Cloud Lab Report: Understanding Virtualization by Installing VirtualBox and Creating a VM (Linux) for a React Application

1. Objective

- **Purpose:** The objective of this experiment is to understand the concept of virtualization by installing Oracle VM VirtualBox and creating a Virtual Machine (VM) with a Linux operating system. This VM will be used to host and deploy a React application, providing hands-on experience with setting up and managing virtual environments for software development and deployment.

2. Background

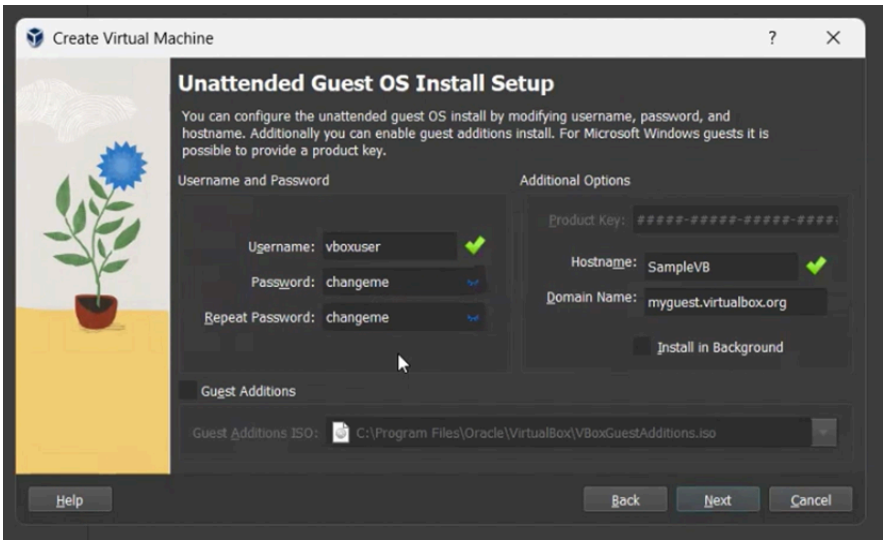
- **Theory/Concepts:**
    - **Virtualization:** Virtualization allows multiple operating systems to run on a single physical machine by creating virtual versions of resources like servers. A Virtual Machine (VM) is a software emulation of a computer, running an OS and applications in isolation.
    - **Oracle VM VirtualBox:** A type 2 hypervisor that manages VMs.
    - **React:** A JavaScript library for building dynamic user interfaces, typically used in modern web development.
- **Context:**
    - In this experiment, Oracle VM VirtualBox will be used to create a Linux VM. The Linux environment will be set up to run a React application, demonstrating the use of virtualization in software development.

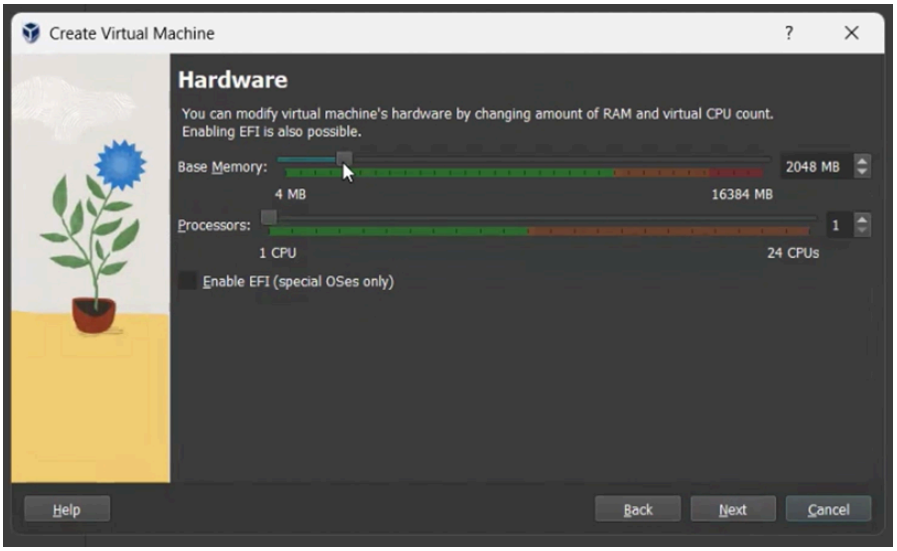
3. Tools and Services

- **Cloud Services:** None (This experiment is conducted locally using VirtualBox).
- **Software/Tools:**
    - **Oracle VM VirtualBox**
    - **Linux ISO image (e.g., Ubuntu)**
    - **Node.js and npm**
    - **React application code**
    - **Text editors (e.g., nano, vim, or Visual Studio Code)**

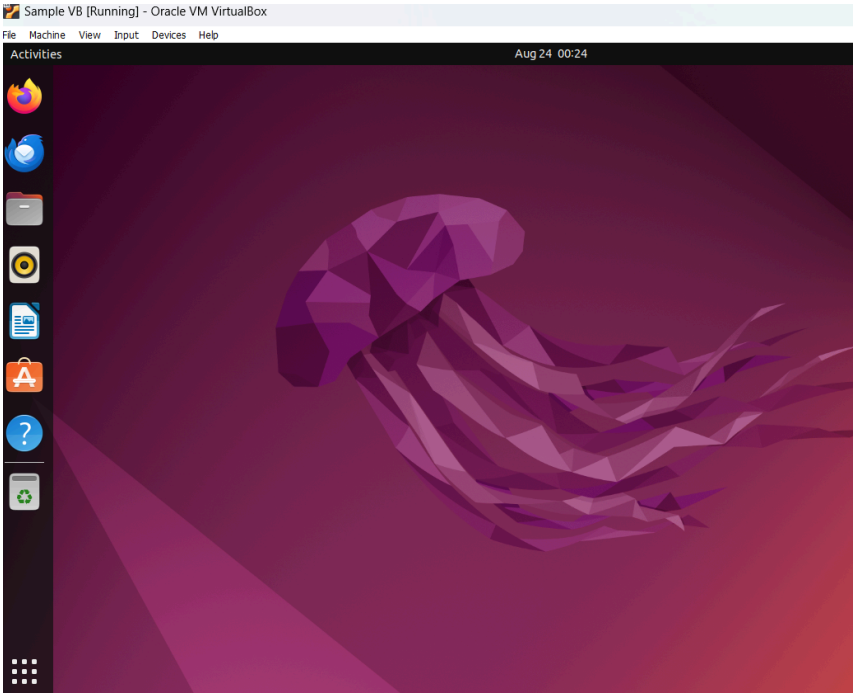
4. Experiment Setup

- **Step-by-Step Configuration:**
    1. **Cloud Account Setup:** Not applicable.
    2. **Environment Configuration:**
        - **Install VirtualBox:**
            - Download and install Oracle VM VirtualBox from the official website.
        - **Download Linux ISO:**
            - Choose and download a Linux distribution (e.g., Ubuntu) ISO image.
        - **Create a New VM:**
            - Open VirtualBox, create a new VM, allocate resources (memory, disk), and configure it to boot from the Linux ISO.

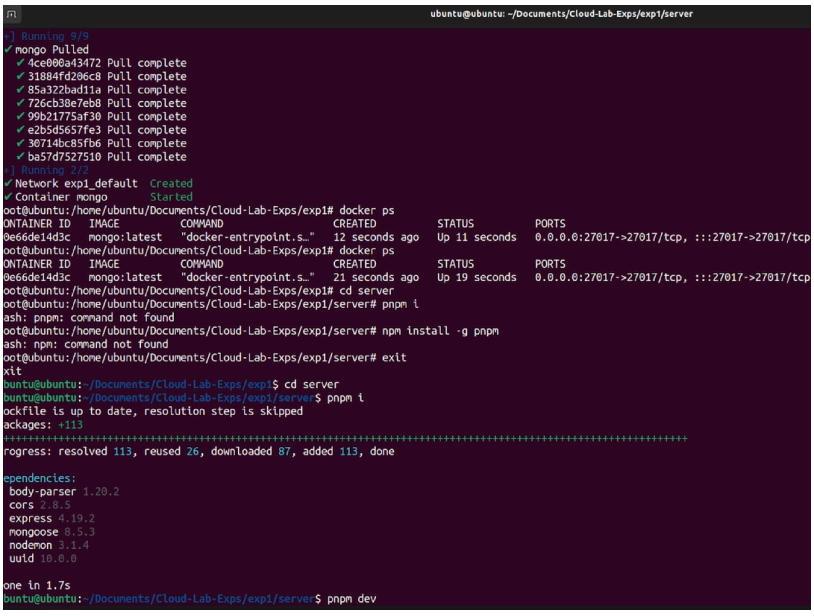




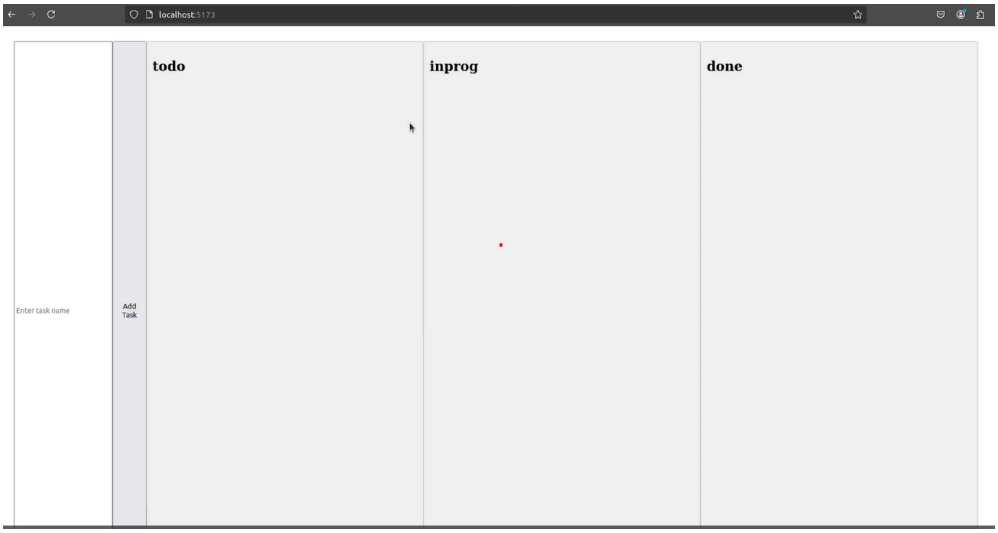
- **Install Linux on the VM:**
  - Boot the VM with the ISO and follow the installation instructions to set up Linux.



- **Set Up Linux Environment:**
  - Update the Linux package manager and install necessary updates.
- **Install Node.js and npm:**
  - Use the Linux terminal to install Node.js and npm.



3. **Service Deployment:**
    - **Transfer React Application:** Transfer or clone the React application code into the VM.
    - **Install Dependencies:** Run `npm install` to install the necessary dependencies.
4. **Security Settings:** Not applicable for this local setup.
5. **Diagrams:**
    - Include architecture diagrams if applicable, showing the VM setup, network configuration, and React app deployment.



5. Execution

- **Tasks Performed:**
    - **Install VirtualBox and Linux:** Successfully set up a Linux VM on VirtualBox.
    - **React Application Deployment:** Configured the Linux environment, installed Node.js, and deployed a React application.
- **Monitoring:**
    - Not applicable, but the functionality of the application can be confirmed by accessing it via a web browser.

6. Observations

- **Data Collected:**
    - Observations include the successful installation of VirtualBox, Linux VM, Node.js, npm, and the React application. Screenshots or terminal outputs can be documented.
- **Tables/Graphs:**
    - No specific performance data, but you could include a table summarizing the steps and their outcomes.

7. Results

- **Outcome:**
    - VirtualBox and the Linux VM were successfully set up, and the React application was deployed and ran correctly within the VM environment.
- **Performance Metrics:**
    - Since this is a local setup, performance metrics like response time or resource usage can be observed but are not the primary focus.
- **Scalability and Resilience:**
    - While not directly tested in this experiment, virtualization demonstrates the potential for scalable and isolated environments.

Cloud Lab Report: Understanding the Concept of Docker-Container

1. Objective

- **Purpose:** The objective of this report is to provide a clear understanding of Docker and its container technology. This will include covering the key concepts, architecture, and benefits of Docker, demonstrating how containers enable efficient software development and deployment. The report will also differentiate Docker containers from traditional virtual machines, explaining why Docker is widely adopted in modern software engineering.

2. Background

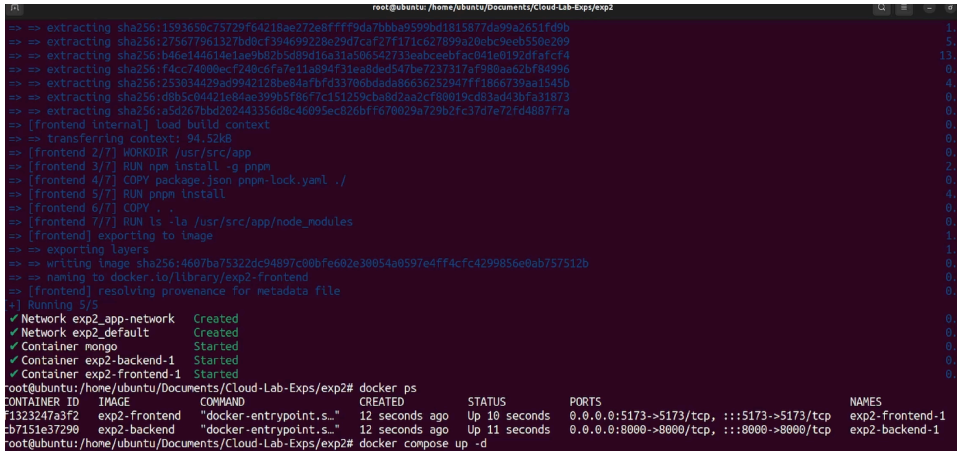
- **Theory/Concepts:**
    - **Docker:** Docker is an open-source platform that automates the deployment, scaling, and management of applications using containerization. Containers are lightweight, standalone software packages that include everything needed to run an application, ensuring consistent performance across different environments.
    - **Containerization vs. Virtualization:** Unlike virtual machines, containers share the host system's kernel, making them more efficient and faster. Docker simplifies the creation and management of these containers with tools like Docker Engine, Docker Images, and Docker Hub.
    - **Importance in DevOps:** Docker is crucial in modern software development, particularly within DevOps and microservices architectures, enabling seamless integration and continuous deployment practices.
- **Context:**
    - This experiment will involve using Docker Desktop on a local machine to create, manage, and interact with Docker containers. The focus will be on understanding how Docker containers differ from traditional virtual machines and their role in modern software development and deployment.

3. Tools and Services

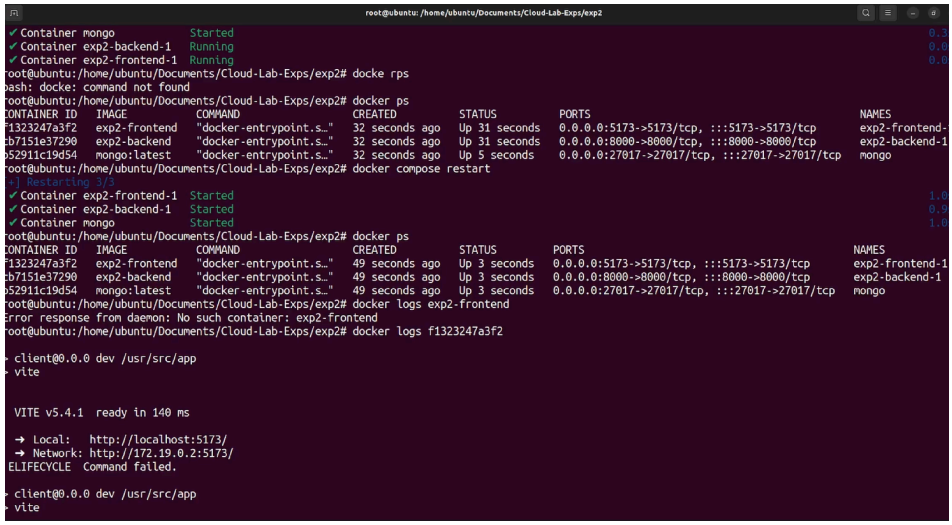
- **Cloud Services:** None (This experiment is conducted locally using Docker Desktop).
- **Software/Tools:**
    - **Docker Desktop**
    - **Internet connection** (for downloading Docker images and accessing Docker Hub)
    - **Text editor or IDE** (for Dockerfiles and application management)
    - **Basic CLI tools** (for running Docker commands)

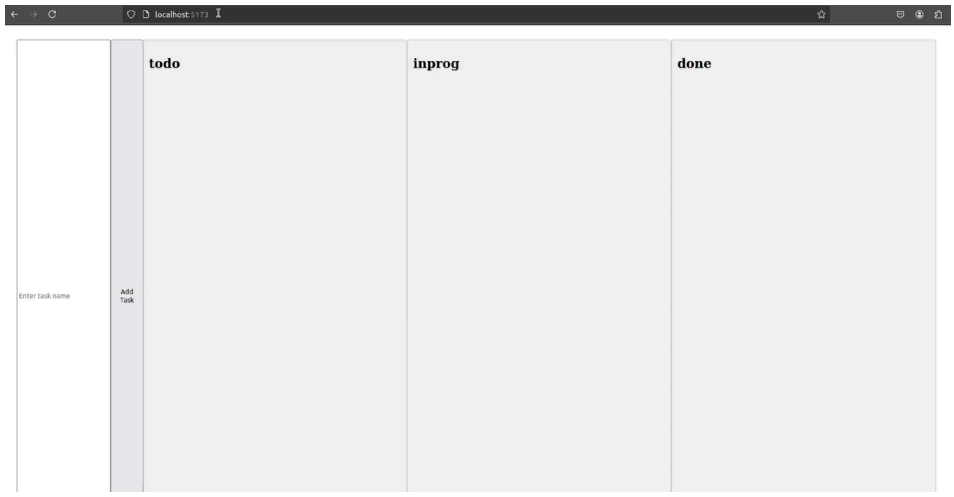
4. Experiment Setup

- **Step-by-Step Configuration:**
    1. **Cloud Account Setup:** Not applicable.
    2. **Environment Configuration:**
        - **Install Docker Desktop:**
            - Download Docker Desktop from the official Docker website.
            - Follow the installation instructions for your operating system (Windows, macOS, or Linux).
        - **Verify Docker Installation:**
            - Open a terminal or command prompt.
            - Run `docker --version` to check if Docker is installed correctly.
    3. **Service Deployment:**
        - **Pull a Docker Image:**
            - Use the command `docker pull <image-name>` to download a Docker image from Docker Hub. For example, `docker pull nginx` to download the NGINX web server image.
        - **Run a Docker Container:**
            - Start a container using the pulled image with the command `docker run -d --name <container-name> <image-name>`. For example, `docker run -d --name mynginx nginx`.



- **List Running Containers:**
  - Use the command `docker ps` to see all active containers on your system.





- **Access the Running Container:**
  - Use `docker exec -it <container-name> /bin/bash` to access the terminal of the running container.
- **Stop the Docker Container:**
  - Stop the container with `docker stop <container-name>`.
- **Remove the Container:**
  - Remove the stopped container using `docker rm <container-name>`.

        - **Clean Up:**
            - Remove unused images with `docker rmi <image-name>` to free up space.
    4. **Security Settings:** Not applicable for this local setup.
    5. **Diagrams:**
        - Include any relevant architecture diagrams that show the Docker container lifecycle or network configuration if applicable.

5. Execution

- **Tasks Performed:**
    - **Docker Installation:** Successfully installed Docker Desktop.
    - **Docker Image Management:** Pulled Docker images from Docker Hub and managed containers using Docker commands.
    - **Container Interaction:** Accessed and interacted with running Docker containers through the command line.
- **Monitoring:**
    - Monitored the status and performance of running containers using the `docker ps` command.

6. Observations

- **Data Collected:**
    - Observations include the successful installation and verification of Docker, downloading and running Docker images, and managing containers. Screenshots or command outputs can be documented to confirm each step.
- **Tables/Graphs:**
    - A table summarizing Docker commands and their outcomes can be included to track progress and results.

7. Results

- **Outcome:**
    - Docker Desktop was installed successfully, and Docker containers were created, managed, and interacted with as expected. The experiment confirmed the efficiency and ease of use of Docker compared to traditional virtualization methods.
- **Performance Metrics:**
    - Observations noted the quick startup times and low resource usage of Docker containers.
- **Scalability and Resilience:**
    - While not directly tested in this experiment, Docker's scalability was implied through the efficient handling of multiple containers.

Title: Lab Report on AWS: Creating an EC2 Instance and Managing S3 Object Storage

Objective:

The objective of this lab was to understand and work with AWS services by creating an EC2 instance and managing object storage using AWS S3. By completing this lab, I gained hands-on experience with setting up a virtual server (EC2 instance) and securely storing data using S3, which is fundamental for leveraging AWS for various cloud computing needs.

Prerequisites:

- AWS Free Tier Account (Registered at https://aws.amazon.com/free/)
- Basic Understanding of Cloud Concepts
- Internet Access
- Web Browser

Theory:

AWS (Amazon Web Services) is a cloud computing platform that provides scalable and flexible computing resources. This lab focused on:

- **EC2 (Elastic Compute Cloud):** A service that allows users to create and manage virtual servers (instances) that can run applications and handle web traffic. Different instance types are available based on computing needs.
- **S3 (Simple Storage Service):** An object storage service designed to store and retrieve any amount of data from anywhere on the web. It offers high durability and scalability, suitable for storing various types of data such as backups and media files.

