaling Triggers | 16 |

1. Objective

Create a local VM and implement a mechanism to monitor resource usage. Configure it to auto-scale to a public cloud (e.g., GCP, AWS, or Azure) when resource usage exceeds 75%.

2. Introduction

In this project, I worked on building a simple but powerful system that automatically scales from a local virtual machine to the cloud — specifically, AWS — based on system resource usage.

The idea is pretty straightforward: I have an Ubuntu VM running locally, and I wrote a Python script that keeps an eye on its CPU and RAM. If either one goes above 75%, the script automatically launches an EC2 instance on AWS. It's like giving the system the ability to call for backup when it's overloaded.

What's cool about this approach is that it starts small — using local resources — and only reaches out to the cloud when it's actually needed. That makes it efficient and cost-effective. I also added the option to deploy a simple Flask app on the EC2 instance to show that it's ready to handle real workloads.

Overall, this project combines local monitoring with cloud automation, using tools like psutil, boto3, and a bit of scripting to bridge the gap between on-prem and cloud infrastructure. In the coming slides we will be discussing about following points

- 1) Step-by-Step Instructions for Implementation:
- 2) Creation of a local VM (using VirtualBox, VMware, or similar).
- 3) Implementation of resource monitoring (using tools like Prometheus, Grafana, or a custom script).
- 4) Configuration of cloud auto-scaling policies and resource migration.
- 5) Deployment of a sample application to demonstrate the process.

I will explain my document to you in parts. First, I will explain the objective using an architecture diagram. Then, step by step, I will explain how I implemented it practically — such as how I installed the VM, how I created a login in the AWS Console, how I configured the AWS Command Line through authentication and access keys, how I ran the Python monitoring script, how I simulated the load, and finally, how the output was generated.

3. Architecture of VM Monitoring & Auto-Scaling to AWS

This diagram illustrates the workflow of a resource-aware auto-scaling system. It starts with a local virtual machine (Ubuntu on VirtualBox), where a monitoring service (via Cron or systemd) runs a Python script using psutil or shell commands to collect system metrics like CPU, RAM, and disk usage. If resource usage exceeds the defined threshold (e.g., CPU/RAM > 75%), the system triggers an action. It uses Boto3 or AWS CLI to call the AWS API, which then launches a new EC2 instance using a pre-configured AMI.

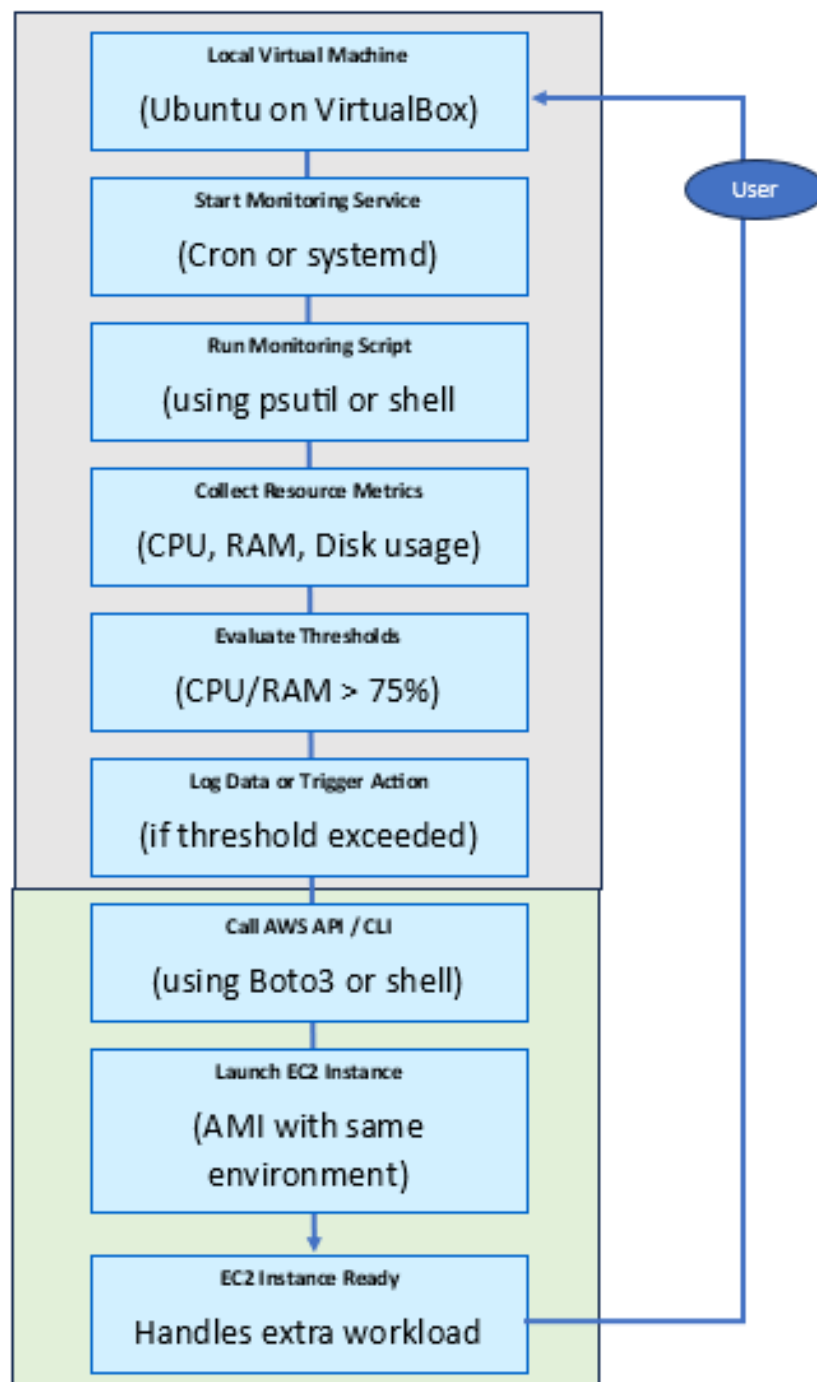


Figure 1: Workflow Diagram of Resource-Aware Auto-Scaling from Local VM to AWS EC2

4. Installation of Virtual Box and VM

To install VirtualBox, start by visiting the official website at <https://www.virtualbox.org>. Click on the "Downloads" section and choose the version that matches your operating system—whether it's Windows, macOS, or Linux.

