EX NO : 1 SETTING UP AN AWS ACCOUNT AND IMPLEMENTING MULTI – FACTOR AUTHENTICATION

AIM:

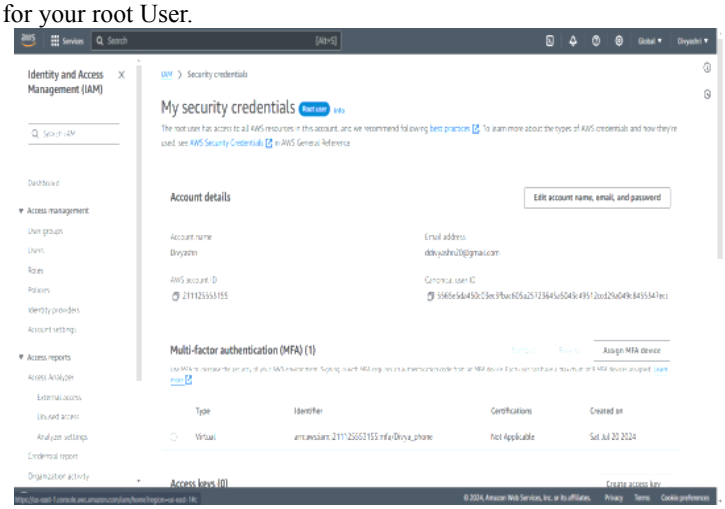
Setting up an AWS account and implementing Multi – Factor Authentication (MFA)

Setting up an AWS Account

1. Visit AWS, click “Create an AWS account”, enter e-mail, password, account name, and click “Continue”.
2. Fill in contact details, choose account type, agree to AWS customer agreement and click “Continue”.
3. Enter your Credit/Debit card Information and click “Verify and Add”.
4. Verify your phone number via SMS or voice call, and enter the received code.
5. Choose a support plan and click “Complete sign up”.
6. Go to the AWS management Console and sign in with your email and password.

Implementing Multi Factor Authentication (MFA)

1. Sign in to the AWS Management Console, click your account name at the top right and select “ My Security credentials”.
2. In Security credentials scroll to Multi Factor Authentication (MFA) and click “Assign MFA”.
3. Specify the MFA device name and choose the type of MFA device and click Next.
4. Download an MFA app like Google Authenticator scan the QR code, enter two consecutive OTPs from the app into AWS, and click Assign MFA.
5. Verify Successful configuration with a confirmation message indicating MFA is enabled for your root User.



RESULT:

EX NO: 2 LAUNCHING EC2 INSTANCE WITH CUSTOM AMIs AND USER DATA SCRIPTS

AIM:

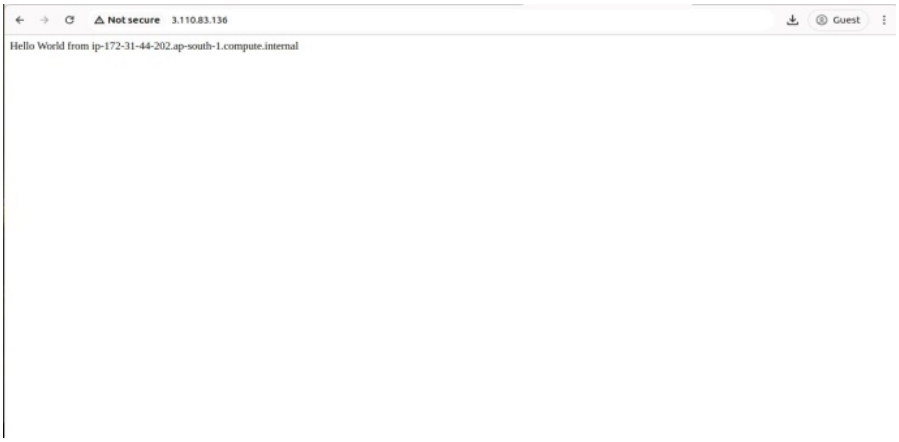
Launching EC2 instances with custom AMIs and user data scripts

PROCEDURE:

1. Open AWS Management Console and Login in with your AWS credentials.
2. Click services on top left and then click “Compute”
3. In Compute click on “EC2” to go the EC2 dashboard.
4. Click the “Launch Instance” button on the EC2 dashboard.
5. Specify an instance name and choose an Amazon Machine Image (AMIs).

6. Under the “Advance Details” section enter your user data script in the “User data” field.
7. Select an existing security group or create a new one , ensuring necessary port are open (eg. SHH,HTTP,HTTPS).
8. Review all instance settings and click “Launch”.
9. Select a key pair or without key pair and click “ Launch Instance”.
10. Click “View Instance” to go to the EC2 dashboard and see your newly launched instance.

OUTPUT:



RESULT:

EX NO: 3 IMPLEMENTING S3 WITH VERSIONING, LIFECYCLE POLICIES, AND CROSS REGION REPLICATION

AIM:

Implementing S3 with versioning , lifecycle policies and cross region replication.

PROCEDURE: Setting up S3 bucket with versioning.

1. Open the AWS Management Console and login with your credentials.
2. Navigate to S3 dashboard, Click “Create bucket “, enter a unique bucket name.
3. Scroll down to “Bucket Versioning” and Enable versioning and then click “Create bucket”.
4. Similarly follow the above steps to create another bucket for replication process.

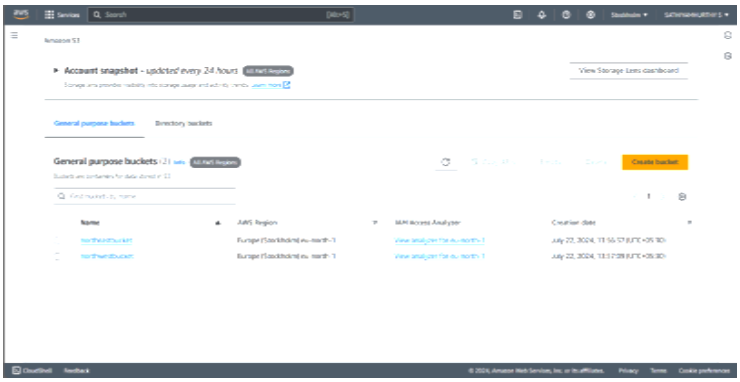
Set up Lifecycle Policies

1. In S3 dashboard, select the bucket for which you want to configure lifecycle policies.
2. Click the “Management” tab, then under “Lifecycle rules“, click “Create Lifecycle rule”.
3. Enter a rule name, specify the rule scope and click “Create rule”.

Setting up Cross Region Replication

1. In the s3 dashboard, select the bucket you want to replicate from ie . Source bucket.
2. Click the “Management” tab, then under “Replication rules”, click “Create replication rule”.
3. Enter a rule name, Select the destination bucket name, specify the rule scope as “apply to all objects in the bucket”
4. Select the IAM role as “Create new role” and then click “Save”.
5. Finally select the source bucket and upload any file for replication and then check the destination bucket for the completion of replicated file.

OUTPUT:



RESULT :

EX NO: 4 CONFIGURING A HIGHLY AVAILABLE WEB APPLICATION USING EC2 ELB AND AUTO SCALING

AIM:

To Configure a Highly available Web application using EC2 ELB and Auto Scaling.

PROCEDURE:

Creating Security Groups.

5. Open the AWS EC2 console, choose groups in the navigation pane.
6. Choose “Create Security Group”,
   - Group name : SGForALB
   - Inbound rule :
     - ➔ Type : HTTP,
     - ➔ Source : Anywhere IPv4,

   then click “Choose Security Group”.
7. Create a security group for Auto Scaling Instance
   - Group name : SGForAutoScaling
   - Inbound rule :
     - ➔ Type : All TCP,
     - ➔ Source : Custom (Reference – SGForALB),
     - ➔ Type : SSH,
     - ➔ Source : Anywhere IPv4,

   then click “Choose Security Group”.

Creating Auto Scaling Group

4. Open the Amazon EC2 console , choose “Auto Scaling Groups” in navigation pane.
5. Choose “ Create Auto Scaling Group” and enter name as “ Demo Auto Scaling “.
6. Click on “Create Launch Template”,
   - Template name : Test Template
   - Under Amazon Machine Image (AMI) ,click Amazon Linux
   - Instance Type : t3.micro
   - Network Setting : SGForAutoScaling in security groups
   - Add the User data script in Advance Details section.

```
#!/bin/bash
# install httpd (Linux 2 version)
yum update -y
yum install -y httpd.x86_64
systemctl start httpd.service
systemctl enable httpd.service
echo "Hello World from $(hostname -f)" > /var/www/html/index.html
```

   then click on “ Create Launch Template”.
7. Select the created template “ TestTemplate” and then select VPC and all the available zones then click “ Next “.

8. Choose Attach to a New Load Balancer and select internet facing.
9. Under Listeners and routing , select the new target group as “demoautoscaling” and then enable “ Elastic load Balancing health checks”.
10. Set the scaling policy with min derived capacity as 1 and max derived capacity 2.
11. Follow the prompts to Next through the remaining steps and create the Auto Scaling Group.

Configure Load Balancer

6. Open the Amazon EC2 console , choose Load Balancer in navigation pane.
7. Select the load balancer “demoautoscaling”, go to security and choose “Edit” and select the security group “SGForALB” and save the configuration.

Verify Auto Scaling Group

1. Go to Auto Scaling Group and select “demoautoscaling”.
2. Go to Activity and review the Activity History to ensure there are 2 successful message indicating the instances are launched and in service.

Check Load Balancer DNS

1. Go to Load Balancer and select “demoautoscaling”.
2. Copy the DNS name and open it in a new browser tab to check the output.

Verify Dynamic Scaling Group

1. Go to Auto Scaling Group and select “demoautoscaling”.
2. Choose Automatic scaling tab and select “Create Dynamic Scaling Policy” and then set the target value to 30 and create the policy.

Verify Cloud Watch Alarms

1. Open the Amazon Cloud Watch Console and ensure there are two alarms created by choosing “All Alarms” in navigation pane

Connect to EC2 Instance

1. Open Amazon EC2 console , choose the instance from the Auto Scaling Group and click on “Connect”.
2. In the terminal give the following command and run :
    - ➢ sudo yum install stress -y
    - ➢ sudo stress --cpu 12 --timeout 240s

Monitor Auto Scaling

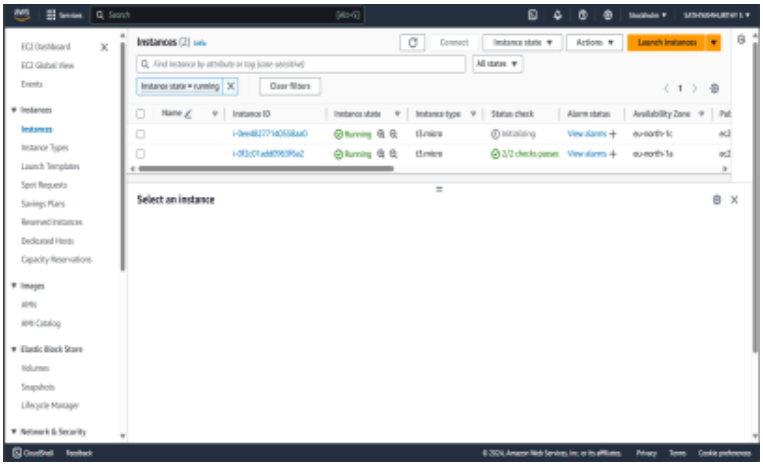
1. Open Amazon EC2 console and choose Auto Scaling Group and select “demoautoscaling”.
2. Go to the Monitoring tab to view the graphs.
3. Go to the Activity tab and check the Activity history for a waiting instance.

Verify Running Instance

1. Open the Amazon EC2 console and choose “Instances” in navigation pane.
2. Ensure there are two instances running.

Load Balancer DNS

1. In the already opened DNS name tab ,refresh the page and check if the IP address get changed.



RESULT:

Ex : 5 DESIGNING A MULTI-TIER VPC ARCHITECTURE WITH NAT GATEWAYS AND VPN CONNECTION ON AWS

AIM :

The aim is to design and implement a robust and secure **Multi-Tier Virtual Private Cloud (VPC) Architecture** on AWS that includes **NAT Gateways** and a **VPN Connection**. The goal is to create a scalable and highly available network environment that separates application components into distinct tiers while providing secure access to on-premises resources.