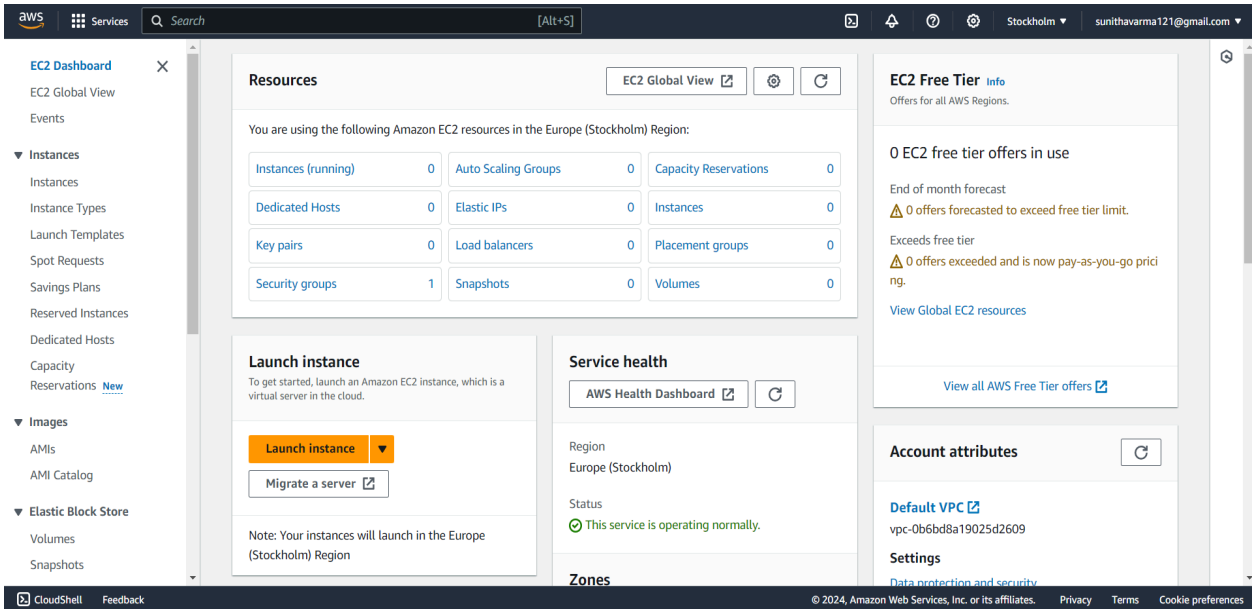
Materials and Equipment:

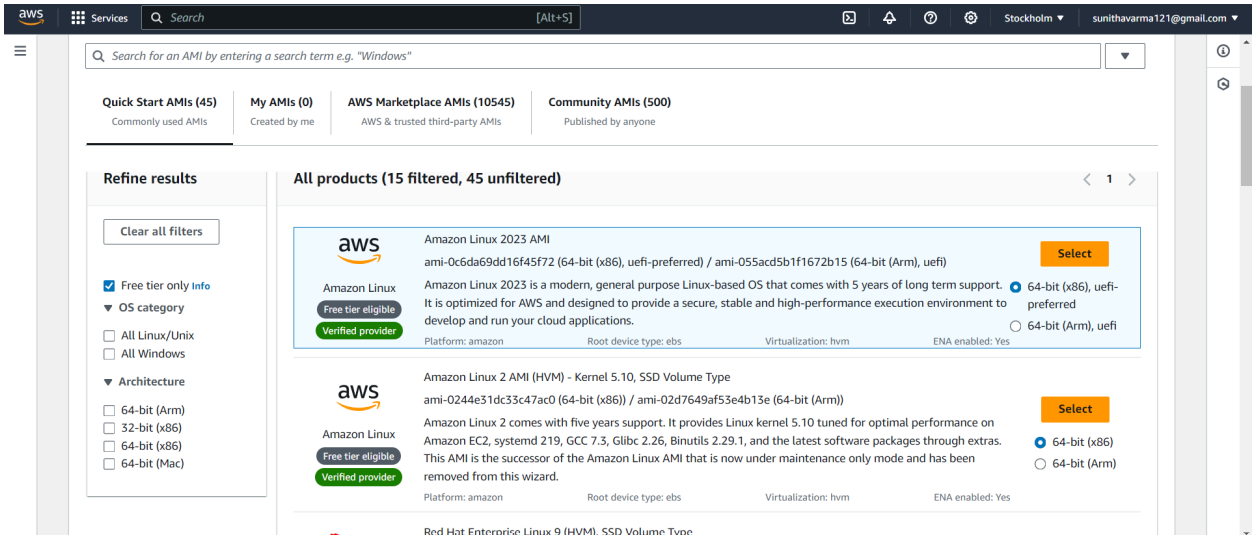
- Stable Internet Connection
- AWS Free Tier Account
- Web Browser

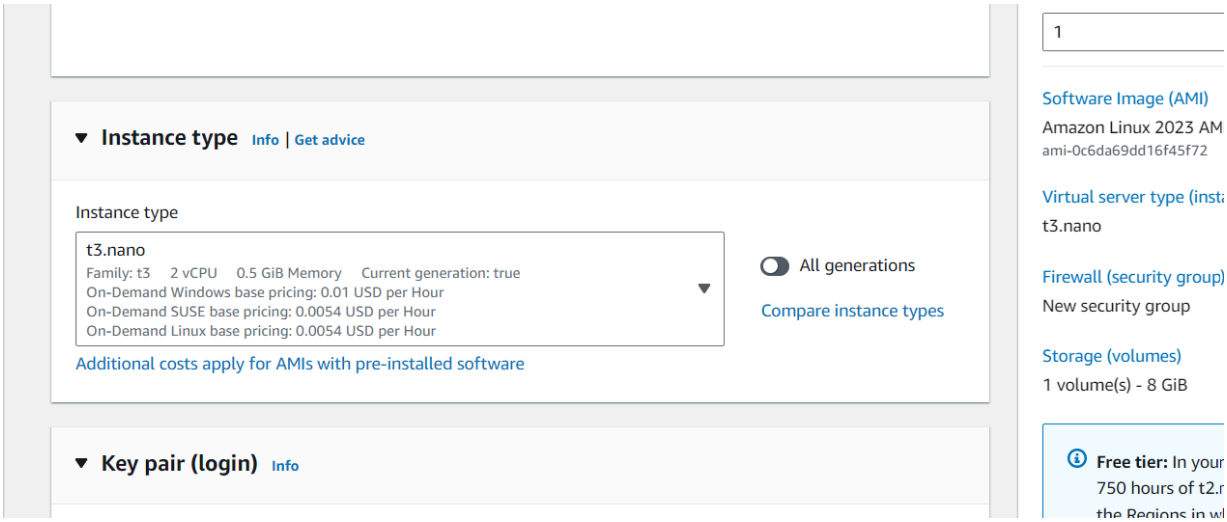
Procedure:

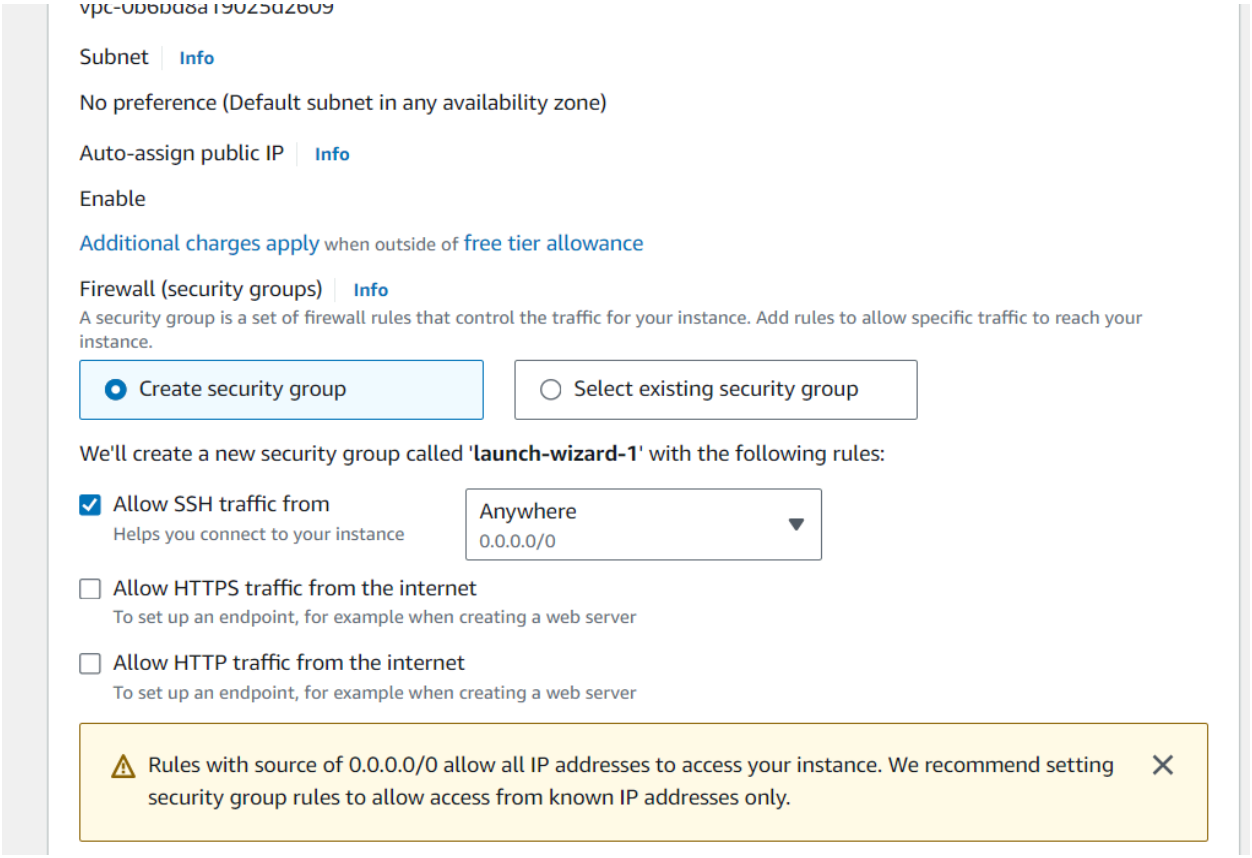
1. **AWS Instance Creation**
    - **Log in to AWS Free Tier Account:**
        - I accessed the AWS Management Console at https://aws.amazon.com/console/.
        - Signed in using my AWS Free Tier credentials.
    - **Navigate to EC2 Dashboard:**
        - From the AWS Management Console, I searched for "EC2" and selected it to open the EC2 dashboard.

- **Launch a New Instance:**
  - Clicked on "Launch Instance."
  - Chose an Amazon Machine Image (AMI) such as Amazon Linux 2.
  - Selected an instance type eligible for the Free Tier, like t2.micro.
  - Configured instance settings including network, IAM roles, and monitoring options.
  - Set up the security group to manage traffic rules.
  - Reviewed and launched the instance, selecting the appropriate key pair for secure access.
- **Connect to the Instance:**
  - After launching the instance, I selected it from the EC2 dashboard.
  - Used the "Connect" option and followed the provided instructions to SSH into the instance using the key pair.






1
Software Image (AMI)
Amazon Linux 2023 AM
ami-0c6da69dd16f45f72
Virtual server type (inst
t3.nano
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 8 GiB
Free tier: In your
750 hours of t2.
Instance type Info | Get advice
Instance type
t3.nano
Family: t3 2 vCPU 0.5 GiB Memory Current generation: true
On-Demand Windows base pricing: 0.01 USD per Hour
On-Demand SUSE base pricing: 0.0054 USD per Hour
On-Demand Linux base pricing: 0.0054 USD per Hour
All generations
Compare instance types
Additional costs apply for AMIs with pre-installed software
Key pair (login) Info


Subnet Info
No preference (Default subnet in any availability zone)
Auto-assign public IP Info
Enable
Additional charges apply when outside of free tier allowance
Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.
Create security group
Select existing security group
We'll create a new security group called 'launch-wizard-1' with the following rules:
Allow SSH traffic from
Helps you connect to your instance
Anywhere
0.0.0.0/0
Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server
Allow HTTP traffic from the internet
To set up an endpoint, for example when creating a web server
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Configure storage Info

Advanced

1x 8 GiB gp3 Root volume (Not encrypted)

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage

Add new volume

Click refresh to view backup information

The tags that you assign determine whether the instance will be backed up by any Data Lifecycle Manager policies.

0 x File systems Edit

Summary

Number of instances Info

1

Software Image (AMI)

Amazon Linux 2023 AMI

ami-0c6da69dd16f45f72

Virtual server type (instance type)

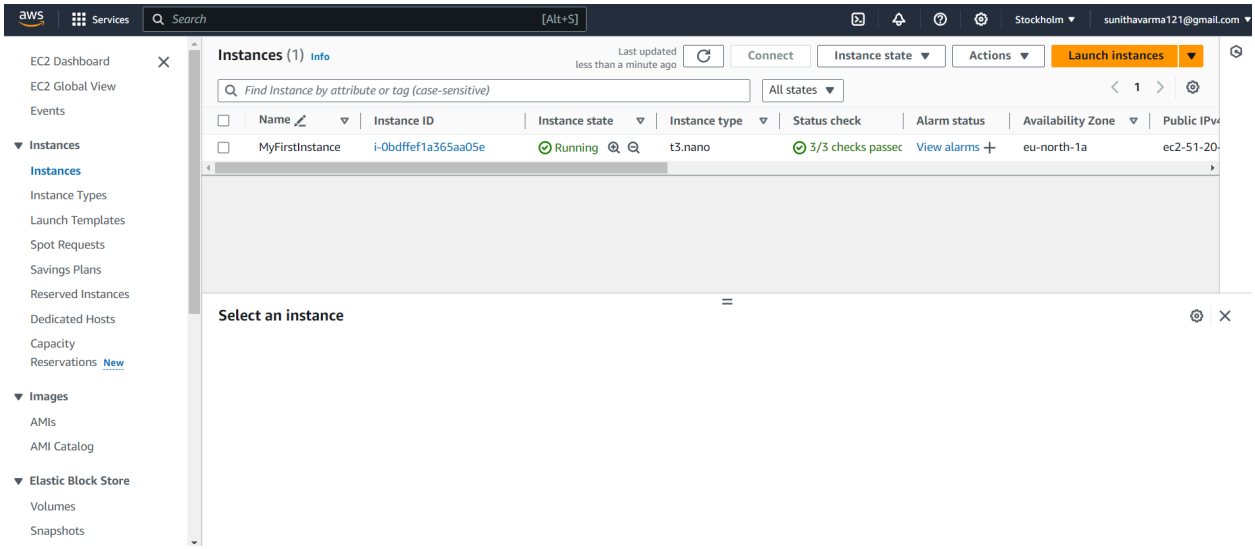
t3.nano

Firewall (security group)

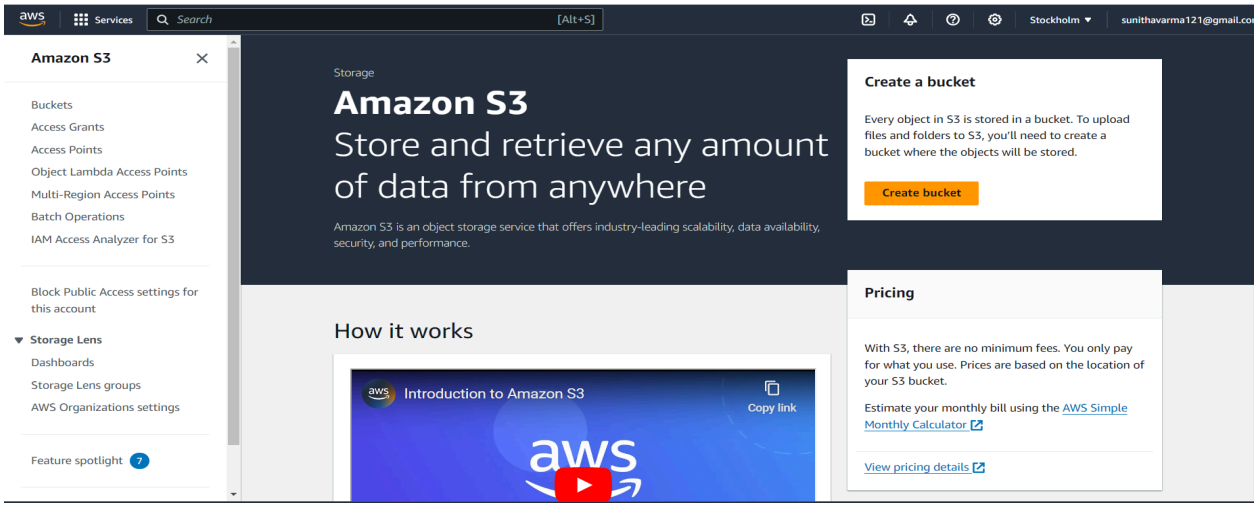
New security group

Storage (volumes)

1 volume(s) - 8 GiB



2. **AWS S3 Object Storage**
    - **Access S3 Service:**
        - From the AWS Management Console, I searched for "S3" and opened the S3 dashboard.
    - **Create a New S3 Bucket:**
        - Clicked on "Create bucket."
        - Entered a unique name for the bucket and selected a region.
        - Configured settings such as versioning, encryption, and access permissions.
        - Reviewed the settings and created the bucket.
    - **Upload Objects to S3 Bucket:**
        - Selected the newly created bucket.
        - Clicked on "Upload" and chose the files and folders I wanted to store.
        - Configured permissions and properties as needed.
        - Started the upload process.



General configuration

AWS Region

Europe (Stockholm) eu-north-1

Bucket type Info

- (●) General purpose
  Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.
- ( ) Directory
  Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name Info

myfirstawsbucket

Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Choose bucket

Format: s3://bucket/prefix

Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

- (●) ACLs disabled (recommended)
  All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.
- ( ) ACLs enabled
  Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Object Ownership

Bucket owner enforced

Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

- [x] **Block *all* public access**
  Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.
  - [x] **Block public access to buckets and objects granted through *new* access control lists (ACLs)**
    S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
  - [x] **Block public access to buckets and objects granted through *any* access control lists (ACLs)**
    S3 will ignore all ACLs that grant public access to buckets and objects.

Default encryption Info

Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption type Info

- (•) Server-side encryption with Amazon S3 managed keys (SSE-S3)
- ( ) Server-side encryption with AWS Key Management Service keys (SSE-KMS)
- ( ) Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)

  Secure your objects with two separate layers of encryption. For details on pricing, see **DSSE-KMS pricing** on the **Storage** tab of the Amazon S3 pricing page.

Bucket Key

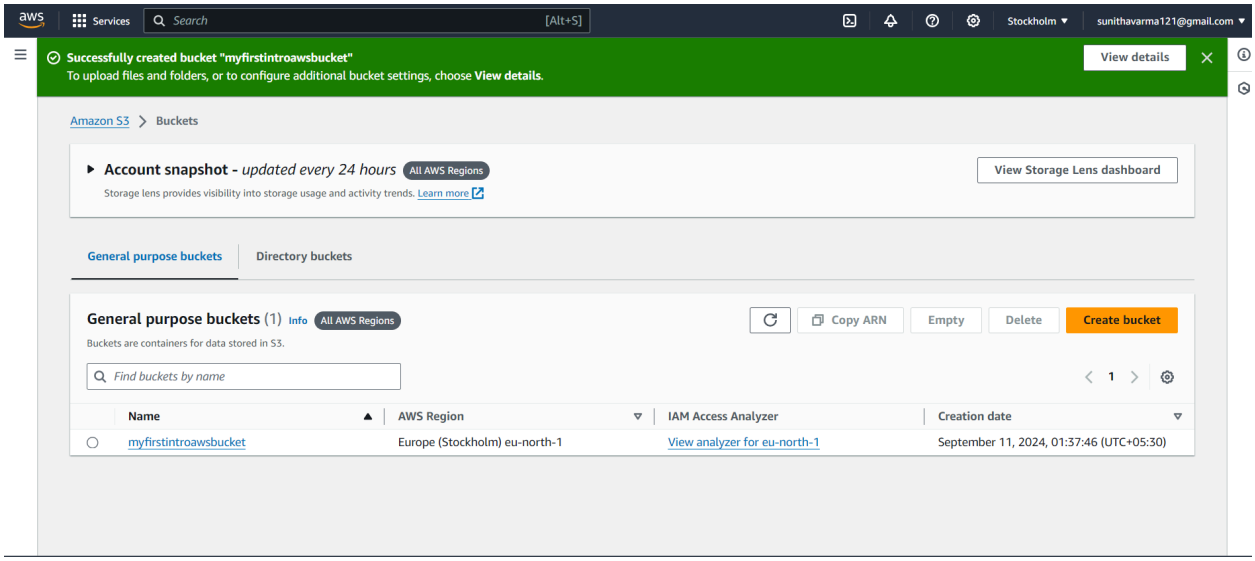
Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. Learn more