Once downloaded, run the installer file. For Windows, it will be a .exe file; for macOS, a .dmg; and for Linux, either a .deb or .rpm package depending on your distro.

Follow the instructions in the setup wizard. Unless you have specific needs, it's best to go with the default settings.

The various steps involved in the installation process will be shown below:

- 1) To install VirtualBox, start by visiting the official website at <https://www.virtualbox.org>. Click on the "Downloads" section and choose the version that matches your operating system—whether it's Windows, macOS, or Linux.
- 2) Once downloaded, run the installer file. For Windows, it will be a .exe file; for macOS, a .dmg; and for Linux, either a .deb or .rpm package depending on your distro.
- 3) Follow the instructions in the setup wizard. Unless you have specific needs, it's best to go with the default settings.
- 4) The various steps involved in the installation process will be shown in the next slide.

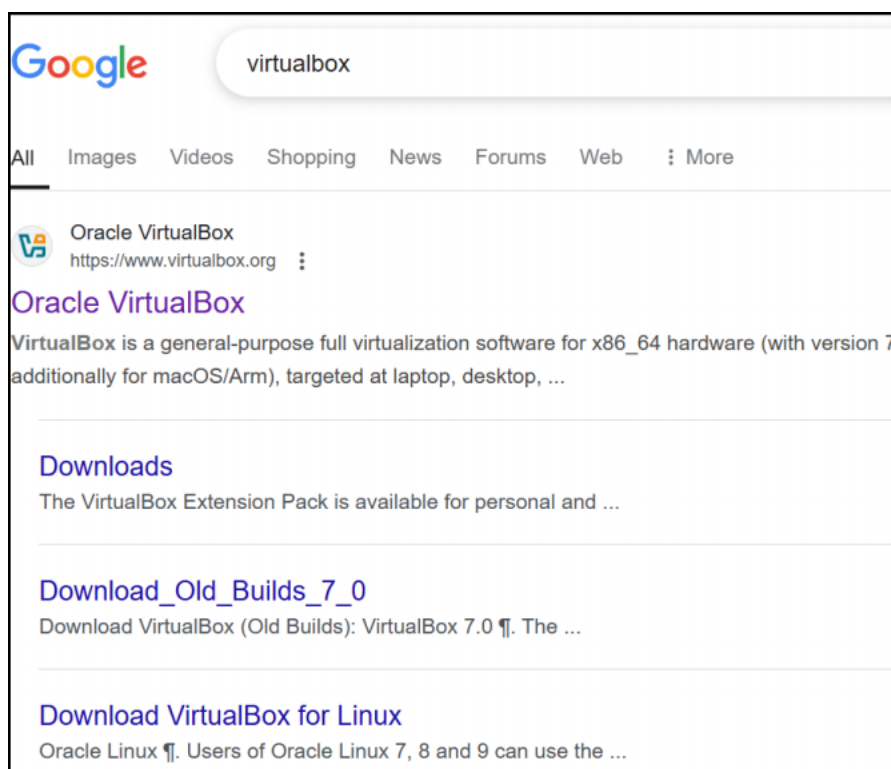


Figure 2: Google Search Result for Oracle VirtualBox Download Page

4.2 Various Steps Involved During Installation

Following are the steps performed for the installation of Virtual Box:

- 1) **Select Your Platform:** Visit the VirtualBox website and click on “Windows hosts” under the Platform Packages section to download the Windows installer.
- 2) **Start the Installer:** After downloading, double-click the .exe file to launch the setup wizard.
- 3) **Welcome Screen:** The installer opens with a welcome message. Click Next to continue the installation process.
- 4) **Custom Setup Options (Optional):** You may be shown options to customize the installation path or components. In most cases, just click Next to proceed with default settings.
- 5) **Ready to Install:** You’ll see a confirmation screen that the setup is ready to begin. Click the Install button to start the installation.
- 6) **Installation in Progress:** The setup will install VirtualBox on your system. This might take a few moments.
- 7) **Launch VirtualBox:** Once installed, VirtualBox will open with the main interface. It welcomes you and asks you to choose an experience mode:

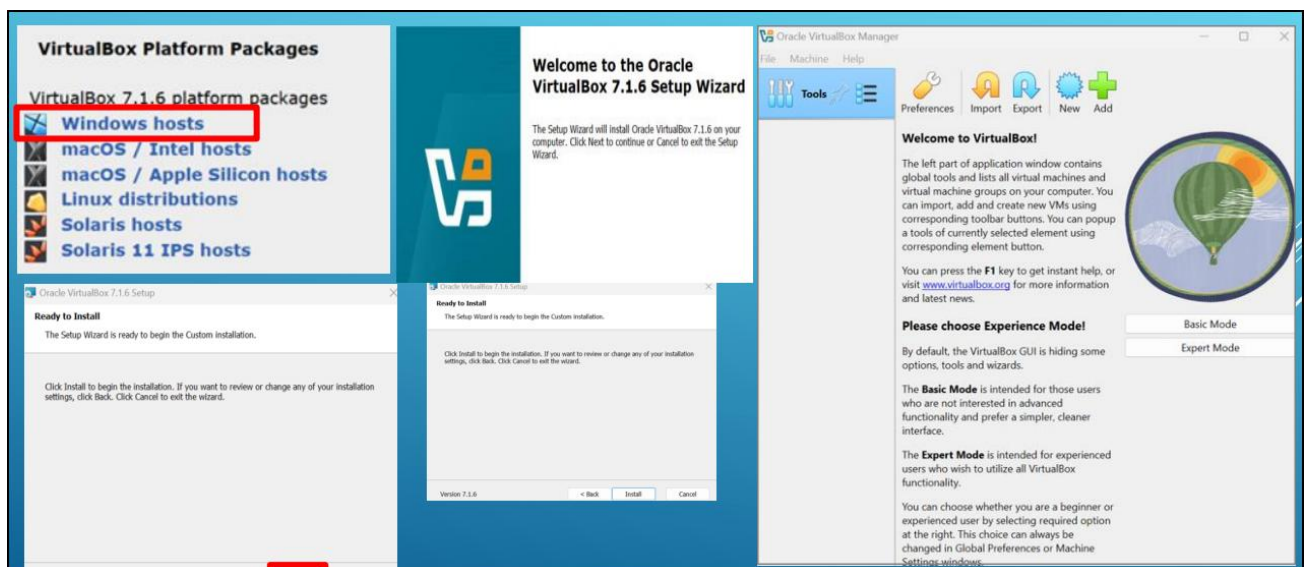


Figure 3: Steps for Downloading and Installing VirtualBox on Windows

5. Downloading Ubuntu 24.04.1 LTS File for Installation

The Ubuntu 24.04.1 LTS ISO file is being downloaded from the official Ubuntu website. This ISO file is essential for installing the Ubuntu operating system inside a virtual machine. It serves as the bootable image that VirtualBox will use to create the VM environment. The image also lists the system requirements for running Ubuntu, such as a dual-core processor, 4 GB of RAM, and sufficient disk space. Once the download is complete, the ISO file will be used to set up the virtual machine in VirtualBox.

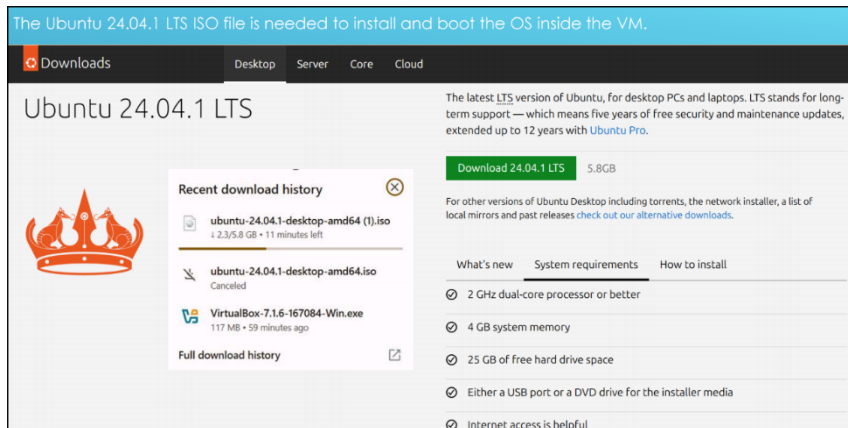


Figure 4: Downloading Ubuntu 24.04.1 LTS ISO File for Virtual Machine Installation

6. Creation of Virtual Machine

This screen represents the Welcome page of Oracle VirtualBox Manager, which serves as a starting point for new users by offering guidance and key options. It begins with an introduction to VirtualBox, explaining that it's a tool used to manage virtual machines. Users can easily import, add, or create new VMs using the toolbar buttons provided.

- I. Basic Mode offers a simplified interface with fewer options, suitable for beginners.
- II. Expert Mode provides full functionality for advanced users who want complete control.



Figure 5: Oracle VirtualBox Manager Welcome Screen and Experience Mode Selection

6.1 Configuring Virtual Machine

The image illustrates the basic steps involved in creating a new virtual machine in VirtualBox for installing Ubuntu 24.04.1 LTS. Name is assigned to the VM (e.g., “Virtual Machine-1”), and a save location is selected — either the default folder or a custom directory. Next, the Ubuntu 24.04.1 ISO image is selected as the installation media.

The Operating System Type is set to Linux, and the version is chosen as Ubuntu 64-bit, or another relevant Linux distribution depending on the ISO file. After clicking “Next”, the base memory for the VM is configured — typically 4096 MB (4 GB) for smooth Ubuntu performance. The default processor settings are used in this case, which are generally sufficient for basic tasks.

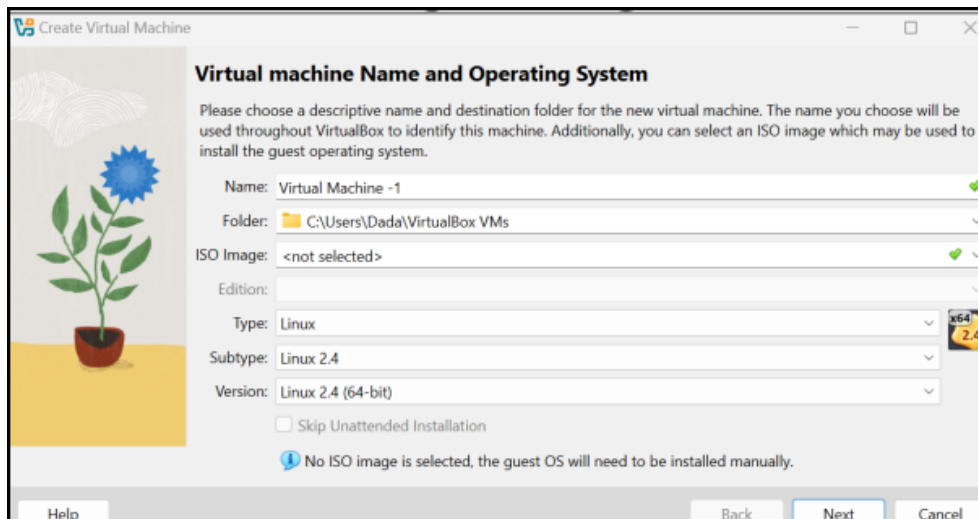


Figure 6: Creating a New Virtual Machine – Setting Name and Operating System

During virtual machine setup in VirtualBox, selecting "Create a Virtual Hard Disk Now" allows the creation of a new virtual disk for the VM. You can then set the disk size, typically around 25 GB (with a minimum of 2 GB as shown in the image). Optionally, you can enable “Pre-allocate Full Size” for better performance, or leave it unchecked to allow dynamic disk allocation, which saves space on your physical drive.