PROCEDURE : Step 1: Create a Virtual Private Cloud (VPC)

1. Log in to the **AWS Management Console**.
2. Navigate to **VPC Dashboard**.
3. Click on **Create VPC**.
4. Configure the VPC:
    - **Name tag**: MyVPC
    - **IPv4 CIDR block**: 10.0.0.0/16 or 192.168.0.0/16
    - Leave the default settings for IPv6, Tenancy, and other options.
5. Click **Create VPC**.

Step 2: Create Subnets

2.1: Create Public Subnet

1. In the VPC Dashboard, click **Subnets** on the left panel.
2. Click **Create Subnet**.
3. Configure the public subnet:
    - **Name tag**: PublicSubnet
    - **VPC**: Select MyVPC.
    - **Availability Zone**: Select a preferred zone (e.g., us-east-1a).
    - **IPv4 CIDR block**: 10.0.1.0/24.
4. Click **Create Subnet**.

2.2: Create Private Subnet

1. Click **Create Subnet** again to create the private subnet.
2. Configure the private subnet:
    - **Name tag**: PrivateSubnet
    - **VPC**: Select MyVPC.
    - **Availability Zone**: Select the same zone (e.g., us-east-1a).
    - **IPv4 CIDR block**: 10.0.2.0/24.
3. Click **Create Subnet**.

Step 3: Create and Attach an Internet Gateway (IGW)

1. In the VPC Dashboard, click **Internet Gateways**.
2. Click **Create Internet Gateway**.
3. Name it, e.g., MyIGW.
4. Click **Create Internet Gateway**.
5. Once created, click **Actions** > **Attach to VPC**.
6. Select MyVPC and click **Attach Internet Gateway**.

Step 4: Create Route Tables - 4.1: Create Public Route Table

1. In the VPC Dashboard, click **Route Tables**.
2. Click **Create Route Table**.
3. Configure:
    - **Name tag**: PublicRouteTable.
    - **VPC**: Select MyVPC.
4. Click **Create Route Table**.
5. Select the newly created route table, click **Edit Routes**:
    - Click **Add Route**:
        - **Destination**: 0.0.0.0/0 (default route for all internet traffic).
        - **Target**: Select **Internet Gateway (IGW)**.
6. Click **Save routes**.
7. Associate the route table with the **PublicSubnet**:
    - Click **Subnet Associations** > **Edit subnet associations**.
    - Select PublicSubnet and click **Save**.

*4.2: **Create Private Route Table***

1. Click **Create Route Table** again for the private subnet.

2. Configure:
    - **Name tag**: PrivateRouteTable.
    - **VPC**: Select MyVPC.
3. Click **Create Route Table**.

Step 5: Create a NAT Gateway

1. In the VPC Dashboard, click **NAT Gateways**.
2. Click **Create NAT Gateway**.
3. Configure:
    - **Subnet**: Select the PublicSubnet.
    - **Elastic IP Allocation**: Click **Allocate Elastic IP** and then **Allocate**.
4. Click **Create NAT Gateway**.
5. Once created, edit the **PrivateRouteTable** to route outbound traffic via the NAT Gateway:
    - Select PrivateRouteTable and click **Edit Routes**.
    - Add a route:
        - **Destination**: 0.0.0.0/0.
        - **Target**: Select the NAT Gateway.
    - Click **Save routes**.

Step 6: Create a VPN Gateway (VGW)

1. In the VPC Dashboard, click **VPN Gateways**.
2. Click **Create VPN Gateway**.
3. Configure:
    - **Name tag**: MyVGW.
    - **Autonomous System Number (ASN)**: Leave default or enter custom ASN.
4. Click **Create VPN Gateway**.
5. After creation, click **Actions** > **Attach to VPC**.
6. Select MyVPC and click **Attach**.

Step 7: Create Customer Gateway (CGW)

1. In the VPC Dashboard, click **Customer Gateways**.
2. Click **Create Customer Gateway**.
3. Configure:
    - **Name tag**: MyCGW.
    - **IP Address**: Enter the public IP address of your on-premises router or firewall.
    - **Routing**: Static or Dynamic (depending on your routing protocol).
4. Click **Create Customer Gateway**.

Step 8: Create a VPN Connection

1. In the VPC Dashboard, click **VPN Connections**.
2. Click **Create VPN Connection**.
3. Configure:
    - **Name tag**: MyVPNConnection.
    - **Target Gateway**: Select **Virtual Private Gateway** and choose MyVGW.
    - **Customer Gateway**: Select the previously created MyCGW.

- **Routing Options**: Choose static or dynamic.

4. For **Static Routing**:
   - Add your on-premises CIDR block(s) (e.g., 192.168.1.0/24).
5. Click **Create VPN Connection**.

Step 9: Download VPN Configuration

1. After creating the VPN connection, click on your VPN connection and download the configuration file.
2. Use the configuration to set up the VPN on your on-premises router/firewall.

Step 10: Update Route Tables for VPN Traffic

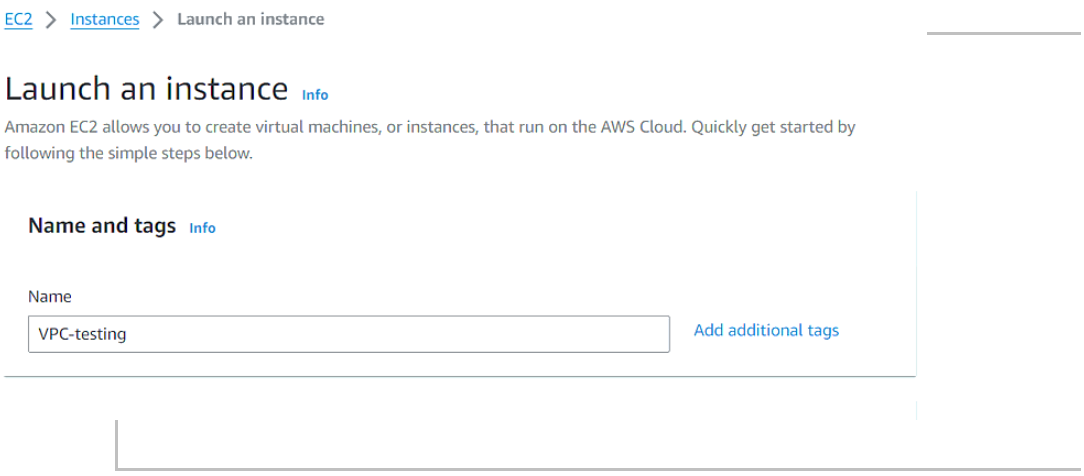
1. Edit the route tables in the **VPC Dashboard**.
2. For **PrivateRouteTable**, add a route for the on-premises network (e.g., 192.168.1.0/24):
   - **Destination**: 192.168.1.0/24.
   - **Target**: Select the **Virtual Private Gateway (VGW)**.
3. Click **Save routes**.

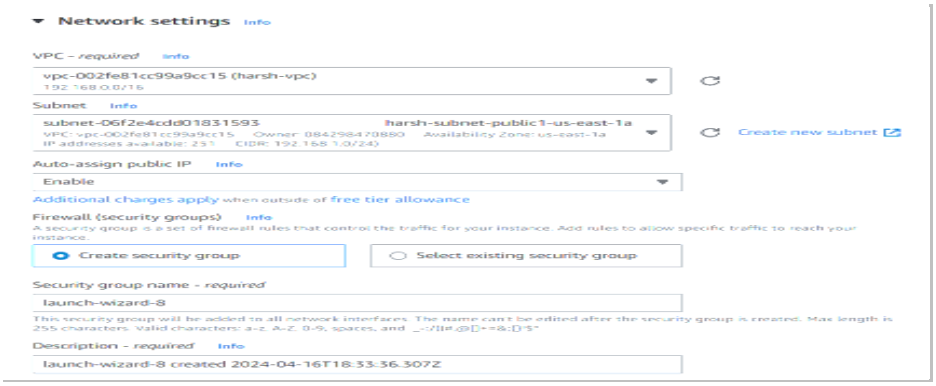
Step 11: Configure Security Groups

1. Navigate to **Security Groups**.
2. Set up security group rules for each tier.
   - **PublicSubnet Security Group**:
     - Allow inbound HTTP (80), HTTPS (443), SSH (22), as needed.
   - **PrivateSubnet Security Group**:
     - Allow inbound traffic only from the public subnet.
     - Allow outbound traffic to the NAT Gateway for internet access.
   - **VPN Security Group**:
     - Allow inbound and outbound traffic from/to the on-premises CIDR block.

Step 12: Test Connectivity

1. Launch instances in the **Public** and **Private** subnets to verify:
   - Instances in the **Public Subnet** can access the internet directly.
   - Instances in the **Private Subnet** can access the internet via the **NAT Gateway**.
   - Instances in the **Private Subnet** can communicate with on-premises infrastructure via the **VPN Gateway**.





RESULT :

EX : 6 IMPLEMENTING AUTO SCALING WITH CUSTOM METRICS AND SCHEDULED ACTIONS

AIM :

The aim is to design and implement an **Auto Scaling** solution in AWS that utilizes **custom metrics** and **scheduled actions** to automatically adjust the capacity of resources based on demand. The goal is to ensure optimal performance, maintain cost efficiency, and enhance the availability of applications by dynamically scaling resources based on real-time usage patterns and predefined schedules.

PROCEDURE : 1. Initial Setup

a. Launch EC2 Instances or Desired Resource

- Launch your EC2 instances or any other resource (e.g., ECS, Lambda) where auto scaling needs to be applied.
- Ensure your instances have proper IAM roles with permissions for CloudWatch, Auto Scaling, and other relevant services.

b. Install CloudWatch Agent (for EC2 Instances)

- If you're using EC2 and want to track custom metrics, install the CloudWatch agent on each instance to send custom metrics to Amazon CloudWatch.

Steps:

1.SSH into your EC2 instance and install the CloudWatch agent:

sudo yum install amazon-cloudwatch-agent

2 . Configure the CloudWatch agent:

sudo vi /opt/aws/amazon-cloudwatch-agent/bin/config.json

3.Start the CloudWatch agent:

sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl \

-a start

2. Set Up Custom Metrics

a. Create and Push Custom Metrics to CloudWatch

- Identify the custom metric you want to scale on (e.g., memory usage, application performance).

- Use the AWS SDK (e.g., AWS CLI, Python Boto3) to publish custom metrics to CloudWatch.

```
aws cloudwatch put-metric-data \
 --metric-name MemoryUtilization \
 --namespace CustomMetrics \
 --unit Percent \
 --value 78.2
```

Ensure this metric is regularly updated from your instances using CloudWatch Agent or your own scripts.

3. Create an Auto Scaling Group

a. Define Launch Configuration or Launch Template

- Create a launch configuration or launch template for your EC2 instances with the necessary AMI, instance type, security groups, etc.

Steps:

- Navigate to **EC2 Dashboard** > **Auto Scaling** > **Launch Configurations** or **Launch Templates**.
- Fill in the required parameters such as AMI ID, instance type, key pair, and security group.

b. Create an Auto Scaling Group

- Create an Auto Scaling Group using the launch configuration/launch template created above.
- Set the minimum, maximum, and desired instance count for your Auto Scaling Group.

Steps:

- Go to **Auto Scaling Groups** and select **Create Auto Scaling Group**.
- Associate it with the launch configuration/launch template.
- Set up scaling policies and health checks as needed.

4. Configure Auto Scaling Based on Custom Metrics

a. Define CloudWatch Alarms for Custom Metrics

- Create CloudWatch Alarms that trigger scaling based on custom metrics.

Steps:

1. Go to **CloudWatch Dashboard** > **Alarms** > **Create Alarm**.
2. Select the custom metric (e.g., MemoryUtilization) from the list of available metrics.
3. Set the threshold (e.g., scale out if MemoryUtilization > 80% for 2 data points).
4. Choose **Auto Scaling Group** as the alarm action and specify whether the action is a scale-in or scale-out.

b. Set Up Target Tracking Scaling Policy

- Use target tracking policies to automatically adjust your instance count to maintain the target value for your custom metric.