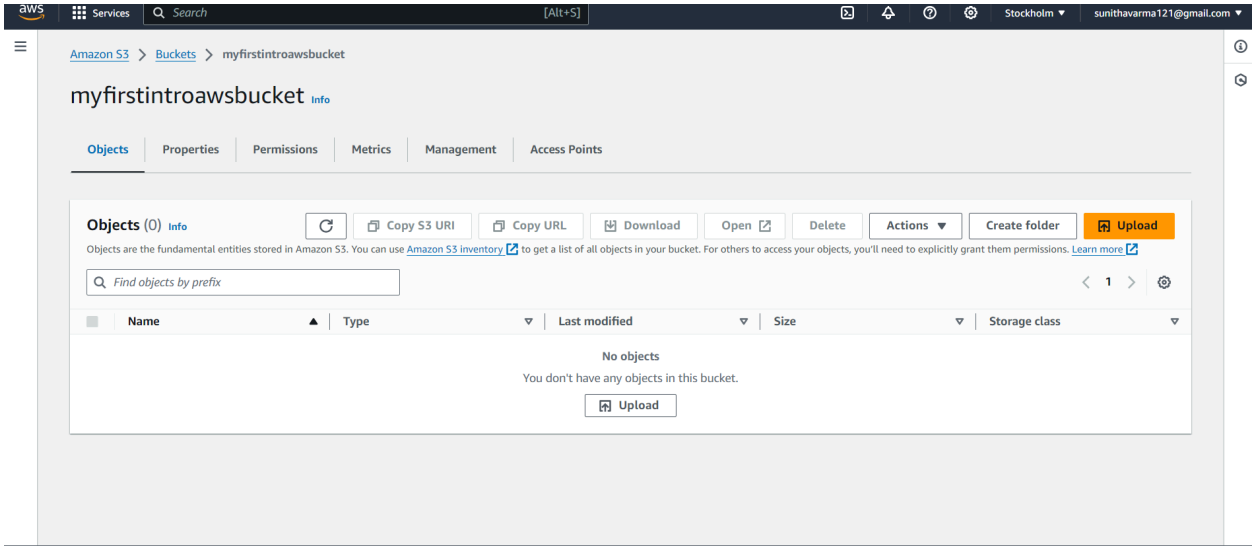
- ( ) Disable
- (•) Enable

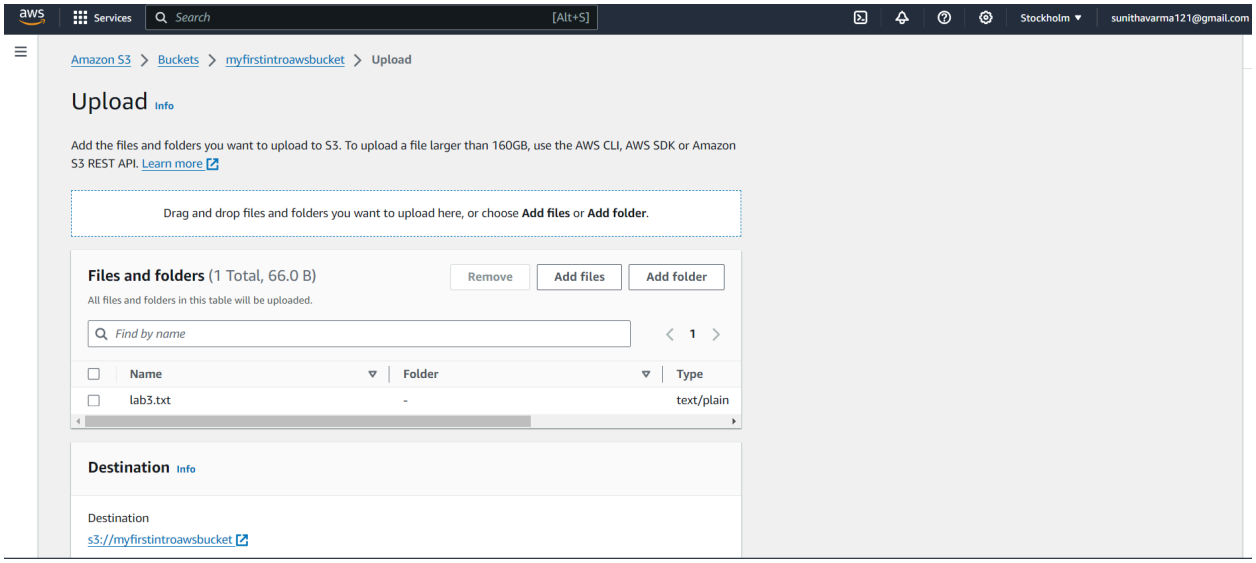
▶ **Advanced settings**

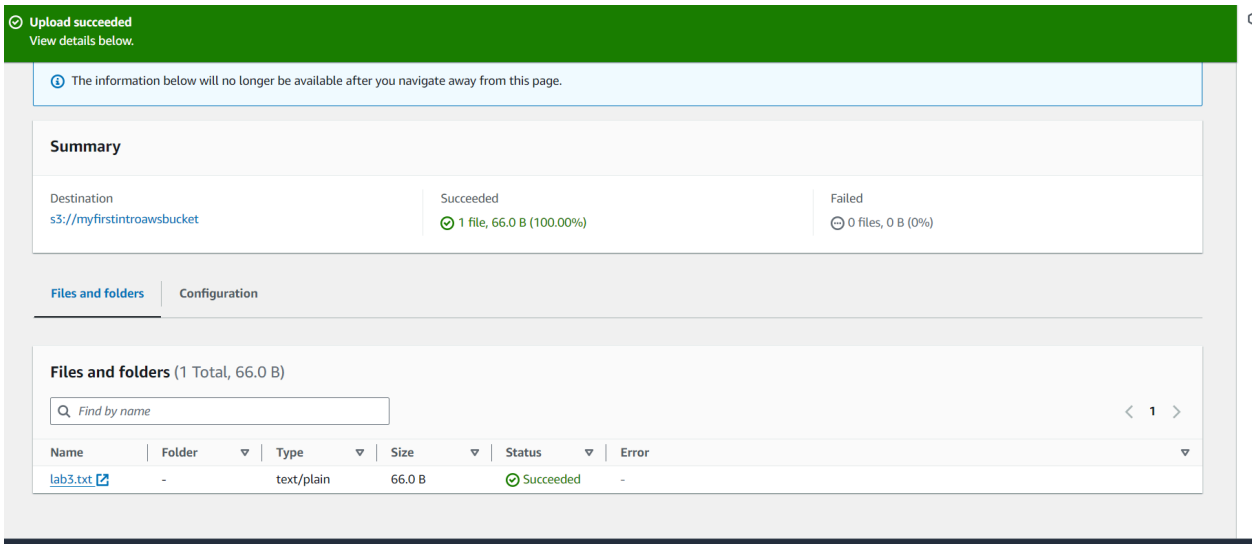
After creating the bucket, you can upload files and folders to the bucket, and configure additional bucket settings.

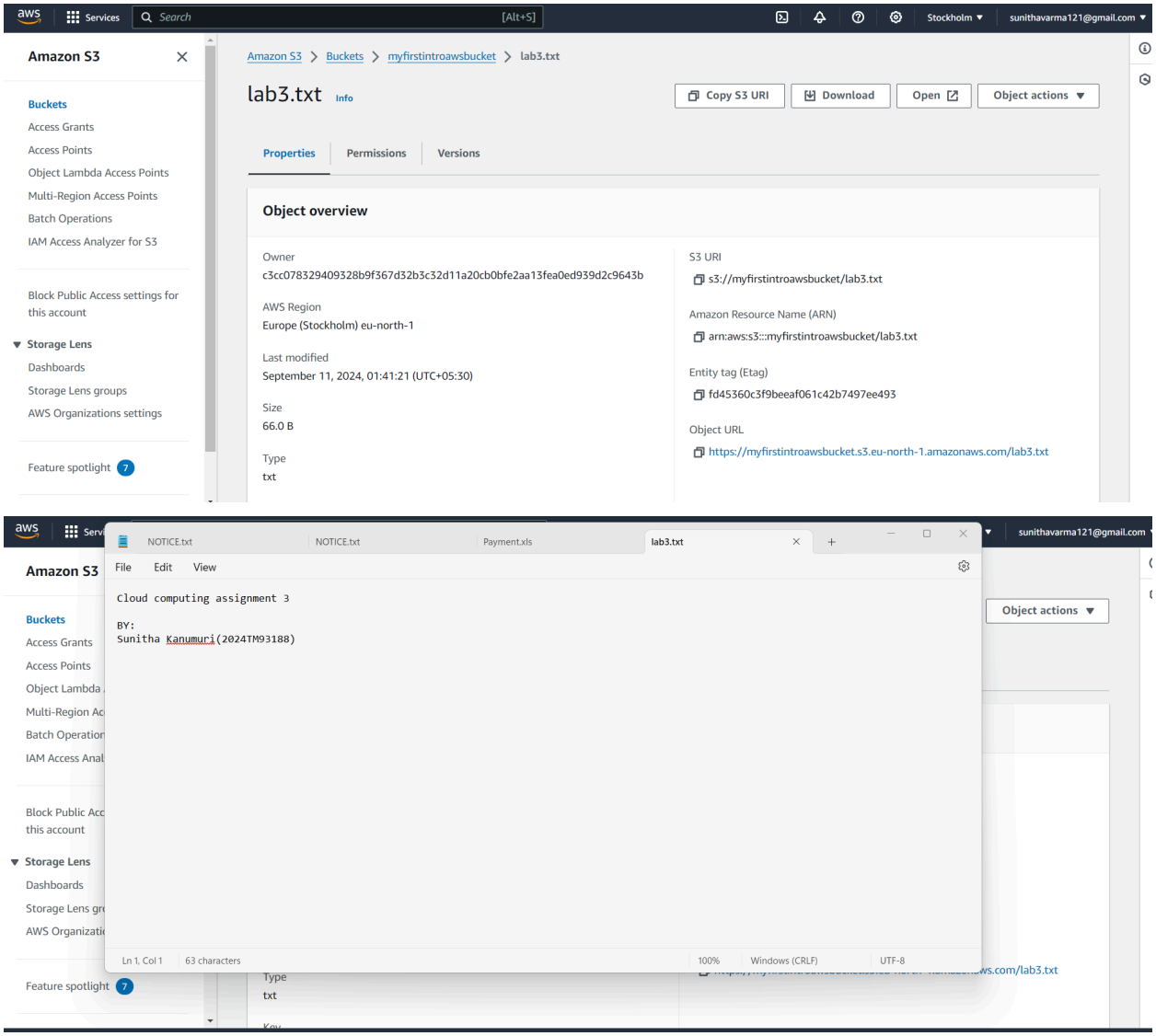
Cancel | Create bucket




aws
Services
Search
[Alt+S]
Stockholm
sunithavarma121@gmail.com
Amazon S3 > Buckets > myfirstintroawsbucket
myfirstintroawsbucket Info
Objects
Properties
Permissions
Metrics
Management
Access Points
Objects (0) Info
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Find objects by prefix
Name
Type
Last modified
Size
Storage class
No objects
You don't have any objects in this bucket.
Upload


aws
Services
Search
[Alt+S]
Stockholm
sunithavarma121@gmail.com
Amazon S3 > Buckets > myfirstintroawsbucket > Upload
Upload Info
Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. Learn more
Drag and drop files and folders you want to upload here, or choose Add files or Add folder.
Files and folders (1 Total, 66.0 B)
Remove
Add files
Add folder
All files and folders in this table will be uploaded.
Find by name
Name
Folder
Type
lab3.txt
-
text/plain
Destination Info
Destination
s3://myfirstintroawsbucket


Upload succeeded
View details below.
The information below will no longer be available after you navigate away from this page.
Summary
Destination
s3://myfirstintroawsbucket
Succeeded
1 file, 66.0 B (100.00%)
Failed
0 files, 0 B (0%)
Files and folders
Configuration
Files and folders (1 Total, 66.0 B)
Find by name
Name
Folder
Type
Size
Status
Error
lab3.txt
-
text/plain
66.0 B
Succeeded
-



Expected Output:

1. **AWS Instance Creation**
   - The EC2 instance appeared in the EC2 dashboard with a "running" status. I could view the instance details, including type, AMI, and public IP. SSH access to the instance was successful.
2. **AWS S3 Object Storage**
   - The S3 bucket was visible in the S3 dashboard. I was able to upload and access files within the bucket, and manage the objects as needed.

Observations:

- **EC2 Instance:** I observed the instance details and connectivity options. There were no significant issues with SSH access, and the instance functioned as expected.

- **S3 Object Storage:** The bucket creation process went smoothly, and the file uploads were efficient. I managed the permissions and organization of objects without any major issues.

Result:

I successfully created an EC2 instance and an S3 bucket using my AWS Free Tier account. The instance was operational with functioning SSH access, and the S3 bucket allowed me to securely store and manage data effectively.

Title: To create a VPC and associate an Application Load Balancer to it

Objective:

The objective of this task is to demonstrate the creation of a Virtual Private Cloud (VPC) and the association of an Application Load Balancer (ALB) within that VPC. This involves setting up a VPC with appropriate subnets and routing, and configuring an ALB to distribute incoming traffic across multiple targets within the VPC. This process is crucial for ensuring scalable and efficient management of application traffic in a secure network environment.

Theory:

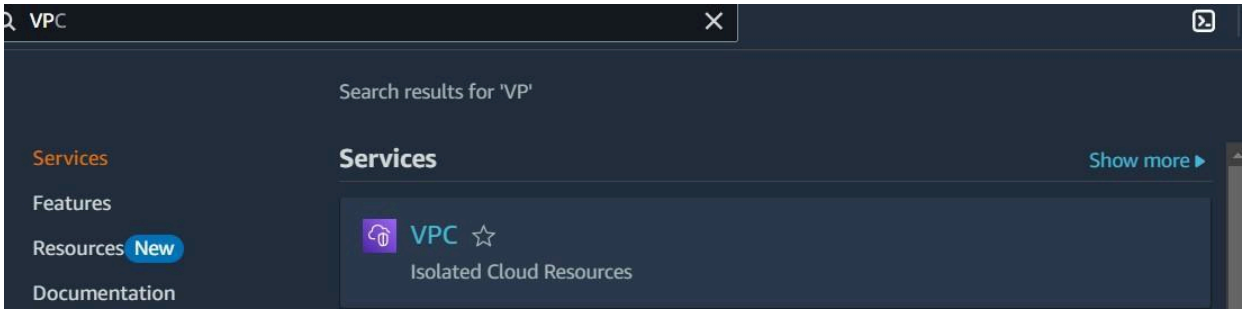
Virtual Private Cloud (VPC): Amazon VPC allows users to create an isolated network within AWS, controlling IP address ranges, subnets, and routing to manage secure communication between resources.

Application Load Balancer (ALB): An ALB operates at the application layer (Layer 7), routing HTTP and HTTPS traffic based on content. It distributes traffic across multiple targets, such as EC2 instances, and supports advanced routing and security features.

Procedure:

Create a VPC:

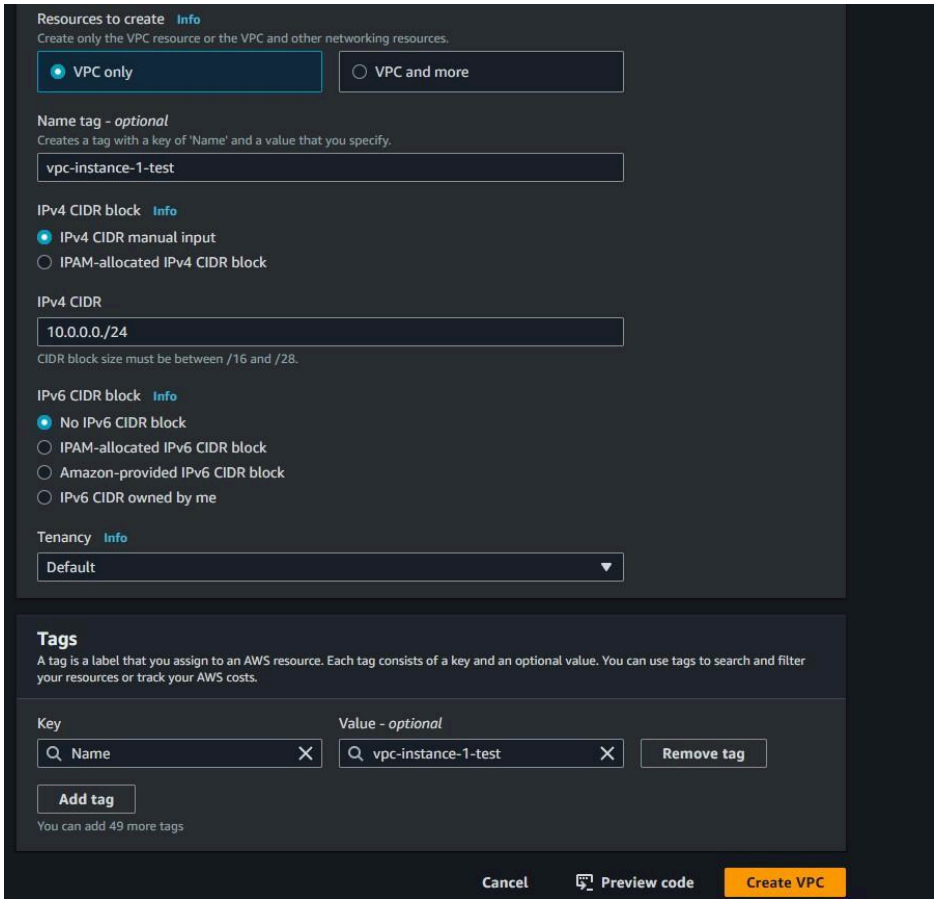
➔ **Log in to AWS Management Console** and navigate to **VPC**.

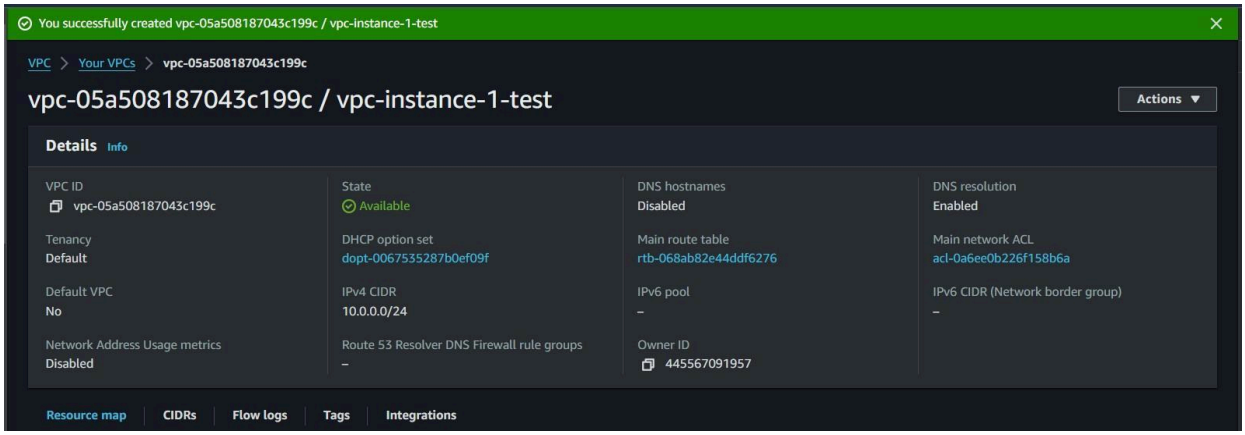


➔ Click on **Create VPC**.

➔ Enter a **Name** for the VPC and an **IPv4 CIDR block** (e.g. 12.0.0.0/16).

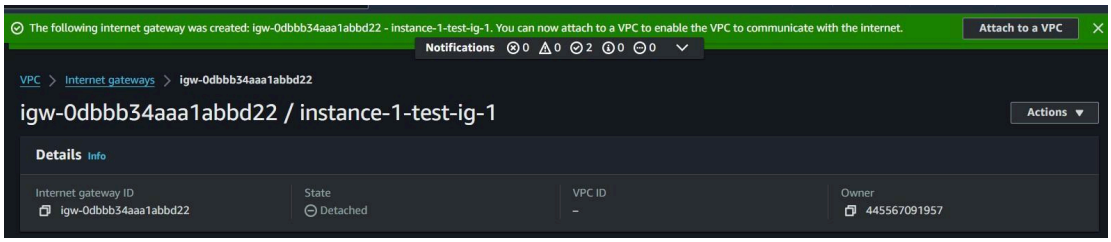
➔ Click **Create VPC**.

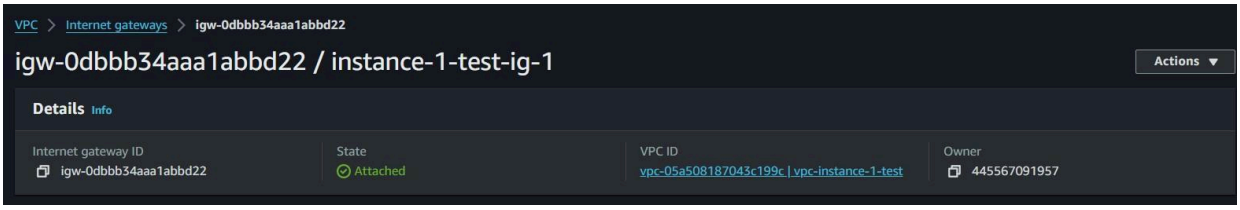




Set Up an Internet Gateway:

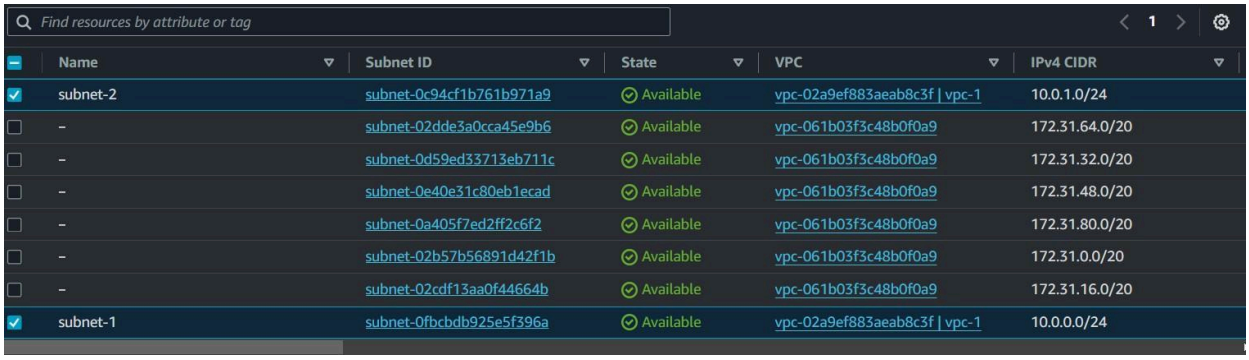
- ➔ Go to **Internet Gateways** in the VPC dashboard.
- ➔ Click **Create internet gateway**, enter a **Name**, and click **Create**.
- ➔ Attach the internet gateway to your VPC by selecting it and clicking **Actions > Attach to VPC**.