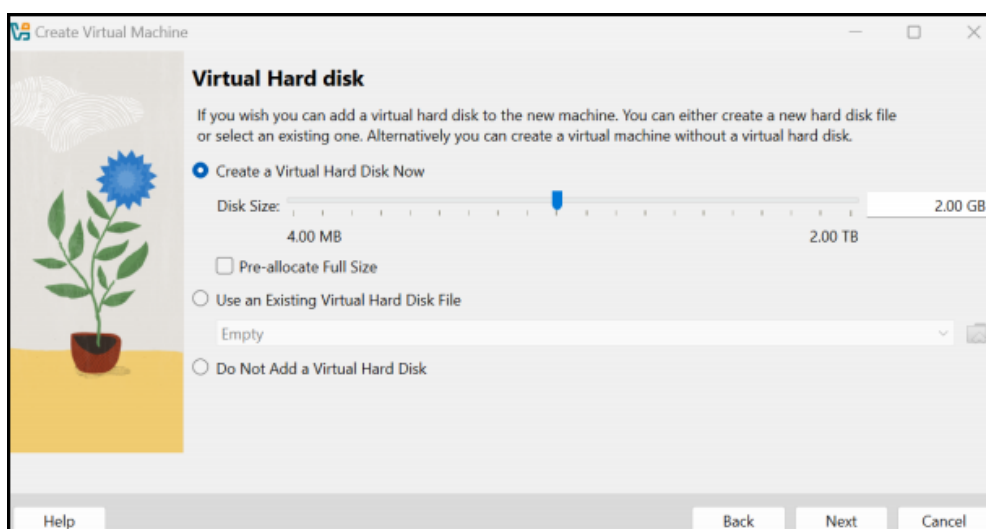


Figure 7: Creating a Virtual Hard Disk for the New Virtual Machine

Click **"Next"** to confirm the virtual disk settings, then click **"Finish"** to complete the virtual machine creation process.

This image shows the VirtualBox Manager interface after successfully creating a VM. It displays key configuration details such as system memory, storage, OS type, and network settings for "Virtual Machine -1".



Figure 8: Virtual Machine Configuration Summary in Oracle VirtualBox Manager

This image outlines the startup and initialization process of an Ubuntu Virtual Machine in VirtualBox. It includes steps from powering on the VM, loading the Ubuntu kernel (GRUB), and starting essential services, to reaching the login screen. It also covers network detection, IP assignment, internet connectivity checks, and system readiness. These steps ensure the VM is fully operational and ready for use. The same sequence will be followed for all three VMs in the setup.

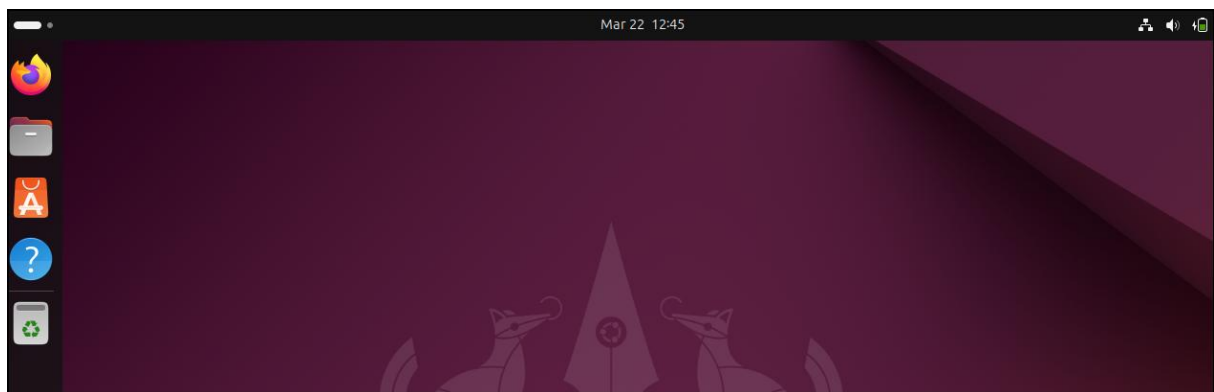


Figure 9: Ubuntu Desktop Environment Successfully Loaded in Virtual Machine

Internal connectivity of Virtual Machines (VMs) using IPv4 allows multiple VMs on the same host to communicate with each other over a private virtual network. This setup is typically done using NAT Network, Host-Only Adapter, or Internal Network in VirtualBox. Each VM is assigned a unique private IPv4 address, enabling ping, SSH, or file sharing between them without requiring internet access. This is essential for simulating multi-node environments or client-server setups within a single system.

7. Creation of AWS Consol

The AWS Console is required in this project to visually manage and monitor EC2 instances triggered by the local VM's auto-scaling script. It allows you to verify that instances are launched successfully, check their status, view public IPs, configure security groups, and terminate instances when needed. While the Python script handles automation via Boto3, the AWS Console provides a graphical interface for validation, troubleshooting, and manual control of cloud resources.