Steps:

1. Go to **Auto Scaling Group** > **Scaling Policies**.
2. Select **Add Policy** > **Target Tracking Scaling**.
3. Choose the custom CloudWatch metric (e.g., MemoryUtilization).
4. Set the target value (e.g., maintain MemoryUtilization at 75%).
5. Configure the scaling behavior (e.g., cooldown period).

5. Implement Scheduled Actions

a. Create Scheduled Actions in Auto Scaling

- Define specific times when the Auto Scaling Group should increase or decrease the number of instances, regardless of current metrics.

Steps:

1. Navigate to the **Auto Scaling Group** in the AWS console.
2. Under the **Scheduled Actions** tab, choose **Create Scheduled Action**.
3. Set up the action with:
    - **Action Name** (e.g., ScaleOutMorning).
    - **Recurrence** (CRON expression or predefined schedule).
    - **Desired Capacity** to adjust the number of instances.
    - **Start and End Time** for when the scaling should occur.

aws autoscaling put-scheduled-update-group-action \

--auto-scaling-group-name my-asg \

--scheduled-action-name scale-in-night \

--recurrence "0 18 * * *" \
--desired-capacity 1

b. Test Scheduled Actions

- Ensure your scheduled actions are functioning correctly by testing with temporary schedules. For example, schedule an action in the next 5 minutes to observe its execution.

6. Monitor and Fine-Tune

a. Monitor Auto Scaling Activity

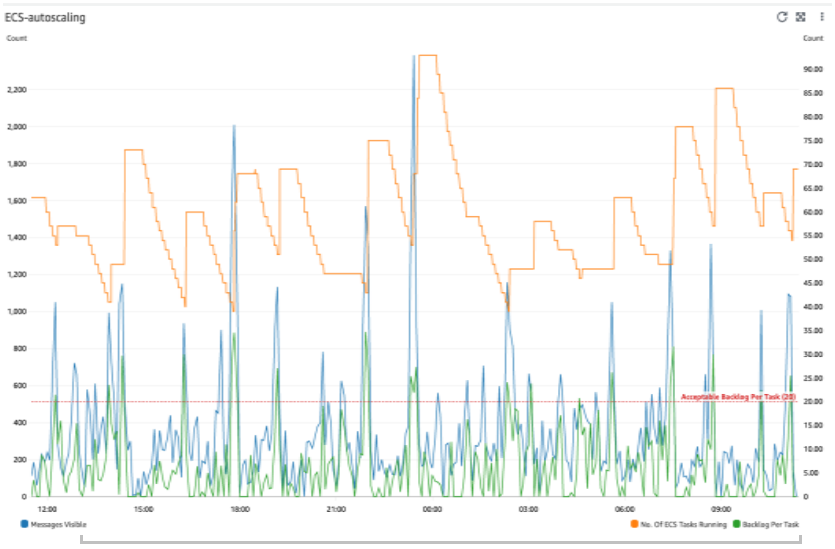
- Go to the **Auto Scaling Group** > **Activity** tab to see a log of scaling events.
- Check **CloudWatch Alarms** to ensure alarms are firing correctly and triggering the desired scaling actions.

b. Fine-Tuning

- Adjust the thresholds for custom metric alarms as needed.
- Modify scaling cooldowns if scaling is happening too frequently or too slowly.
- Refine scheduled actions to match peak traffic times accurately.

7. Automate and Optimize

- Set up notifications for scaling events (e.g., using SNS).
- Consider using **Auto Scaling Lifecycle Hooks** to add custom actions before scaling in or out, such as draining connections from instances.



RESULT :

EX NO 7 SETTING UP AN APPLICATION LOAD BALANCER WITH PATH-BASED ROUTING AND SSL/TLS

AIM:

The aim is to design and implement an **Application Load Balancer (ALB)** in AWS that utilizes **path-based routing** and **SSL/TLS** to enhance the performance, security, and availability of web applications. The goal is to ensure that traffic is efficiently distributed among multiple targets based on the requested URL path while maintaining secure connections for end users.

PROCEDURE : 1. Initial Setup

a. Launch EC2 Instances or Services

- Ensure you have EC2 instances or target services (like ECS) ready to be used as the backend targets for the load balancer.
- Confirm that these instances have a working web application that you'll use for path-based routing (e.g., /api goes to Service A, /admin goes to Service B).

2. Create a Target Group

A **Target Group** is a set of targets (EC2 instances, IPs, or Lambda functions) that ALB directs traffic to.

a. Define Target Groups

- Navigate to the **EC2 Dashboard** > **Target Groups** > **Create Target Group**.
- Select the target type:
  - **Instances** (for EC2 instances)
  - **IP Addresses** (for external IPs)
  - **Lambda functions** (for serverless apps)
- For **Target Group Name**, choose a name like app-service-target-group.

b. Configure Target Group

- Select the protocol (HTTP or HTTPS) and port your instances are listening on (e.g., HTTP:80 or HTTPS:443).
- Set the **Health Check Protocol** (usually HTTP) and path (e.g., /healthcheck).

c. Register Targets

- Add your EC2 instances (or other targets) to the target group.
- Choose **Add to registered** for all instances that should be part of this group.

Repeat the process for other services if you plan to route traffic to multiple services based on paths.

3. Set Up the Application Load Balancer (ALB)

a. Create a Load Balancer

1. Go to the **EC2 Dashboard** > **Load Balancers** > **Create Load Balancer**.
2. Select **Application Load Balancer**.
3. For **Load Balancer Name**, choose a name like my-app-alb.
4. Choose the **Scheme**:
   - **Internet-facing** for public traffic.
   - **Internal** if the ALB is for private use (e.g., inside a VPC).
5. Select **IP Address Type**: IPv4 (or dual stack if needed).

b. Configure Network

- Select at least two subnets in different availability zones (for high availability).
- Ensure that the selected subnets are in the same VPC as your instances.

c. Configure Listeners and Protocols

- For **Listeners**, choose **HTTPS** for SSL/TLS termination.
- Specify a security group for the ALB that allows HTTPS traffic on port 443.

4. Set Up SSL/TLS

a. Obtain an SSL Certificate

You need an SSL certificate to enable HTTPS on your load balancer. You can either upload a certificate from a trusted provider or use **AWS Certificate Manager (ACM)** to issue a free certificate.

Using ACM:

1. Go to **AWS Certificate Manager** (ACM) > **Request a Certificate**.
2. Select **Request a public certificate**.
3. Enter your domain name (e.g., www.example.com).
4. Choose **DNS Validation** (simpler) or **Email Validation**.
5. Complete the certificate request process. Once validated, the certificate will be available in ACM.

b. Add the SSL Certificate to ALB

1. In the ALB creation wizard, under **Listeners**, select **HTTPS** and associate the ACM certificate:
   - Click on **Select a certificate from ACM** and choose the one you validated.
2. Choose **TLS Security Policies** based on your requirements (e.g., **ELBSecurityPolicy-2016-08**).

5. Configure Path-Based Routing

a. Create Listener Rules for Path-Based Routing

1. After the ALB is created, go to the **Listeners** tab.
2. Select the **HTTPS Listener** (port 443) and click **View/Edit Rules**.
3. Set up path-based routing by defining rules for different paths.

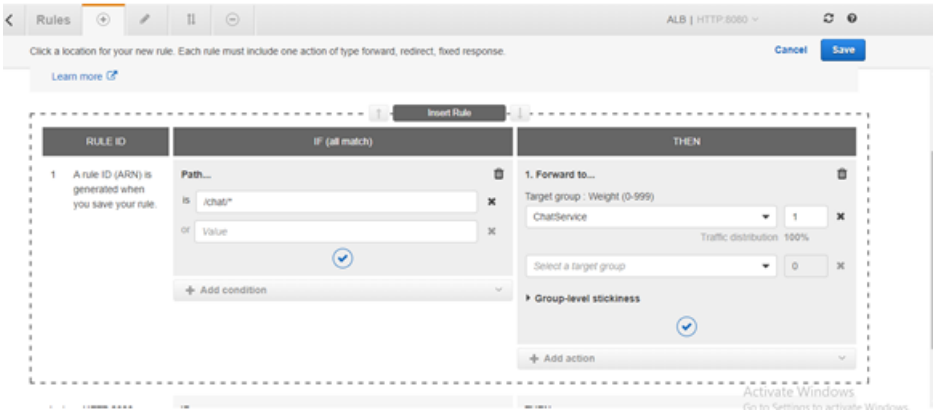
b. Define Rules

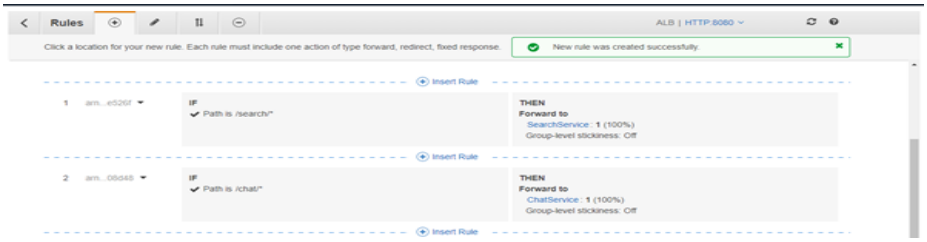
- Click + **Add Rule** > **Insert Rule**.
- In **IF** conditions, choose **Path** > **is /api/** (or any specific path like /admin/).
- In **THEN** actions, select **Forward to** and choose the appropriate **Target Group** (e.g., the group for /api traffic).

Repeat the process for each path that you want to configure.

c. Save and Test

- Save the rule configuration.
- Visit your domain (e.g., https://www.example.com/api/) to verify that traffic is directed to the correct service.





6. Configure Security Groups

Ensure that both your ALB and EC2 instances have the correct security groups:

a. ALB Security Group

- The security group for the ALB should allow inbound traffic on port 443 (HTTPS).
- Outbound traffic should be allowed on all ports to forward traffic to EC2 instances.

b. EC2 Instance Security Group

- The EC2 instances should allow inbound traffic from the ALB's security group on the application port (e.g., port 80 if the instances are running HTTP).

7. Test the Setup

a. Verify Path-Based Routing

- Navigate to the paths you defined, such as https://www.example.com/api/ or https://www.example.com/admin/.
- Ensure that the correct EC2 instances or services handle the traffic.

b. Verify SSL/TLS

- Ensure your browser shows a secure connection (HTTPS with a padlock) when accessing your application via the ALB.
- Check the SSL certificate by clicking the padlock icon in your browser and confirming the certificate details.

8. Monitor and Fine-Tune

a. Enable Access Logs

- Go to the ALB settings and enable access logging to track incoming traffic.
- Choose an S3 bucket to store the logs.

b. CloudWatch Metrics

- Use **Amazon CloudWatch** to monitor the ALB's performance (e.g., response times, traffic, error rates).
- Set up alarms for unhealthy targets or high latency.

RESULT :

EX 8 DEPLOYING A CONTAINERIZED APPLICATION USING AMAZON ECS WITH FARGATE

AIM :

The aim is to design and implement a containerized application deployment strategy using **Amazon Elastic Container Service (ECS)** with **Fargate** to achieve a fully managed, serverless infrastructure for running containers. The goal is to simplify the deployment and management of microservices architectures while ensuring scalability, availability, and security.

PROCEDURE :

1. **Set Up a Docker Image**

*a. **Create Your Application***

- Develop your containerized application, such as a simple web server, API, or any microservice.