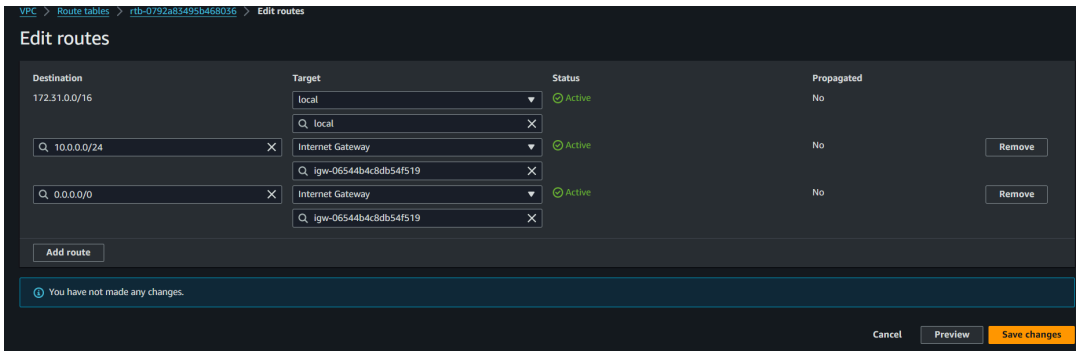
Create Subnets:

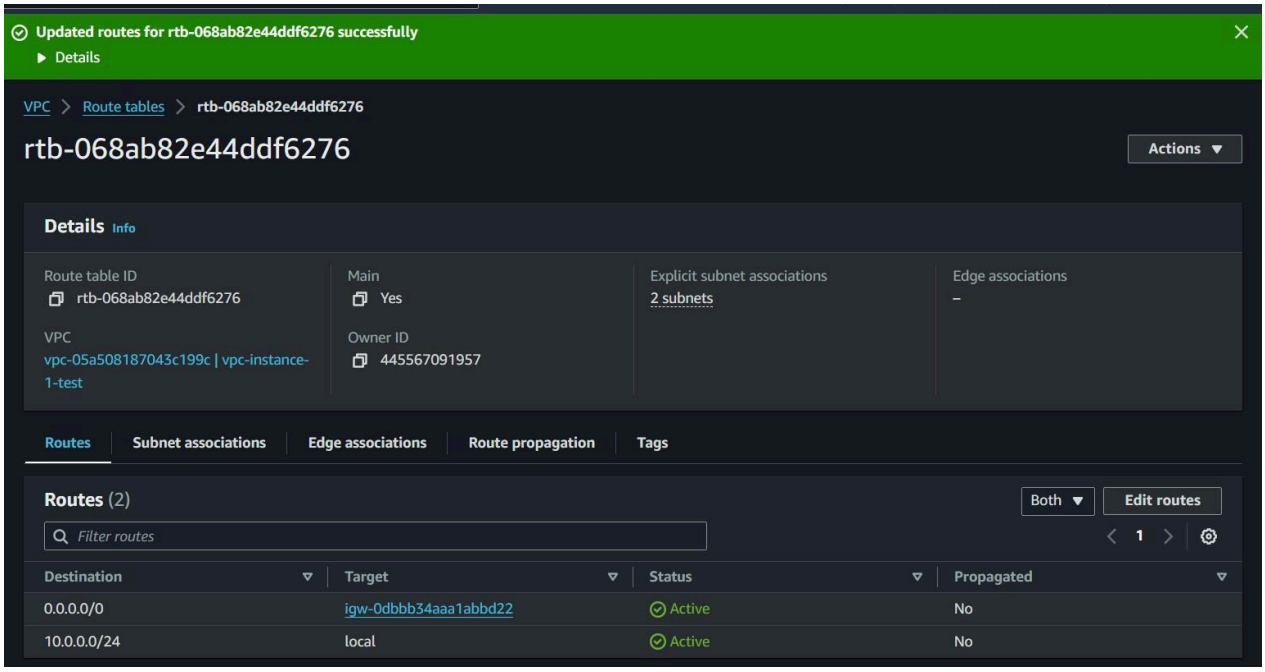
- ➔ Go to **Subnets** in the VPC dashboard.
- ➔ Click **Create subnet**.
- ➔ Select the VPC created earlier, provide a **Name**, and specify an **IPv4 subnet CIDR block** for the subnet (e.g.10.0.1.0/24).
- ➔ Repeat the same for another subnet and specify the IPv4 subnet CIDR as 10.0.0.0/24



Configure Route Tables:

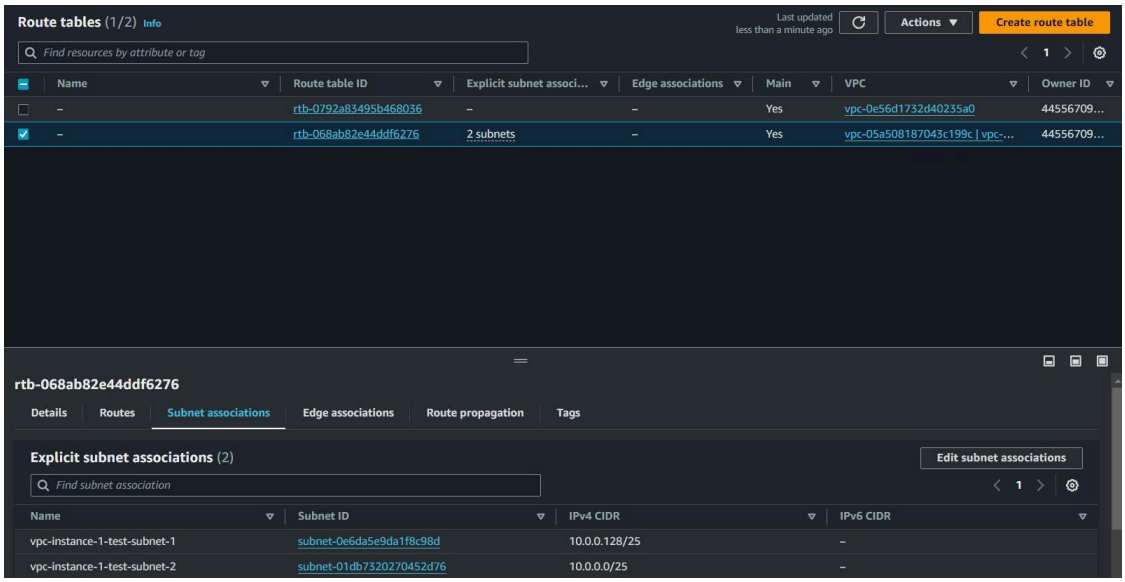
- ➔ Navigate to **Route Tables** in the VPC dashboard.
- ➔ Select the route table associated with your VPC.
- ➔ Click on **Routes** and **Edit routes**.
- ➔ Add a route to direct traffic to the internet gateway (for public subnets) or other destinations as needed.





Add Subnet Associations:

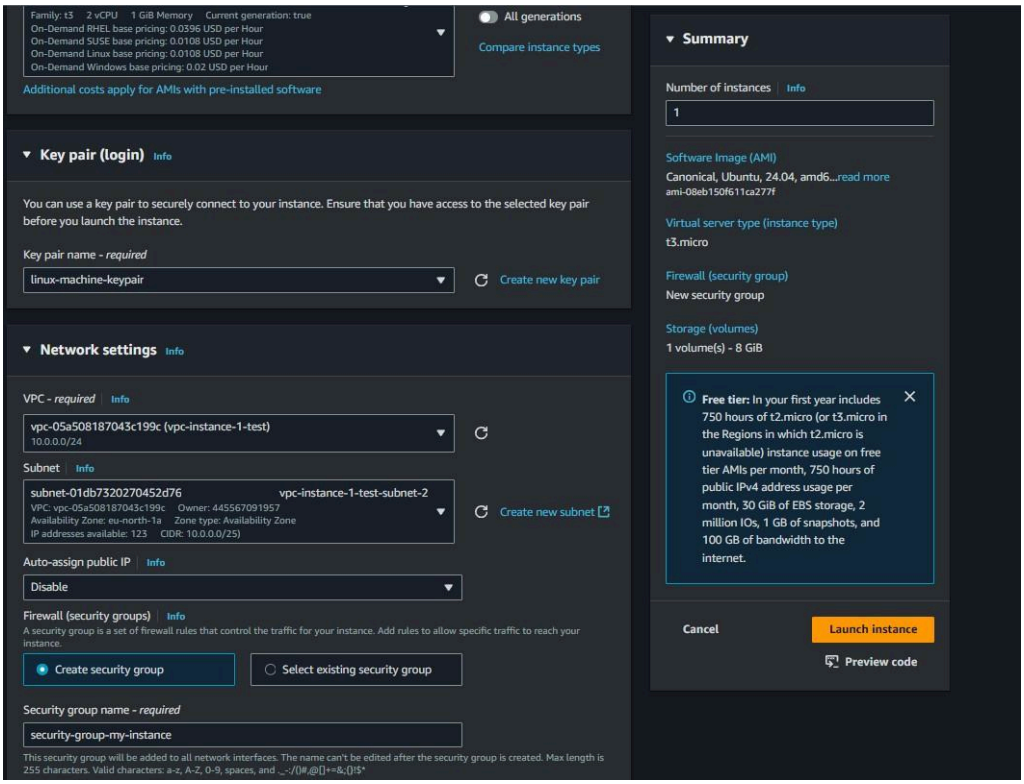
- ➔ Go to the **Subnet Associations** tab in the selected route table.
- ➔ Click **Edit subnet associations**.
- ➔ Select the subnets that should use this route table and click **Save**.



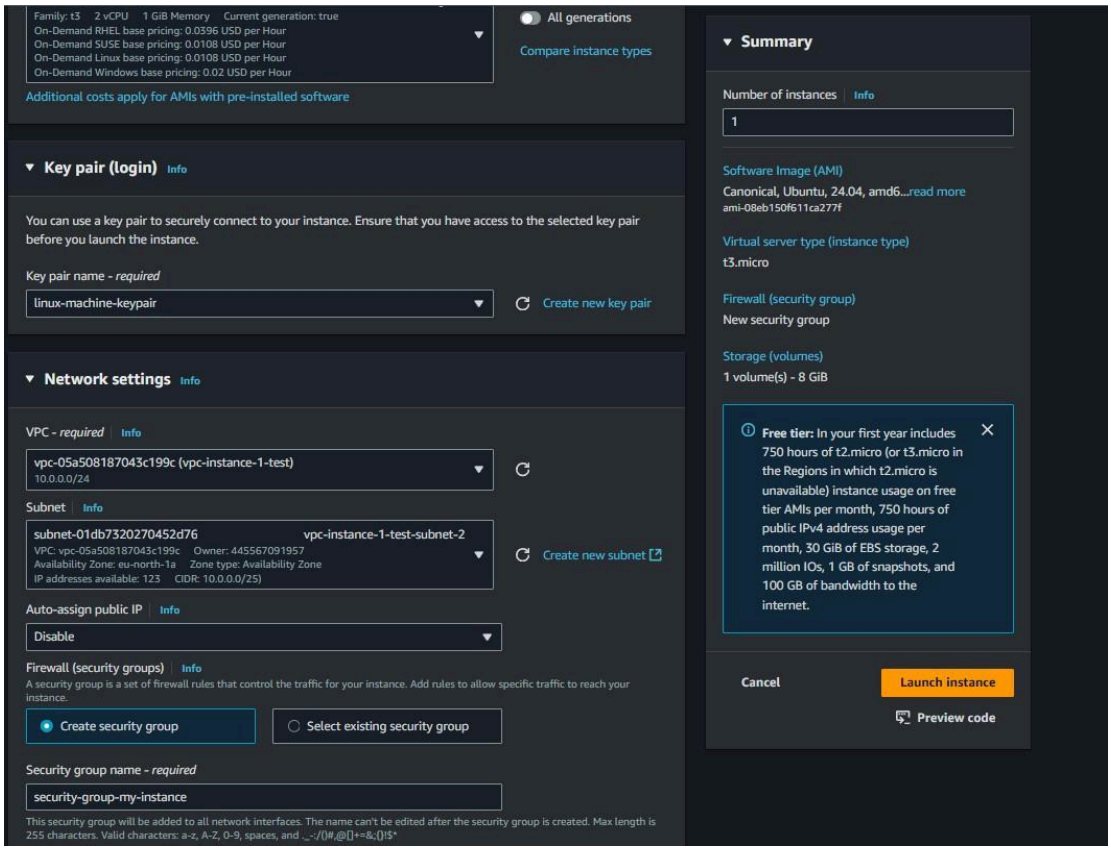
Launch EC2 Instances in Each Subnet:

- ➔ Go to the **EC2 Dashboard** and click **Launch Instance**.
- ➔ Select an **Amazon Machine Image (AMI)** and an **Instance Type**.
- ➔ Configure **Instance Details**: Choose the VPC and select the subnet where the instance will be launched.
- ➔ Add **Storage** if needed.
- ➔ **Configure Security Group** settings to allow required traffic (e.g., HTTP/HTTPS).

➔ Review and **Launch** the instance.

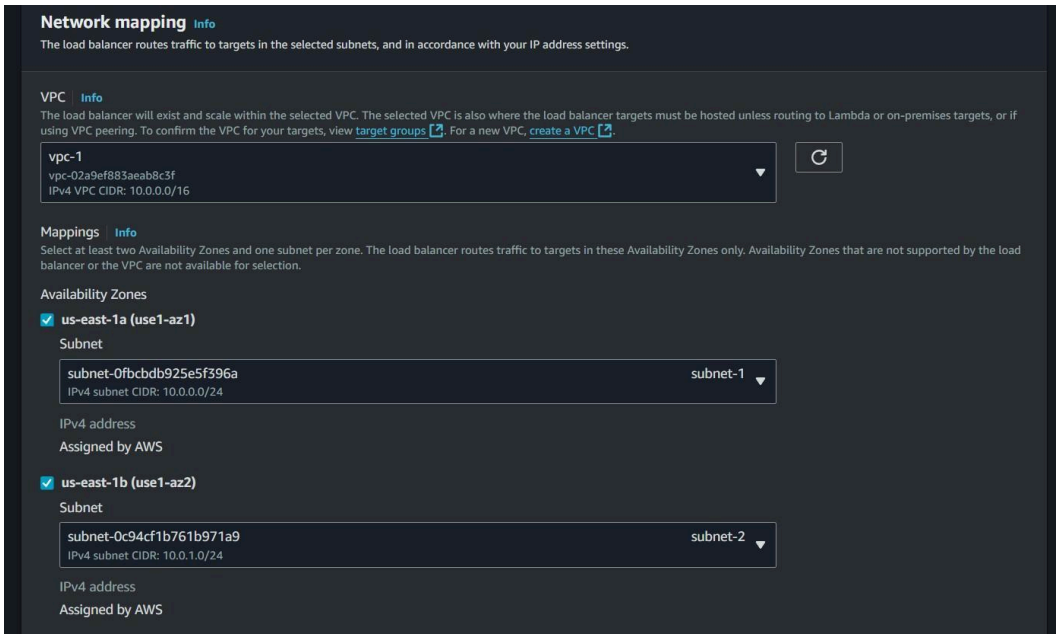


➔ Repeat for the other subnet.



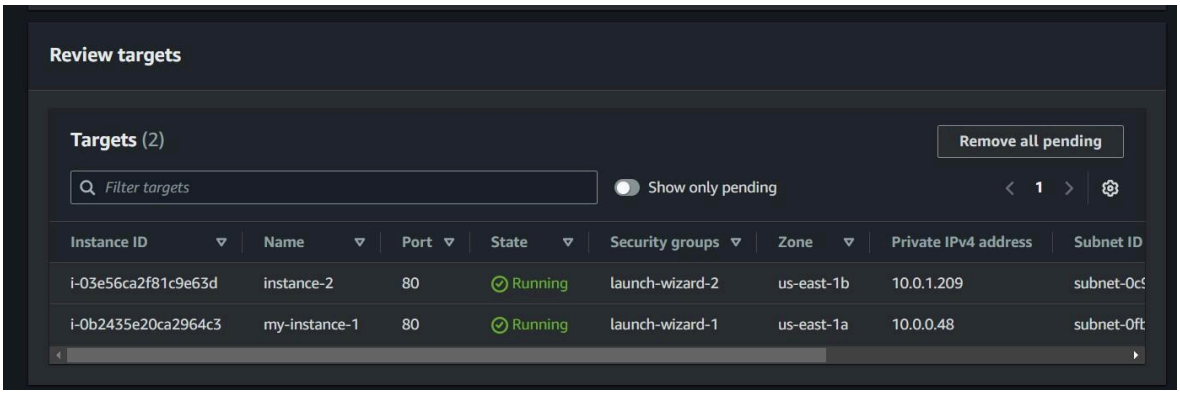
Create an Application Load Balancer:

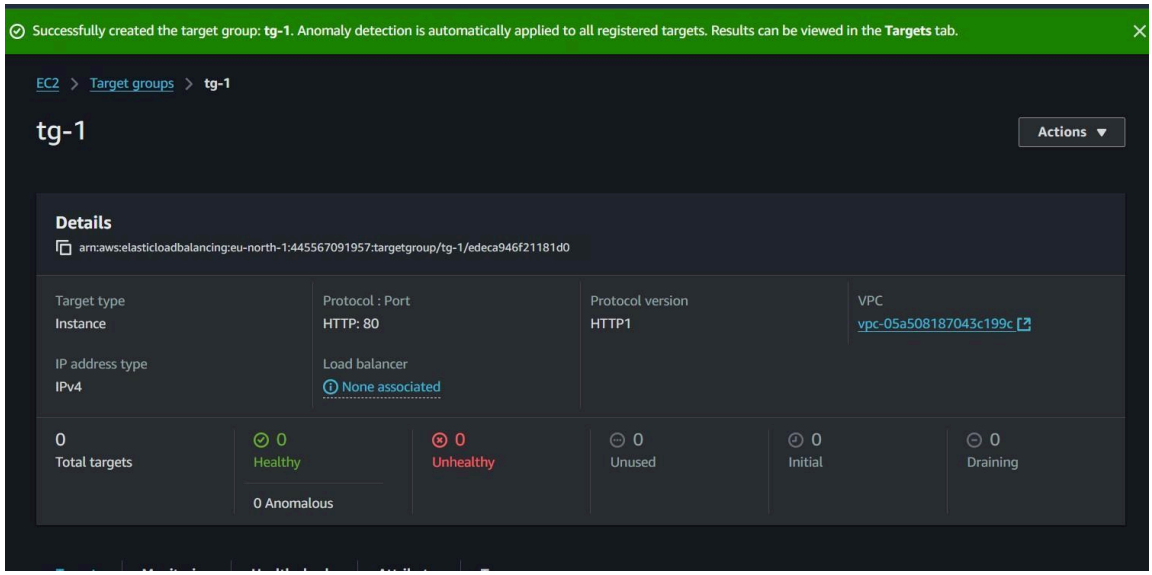
- ➔ Go to the **EC2 Dashboard** and select **Load Balancers**.
- ➔ Click **Create Load Balancer** and choose **Application Load Balancer**.
- ➔ Provide a **Name** and select **internet-facing**.
- ➔ Choose the VPC and subnets where the ALB will operate.
- ➔ Configure **Listeners** (e.g., HTTP on port 80).
- ➔ Create or select **Target Groups** for routing traffic to your EC2 instances.
- ➔ Configure **Health Checks** and **Security Groups** as needed.
- ➔ Review and create the ALB.

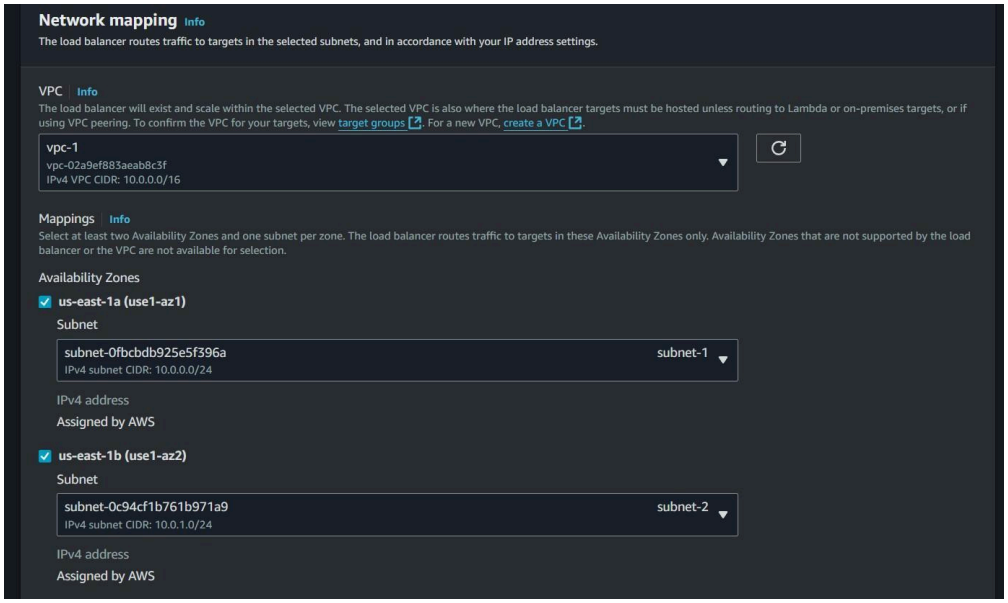


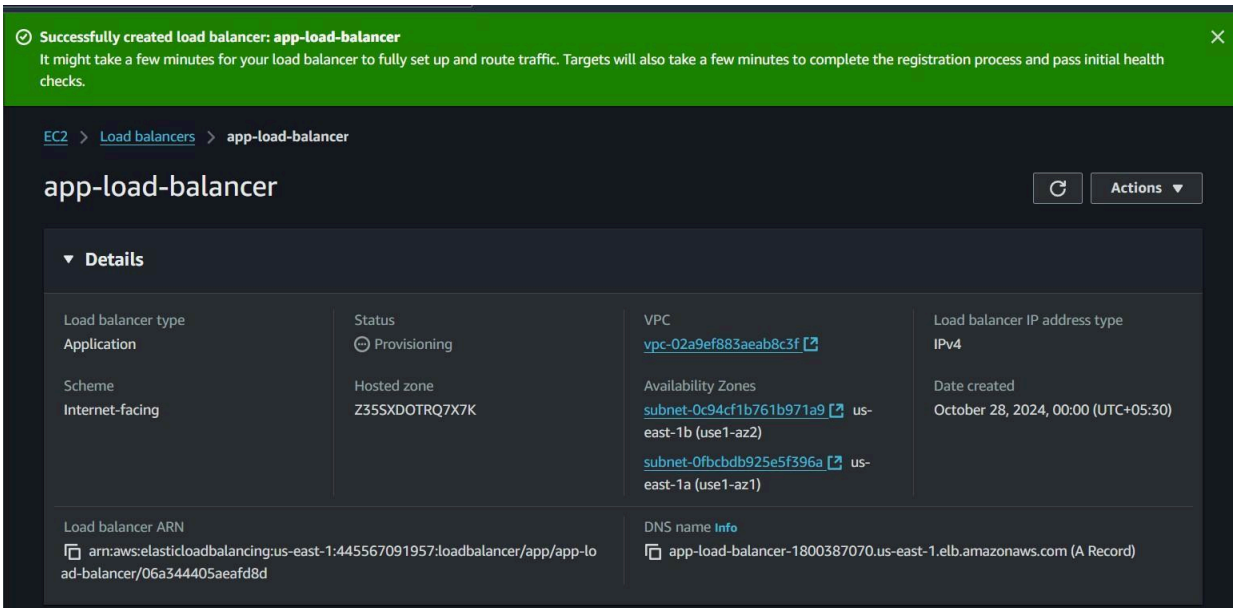
Register Targets:

- In the **Target Groups** section, register your EC2 instances or other resources as targets for the ALB.