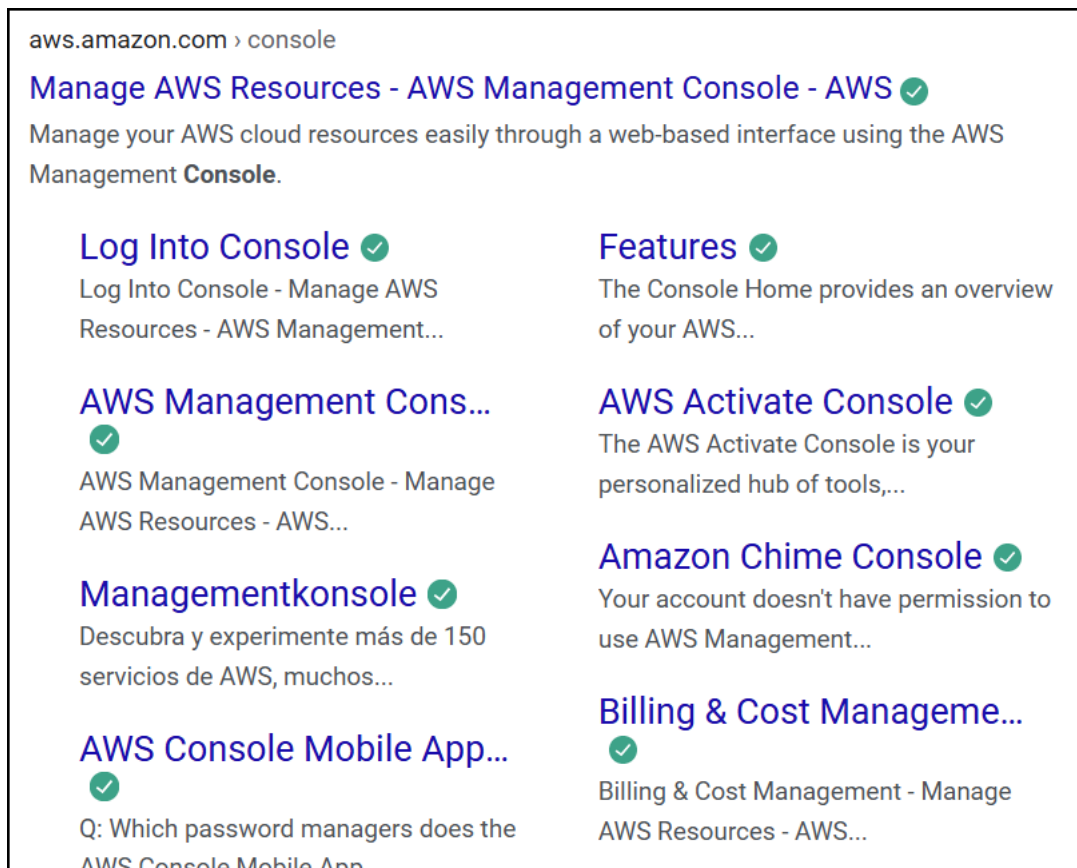


Figure 10: Google Search Results for Accessing AWS Management Console

7.1 Steps to Perform in AWS for Auto-Scaling Project

The image shows the IAM user sign-in page for AWS. The Account ID (e.g., 774335387272) uniquely identifies the AWS account. Within this account, multiple IAM users can be created to provide secure and controlled access to AWS services. Each IAM user (like G24AI1009_USER) has their own username, password, and set of permissions defined by the admin. IAM allows organizations to manage access securely without sharing root credentials, promoting best practices for user-specific access control.

The screenshot shows the 'IAM user sign in' page. It includes a header with the title and an information icon. Below is a form with the following fields and options:

- Account ID or alias** (with a link '(Don't have?)'): A text box containing '774335387272'.
- Remember this account**: A checked checkbox.
- IAM username**: A text box containing 'G24AI1009_USER'.
- Password**: A text box with masked characters '.....'.
- Show Password**: An unchecked checkbox.
- Having trouble?**: A link.
- Sign in**: A large orange button.
- Sign in using root user email**: A button.
- Create a new AWS account**: A link at the bottom.

Figure 11: IAM User Login Screen for AWS Management Console Access

The AWS Console Home page serves as the central dashboard for managing your AWS services. It provides quick access to recently visited services like EC2 and IAM, helping users easily return to important sections without searching. The console is also personalized for the current IAM user and selected region — in this case, G24AI1009_USER under the Asia Pacific (Mumbai) region.

On the right, the Applications panel allows users to create and manage cloud applications, giving a clear overview of deployed services. The “Create application” button helps you quickly start building a new environment.

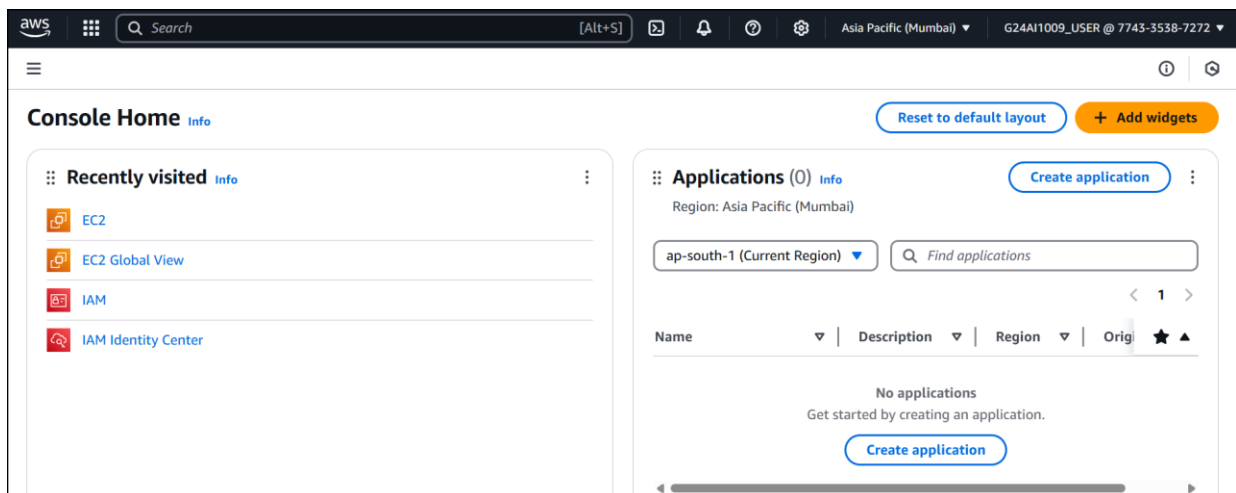


Figure 12: AWS Console Home Page Showing User Dashboard and Recently Visited Services

7.2 Steps Create a Key Pair

A Key Pair in AWS is a security credential used for securely connecting to EC2 instances. It consists of two parts:

Public Key: Stored by AWS and attached to the EC2 instance.