Node.js

```
const express = require('express');
const app = express();
app.get('/', (req, res) => res.send('Hello World from ECS Fargate!'));
app.listen(3000, () => console.log('App running on port 3000'));
```

b. **Create a Dockerfile**

Node.js

Create a Dockerfile to package your application

```
# Use an official Node.js image
FROM node:14
# Set the working directory
WORKDIR /app

# Copy package.json and install dependencies
COPY package*.json ./
RUN npm install
# Copy the rest of the application code
COPY . .
# Expose the port your app will run on
EXPOSE 3000
# Command to run the app
CMD ["node", "app.js"]
```

c. **Build and Test the Docker Image Locally**

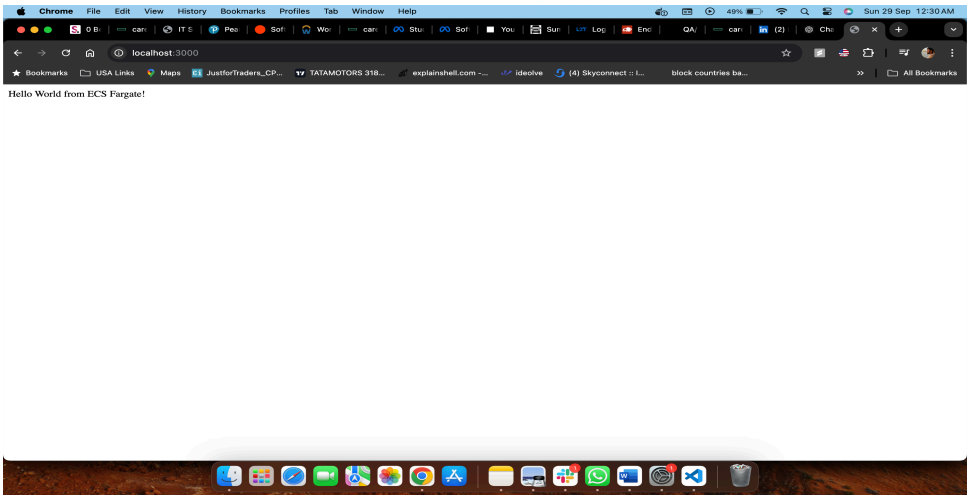
Build the Docker image using the following command

docker build -t my-app .

Run the container locally to ensure everything is working:

docker run -p 3000:3000 my-app

Access your application at http://localhost:3000 to verify it works.



2. Push Docker Image to Amazon ECR

ECS uses Amazon Elastic Container Registry (ECR) to store Docker images. Follow the

steps to push your image to ECR.

a. ***Create an ECR Repository***

1. Go to the **ECR Dashboard** in the AWS Management Console.
2. Click on **Create repository**.
3. Name your repository (e.g., my-app-repo) and select any required settings.
4. Click **Create repository**.

b. **Authenticate Docker to ECR**

• aws configure

• Use the AWS CLI to authenticate Docker with ECR:

aws ecr get-login-password --region <region> | docker login --username AWS --password-stdin <aws_account_id>.dkr.ecr.<region>.amazonaws.com

c. ***Tag and Push Your Docker Image***

Tag your image with the ECR repository URI:

docker tag my-app:latest <aws_account_id>.dkr.ecr.<region>.amazonaws.com/my-app-repo:latest

Push the image to ECR:

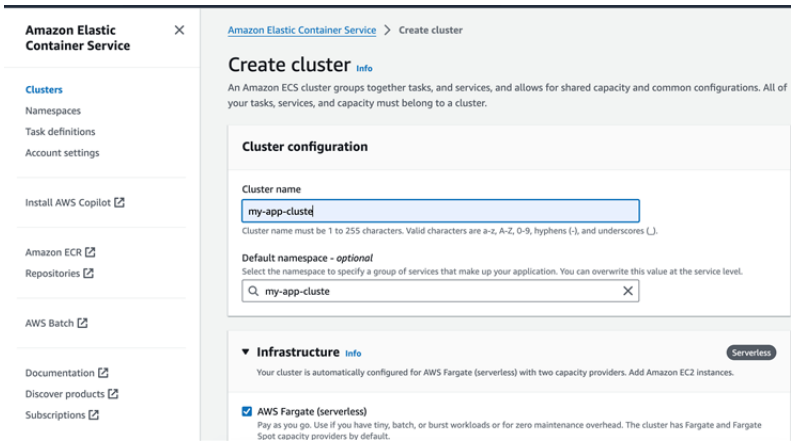
docker push <aws_account_id>.dkr.ecr.<region>.amazonaws.com/my-app-repo:latest

3. Set Up ECS Cluster

ECS requires a cluster to manage services and tasks. Fargate runs containers within this cluster without the need to manage infrastructure.

a. ***Create an ECS Cluster***

1. Go to the **ECS Dashboard** in the AWS Management Console.
2. Click on **Clusters** > **Create Cluster**.
3. Choose **Networking only (Fargate)** and click **Next**.
4. Name the cluster (e.g., my-app-cluster).
5. Click **Create**.



4. Create a Task Definition

A Task Definition defines how Docker containers are launched and configured in ECS.

a. ***Create a New Task Definition***

1. In the ECS Dashboard, go to **Task Definitions** > **Create new Task Definition**.
2. Select **Fargate** as the launch type.
3. Enter the following details:

- **Task Definition Name**: my-app-task.
- **Task Role**: Choose an IAM role with the necessary permissions (or create one).
- **Execution Role**: Select the ECS task execution role (ecsTaskExecutionRole).

*b. **Configure the Container***

1. Under the **Container Definitions** section, click **Add container**.
2. Enter:
   - **Container Name**: my-app.
   - **Image**: The ECR image URL (e.g.,
   - <aws_account_id>.dkr.ecr.<region>.amazonaws.com/my-app-repo:latest).
   - **Memory Limits**: Set soft and hard memory limits (e.g., 512 MiB).
   - **Port Mappings**: Map the container port (e.g., container port 3000, host port 0 as it's dynamically assigned).
3. Click **Add**.

c. **Set Task Size**

• Specify task memory and CPU requirements under **Task Size**.

- **Task Memory**: 1 GB.
- **Task CPU**: 0.5 vCPU (or choose based on the application's needs).

4. Click **Create**.

5. Create a Service in ECS

An ECS Service manages running tasks and maintains the desired number of instances.

*a. **Create a New Service***

1. Go to the **ECS Dashboard**, select your cluster, and click **Create Service**.
2. Choose **Fargate** as the launch type.
3. Select your **Task Definition** (e.g., my-app-task) and **Cluster** (e.g., my-app-cluster).
4. For **Service name**, use my-app-service.
5. Set **Number of tasks** (e.g., 1).

b. **Configure Network**

• Select a **VPC** and at least two **Subnets** for high availability.

• Choose **Auto-assign public IP** if your app needs to be publicly accessible.

• Select the appropriate **Security Group** to allow inbound traffic (e.g., HTTP on port 80 or 3000).

6. Click **Next** and **Create Service**.

6. Set Up Load Balancing (Optional)

You can configure an **Application Load Balancer (ALB)** to distribute traffic to the running containers. This step is optional but recommended for production environments.

*a. **Create an Application Load Balancer (ALB)***

1. Go to the **EC2 Dashboard** > **Load Balancers** > **Create Load Balancer**.
2. Select **Application Load Balancer**.
3. Name your ALB (e.g., my-app-alb) and choose **Internet-facing**.
4. Select at least two subnets in different Availability Zones.

*b. **Configure Security Group***

• Create or use an existing security group to allow HTTP/HTTPS traffic.

*c. **Configure Listeners and Target Group***

1. Add a listener for HTTP on port 80.
2. Create a new **Target Group** (e.g., my-app-tg) and register your ECS tasks as targets.
3. Go back to your ECS service, and under the **Load Balancing** section, select the newly created ALB

7. Test Your Application

*a. **Access the Application***

- Once the tasks are running, you can access the application via the public IP or DNS name of the Fargate tasks or ALB.
- If you're using a load balancer, navigate to the DNS name of the ALB to access your app.

*b. **Check Logs and Monitoring***

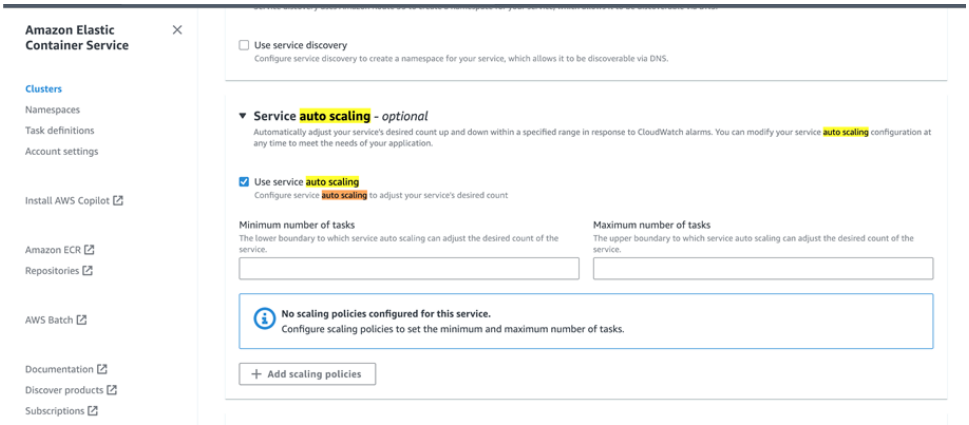
- Use **CloudWatch Logs** to monitor the application logs.
- Ensure **CloudWatch Metrics** are configured to track your container performance.

8. Scaling and Auto Scaling (Optional)

You can enable auto-scaling to scale your services based on CPU, memory utilization, or custom metrics.

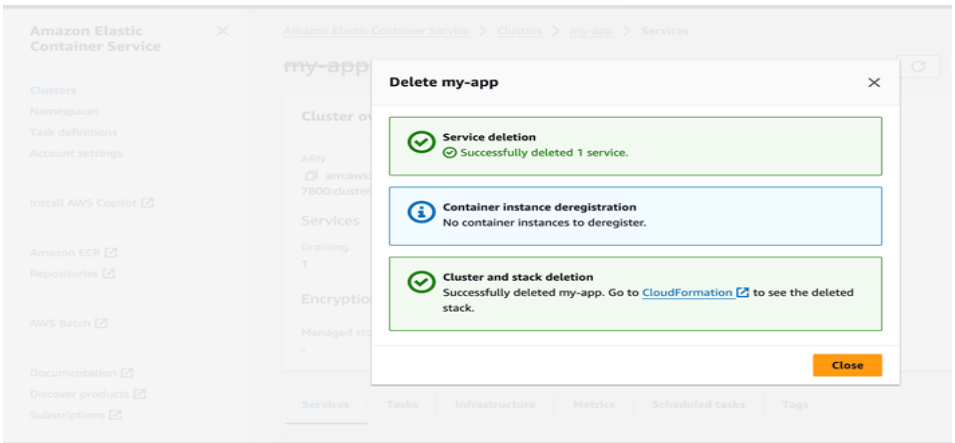
*a. **Enable Service Auto Scaling***

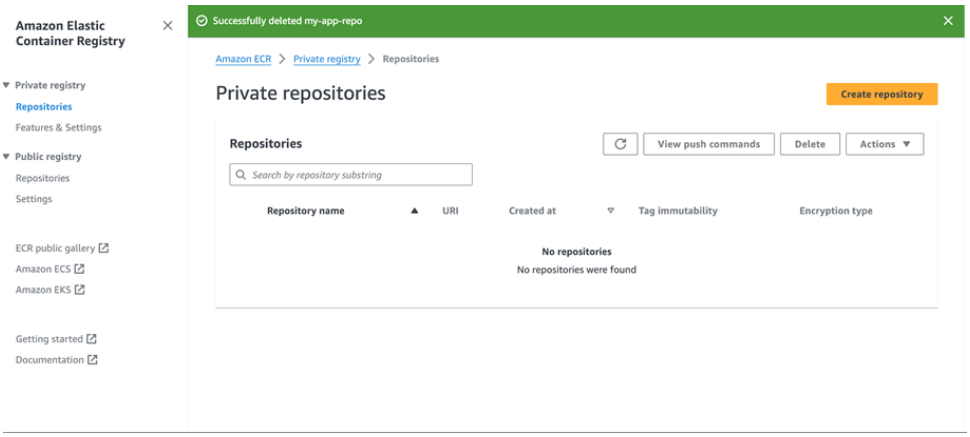
1. Go to the ECS service in the AWS console.
2. Click **Update Service** and enable **Auto Scaling**.
3. Set scaling policies based on CPU or memory usage thresholds.



9. Clean Up (Optional)

Once you're done testing, remember to clean up resources to avoid unnecessary charges. Delete the ECS service, cluster, load balancer, and ECR repository.





RESULT :

EX 9 IMPLEMENTING A CI/CD PIPELINE WITH JENKINS, AWS CODEPIPELINE, AND BLUE/GREEN DEPLOYMENT

AIM :

The aim is to design and implement a Continuous Integration/Continuous Deployment (CI/CD) pipeline using **Jenkins**, **AWS CodePipeline**, and **Blue/Green Deployment** to automate the build, test, and deployment process for applications. The goal is to ensure fast, reliable, and automated delivery of new features and updates while minimizing downtime and reducing the risk of deployment errors.