Successfully created the target group: tg-1. Anomaly detection is automatically applied to all registered targets. Results can be viewed in the Targets tab.
EC2 > Target groups > tg-1
tg-1
Actions
Details
arn:aws:elasticloadbalancing:eu-north-1:445567091957:targetgroup/tg-1/edeca946f21181d0
Target type
Instance
Protocol : Port
HTTP: 80
Protocol version
HTTP1
VPC
vpc-05a508187043c199c
IP address type
IPv4
Load balancer
None associated
0
Total targets
0
Healthy
0 Anomalous
0
Unhealthy
0
Unused
0
Initial
0
Draining


Network mapping Info
The load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address settings.
VPC Info
The load balancer will exist and scale within the selected VPC. The selected VPC is also where the load balancer targets must be hosted unless routing to Lambda or on-premises targets, or if using VPC peering. To confirm the VPC for your targets, view target groups. For a new VPC, create a VPC.
vpc-1
vpc-02a9ef883aeab8c3f
IPv4 VPC CIDR: 10.0.0.0/16
Mappings Info
Select at least two Availability Zones and one subnet per zone. The load balancer routes traffic to targets in these Availability Zones only. Availability Zones that are not supported by the load balancer or the VPC are not available for selection.
Availability Zones
us-east-1a (use1-az1)
Subnet
subnet-0fbcbdb925e5f396a
IPv4 subnet CIDR: 10.0.0.0/24
subnet-1
IPv4 address
Assigned by AWS
us-east-1b (use1-az2)
Subnet
subnet-0c94cf1b761b971a9
IPv4 subnet CIDR: 10.0.1.0/24
subnet-2
IPv4 address
Assigned by AWS


Successfully created load balancer: app-load-balancer
It might take a few minutes for your load balancer to fully set up and route traffic. Targets will also take a few minutes to complete the registration process and pass initial health checks.
EC2 > Load balancers > app-load-balancer
app-load-balancer
Actions
Details
Load balancer type
Application
Status
Provisioning
VPC
vpc-02a9ef883aeab8c3f
Load balancer IP address type
IPv4
Scheme
Internet-facing
Hosted zone
Z35SXDOTRQ7X7K
Availability Zones
subnet-0c94cf1b761b971a9 us-east-1b (use1-az2)
subnet-0fbcbdb925e5f396a us-east-1a (use1-az1)
Date created
October 28, 2024, 00:00 (UTC+05:30)
Load balancer ARN
arn:aws:elasticloadbalancing:us-east-1:445567091957:loadbalancer/app/app-load-balancer/06a344405aeafd8d
DNS name Info
app-load-balancer-1800387070.us-east-1.elb.amazonaws.com (A Record)

Expected Output:

The expected output includes a fully configured VPC with the specified CIDR block and associated subnets, each with appropriate routing and subnet associations. The internet gateway will be attached and functional, enabling internet access where required. The Application Load Balancer will be created and operational, correctly routing traffic to the target groups within the VPC. Additionally, EC2 instances will be launched in each subnet, accessible as per the configured security groups, and visible in the EC2 dashboard.

Observations:

The VPC should appear as configured, with all specified subnets visible and properly segmented according to their CIDR blocks. An internet gateway should be successfully attached to the VPC, facilitating internet connectivity for public subnets. The route table should correctly reflect the added routes and subnet associations. The Application Load Balancer should be operational, effectively routing traffic to the registered EC2 instances across the different subnets. The EC2 instances should be running in their respective subnets, accessible as per the security group rules, and functioning correctly based on their configuration.

Result -The VPC, subnets, internet gateway, route table, Application Load Balancer, and EC2 instances will be correctly set up and operational as configured.

Title: Create an EC2 Instance and Associate an EBS Volume

Objective:
The purpose of this experiment was to create and configure an EC2 instance in AWS and demonstrate how to attach and manage an Elastic Block Store (EBS) volume. This exercise provided hands-on experience in deploying virtual servers in the cloud and utilizing persistent storage solutions, which are crucial for maintaining scalable and reliable cloud infrastructure.

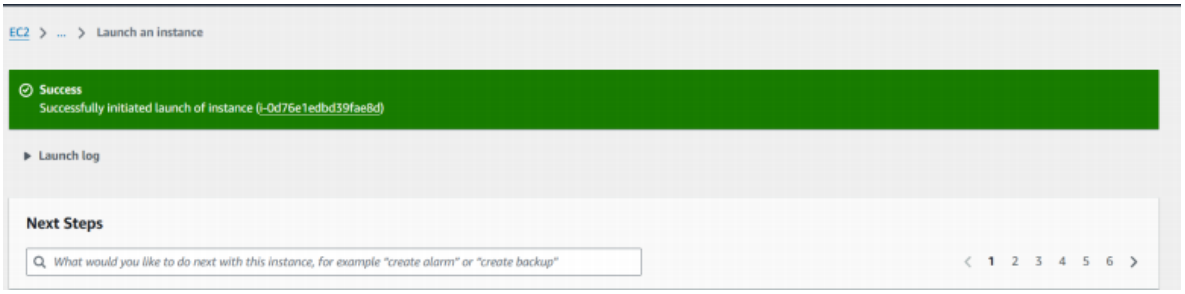
Prerequisites:

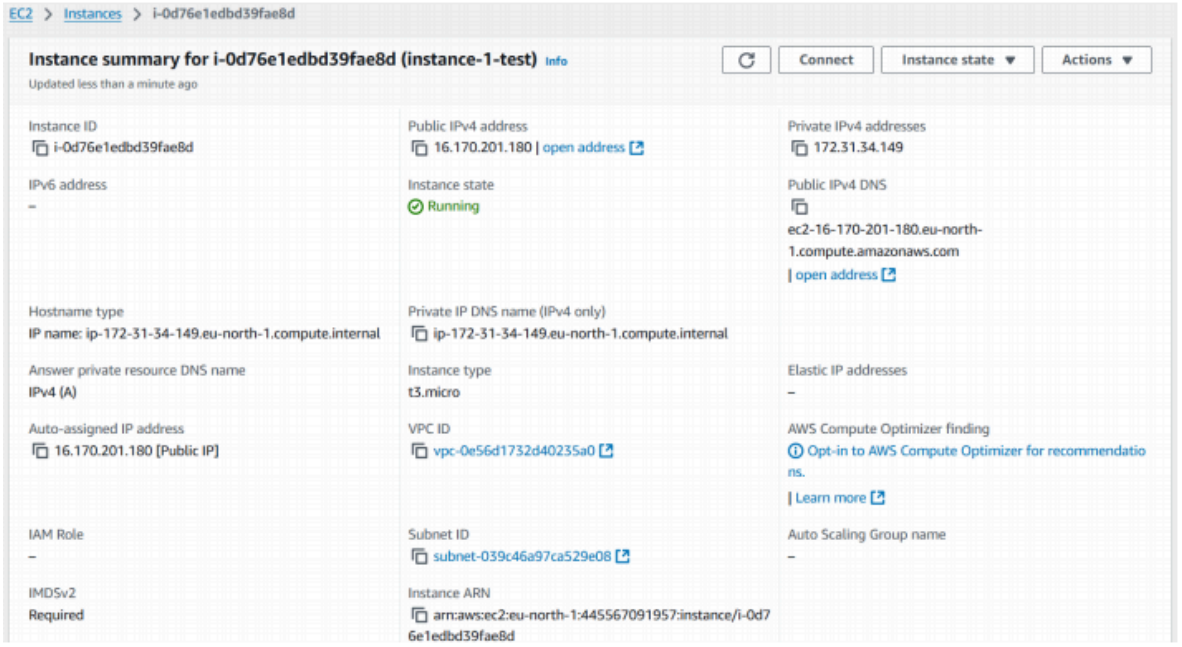
- AWS Account
- Internet Access

Theory:
Amazon EC2 (Elastic Compute Cloud) is a web service that provides scalable compute capacity in the cloud, allowing users to run virtual servers—called instances—with flexible configurations to meet varying computational needs. EC2 offers different instance types tailored for specific tasks, enabling users to scale resources up or down as needed while paying only for the resources utilized. Complementing EC2, Elastic Block Store (EBS) offers persistent block storage that retains data beyond instance termination, suitable for applications that require frequent data updates. EBS volumes come in several types, such as SSD and magnetic, optimized for different performance and cost requirements. Together, EC2 and EBS create a robust, scalable, and cost-effective cloud infrastructure.

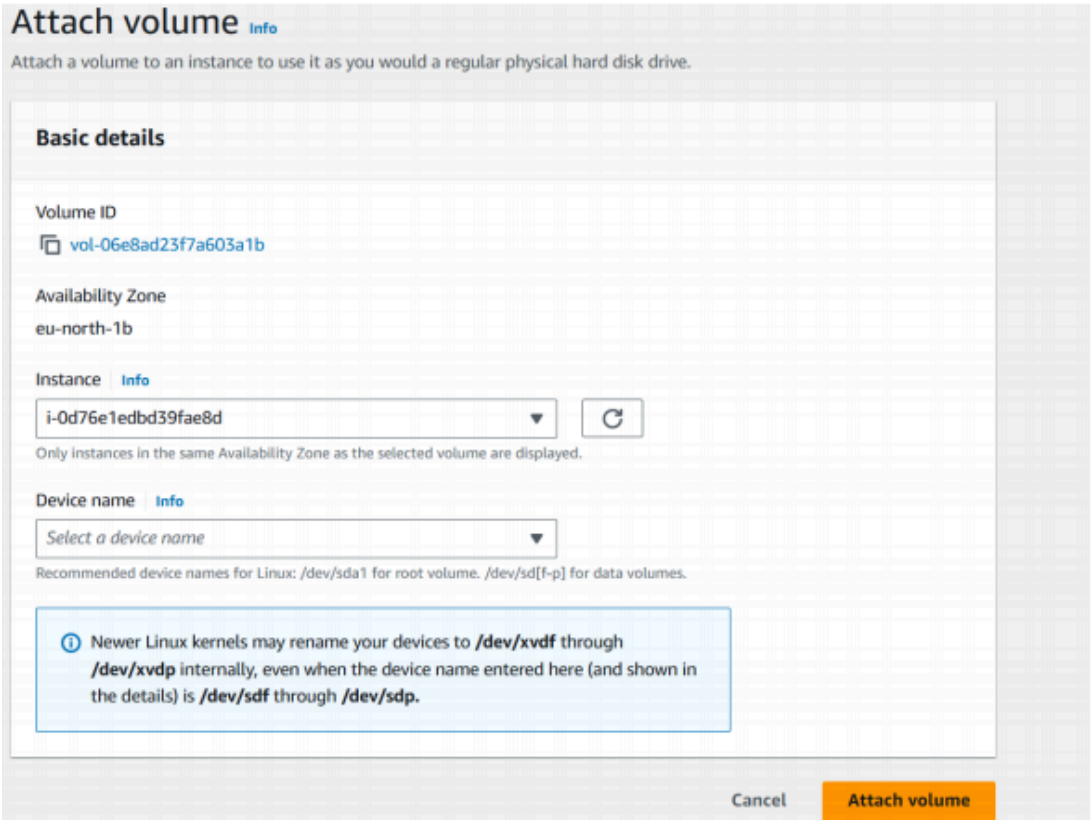
Procedure:

1. **Log in to AWS Management Console:**
    - Open the AWS Management Console.
    - Sign in with AWS credentials.
2. **Launch an EC2 Instance:**
    - Navigate to the EC2 Dashboard from the services menu.
    - Click on "Launch Instance."
    - Select an Amazon Machine Image (AMI) that meets requirements.
    - Choose an instance type based on required compute capacity and performance.
    - Configure instance details like network settings and IAM roles (default settings are typically sufficient).
    - Review and modify the default root volume size, if needed.
    - Set up the security group, defining inbound and outbound rules for network access.
    - Review the instance configuration and click "Launch."
    - Select or create a new key pair for secure access to the instance, then click "Launch Instances."

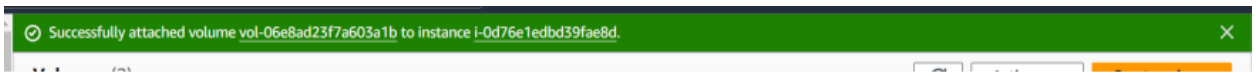




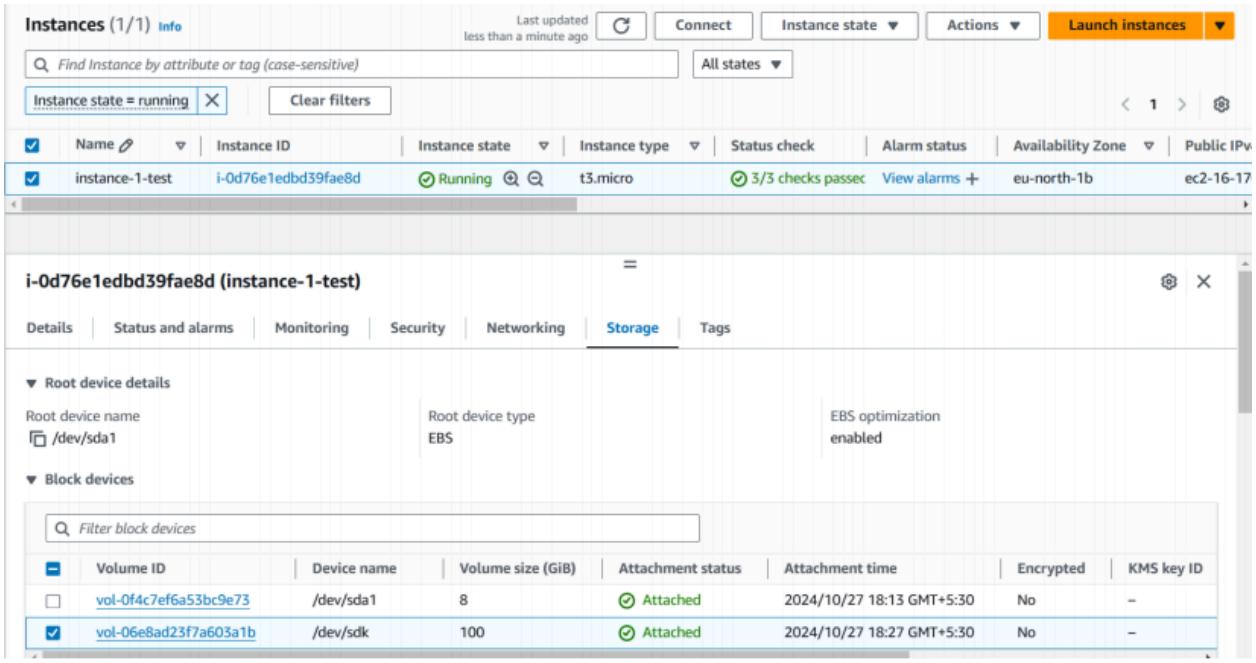
3. **Create and Attach an EBS Volume:**
    - Go to the "Volumes" section under Elastic Block Store in the EC2 Dashboard.
    - Click on "Create Volume."
    - Specify the volume type, size, and availability zone (matching the EC2 instance's availability zone).
    - Click "Create Volume."



4. **Attach EBS Volume to EC2 Instance:**
    - Select the newly created EBS volume from the list.
    - Click "Actions" and select "Attach Volume."
    - Choose the EC2 instance from the dropdown menu.
    - Specify the device name and click "Attach Volume."



Attached: The volume is currently attached to an EC2 instance and is in use



Expected Output:
The expected outcome included a running EC2 instance with the specified configuration, accessible via its public DNS or IP address based on the security group settings. The attached EBS volume should be visible in the EC2 dashboard with an "in-use" status. On the instance, the volume should be identifiable and accessible, appearing in disk management tools. If formatted and mounted (for Linux), the volume should be available at the specified mount point and ready for data storage and use.

Observations:
The EC2 instance should run as configured and be accessible. The EBS volume appeared in the dashboard with an "in-use" status and was recognized on the instance. For Linux instances, once formatted and mounted, the volume was integrated into the file system and accessible at the designated mount point.

Result:
The EC2 instance was operational with the EBS volume attached, visible, accessible, and usable as expected.

Title 1: To create an EC2 instance via CloudFormation

Objective:

The objective of this CloudFormation template is to automate the creation of an EC2 instance with specified properties, including instance type, AMI ID, and key pair name. This allows for consistent and repeatable deployment of EC2 instances within an AWS environment, reducing manual configuration efforts and ensuring infrastructure as code practices.

Theory:

AWS CloudFormation: AWS CloudFormation is a service that enables users to define and provision AWS infrastructure using code. By creating templates in JSON or YAML format, users can automate the deployment of AWS resources, ensuring consistent and repeatable infrastructure setups. CloudFormation templates can define a wide range of AWS resources, including EC2 instances, VPCs, and security groups, and manage their configurations and dependencies.

Procedure:

Prepare Your CloudFormation Template:

- ➔ Create a YAML or JSON file with your CloudFormation template. For example, use the provided YAML template to define an EC2 instance.

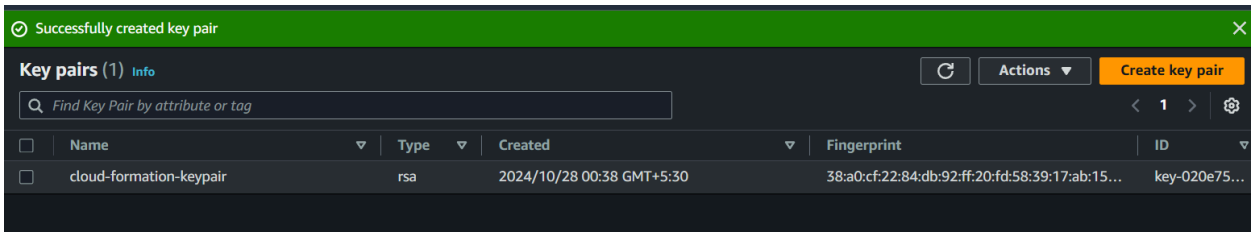
YAML

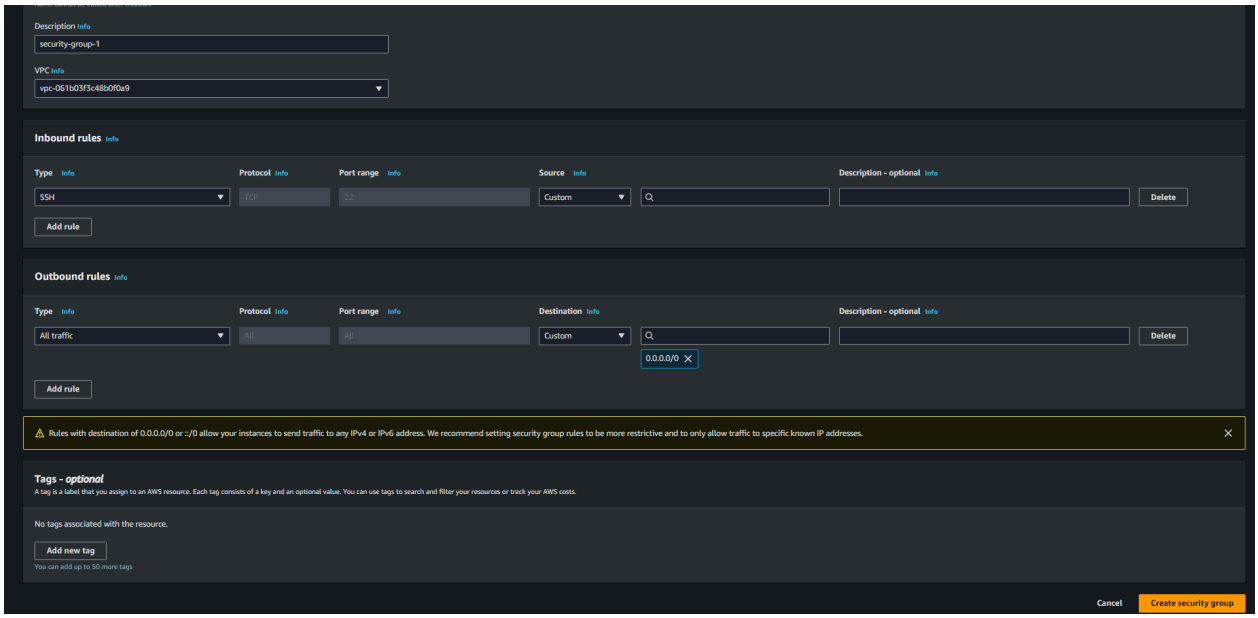
```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Creates an EC2 Instance'

Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:

      ImageId: ami-0c55b159cbfafe1f0  # Replace with the desired AMI ID
      InstanceType: t3.micro
      KeyName: MyKeyPair  # Replace with your key pair name
      SecurityGroupIds:
        - sg-0123456789abcdef0  # Replace with your security group ID
```

Log in to AWS Management Console:

- ➔ Open the AWS Management Console.
- ➔ Sign in with your AWS credentials.

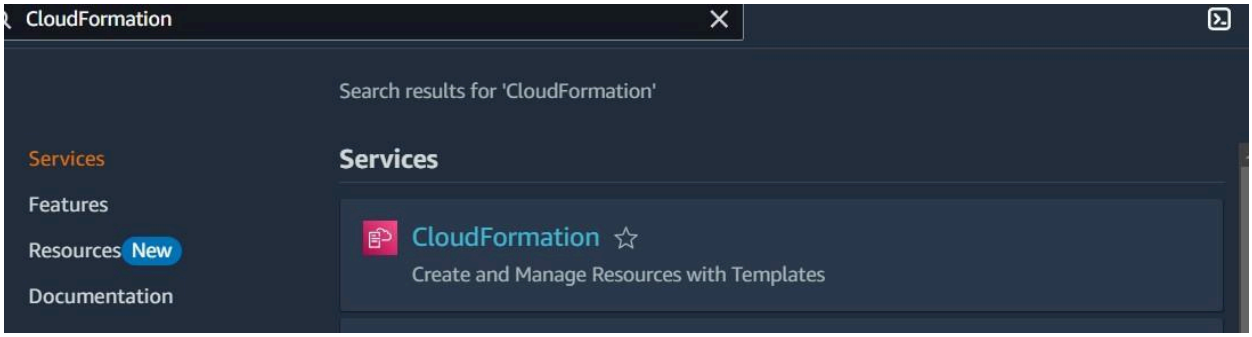
Create a Key pair and a security Group to add the values in the JSON file





Navigate to CloudFormation:

- ➔ Go to the **CloudFormation** service from the AWS Management Console.