Private Key: Downloaded by the user and used to authenticate via SSH.

When launching an EC2 instance, AWS uses the key pair to ensure only users with the correct private key can access the instance. This is a critical part of secure remote access in AWS.

- I. EC2 Dashboard → Key Pairs (under Network & Security).
- II. “Create key pair”.
- III. Name it (e.g., autoscale-key) and select .pem format.
- IV. Download and save the file securely — you’ll use it to SSH into EC2.

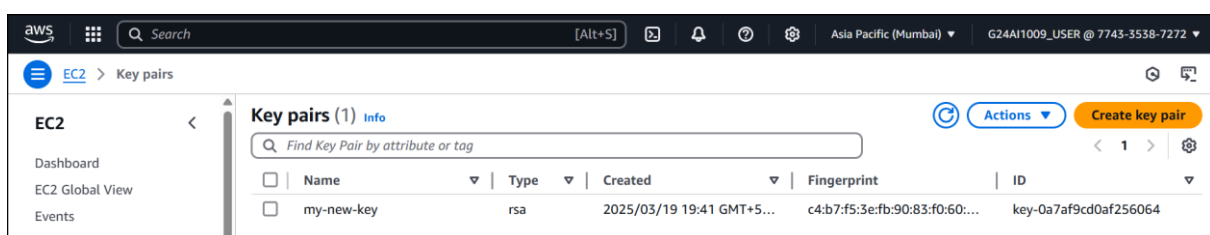


Figure 13: Key Pair Management in AWS EC2 Console

7.3 Steps Create a Security Group

A Security Group in AWS acts as a virtual firewall for EC2 instances. It controls inbound and outbound traffic based on defined rules. You can allow or restrict access by specifying IP addresses, ports, and protocols.

Allow SSH access (port 22) from your IP.

Enable HTTP access (port 80) for web servers.

Security groups help ensure that only authorized traffic can reach your instances, enhancing the overall security of your cloud environment.

- I. In EC2, go to Security Groups.
- II. Click “Create security group”.
- III. Add a rule to allow:
- IV. SSH (port 22) from your IP
- V. HTTP (port 80) if you are running a Flask app

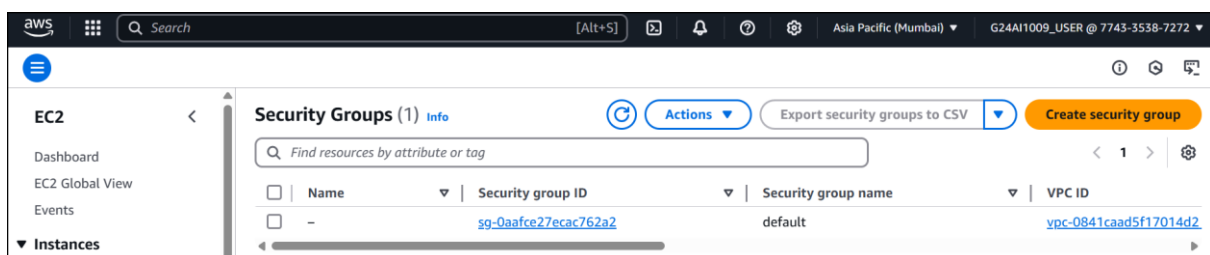


Figure 14: Security Group Configuration Section in AWS EC2 Console

7.4 Find a Valid AMI ID

An AMI ID (Amazon Machine Image ID) is a unique identifier for a pre-configured virtual machine image in AWS. It contains the operating system, application server, and software configuration needed to launch an EC2 instance.

When creating an EC2 instance, the AMI ID tells AWS which image to use as the base. For example, an Ubuntu 22.04 LTS AMI ID will launch an instance with that specific OS pre-installed.

In short, the AMI ID is essential for defining the environment of your EC2 instance at the time of launch.

- I. Go to EC2 → AMIs
- II. Search for Ubuntu 22.04 LTS (x86_64) or a similar image.
- III. Copy the AMI ID (e.g., ami-0aa7d40eeae50c9a9 for Mumbai region).