PROCEDURE:

Step 1: Set Up Jenkins as a CI Tool

*a. **Install Jenkins***

1. **Launch an EC2 instance**:
   - Use an Amazon Linux 2 or Ubuntu instance.
   - Ensure it has proper inbound security group rules (allow port 8080 for Jenkins UI).
2. **Install Jenkins**:
   - SSH into your EC2 instance and install Jenkins.

For Amazon Linux:

```
sudo yum update -y
sudo amazon-linux-extras install java-openjdk11
sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
sudo yum install jenkins -y
sudo systemctl start Jenkins
```

For Ubuntu:

```
sudo apt update -y
```

sudo apt install openjdk-11-jre -y

wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -

sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt update

sudo apt install jenkins -y

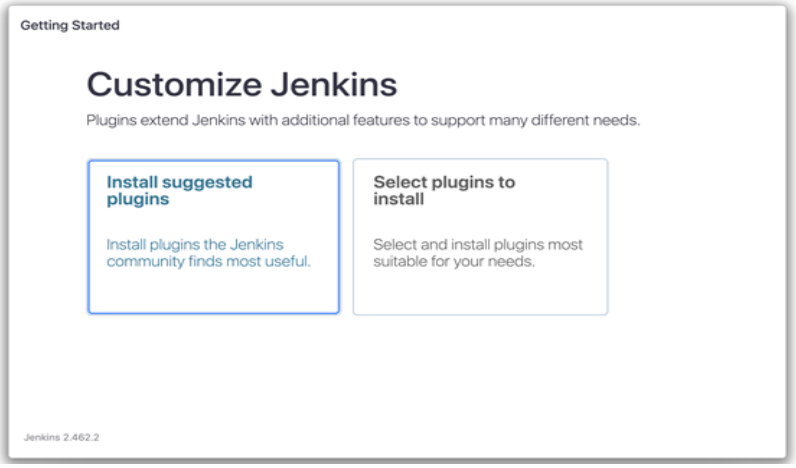
sudo systemctl start Jenkins

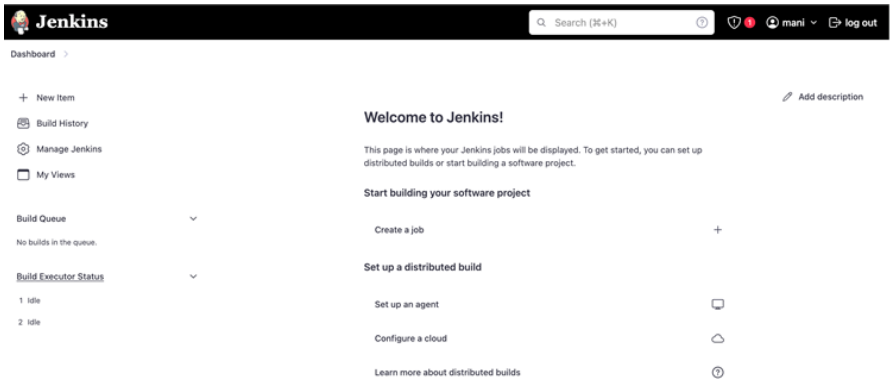
3. **Access Jenkins**:
    - Open your browser and navigate to http://<public_IP>:8080.
    - Complete the setup by unlocking Jenkins (retrieve the admin password from /var/lib/jenkins/secrets/initialAdminPassword).

b. ***Install Jenkins Plugins***

• Install necessary plugins for AWS and Git integration:

- **AWS CodePipeline** Plugin.
- **Pipeline** Plugin.
- **Git** Plugin.
- **Blue Ocean** Plugin (for a better pipeline visualization).

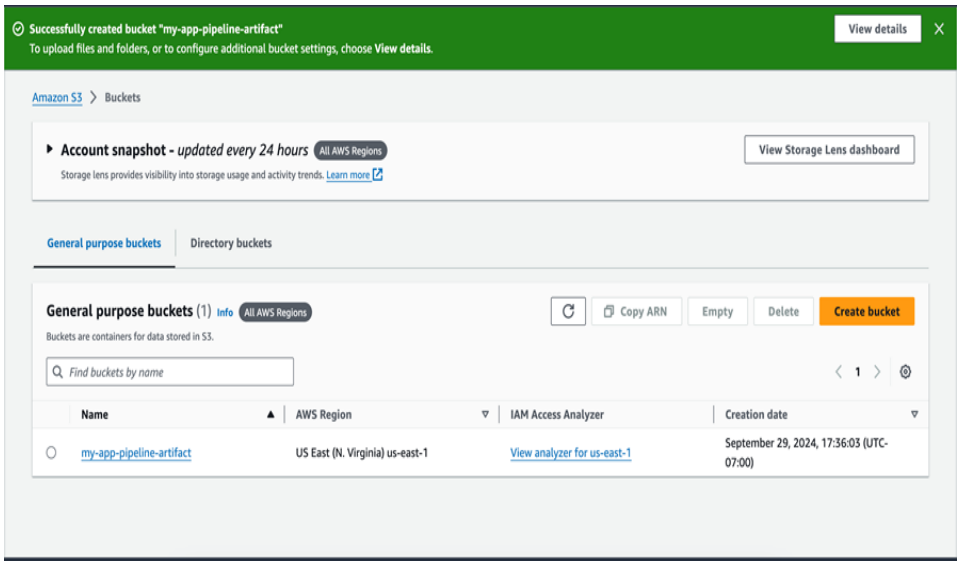




Step 2: Configure AWS CodePipeline for CI/CD

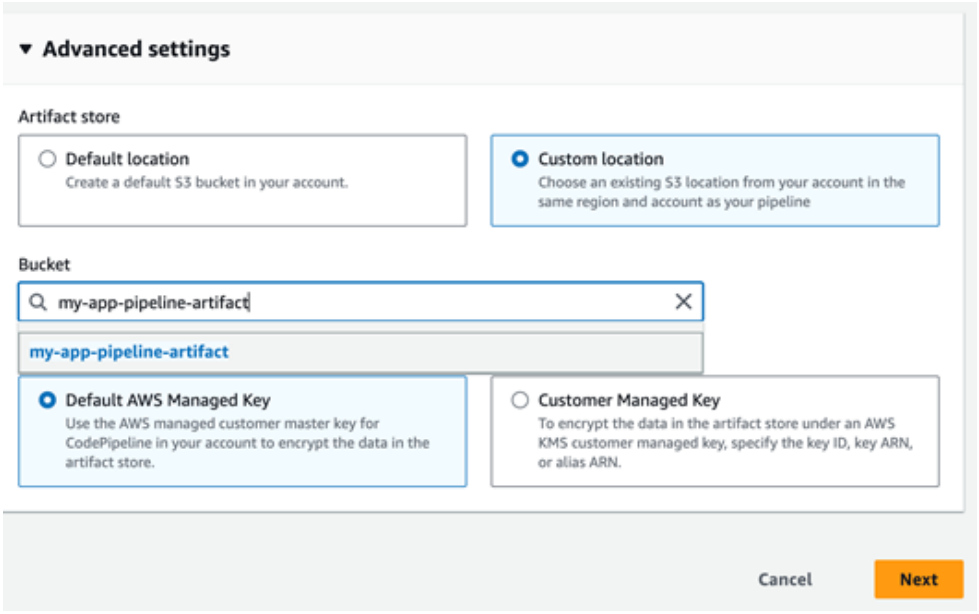
*a. **Create an S3 Bucket for Artifact Storage***

1. Go to the **S3 Dashboard** in AWS.
2. Click **Create Bucket**, name it (e.g., my-app-pipeline-artifacts), and configure it for artifact storage.



*b. **Create an AWS CodePipeline***

1. Navigate to **AWS CodePipeline** > **Create Pipeline**.
2. Enter the pipeline name (e.g., my-app-pipeline).



3. **Source Stage**:
   - Choose **Source Provider**: **GitHub** or **CodeCommit**.
   - Authenticate and select your repository.
   - Specify the branch that triggers the pipeline (e.g., main).

4. **Build Stage**:
    - Select **Jenkins** as the build provider.
    - Configure Jenkins with an **AWS CodePipeline** integration using the Jenkins plugin.
    - Enter the **Project Name** (pipeline job in Jenkins).
    -
5. **Deploy Stage**:
    - Choose **AWS Elastic Beanstalk** or **ECS**.
    - Select or create a new application and environment.
    - Specify deployment settings for Blue/Green deployment.

Step 3: Set Up Jenkins Job for the Build Stage

*a. **Create a Jenkins Pipeline Job***

1. Open the Jenkins dashboard and create a new **Pipeline Job**.
    - Go to **New Item** > **Pipeline** and name it (e.g., my-app-build).
2. **Configure Git Repository**:
    - In the **Pipeline** section, select **Pipeline script from SCM**.
    - Choose **Git** as the SCM and provide the GitHub or CodeCommit repository URL.
    - Add credentials if the repository is private.
3. **Define Jenkinsfile (Pipeline Script)**:
    - In your repository, add a Jenkinsfile to define the CI/CD pipeline. The Jenkinsfile should include stages for building, testing, and packaging the application.

Sample Jenkinsfile:

```
pipeline {
 agent any
  stages {
    stage('Build') {
      steps {
        echo 'Building the application...'
        sh 'npm install'
        sh 'npm run build'
      }
    }

    stage('Test') {
      steps {
        echo 'Running tests...'
        sh 'npm test'
      }
    }

    stage('Package') {
      steps {
        echo 'Packaging the application...'
        sh 'zip -r build.zip .'
```

```
            archiveArtifacts artifacts: 'build.zip'
        }
    }
  }
}
```

4.Save and Run:

- Trigger the pipeline manually to ensure it builds and packages the application correctly.

Step 4: Set Up Blue/Green Deployment in AWS

Blue/Green deployment allows for zero-downtime releases by directing traffic between two environments (production and staging).

a. ***Create an Elastic Beanstalk Application (or ECS Service)***

1. Go to **Elastic Beanstalk Dashboard** > **Create Application**.
   - Name your application (e.g., my-app).
   - Choose a platform (e.g., **Node.js**, **Python**, or **Docker**).
   - Choose the environment type: **Web server environment**.
   - Use **Fargate** for ECS or **Elastic Beanstalk** for simple applications.
2. **Deploy a Sample App** to create the environment initially.

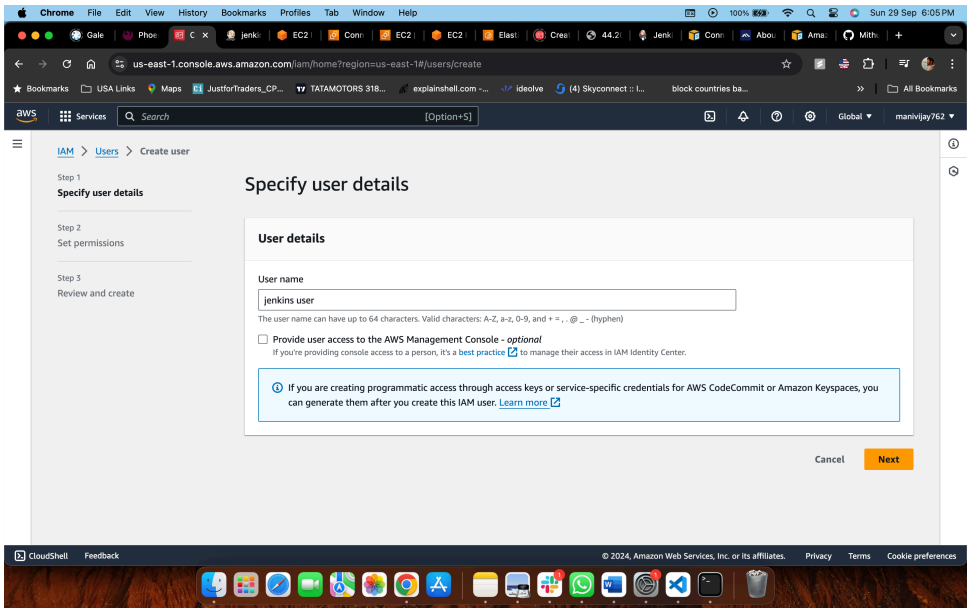
b. **Configure Blue/Green Deployment Settings**

1. In Elastic Beanstalk, go to **Configuration** > **Rolling updates and deployments**.
2. Enable **Blue/Green Deployment** and **Immutable Updates**.
   - This will create a new environment for each deployment and switch the DNS after verification.
3. **Add Elastic Load Balancer** (optional but recommended for handling traffic):
   - Attach a load balancer to your Beanstalk environment.
   - This ensures traffic is smoothly routed during the environment swap.