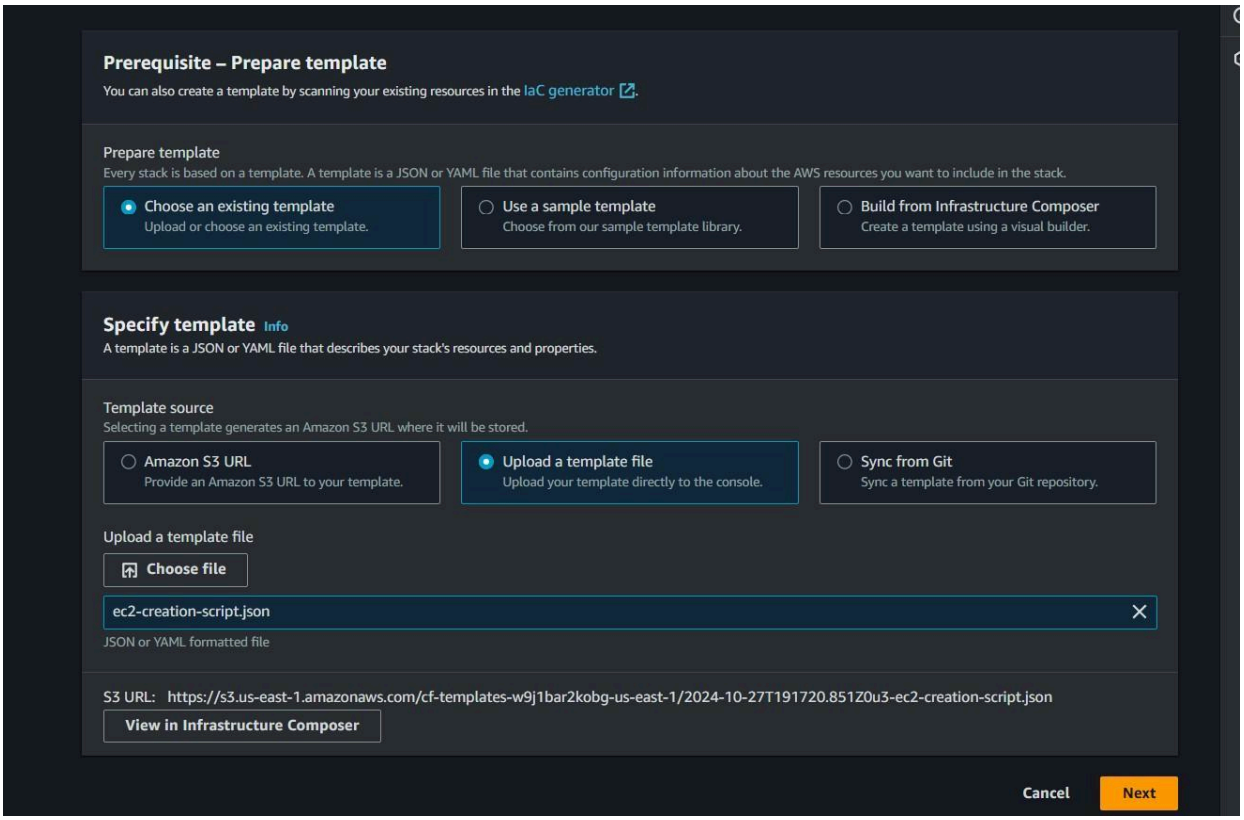


Create a New Stack:

- ➔ Click on **Create stack**.
- ➔ Select **With new resources (standard)**.

Specify Template:

- ➔ Choose **Upload a template file**.
- ➔ Click **Choose file** and upload your CloudFormation template file(.yaml).
- ➔ Click **Next**.

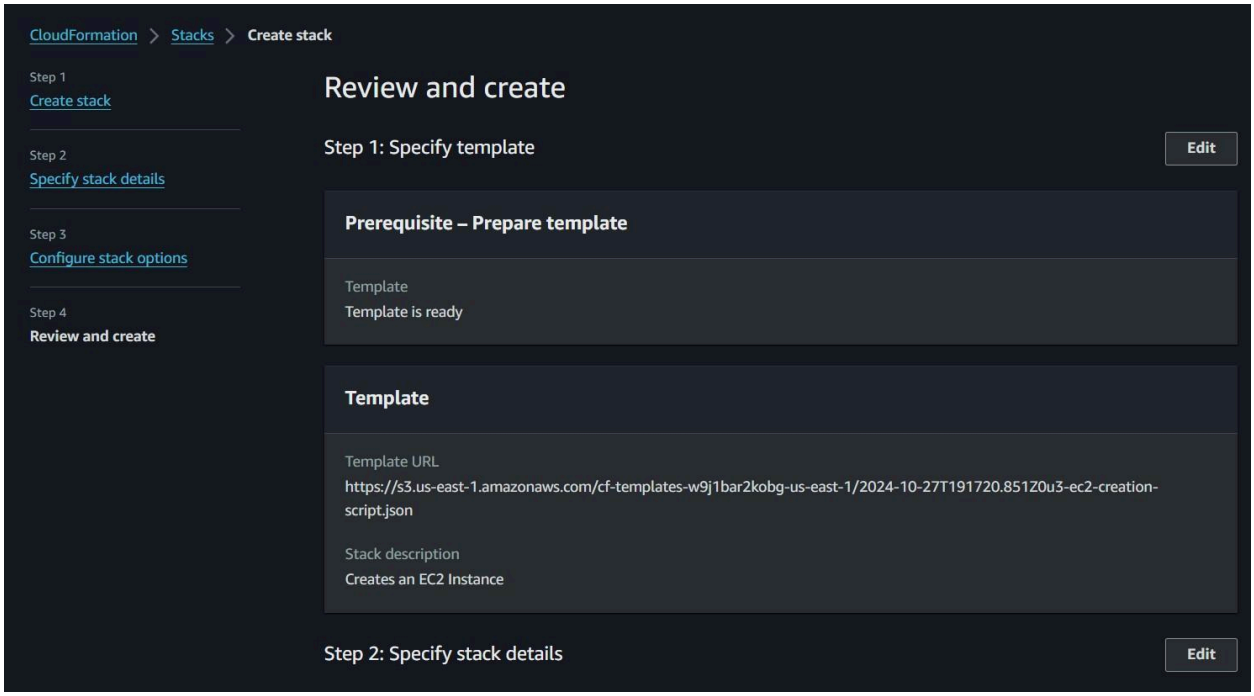


Configure Stack Details:

- ➔ Enter a **Stack name**
- ➔ Review other optional settings if needed, and click **Next**.

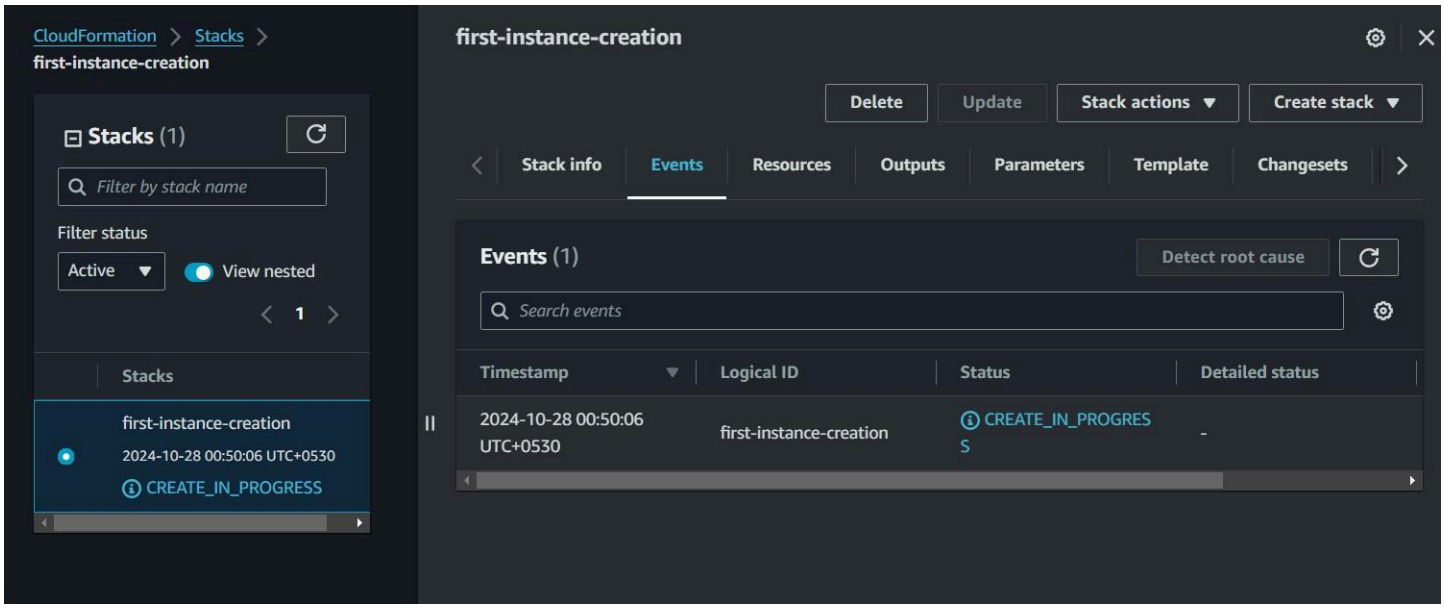
Review and Create:

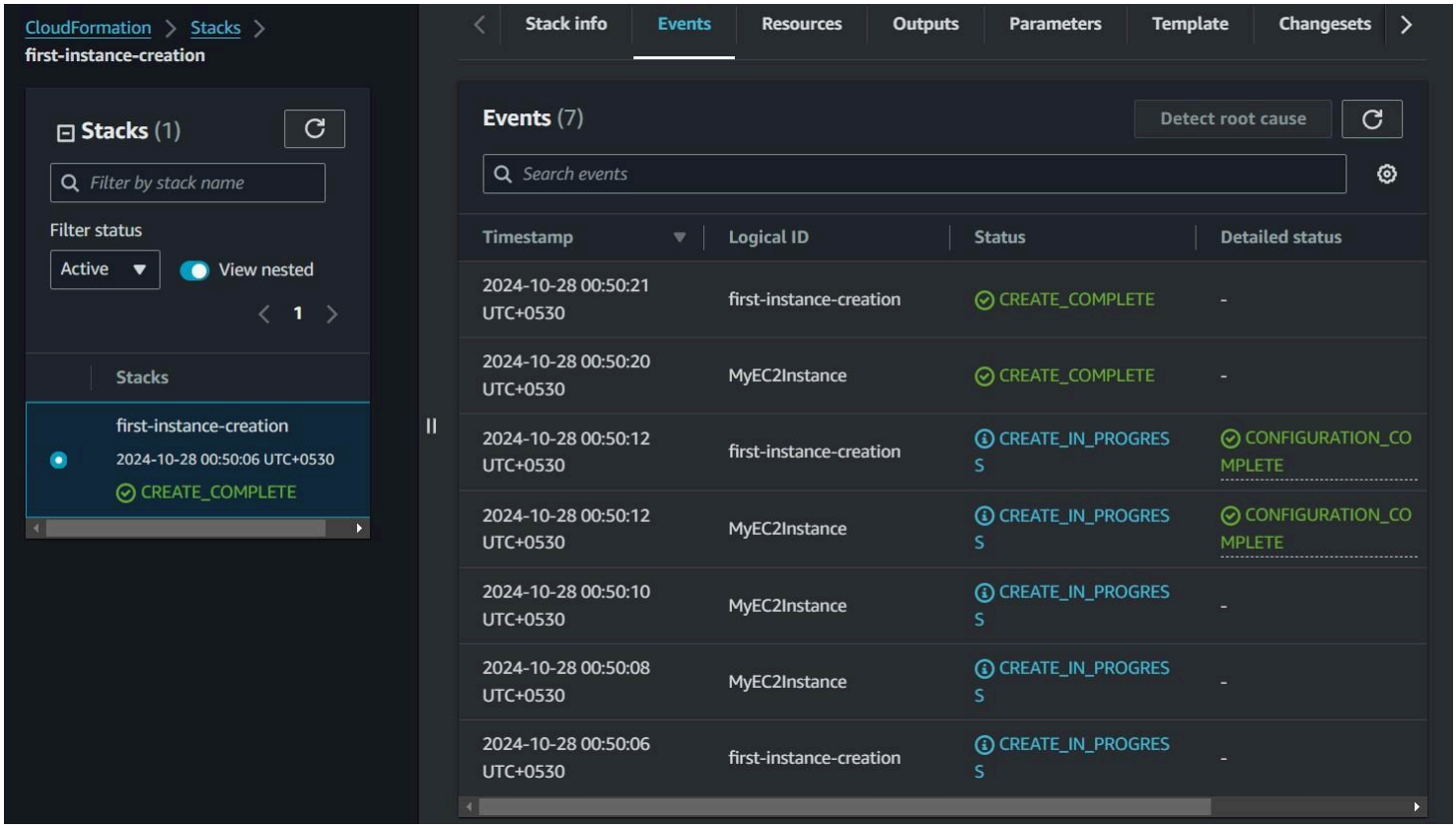
- ➔ Review the details of your stack.
- ➔ Click **Create stack** to start the stack creation process.



Monitor Stack Creation:

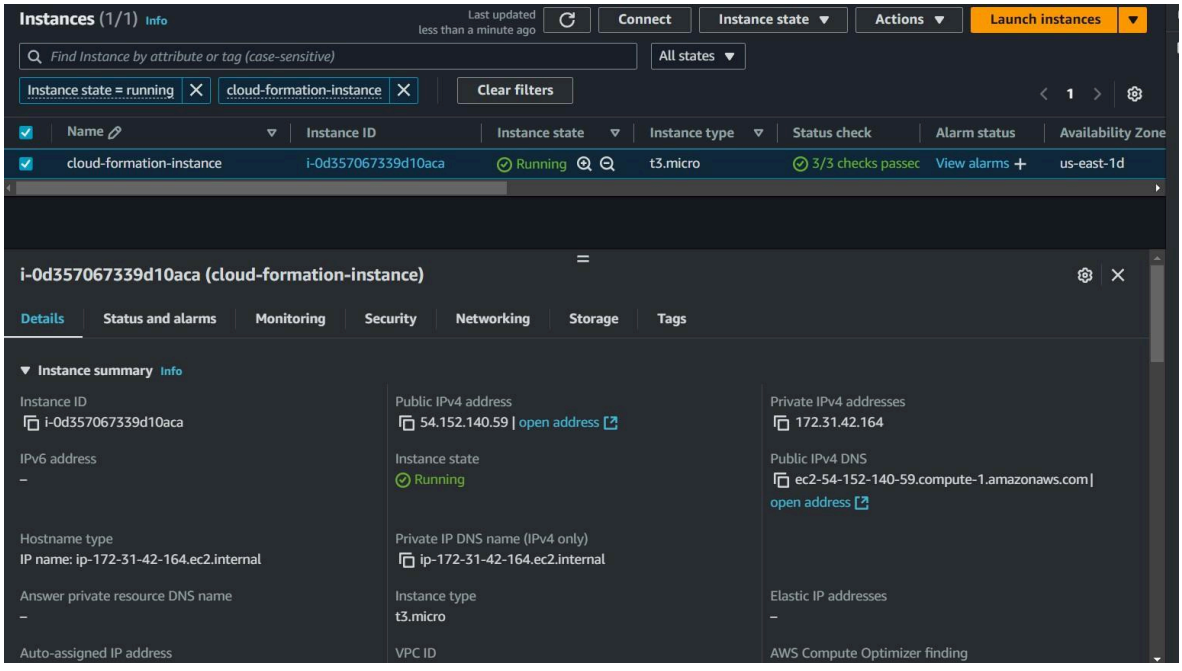
- ➔ Monitor the stack creation progress in the **CloudFormation** dashboard.
- ➔ Once the stack status is **CREATE_COMPLETE**, the EC2 instance will be created.





Verify EC2 Instance:

- ➔ Go to the **EC2 Dashboard**.
- ➔ Verify that the EC2 instance is created and running with the properties specified in the CloudFormation template.



Expected Output:

The expected output of the CloudFormation stack creation is an EC2 instance successfully launched with the properties specified in the template. The instance will be of type t2.micro, use the specified AMI ID, and be associated with the provided key pair for secure access. The instance will appear in the EC2 Dashboard with a status of **running**, reflecting the configurations set in the CloudFormation template.

Observations:

Upon successful execution of the CloudFormation stack, the EC2 instance should be visible in the EC2 Dashboard with the specified attributes, including instance type, AMI ID, and key pair. The instance should be in a **running** state, indicating that it has been properly launched and is operational. Verify that the instance details match the configuration provided in the CloudFormation template, and ensure that it is accessible using the defined key pair.

Result:

The CloudFormation stack will result in a running EC2 instance configured with the specified instance type, AMI ID, and key pair, as defined in the template.

Title 2: EC2 Provisioned Web Server with User Data

Objective:

To provision an EC2 instance configured as a web server, using user data for automatic setup upon launch.

Theory:

User Data: User Data allows you to automate the configuration of EC2 instances during launch. It can be used to run scripts or commands that configure the instance, such as installing software, updating packages, or setting up services. This is typically done using shell scripts or cloud-init directives for Linux instances.

Web Server Configuration: A web server, such as Apache or Nginx, can be set up on an EC2 instance to serve web content. User Data scripts can automate the installation and configuration of these web servers, making the instance ready for hosting web applications.

Procedure:

Prepare User Data Script:

- ➔ Create a script to configure the web server. For example, a shell script to install Apache and set up a basic HTML page:

```
#!/bin/bash
yum update -y
yum install -y httpd
systemctl start httpd
systemctl enable httpd
echo "Hello, World!" > /var/www/html/index.html
```

Log in to AWS Management Console:

- ➔ Open the AWS Management Console.
- ➔ Sign in with your AWS credentials.

Navigate to EC2 Dashboard:

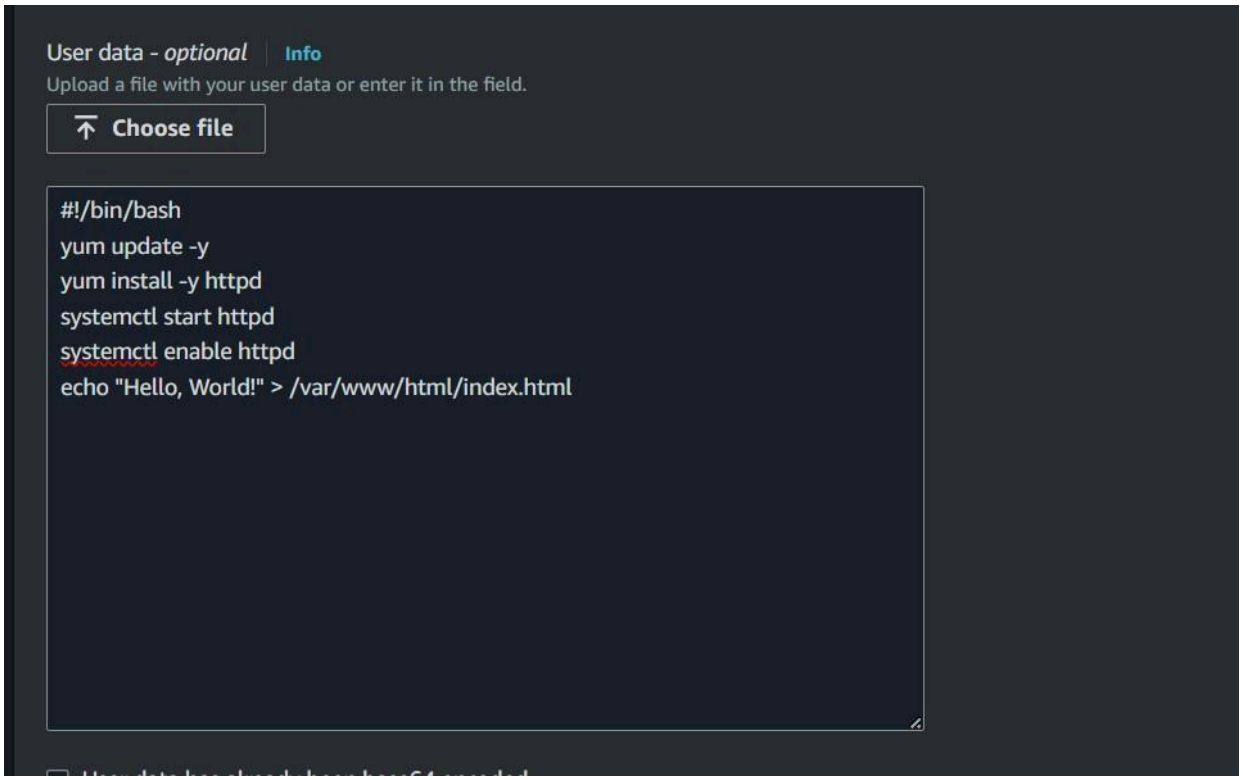
- Go to the **EC2** service from the AWS Management Console.

Launch a New Instance:

- ➔ Click **Launch Instance**.
- ➔ Choose an **Amazon Machine Image (AMI)**
- ➔ Select an **Instance Type**
- ➔ Click **Next: Configure Instance Details**.

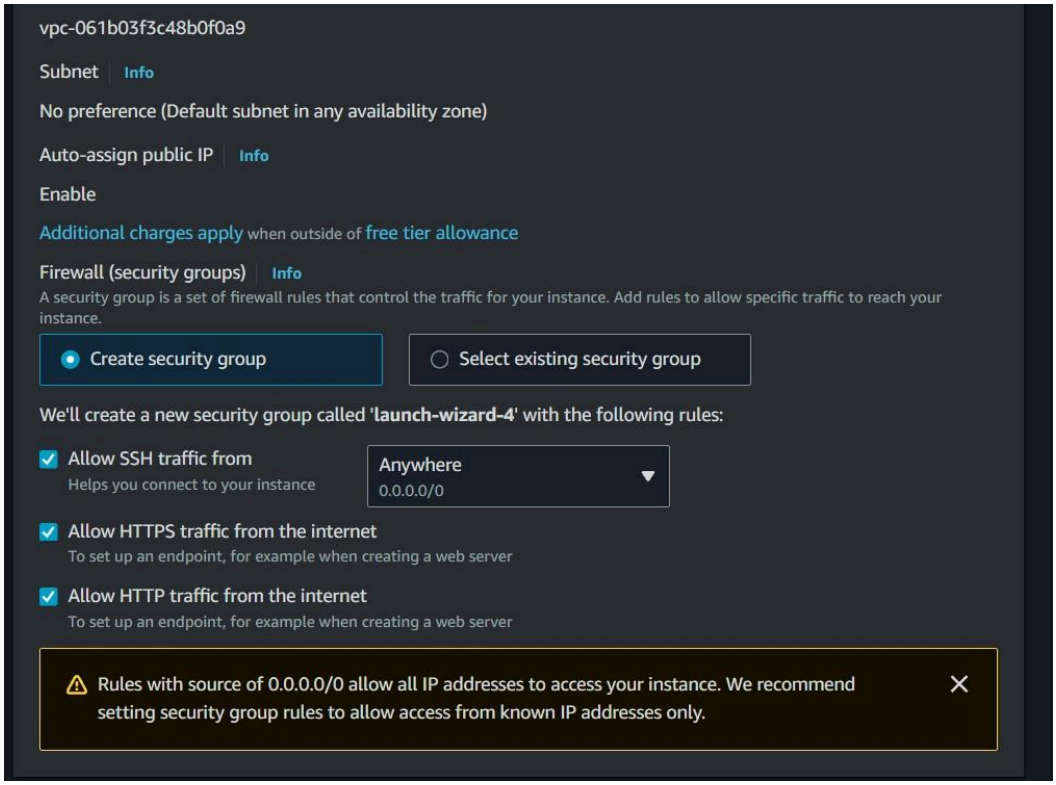
Configure User Data:

- ➔ In the **Configure Instance** section, scroll down to **Advanced Details**.
- ➔ Paste your User Data script into the **User data** field.