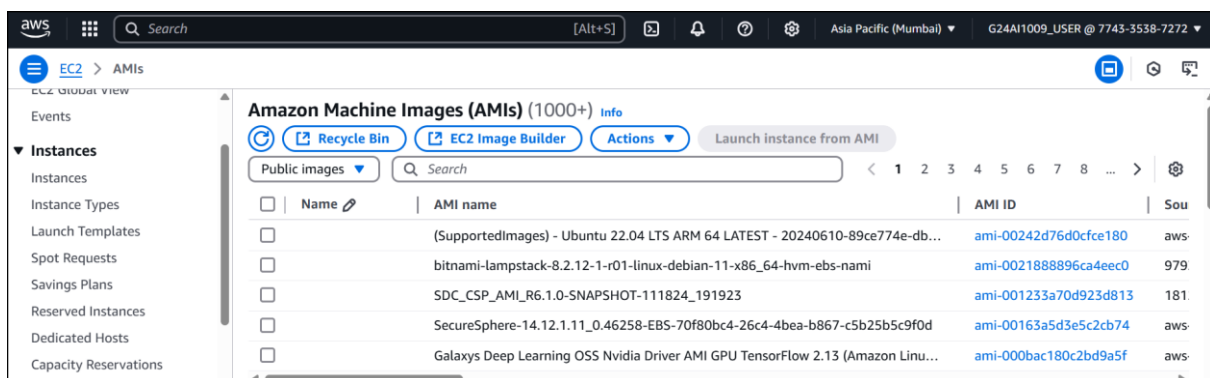


Figure 15: Viewing and Selecting AMI IDs from the AWS EC2 AMI Library

7.5 Configure IAM Permissions (Optional but Recommended)

Configuring IAM permissions ensures that the user or script launching EC2 instances has the necessary access rights. By assigning the appropriate IAM policies (such as ec2:RunInstances, ec2:DescribeInstances, etc.), you can control who can perform what actions on AWS resources.

This step is optional if you're using admin access, but highly recommended for security and role-based access control. It helps follow the principle of least privilege, allowing users to access only what they need.

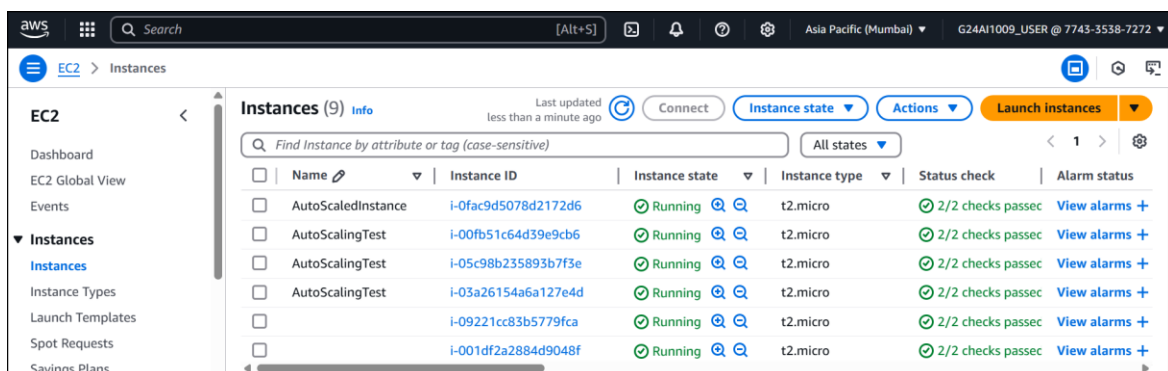


Figure 16: EC2 Instances List Showing Auto-Scaled and Running Instances in AWS Console

8. Installation & Configuration AWS CLI in VM

The following command **installs the AWS Command Line Interface (CLI)** on your Ubuntu system using the apt package manager. The AWS CLI allows you to interact with AWS services directly from your terminal, such as launching EC2 instances, managing S3 buckets, or configuring IAM users — without using the web console.

```
sudo apt install awscli
```

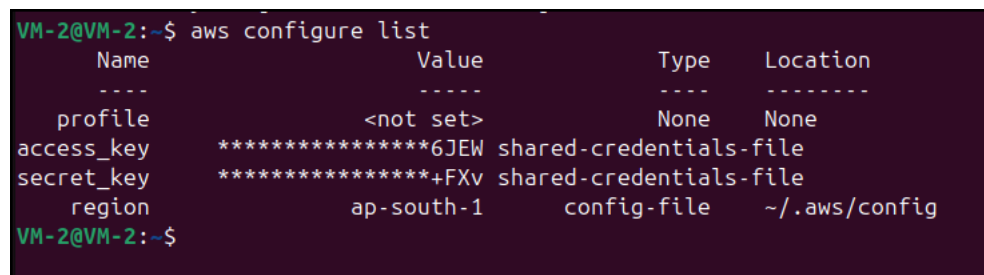
The following command sets up your AWS credentials and configuration so the CLI can authenticate with your AWS account.

```
aws configure
```

It prompts you to enter:

- I. AWS Access Key ID
- II. AWS Secret Access Key
- III. Default Region (e.g., ap-south-1 for Mumbai)
- IV. Default Output Format (optional, like json or table)

Together, these commands prepare your local machine to securely communicate with AWS services from the command line or through scripts (like in your auto-scaling project).



```
VM-2@VM-2:~$ aws configure list
      Name                               Value                               Type      Location
      ----                               -
profile                                <not set>                          None      None
access_key                            *****6JEW                        shared-credentials-file
secret_key                             *****+FXv                        shared-credentials-file
region                                ap-south-1                         config-file  ~/.aws/config
VM-2@VM-2:~$
```

Figure 17: Verifying AWS CLI Configuration Using `aws configure list` Command

9. Implementation of resource monitoring using custom script.

The `monitor_and_scale.py` script is the core component of the auto-scaling project. It is written in Python and performs two main tasks:

1. Resource Monitoring
The script uses the `psutil` library to continuously monitor the local VM's CPU and RAM usage. It checks at regular intervals whether resource usage has crossed a defined threshold (usually 75%).
2. Cloud Auto-Scaling Trigger
If the threshold is exceeded, the script uses the `boto3` library (AWS SDK for Python) to automatically launch a new EC2 instance on AWS. It includes required AWS configurations like the AMI ID, instance type, key pair name, and security group ID.