Step 5: Integrate Jenkins with AWS CodePipeline

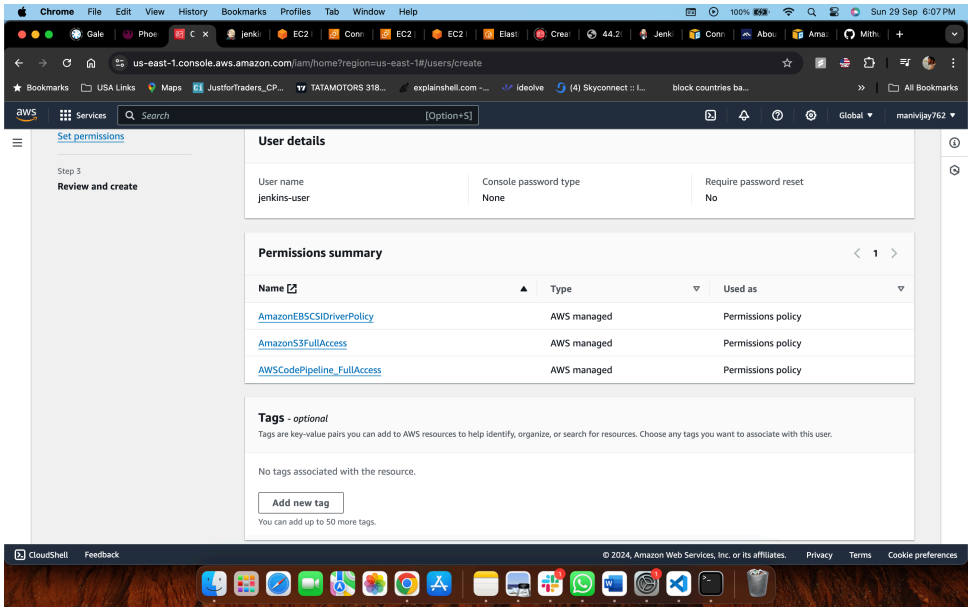
a. ***Set Up AWS IAM Roles for Jenkins***

1. Create an IAM role with permissions to access CodePipeline, Elastic Beanstalk (or ECS), and S3.
2. Attach the IAM role to your Jenkins instance or configure AWS credentials using the ~/.aws/credentials file on your Jenkins server.



b. ***Configure Jenkins Pipeline Job with AWS CodePipeline***

1. In Jenkins, edit your existing pipeline job and configure it to trigger based on CodePipeline webhook events.
    - Use the **AWS CodePipeline Plugin** to integrate Jenkins with the pipeline.
2. In the Jenkinsfile, add steps to upload the build artifact to S3 (for CodePipeline) and trigger deployment.



Example Jenkinsfile with AWS CodePipeline Integration:

```
pipeline {
  agent any
  environment {
    S3_BUCKET = 'my-app-pipeline-artifacts'
    AWS_REGION = 'us-west-2'
  }
```

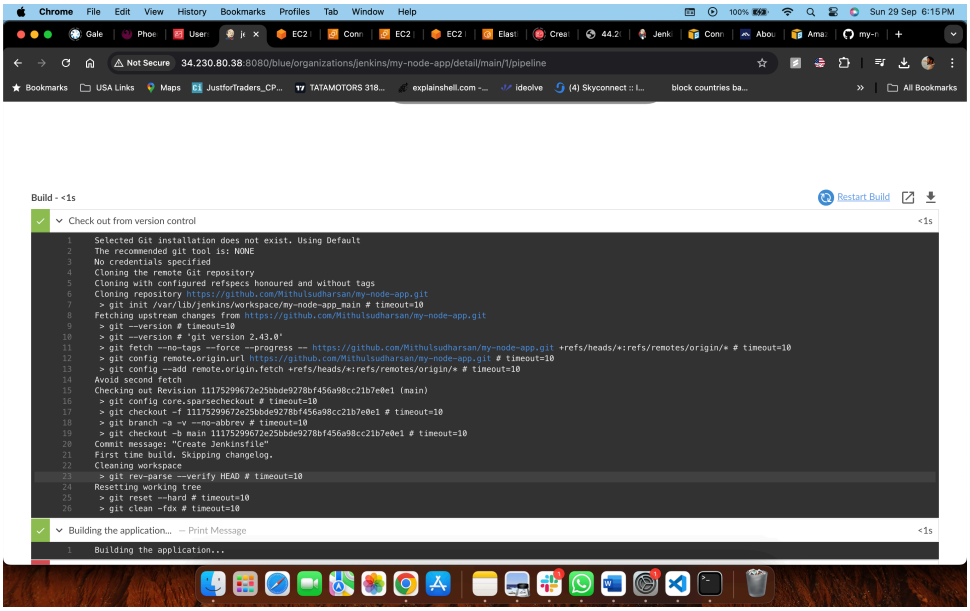
```
stages {
    stage('Build') {
        steps {
            echo 'Building the application...'
            sh 'npm install'
            sh 'npm run build'
        }
    }
    stage('Test') {
        steps {
            echo 'Running tests...'
            sh 'npm test'
        }
    }
    stage('Package') {
        steps {
            echo 'Packaging the application...'
            sh 'zip -r build.zip .'
            archiveArtifacts artifacts: 'build.zip'
        }
    }
    stage('Upload to S3') {
        steps {
            echo 'Uploading build.zip to S3...'
            sh 'aws s3 cp build.zip s3://$S3_BUCKET/'
        }
    }


    stage('Deploy') {
        steps {
            echo 'Deploying application to Elastic Beanstalk...'
                sh 'aws elasticbeanstalk create-application-version --application-name my-app --
version-label v1 --source-bundle S3Bucket=$S3_BUCKET,S3Key=build.zip'
                    sh 'aws elasticbeanstalk update-environment --application-name my-app --
environment-name my-app-env --version-label v1'
        }
    }
  }
}
```

Step 6: Test and Monitor the Pipeline

a. ***Trigger the Pipeline***

- Commit and push changes to your source repository.
- The pipeline will trigger the build in Jenkins, upload artifacts to S3, and initiate the deployment to Elastic Beanstalk (or ECS) with Blue/Green deployment.

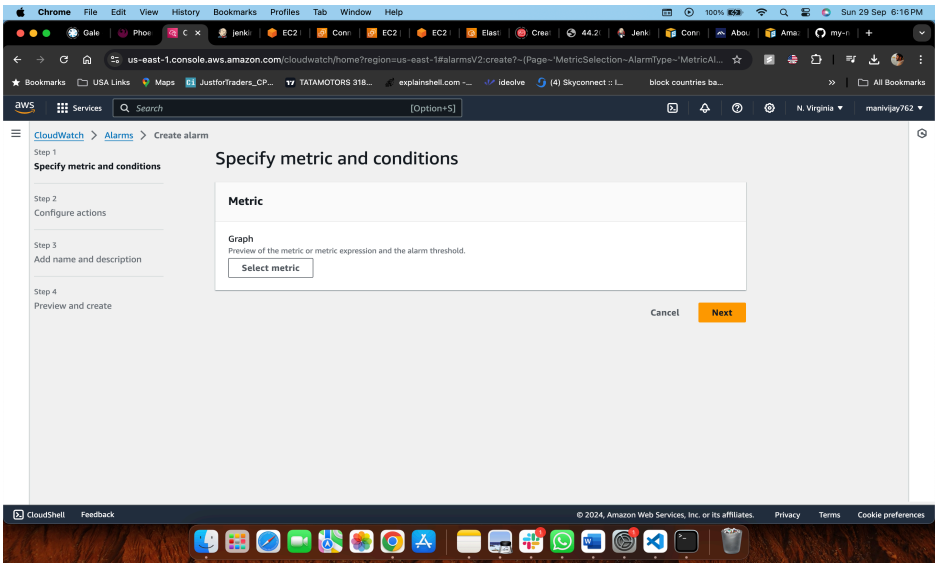


b. ***Monitor Deployment and Traffic Switching***

- Monitor your application deployment in Elastic Beanstalk or ECS. Ensure the Blue/Green deployment completes successfully, and traffic is switched to the new environment.

c. ***CloudWatch Logs and Metrics***

- Set up **CloudWatch Alarms** to monitor build failures, deployment issues, or traffic problems.
- Review logs from Jenkins and Elastic Beanstalk for debugging any issues.



Step 7: Optional Enhancements

- **Add Rollback Mechanism**: Automate rollback if the deployment fails or doesn't meet defined health checks.
- **Auto Scaling**: Enable auto-scaling in your environment to handle traffic changes dynamically.
- **Notifications**: Use **SNS** or **Slack** integrations to notify your team on pipeline events (build success/failure, deployment status).

RESULT :

EX 10 DESIGNING A SERVERLESS ARCHITECTURE WITH API GATEWAY,

AWS LAMBDA, AND DYNAMODB

AIM :

The aim is to design and implement a fully serverless architecture using **Amazon API Gateway**, **AWS Lambda**, and **Amazon DynamoDB** to build a scalable, cost-effective, and highly available application. The goal is to eliminate the need for server management while enabling rapid application deployment, automatic scaling, and high performance.

PROCEDURE :

Step 1: Create a DynamoDB Table

DynamoDB will be used to store data for the application.

a. Go to DynamoDB in AWS Console

1. In the AWS Console, search for **DynamoDB** and navigate to the DynamoDB Dashboard.
2. Click **Create Table**.

b. Define Table Details

1. Enter a **Table Name** (e.g., Users).
2. Specify a **Primary Key** (e.g., userId of type String).
    - You can also add a **Sort Key** if you need to define complex access patterns (optional).
3. Leave other options at their default values and click **Create Table**.

Step 2: Create an AWS Lambda Function

Lambda will handle the backend logic for your API. It will interact with DynamoDB to store, retrieve, update, or delete records.

a. Go to Lambda in AWS Console

1. Navigate to **AWS Lambda**.
2. Click **Create function**.

b. Choose a Function Blueprint

1. Choose **Author from Scratch**.
2. Enter a **Function name** (e.g., UserHandler).
3. Choose the **Runtime** (e.g., **Node.js**, **Python**, or **Java**).
4. Choose or create an **Execution Role**:
    - Ensure the Lambda function has an IAM role that allows it to read/write to DynamoDB. You can attach the **AmazonDynamoDBFullAccess** policy or create a custom policy with the necessary permissions.

c. Write Lambda Function Code

Write the Lambda function code to perform CRUD operations (Create, Read, Update, Delete) on DynamoDB.