Configure Security Group:

➔ Configure the security group to allow HTTP traffic (port 80) and any other necessary ports.



Review and Launch:

➔ Review the instance configuration.
➔ Select an existing key pair or create a new one for SSH access.
➔ Click **Launch**.

Verify Web Server:

➔ Once the instance is running, obtain its public IP address from the **Instances** section of the EC2 Dashboard.
➔ Access the instance via a web browser using its public IP to ensure the web server is serving the HTML page created by the User Data script.

Hello, World!

Expected Output:

The EC2 instance will be provisioned with Apache installed and running. Upon accessing the instance's public IP address via a web browser, you should see a simple web page displaying the text "Hello, World!" This confirms that Apache was successfully installed and the User Data script executed correctly, setting up the web server and serving the HTML content as specified.

Observations:

After running the script, the EC2 instance should have Apache installed and running, with the service configured to start on boot. The HTML file should contain the text "Hello, World!" Accessing the instance's public IP address via a web browser should display the "Hello, World!" message, confirming that the User Data script executed successfully and set up the web server as intended.

Result:

The EC2 instance will display "Hello, World!" when accessed via its public IP address, indicating that the web server is running.

Title:
Deploying a Static Web Application in Microsoft Azure

Objective:
The objective was to deploy a static web application on Azure App Services, ensuring a reliable, scalable, and cost-effective solution for hosting. This deployment leveraged Azure's cloud infrastructure to deliver fast and secure access to web resources, while integrating with other Azure services for monitoring, security, and performance optimization.

Prerequisites:

- Microsoft Azure Account
- Internet Access
- Code for the Application

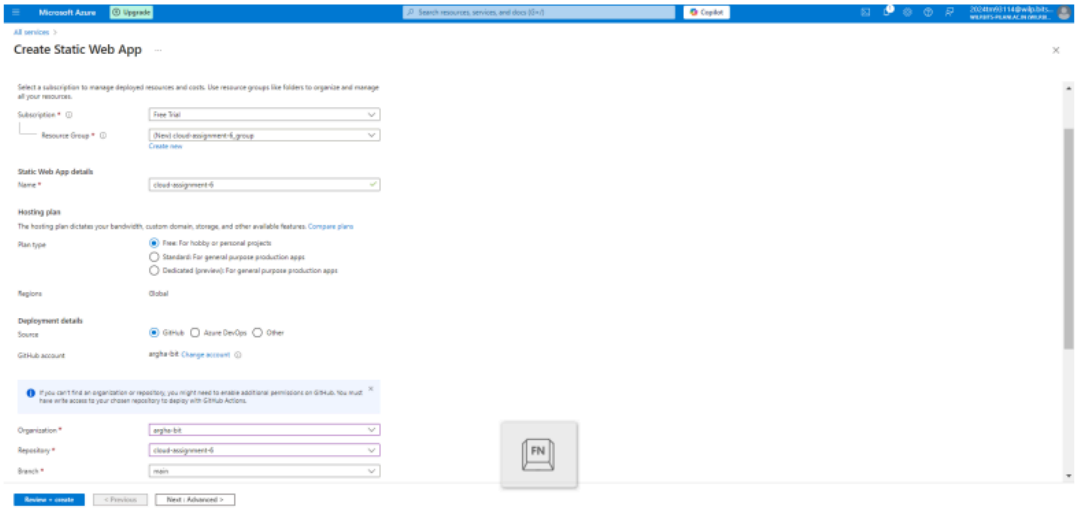
Theory:
Azure is Microsoft's cloud computing platform that offered a broad range of services, including compute, storage, databases, and networking. It provided both infrastructure-as-a-service (IaaS) and platform-as-a-service (PaaS) solutions, allowing businesses to build, deploy, and manage applications globally without the need for server management. Azure App Services made deploying web applications easy, while integrated tools and security features enhanced scalability and resilience. Azure's seamless integration with AI, data analytics, and IoT further delivered a comprehensive cloud solution for modern businesses.

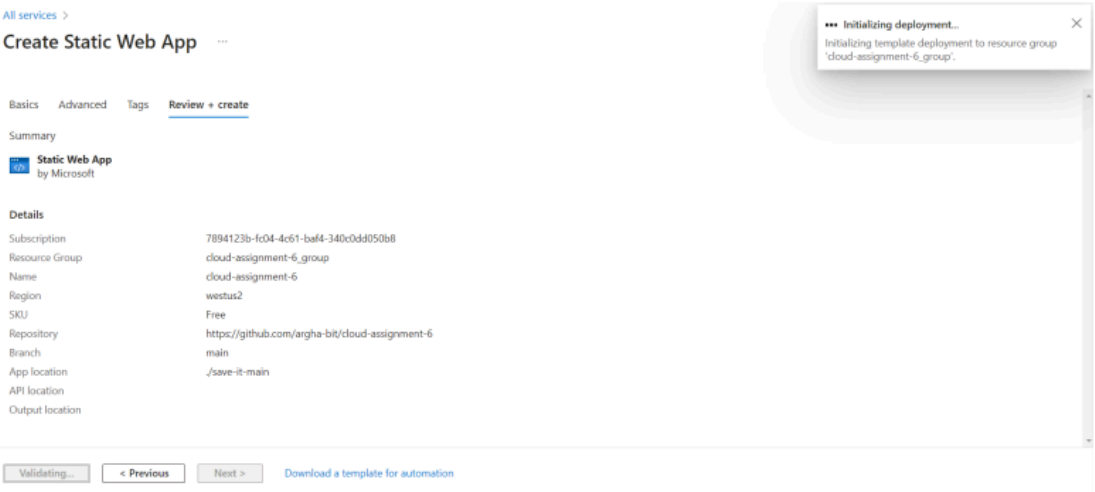
Procedure:

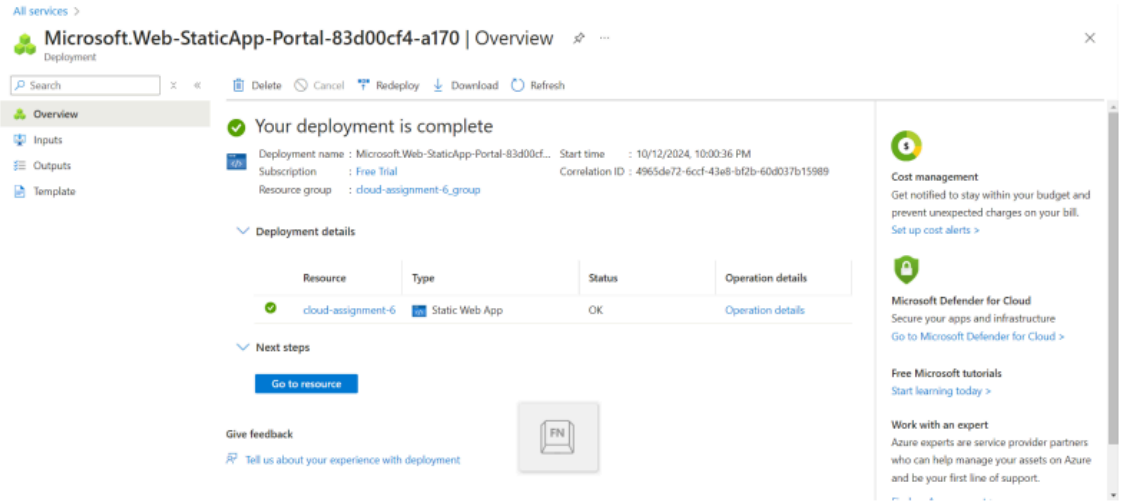
1. **Logged into Azure Portal:**
    - I navigated to the Azure Portal and logged in with my Azure credentials.
2. **Created an App Service:**
    - I clicked on "Create a resource" and selected "App Services."
    - After that, I clicked on "Create" to start the App Service configuration process.
3. **Configured Basic Settings:**
    - I filled in the required fields as follows:
        - **Subscription:** Selected my Azure subscription.
        - **Resource Group:** Created a new resource group or selected an existing one.
        - **Name:** Entered a unique name for the app service.
        - **Region:** Chose a region for the app based on proximity and performance considerations.
        - **Deployment Source:** Configured GitHub as the deployment source for the application's static files.
4. **Configured the App Service Plan:**

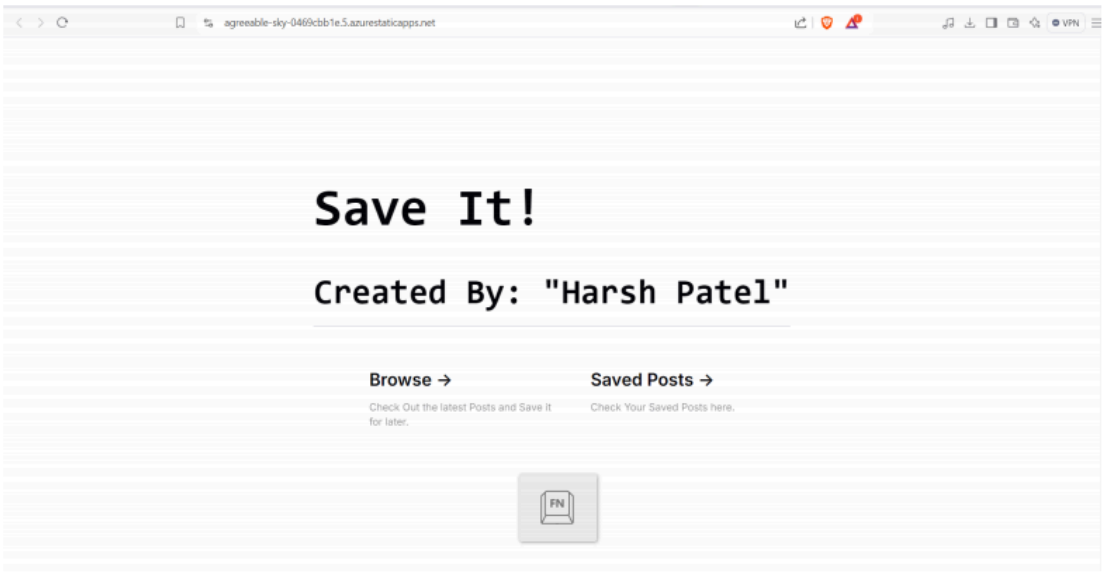
- I chose a pricing plan that suited my needs. In this case, I opted for the **Free** tier, which was sufficient for the static website.
- I reviewed the settings and clicked “Next” to continue.

5. **Reviewed and Created the App Service:**
   - I carefully reviewed all the configuration settings to ensure they were correct, and then clicked “Create” to provision the App Service.
6. **Deployed Static Files:**
   - Once the App Service was successfully created, I deployed the static files (HTML, CSS, JS) through the GitHub URL I had configured earlier. This allowed for seamless integration and quick updates from my source control repository.
7. **Accessed the Deployed Application:**
   - After the deployment, I navigated to the App Service URL (provided by Azure) to view the static web application live. The app was accessible via a public URL, which followed the pattern `https://<app-name>.azurewebsites.net`.









Expected Output:
After successfully deploying the static web app in Azure App Services, I was able to access it through a public URL (like `https://<app-name>.azurewebsites.net`). The static content, including HTML, CSS, and JavaScript files, was displayed as intended. Additionally, Azure's monitoring tools became available, providing insights into the app's performance and deployment logs. There was also the option to set up a custom domain and enable SSL to ensure secure connections over HTTPS, making the application more secure and professional.

Azure’s management dashboard further allowed for easy monitoring, scaling, and configuration of the app service settings.

Observations:
Deploying a static web app in Azure App Services was both efficient and user-friendly. The process facilitated quick setups and seamless updates by integrating with source control platforms like GitHub. Azure’s built-in performance metrics were useful for monitoring and optimizing the application, while features like custom domains and SSL certificates enhanced security and professionalism. Overall, the experience showcased Azure’s strengths in providing scalable and reliable web hosting and management solutions for businesses.

Title:
Exploring SaaS Solutions - Flowlu CRM

Objective:
The objective of this task was to evaluate the feasibility of Flowlu as a comprehensive, free SaaS CRM solution for small businesses. The task involved exploring Flowlu's various features, such as project management, task creation, finance management, and automation capabilities, to determine if it met the needs of small business operations.

Prerequisites:
None.

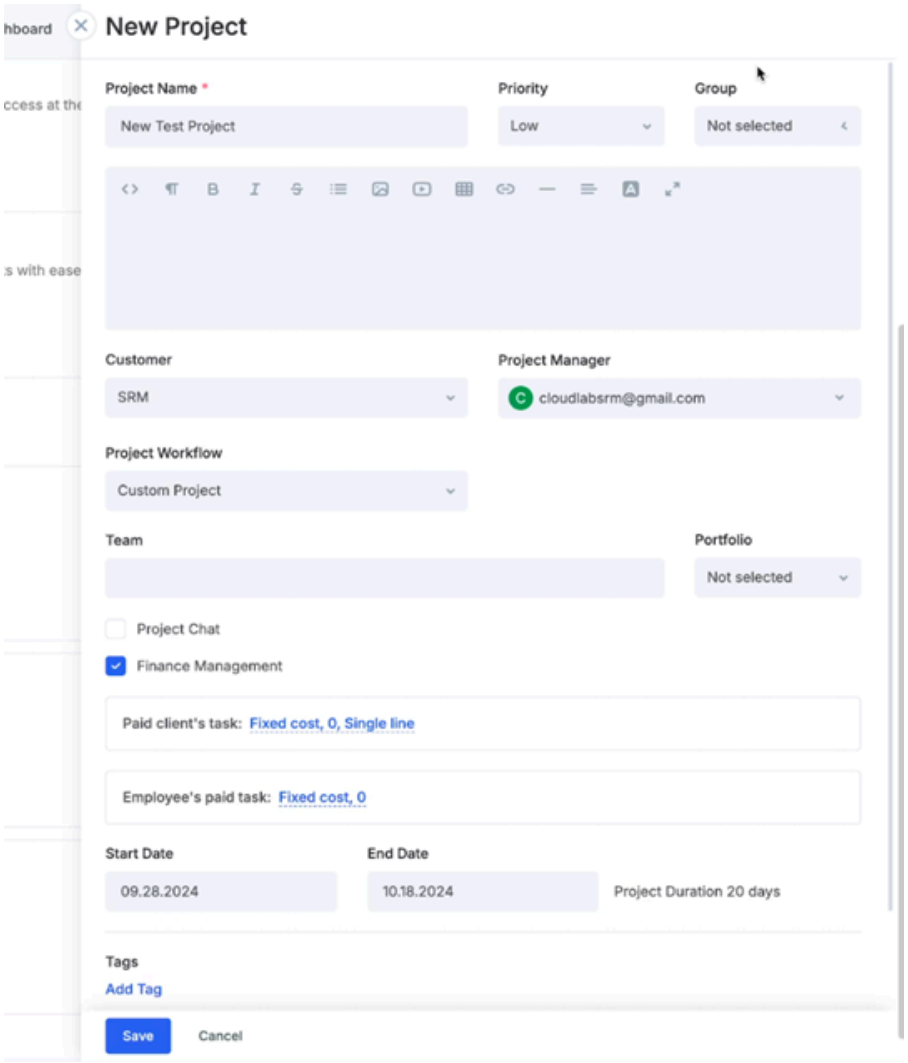
Theory:
The idea of a completely free, all-inclusive SaaS CRM, like Flowlu, challenged the traditional concept of CRM software pricing. Flowlu offered nine different sets of tools that covered every step of the sales and project management process. It promised significant cost savings and scalability benefits for small businesses. However, the integration of these tools with other business systems could present challenges, requiring additional technical expertise. This evaluation aimed to determine whether Flowlu could provide an effective, free CRM solution for small businesses by examining its capabilities, user experience, and potential challenges.

Procedure:

Sign up on the Flowlu Website:
The first step was visiting the Flowlu website and signing up for a free account. This was done through the following URL:
ruby
Copy code
`https://www.flowlu.com/uses/best-free-saas-crm/`

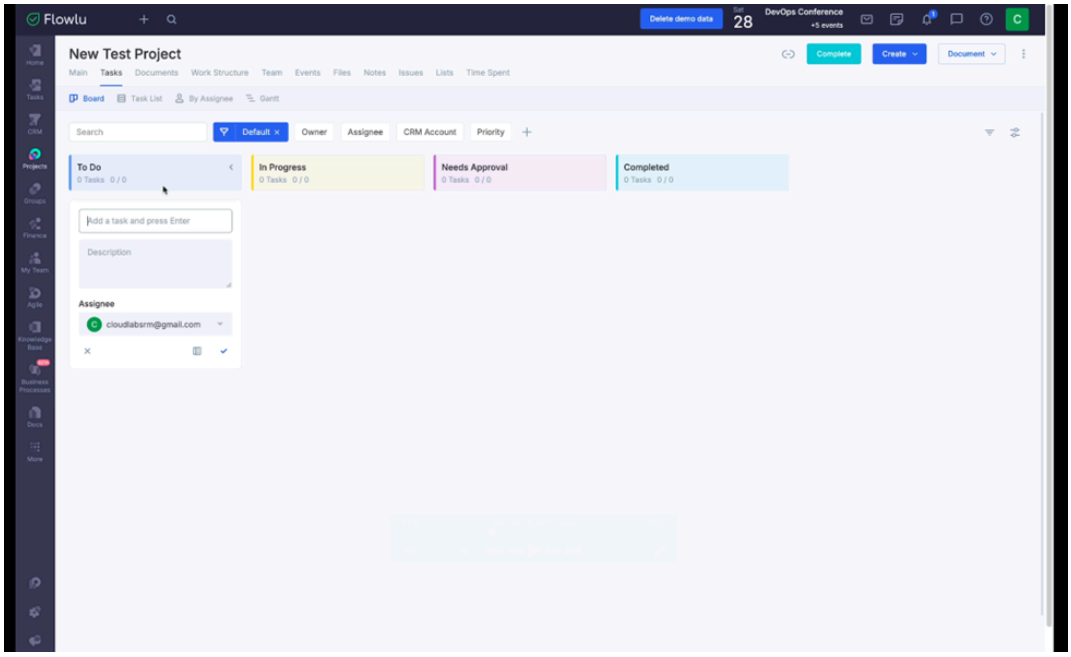
Create an Account and Test Project:
After signing up, a test project was created as per the instructions provided in the onboarding video. This allowed for a hands-on exploration of Flowlu's project management capabilities, bringing the user to Flowlu's home page.

Create a Sample Project:
A sample project was created, incorporating additional information to replicate real-world usage. This step involved defining project details such as deadlines, goals, and tasks to simulate how Flowlu could be used for managing business projects.

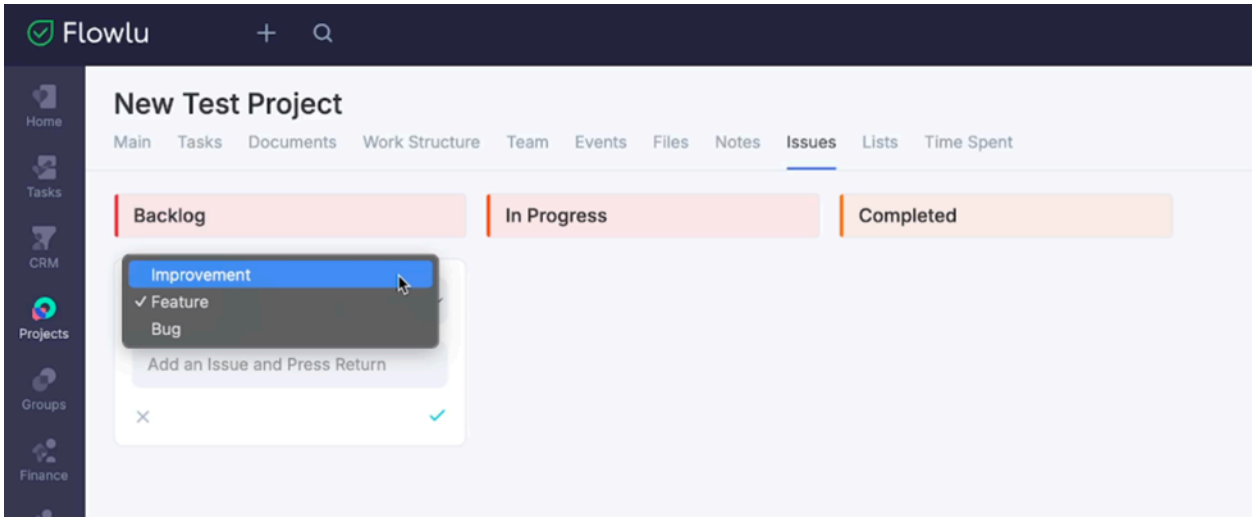


Create Sample Tasks:

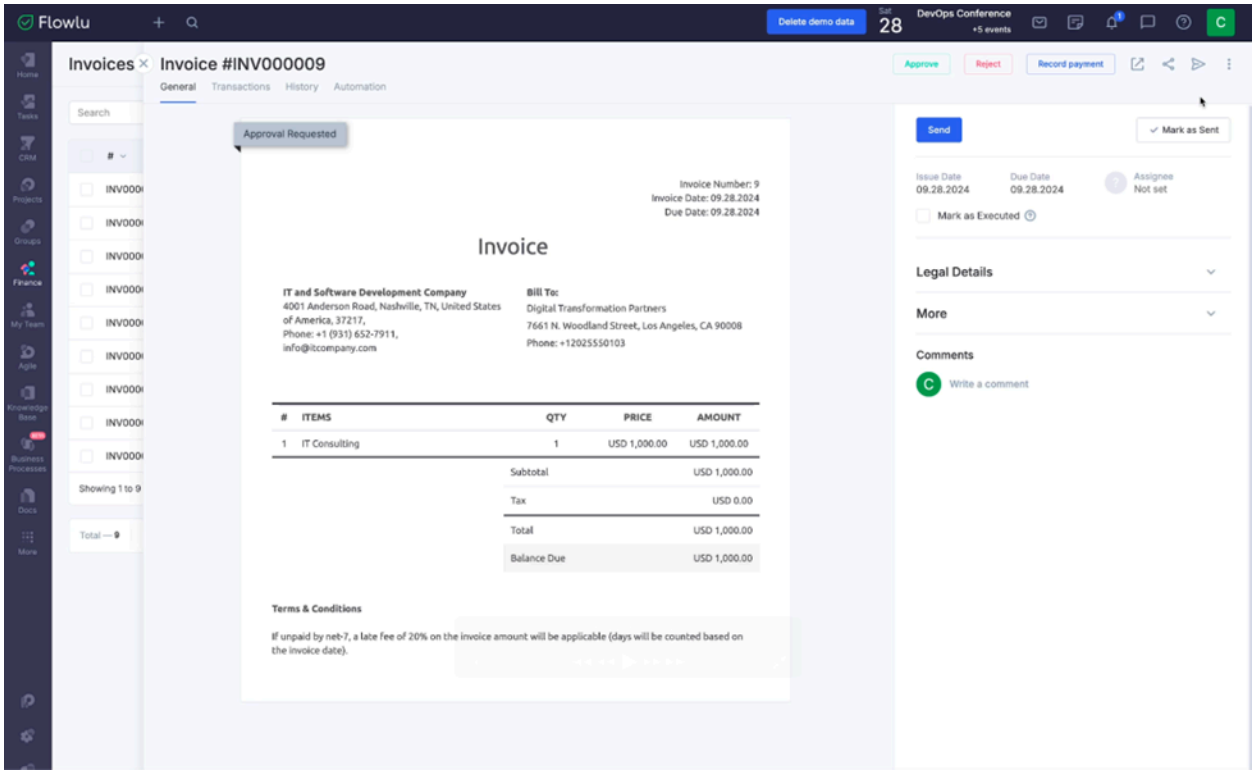
The “Tasks” tab was explored next, where several sample tasks were created. This process demonstrated how to assign tasks to team members, set priorities, and manage deadlines within a project.