Optional functionality can include deploying a Flask app on the launched EC2 instance via a user data script. This script helps simulate real-world dynamic scaling by bridging local infrastructure with cloud services based on system load.

```

import psutil
import time
import boto3
from datetime import datetime

# AWS Setup - replace with your actual values

AMI_ID = "ami-06b6e5225d1db5f46"
INSTANCE_TYPE = "t2.micro"
KEY_NAME = "my-new-key"
SECURITY_GROUP = "sg-0aafce27ecac762a2"
REGION = "ap-south-1"

ec2 = boto3.client('ec2', region_name=REGION)

def launch_instance():
    print("Launching new AWS EC2 instance...")
    response = ec2.run_instances(

        time.sleep(5)

if __name__ == "__main__":
    monitor()

    ImageId=AMI_ID,
    InstanceType=INSTANCE_TYPE,
    KeyName=KEY_NAME,
    SecurityGroupIds=[SECURITY_GROUP],
    MinCount=1,
    MaxCount=1
    )
    instance_id = response['Instances'][0]['InstanceId']
    print(f"Instance {instance_id} launched at {datetime.now()}")

def monitor():
    while True:
        cpu = psutil.cpu_percent(interval=1)
        ram = psutil.virtual_memory().percent

        print(f"CPU: {cpu}% | RAM: {ram}%")

        if cpu > 75 or ram > 75:
            launch_instance()
            break

```

Figure 18: Python script to monitor CPU and RAM Usage

This image shows the execution of the `monitor_and_scale.py` script in an activated Python virtual environment. The script is continuously monitoring CPU and RAM usage of the local Ubuntu VM using the `psutil` library. The live output displays real-time percentages, and once CPU or RAM usage exceeds 75%, the script will automatically trigger the launch of an EC2 instance on AWS using `boto3`.

```

VM-2@VM-2:~$ source myenv/bin/activate
(myenv) VM-2@VM-2:~$ python monitor_and_scale.py
CPU: 1.2% | RAM: 45.1%
CPU: 7.0% | RAM: 45.1%
CPU: 2.6% | RAM: 45.1%
CPU: 4.9% | RAM: 45.1%
CPU: 2.7% | RAM: 45.1%
CPU: 4.8% | RAM: 45.1%
CPU: 3.9% | RAM: 45.1%

```

Figure 19: Running Python Monitoring Script to Track CPU and RAM Usage in Real Time

10. Pushing the VM to Trigger AWS Scaling (Loading and testing the application)

To test the auto-scaling functionality, the script needs to detect high CPU or RAM usage. This can be simulated by generating artificial load on the system. For example, running the command `yes >/dev/null &` repeatedly will spike CPU usage.

Once the CPU usage crosses the defined threshold (75%), the script automatically triggers the launch of an EC2 instance on AWS. This validates that the resource monitoring and cloud integration are working correctly, simulating real-time scaling in response to system load.

```
VM-2@VM-2:~$ yes >/dev/null &
[1] 21231
VM-2@VM-2:~$ bash: /dev/null: No such file or directory
```

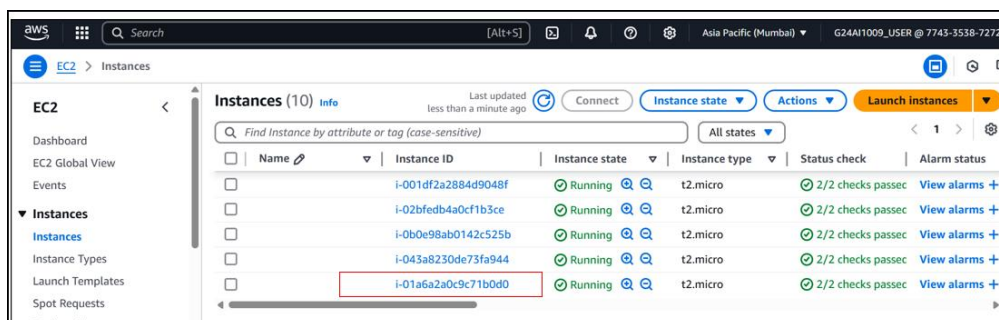
Figure 20: Attempt to Simulate CPU Load Using Background Process in Terminal

This image shows the successful execution of the auto-scaling script. After detecting high CPU usage (91.4%), the script automatically triggered the creation of a new **AWS EC2 instance**. The instance with ID `i-01a6a2a0c9c71b0d0` was launched at the specified timestamp, confirming that the resource-based scaling logic is working as expected.

```
CPU: 4.0% | RAM: 45.1%
CPU: 2.5% | RAM: 45.1%
CPU: 5.3% | RAM: 45.1%
CPU: 1.4% | RAM: 45.1%
CPU: 2.6% | RAM: 45.1%
CPU: 5.2% | RAM: 45.1%
CPU: 2.7% | RAM: 45.1%
CPU: 91.4% | RAM: 45.0%
Launching new AWS EC2 instance...
Instance i-01a6a2a0c9c71b0d0 launched at 2025-03-22 16:42:57.499038
(myenv) VM-2@VM-2:~$
```

Figure 21: EC2 Instance Automatically Launched After CPU Threshold is Exceeded

This image shows the AWS EC2 Instances dashboard, where the instance with ID `i-01a6a2a0c9c71b0d0`—launched by the auto-scaling script—is now listed and in a running state. It confirms that the instance was successfully created in response to high CPU usage detected on the local VM.



| | Name | Instance ID | Instance state | Instance type | Status check | Alarm status |
|--------------------------|------|---------------------|----------------|---------------|-------------------|---------------|
| <input type="checkbox"/> | | i-001df2a2884d9048f | Running | t2.micro | 2/2 checks passed | View alarms + |
| <input type="checkbox"/> | | i-02bfed40c1b3ce | Running | t2.micro | 2/2 checks passed | View alarms + |
| <input type="checkbox"/> | | i-0b0e98ab0142c525b | Running | t2.micro | 2/2 checks passed | View alarms + |
| <input type="checkbox"/> | | i-043a8230de73fa944 | Running | t2.micro | 2/2 checks passed | View alarms + |
| <input type="checkbox"/> | | i-01a6a2a0c9c71b0d0 | Running | t2.micro | 2/2 checks passed | View alarms + |

Figure 22: Newly Launched EC2 Instance Confirmed in AWS Console After Auto-Scaling Trigger