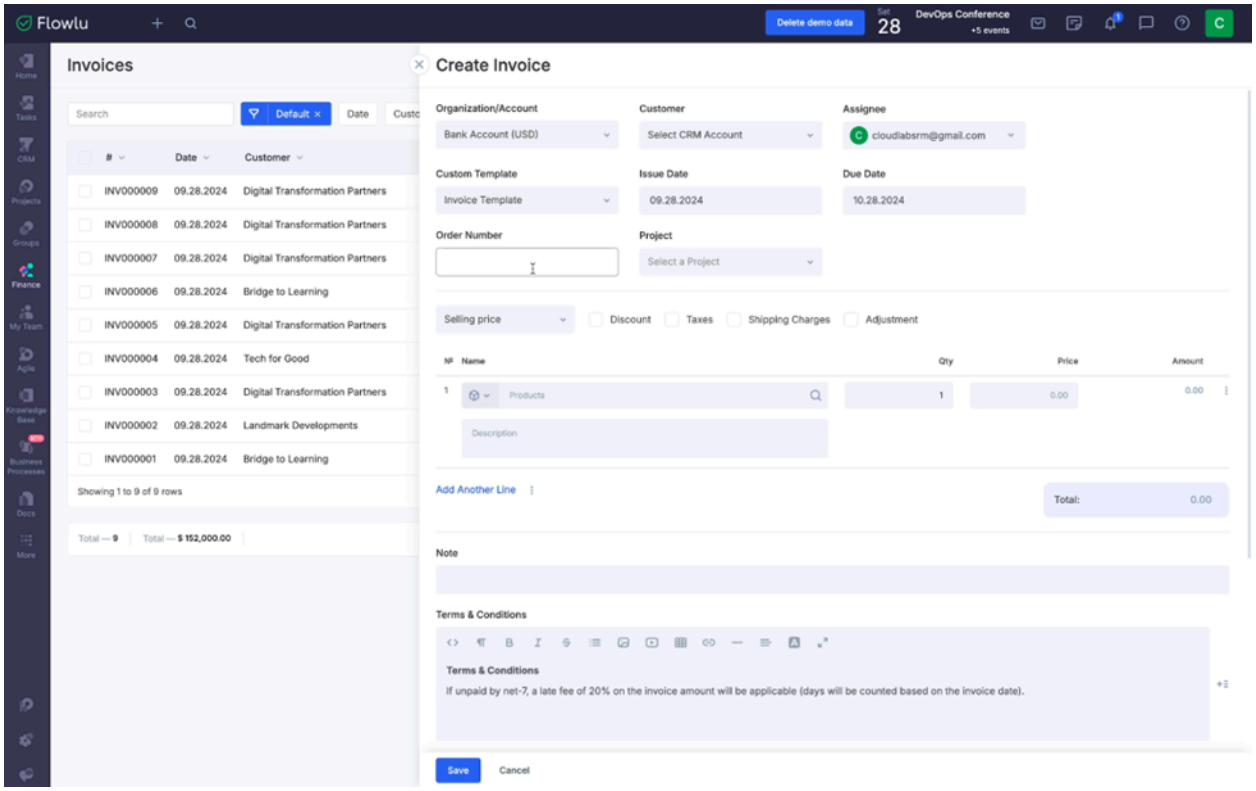


Explore Other Tabs:
Various other tabs within Flowlu were explored, including Documents, Teams, Events, Roles, Files, Notes, Issues, and Lists. Each section was examined to understand how Flowlu handled different aspects of business management, from document storage to team collaboration.



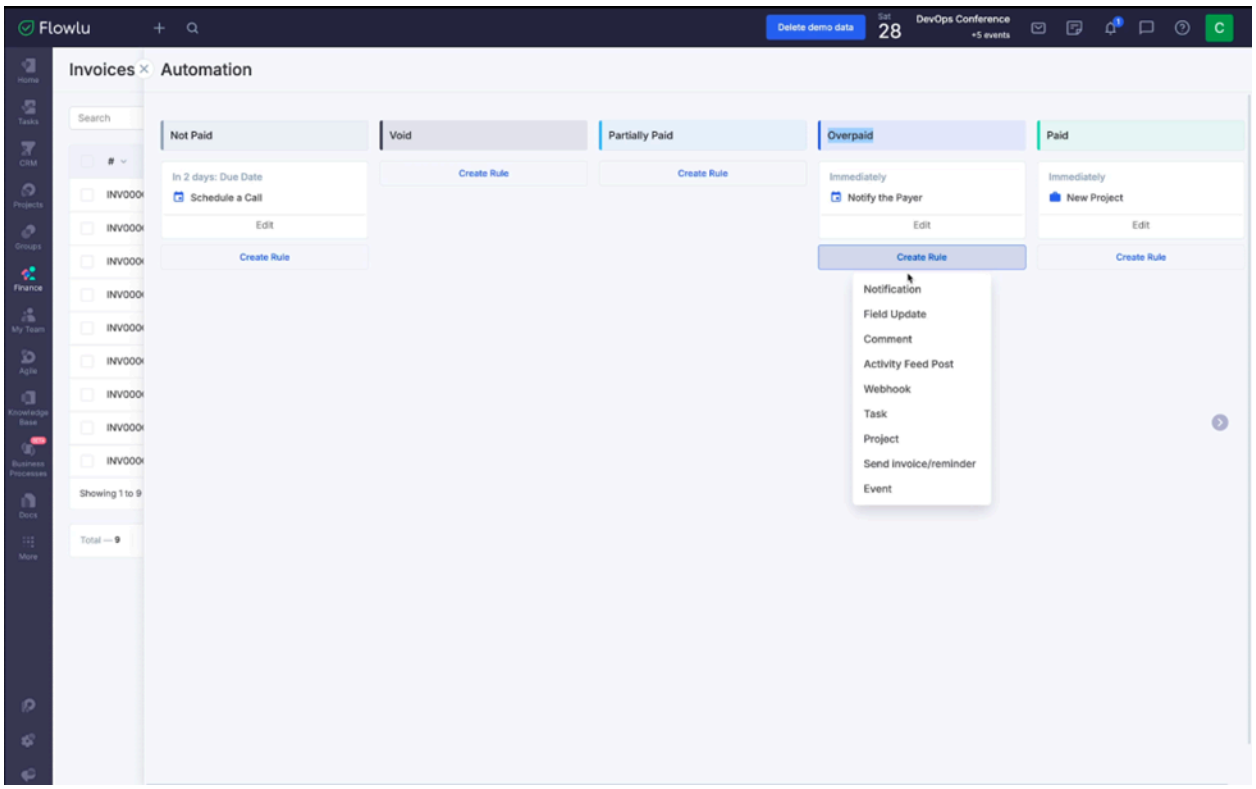
Managing Invoices:
The “Finances” tab was opened to explore Flowlu’s invoice management features. The process of creating and managing invoices was tested, allowing a closer look at Flowlu's ability to handle financial tasks such as tracking expenses and generating invoices for clients.



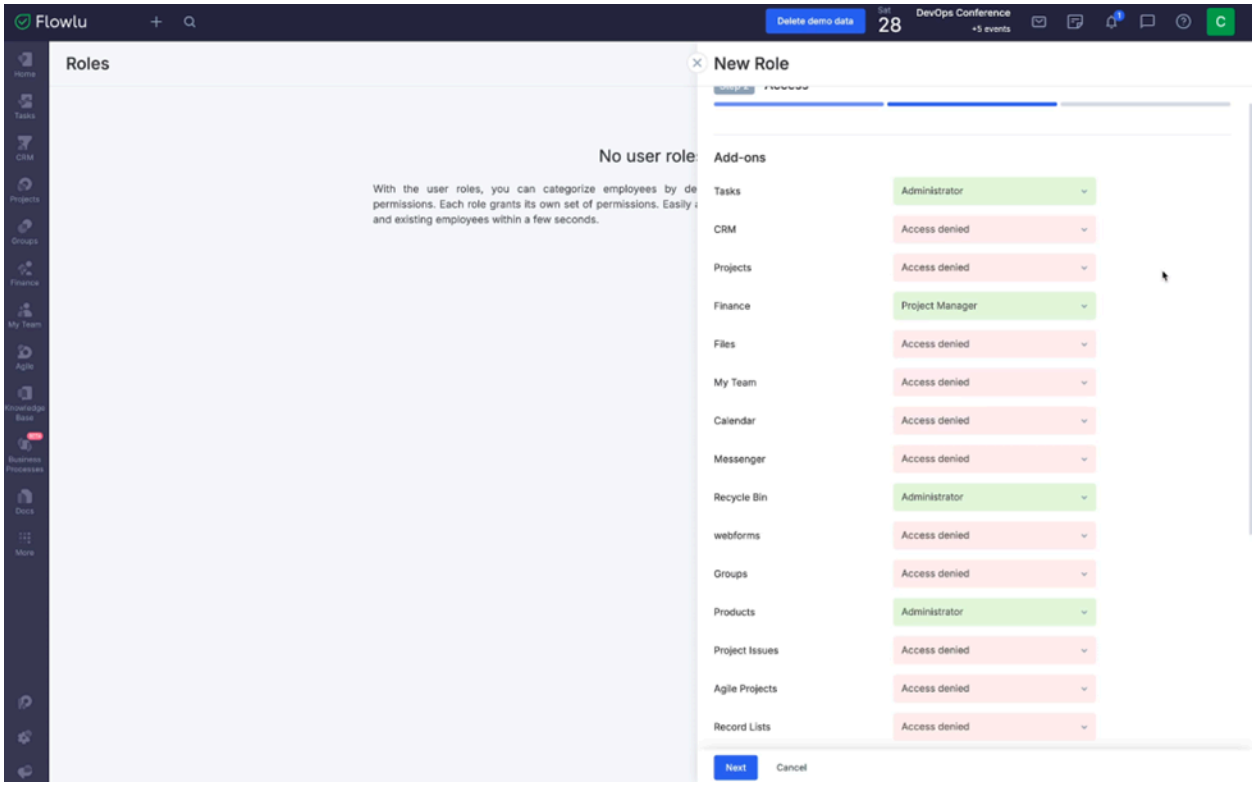


Explore Automations:
Flowlu's automation features were explored, specifically focusing on webhooks and other events

that could trigger automated processes. This demonstrated the potential for automating repetitive tasks and improving efficiency in business operations.



Create and Manage a Role:
A new role was created and managed within the CRM system, showing how Flowlu allowed businesses to assign specific permissions and responsibilities to different team members. This feature provided insight into user management within Flowlu.



Explore Other Features like Groups:
Lastly, additional tabs such as Groups were explored as demonstrated in the onboarding video. This section helped in understanding how team collaboration and group management functioned within Flowlu, providing more insights into the platform's overall capabilities.

Expected Output:
After completing the task, the expected outcome was to gain a clear understanding of how a SaaS CRM solution like Flowlu worked and how it could be applied to manage various business tasks. The exploration provided a hands-on learning experience, demonstrating how Flowlu could help businesses manage projects, tasks, finances, and team collaboration effectively. It offered a glimpse into the powerful features of SaaS CRMs and their potential for improving efficiency in business operations.

Title:
Exploring Word Count Using MapReduce - Market Rating Example

Objective:
The objective of this assignment was to understand the functionality of Apache Hadoop's MapReduce framework by calculating the average rating per market in a given dataset using a custom Java program. The program demonstrated how to split data, map it, and reduce it using MapReduce to perform a basic aggregation task, while illustrating the broader concepts of distributed computing and data processing.

Prerequisites:

- A basic understanding of Hadoop, HDFS (Hadoop Distributed File System), and MapReduce concepts was required.
- A running Hadoop cluster with the ability to compile and run Java programs was necessary. Familiarity with Java programming and command-line operations was also beneficial.

Theory:
MapReduce is a core component of Hadoop's distributed computing model. It provided a powerful framework for processing large datasets in parallel across a distributed cluster of computers. The MapReduce model worked in two primary phases: the Map phase and the Reduce phase.

1. **Map Phase:**
    - During this phase, input data was divided into smaller, manageable chunks. Each chunk was processed by a mapper, which executed the map function to perform data transformation and extraction.
    - In this example, each line in the dataset contained a market name and its associated rating. The `TokenizerMapper` class was responsible for tokenizing each line by splitting it on commas. The first token was the market name, and the second token represented the rating.
    - The mapper emitted key-value pairs, where the market name served as the key, and the rating was the value. This was accomplished using the `context.write()` method, which passed the key-value pairs to the framework for further processing. The `TokenizerMapper` effectively transformed the input lines into a format suitable for aggregation by identifying the relevant fields and preparing them for the next stage.
2. **Reduce Phase:**
    - After the map phase was completed, the intermediate key-value pairs were shuffled and sorted by key. The reducer then processed these pairs to aggregate results.

- The `IntSumReducer` class was designed to handle this aggregation. For each unique market name (key), it received a list of ratings (values) and calculated the sum and count of these ratings.
- The final output from the reducer was the average rating for each market, computed by dividing the total sum of ratings by the count of ratings. The average rating was then written to the output context, making it available for further analysis or reporting. This phase exemplified how MapReduce was used to perform complex calculations and derive insights from large datasets in a highly efficient manner.

3. **Execution Flow:**
   - The main method orchestrated the execution of the MapReduce job. It set up the job configuration, specifying parameters such as the job name, input and output paths, mapper and reducer classes, and the data types for output keys and values.
   - The job was submitted to the Hadoop framework for execution. Upon completion, the results were stored in the specified output path in HDFS, allowing for further analysis or reporting. The process highlighted the simplicity and power of the MapReduce paradigm, enabling developers to focus on the logic of their application while leveraging the underlying framework to handle the complexities of distributed processing.

Procedure:

1. **Prepare the Environment:**
   A running Hadoop cluster was prepared, ensuring all necessary configurations were in place. Libraries and dependencies required to compile and run Java programs were installed.
2. **Create the `MarketRating.java` File:**
   A Java program was written to implement the MapReduce logic. The `TokenizerMapper` class and the `IntSumReducer` class were defined, along with the main method for execution flow.

Copy the Java File to the Namenode:
The Java file was copied to the namenode in the Hadoop cluster using the following command:

```
docker cp MarketRating.java namenode:/path/to/destination
```

Compile the Java Program:
The Java program was compiled on the namenode by executing the following command from the namenode shell:

```
docker exec -it namenode /bin/bash
```

```
cd /path/to/destination
javac -classpath $(hadoop classpath) -d . MarketRating.java
```

Package the Compiled Classes into a JAR File:
The compiled Java classes were packaged into a JAR file using the `jar` command:
bash
Copy code
```
jar -cvf MarketRating.jar -C . .
```

Prepare HDFS for Input:
An input directory was created in HDFS using the following command:
bash
Copy code
```
hdfs dfs -mkdir /input
```

Put the Input Data into HDFS:
The input data file, `MarketRating.txt`, was transferred into the input directory in HDFS using the following command:
bash
Copy code
```
hdfs dfs -put /path/to/MarketRating.txt /input
```

Run the MapReduce Job:
The MapReduce job was executed using the `hadoop jar` command, specifying the input and output paths:
bash
Copy code
```
hadoop jar MarketRating.jar MarketRating /input /output
```

Check the Output:
After the job was completed, the output was verified by listing the contents of the output directory and displaying the results:

```
hdfs dfs -ls /output
hdfs dfs -cat /output/part-r-00000
```

Analyze the Results:
The output data, containing the market names along with their corresponding average ratings, was analyzed. This data could be used for decision-making, identifying trends, or further exploration of the dataset.

Expected Output:

The expected output of the MapReduce job consisted of market names along with their corresponding average ratings, formatted as follows:

```
Market1   4.5
Market2   3.2
Market3   5.0
```

Learning:

By implementing this MapReduce program, several key concepts were learned:

- **Mappers and Reducers:** A deeper understanding of how mappers and reducers played distinct roles in the data processing pipeline was gained, and how they interacted within a distributed computing environment.
- **Data Structuring:** The importance of structuring input data for efficient processing was emphasized, and the role of key-value pairs in performing aggregation tasks was understood.
- **Hadoop Ecosystem:** Familiarity with the Hadoop ecosystem, including HDFS, which facilitated the storage and retrieval of large datasets across a distributed environment, was achieved.
- **Distributed Computing:** Insights into the principles of distributed computing, such as parallel processing, fault tolerance, and scalability, were gained. These principles are essential for building robust big data applications.
- **Java Programming for Big Data:** Practical skills in using Java to write MapReduce programs were developed, further enhancing proficiency in big data programming.

Title:
Word Count Using Apache Pig on HDFS

Objective:
The objective of this assignment was to implement a word count program using Apache Pig that processed a text file stored in HDFS. The program counted the frequency of each word and stored the results in an output file, demonstrating Pig's data processing capabilities on large datasets.

Prerequisites:

- A basic understanding of Apache Pig and HDFS was required.
- A running Hadoop and Pig setup was ensured.
- The input text file was stored in HDFS before beginning the assignment.

Theory:
Apache Pig provided a high-level platform for creating MapReduce programs to be used with Hadoop. Pig used a scripting language called Pig Latin, which simplified the development of complex data processing tasks. One common use case was counting word occurrences in a text file—a problem traditionally solved using MapReduce.

In this assignment, Apache Pig was utilized to:

1. Load the data from HDFS.
2. Tokenize the text into individual words.
3. Group the words together to count their occurrences.
4. Store the result back in HDFS.

This approach allowed for a more abstract and straightforward method of processing data compared to writing a complex MapReduce program in Java.

Procedure:

Store the Input File in HDFS:
The input file, named `input.txt`, was placed into HDFS at the following path:

```
hdfs://namenode:9000/pig-scripts/input.txt
```

Run the Pig Script:
A Pig script was executed, which performed the following steps:

Load the Text Data:
The text file was loaded into Pig from HDFS, with each line of the file treated as a string. This

step was performed using the following command:

```
data = LOAD 'hdfs://namenode:9000/pig-scripts/input.txt' USING
PigStorage(' ') AS (line:chararray);
```

Tokenize the Lines into Words:
Each line of text was tokenized into individual words using the `TOKENIZE` function, and `FLATTEN` was used to convert the nested structure into a flat list of words. The command was:

```
words = FOREACH data GENERATE FLATTEN(TOKENIZE(line)) AS word;
```

Group the Words:
The words were grouped together by their actual values, allowing for counting each word's occurrences. This step was executed with:

```
grouped_words = GROUP words BY word;
```

Count Occurrences of Each Word:
For each group of words, the `COUNT` function was used to determine the number of occurrences. The results were stored as a tuple containing the word and its count. The command for this was:

```
word_count = FOREACH grouped_words GENERATE group AS word,
COUNT(words) AS count;
```

Store the Result in HDFS:
The final output, which contained the word counts, was stored back in HDFS in CSV format. This was accomplished using:

```
STORE word_count INTO 'hdfs://namenode:9000/pig-scripts/output' USING
PigStorage(',');
```

Review the Output:
After the Pig script completed its execution, the output was reviewed by navigating to the specified HDFS output path:

```
hdfs://namenode:9000/pig-scripts/output
```

The output files were inspected to confirm the word counts were generated correctly.

Expected Output:
The expected output of the Pig script consisted of a list of words from the input text file, along

with the corresponding occurrence counts. For example, if the input file `input.txt` contained the following lines:

```
hello world
hello hadoop
hello pig
```

The output generated in HDFS was expected to look like this:

```
hello, 3
world, 1
hadoop, 1
pig, 1
```

Learnings:
By completing this assignment, several key learnings were achieved:

- **Apache Pig for Simplified Data Processing:**
  Apache Pig was shown to abstract many of the complexities associated with MapReduce. The word count program demonstrated how large-scale text processing tasks could be implemented using simple Pig Latin commands rather than complex Java-based MapReduce programs.
- **Integration with HDFS:**
  The integration of Pig with HDFS made it possible to efficiently process large datasets stored across a distributed Hadoop cluster. Pig's ability to read from and write to HDFS provided seamless data flow.
- **Word Count Use Case:**
  The classic word count problem was solved using Pig, reinforcing the concepts of tokenization, grouping, and aggregation within the Hadoop ecosystem.
- **Distributed Processing:**
  Pig's ability to parallelize data processing across the Hadoop cluster was highlighted. This showed the power of distributed computing for large-scale data analysis.

Overall, the assignment demonstrated the use of Apache Pig in simplifying data processing tasks and provided practical experience in working with large datasets stored in HDFS.