Example in Node.js:

```
const AWS = require('aws-sdk');
const dynamo = new AWS.DynamoDB.DocumentClient();
const tableName = process.env.TABLE_NAME;
exports.handler = async (event) => {
  const { httpMethod, pathParameters, body } = event;
  let response;
  switch (httpMethod) {
    case 'GET':
      response = await getUser(pathParameters.userId);
      break;
    case 'POST':
```

```
            response = await createUser(JSON.parse(body));
            break;
        case 'PUT':
            response = await updateUser(pathParameters.userId, JSON.parse(body));
            break;
        case 'DELETE':
            response = await deleteUser(pathParameters.userId);
            break;
        default:
            response = {
                statusCode: 405,
                body: JSON.stringify({ message: 'Method not allowed' })
            }; }
    return response;
};
const getUser = async (userId) => {
    const params = {
        TableName: tableName,
        Key: { userId }
    };
    const result = await dynamo.get(params).promise();
    return {
        statusCode: 200,
        body: JSON.stringify(result.Item)
    };
};
const createUser = async (user) => {
    const params = {
        TableName: tableName,
        Item: user
    };
    await dynamo.put(params).promise();
    return {
        statusCode: 201,
        body: JSON.stringify({ message: 'User created', user })
    };
};
const updateUser = async (userId, updates) => {
    const params = {
        TableName: tableName,
        Key: { userId },
        UpdateExpression: 'set #name = :name, email = :email',
        ExpressionAttributeNames: { '#name': 'name' },
        ExpressionAttributeValues: {
            ':name': updates.name,
            ':email': updates.email
        }
    };
    await dynamo.update(params).promise();
    return {
```

```
    statusCode: 200,
    body: JSON.stringify({ message: 'User updated' })
  };
};
const deleteUser = async (userId) => {
  const params = {
    TableName: tableName,
    Key: { userId }
  };
  await dynamo.delete(params).promise();
  return {
    statusCode: 204,
    body: JSON.stringify({ message: 'User deleted' })
  };};};
```

d. Set Environment Variables

1. Go to your Lambda function's **Configuration** tab and set the **Environment Variables**.
2. Add an environment variable for the table name:
    - TABLE_NAME = Users (or your DynamoDB table name).

e. Test Your Lambda Function

1. Click on **Test**, create a sample event, and test your Lambda function using sample input data.

Step 3: Create API Gateway

API Gateway is the front-end that exposes your Lambda function via HTTP endpoints.

a. Go to API Gateway in AWS Console

1. Search for **API Gateway** in the AWS Console.
2. Click **Create API**.
3. Select **HTTP API** or **REST API**. (For simplicity, HTTP API is used here.)

b. Create a New API

1. Name the API (e.g., UserAPI).
2. Click **Next** and follow the prompts.

c. Set Up Routes

1. Under **Routes**, create new routes that map HTTP methods (GET, POST, PUT, DELETE) to specific Lambda functions.

Example routes:

- GET /users/{userId} – Get user by ID.
- POST /users – Create a new user.
- PUT /users/{userId} – Update a user.
- DELETE /users/{userId} – Delete a user.

2. For each route, specify **Lambda Function** as the integration and select your **UserHandler** Lambda function.

d. Configure Path Parameters (if necessary)

1. For routes like /users/{userId}, you'll need to configure the **path parameter** (userId).
2. Ensure the Lambda function can access the path parameter from the **event** object.

e. Deploy the API

1. Once the routes and integrations are configured, click **Deploy API**.

2. Create a new **Stage** (e.g., dev, prod).

f. Test API Gateway

- Copy the **API Gateway Invoke URL** from the **Stages** section and use a tool like **Postman** or **cURL** to test your API.

Example using cURL:

curl -X GET https://<api-id>.execute-api.<region>.amazonaws.com/dev/users/{userId}

Step 4: Set Up Permissions

a. Lambda Execution Role

Ensure the Lambda function has permissions to access DynamoDB:

1. Attach the AWSLambdaDynamoDBExecutionRole to the Lambda execution role.
2. Optionally, create a custom IAM policy that gives dynamodb:PutItem, dynamodb:GetItem, dynamodb:UpdateItem, and dynamodb:DeleteItem permissions for your specific table.

b. API Gateway Invoke Permissions

- API Gateway should automatically have permission to invoke the Lambda function, but you can double-check this in the **Lambda Function Permissions** settings.

Step 5: Test the Complete Serverless Setup

Now that everything is set up, you can test the end-to-end workflow of the API interacting with Lambda and DynamoDB.

a. Test with Postman or cURL

Create a User (POST request):

curl -X POST https://<api-id>.execute-api.<region>.amazonaws.com/dev/users \

-H "Content-Type: application/json" \

-d '{"userId": "1", "name": "John Doe", "email": "john@example.com"}'

Get a User (GET request):

curl -X GET https://<api-id>.execute-api.<region>.amazonaws.com/dev/users/1

Update a User (PUT request):

curl -X PUT https://<api-id>.execute-api.<region>.amazonaws.com/dev/users/1 \

-H "Content-Type: application/json" \

-d '{"name": "Jane Doe", "email": "jane@example.com"}'

Delete a User (DELETE request):

curl -X DELETE https://<api-id>.execute-api.<region>.amazonaws.com/dev/users/1

Step 6: Optional Enhancements

a. Enable CORS

1. If your API is going to be accessed from a browser (frontend), enable **CORS** in API Gateway.
2. Go to **API Gateway** > **Settings** > **Enable CORS** for each route.

b. Set Up Monitoring and Logging

1. Enable **CloudWatch Logs** for both API Gateway and Lambda.
2. In the Lambda console, ensure **CloudWatch Logs** are turned on to track function execution.
3. You can also set up **CloudWatch Alarms** for Lambda errors or latency issues.

c. Custom Domain Name for API Gateway (Optional)

1. If needed, configure a custom domain for your API Gateway, attach SSL certificates using **AWS Certificate Manager (ACM)**, and map it to your API Gateway stage.

RESULT :

EX 11 IMPLEMENTING A DISASTER RECOVERY SOLUTION USING ROUTE 53 AND CROSS-REGION REPLICATION

AIM :

The aim is to design and implement a highly available and fault-tolerant disaster recovery (DR) solution using **Amazon Route 53** and **Cross-Region Replication** across AWS regions. The goal is to minimize downtime and ensure seamless failover in the event of a disaster or outage in the primary AWS region.

PROCEDURE :

Step 1: Set Up S3 Cross-Region Replication (CRR)

The first step in ensuring data replication between regions is to set up **S3 Cross-Region Replication**.

a. Create the Source and Destination S3 Buckets

1. **Create Source Bucket**:
    - Go to the **S3 Console**.
    - Click **Create Bucket** and name it (e.g., source-bucket).
    - Choose your **primary region** (e.g., **us-east-1**).
    - Enable **Versioning** on the source bucket.
2. **Create Destination Bucket**:
    - Create another S3 bucket in a **different region** (e.g., **us-west-2**), and name it (e.g., destination-bucket).
    - Enable **Versioning** on the destination bucket.

b. Configure Cross-Region Replication

1. Go to the **source bucket** and open the **Management** tab.
2. Click **Create Replication Rule** and name it (e.g., replicate-to-west).
3. Configure the **Rule Scope**:
    - You can replicate the entire bucket or specific prefixes/tags.
4. Set the **Destination**:
    - Choose the destination bucket (destination-bucket) created earlier.
    - Select the destination region (e.g., us-west-2).
5. **IAM Role**:
    - Choose to create a new **IAM Role** that grants replication permissions between buckets.
6. **Review and Create**:
    - Save the replication rule, which will start the replication process for all new objects added to the source bucket.

c. Test Cross-Region Replication

1. Upload files to the **source bucket**.

2. Verify that the files are replicated to the **destination bucket** in the second region.

Step 2: Set Up Route 53 DNS Failover

Once cross-region replication is set up for data, the next step is configuring **Route 53** for automatic DNS failover. This ensures that in case the primary region becomes unavailable, traffic is routed to the secondary region.

a. Create Route 53 Hosted Zone

1. Navigate to the **Route 53 Console**.
2. Click **Create Hosted Zone** and enter the domain name for your website (e.g., example.com).
3. Click **Create** to get a hosted zone with the necessary DNS records.

b. Set Up Health Checks

Health checks will monitor the health of the endpoints (servers in different regions) and trigger failover when necessary.

1. Go to the **Health Checks** section in Route 53.
2. Click **Create Health Check**:
    - **Configure the Endpoint**: Enter the public IP or domain name of your **primary server** in Region A (e.g., us-east-1).
    - **Specify Protocol and Path**: Use **HTTP** or **HTTPS** with the path to a health-check file or endpoint (e.g., /healthcheck).
    - Set up **failure thresholds** and **intervals**.
3. Create another health check for the **secondary region** (Region B).

c. Create Route 53 DNS Records for Failover

1. Go to your **Hosted Zone**.
2. **Create Record Set** for the **primary region**:
    - Enter a record name (e.g., www or leave blank for the root domain).
    - Choose **A Record** (or **CNAME** if applicable).
    - Set the **Routing Policy** to **Failover**.
    - Set the **Failover Record Type** to **Primary**.
    - Associate this with the health check you created for the **primary region**.
3. **Create Record Set** for the **secondary region**:
    - Create another record for the same domain.
    - Set the **Routing Policy** to **Failover**.
    - Set the **Failover Record Type** to **Secondary**.
    - Associate it with the health check for the **secondary region**.
    - This will serve traffic when the health check for the primary region fails.

Step 3: Test the Disaster Recovery Solution

To ensure that your DR solution works effectively, conduct a **disaster recovery drill**.

a. Simulate a Failure

1. Disable or stop the EC2 instances, RDS instances, or other services in the primary region.
2. Monitor the **Route 53 health checks**. Once the health check for the primary region fails, Route 53 will automatically failover and start routing traffic to the secondary region.

b. Verify Cross-Region Replication

1. Check if your data (S3, DynamoDB, RDS) is accessible from the secondary region.
2. Verify that application users can continue interacting with the application hosted in the secondary region without downtime.

c. Restore Normal Operations

1. Once the primary region becomes available, re-enable the services.

2. Route 53 will automatically restore traffic to the primary region if it passes the health checks again.

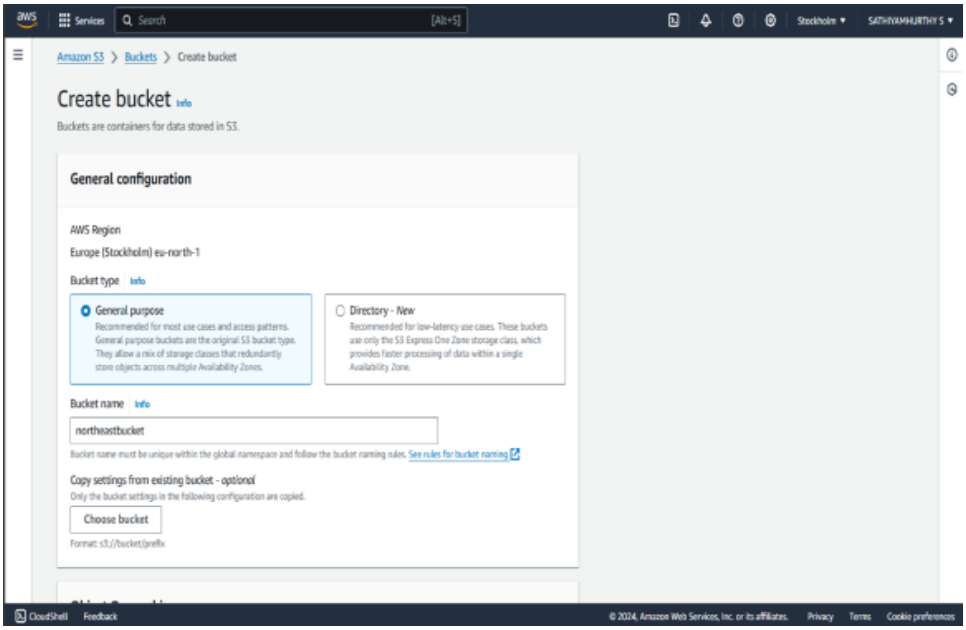
Step 4: Set Up Monitoring and Alerts

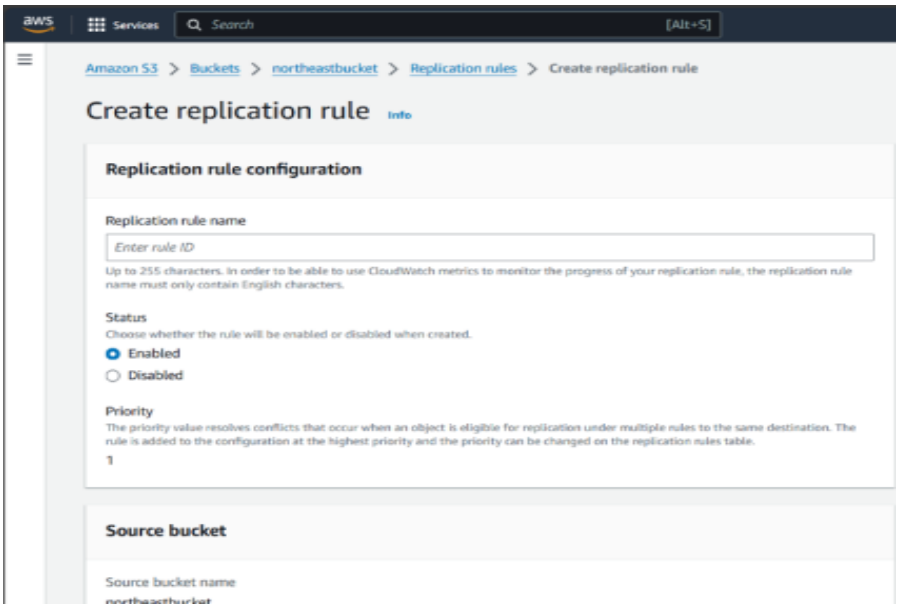
To monitor the disaster recovery setup and get alerted in case of failures:

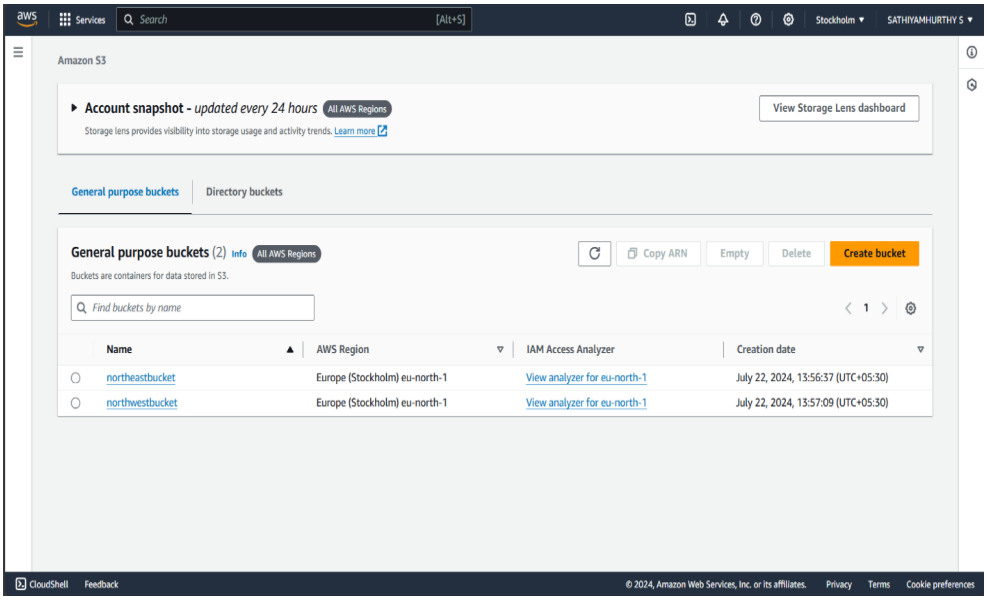
1. **CloudWatch Alarms**:
    - Create **CloudWatch Alarms** for Route 53 health checks to alert you when the health check fails and triggers a failover.
    - You can monitor S3 Cross-Region Replication with **CloudWatch** for any replication failures.
2. **SNS Notifications**:
    - Set up **SNS** notifications to receive alerts when failover occurs or if there are any issues with replication.

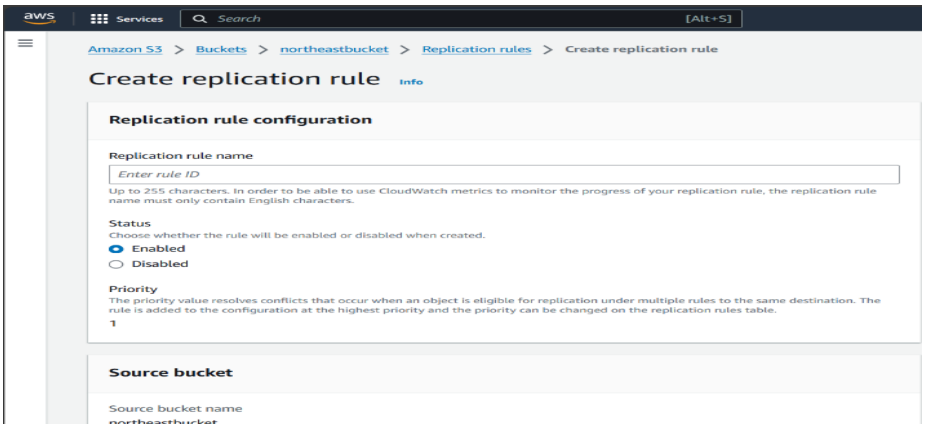
Step 5: Optimize for Cost and Performance

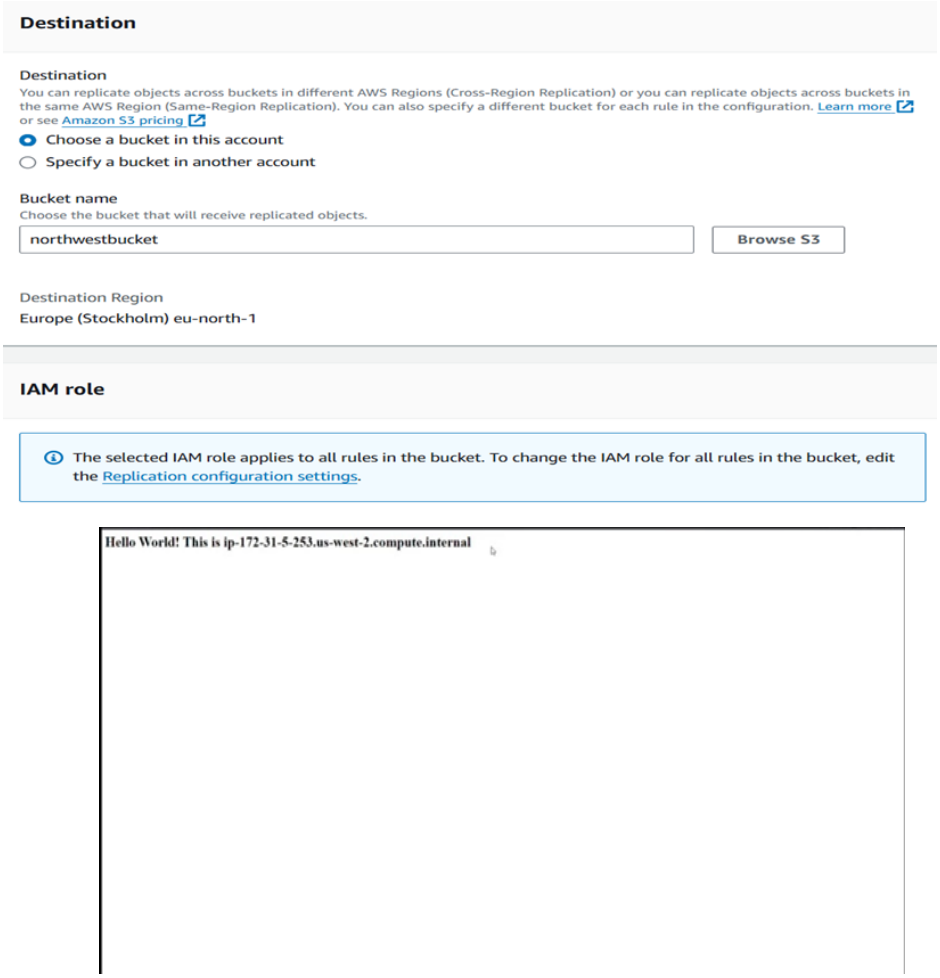
1. **Use Auto Scaling**: Configure **Auto Scaling Groups** in both regions to dynamically adjust compute resources (EC2 instances or ECS tasks) in response to traffic.
2. **Cost Optimization**: Set **Lifecycle Policies** to delete outdated AMIs, snapshots, or objects from S3 buckets to reduce storage costs.
3. **Regular Testing**: Conduct periodic **disaster recovery drills** to ensure the system is always prepared for unexpected outages.




Services
Search
[Alt+S]
Amazon S3 > Buckets > northeastbucket > Replication rules > Create replication rule
Create replication rule Info
Replication rule configuration
Replication rule name
Enter rule ID
Up to 255 characters. In order to be able to use CloudWatch metrics to monitor the progress of your replication rule, the replication rule name must only contain English characters.
Status
Choose whether the rule will be enabled or disabled when created.
Enabled
Disabled
Priority
The priority value resolves conflicts that occur when an object is eligible for replication under multiple rules to the same destination. The rule is added to the configuration at the highest priority and the priority can be changed on the replication rules table.
1
Source bucket
Source bucket name


Services
Search
[Alt+S]
Stockholm
SATHIYAMHURTHY S
Amazon S3
Account snapshot - updated every 24 hours
All AWS Regions
Storage lens provides visibility into storage usage and activity trends. Learn more
View Storage Lens dashboard
General purpose buckets
Directory buckets
General purpose buckets (2) Info
All AWS Regions
Buckets are containers for data stored in S3.
Copy ARN
Empty
Delete
Create bucket
Find buckets by name
Name
AWS Region
IAM Access Analyzer
Creation date
northeastbucket
Europe (Stockholm) eu-north-1
View analyzer for eu-north-1
July 22, 2024, 13:56:37 (UTC+05:30)
northwestbucket
Europe (Stockholm) eu-north-1
View analyzer for eu-north-1
July 22, 2024, 13:57:09 (UTC+05:30)
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Services
Search
[Alt+S]
Amazon S3 > Buckets > northeastbucket > Replication rules > Create replication rule
Create replication rule Info
Replication rule configuration
Replication rule name
Enter rule ID
Up to 255 characters. In order to be able to use CloudWatch metrics to monitor the progress of your replication rule, the replication rule name must only contain English characters.
Status
Choose whether the rule will be enabled or disabled when created.
Enabled
Disabled
Priority
The priority value resolves conflicts that occur when an object is eligible for replication under multiple rules to the same destination. The rule is added to the configuration at the highest priority and the priority can be changed on the replication rules table.
1
Source bucket
Source bucket name



RESULT :

EX N0 12 DEPLOY AND CONNECT RTOS USING AWS IOT FOR EDGE.

AIM :

The aim is to **Deploying and Connecting RTOS (Real-Time Operating System) Using AWS IoT for Edge** is to enable seamless integration of edge devices running RTOS with AWS IoT services, facilitating real-time data collection, processing, and control.

PROCEDURE :

Step 1:

Before you begin, ensure that you have the following:

- **AWS Account**: Set up an AWS account if you don't have one.
- **AWS IoT Core enabled**: IoT Core should be set up in your AWS account.
- **Supported RTOS Device**: RTOS devices such as FreeRTOS, or any other microcontroller that can work with AWS IoT SDK.
- **AWS CLI Installed**: Install the AWS CLI for managing AWS services from the

command line.

- **AWS IoT SDK**: Get the necessary AWS IoT SDK for RTOS.
- **Code editor or IDE**: You will need an IDE for code development.

Step 2: Setup AWS IoT on AWS Management Console

1. **Create an AWS IoT Thing**:
    - Go to the AWS IoT Console.
    - Click on **Manage** > **Things** > **Create Things**.
    - Create a new IoT Thing by providing a name and attributes.
2. **Generate Certificates**:
    - Create a new certificate for your IoT thing. These certificates are used for securing communication between the edge device and AWS IoT.
    - Download the device certificate, private key, and public key. Also, download the **Root CA certificate**.
3. **Attach Policy to Certificate**:
    - Create an IoT policy that grants the required permissions to the device.
    - Attach the policy and certificate to the created IoT Thing.
4. **Activate the Certificate**:
    - After attaching the certificate, make sure to activate it in the IoT Console.

Step 3: Set Up FreeRTOS on the Edge Device

1. **Download FreeRTOS**:
    - You can download the FreeRTOS SDK from the FreeRTOS official site.
    - FreeRTOS offers pre-configured demos to easily connect to AWS IoT Core.
2. **Configure Wi-Fi and AWS IoT Settings**:
    - In the downloaded FreeRTOS SDK, locate the aws_clientcredential.h and aws_clientcredential_keys.h files.
    - Update the files with the following:
        - Wi-Fi SSID and password.
        - AWS IoT Core endpoint URL.
        - Device certificate, private key, and root CA.
3. **Compile FreeRTOS for Your Device**:
    - Use an IDE such as Eclipse or VS Code to compile the FreeRTOS firmware for your microcontroller.

Step 4: Flash the RTOS Firmware onto the Device

1. **Flash the Binary**:
    - Once compiled, flash the firmware onto your edge device using the appropriate tools (like STLink, JTAG, or USB for development boards).
2. **Monitor Serial Output**:
    - Connect to the device over a serial connection (e.g., using PuTTY or a similar tool) to verify that the device is booting correctly and attempting to connect to AWS IoT Core.

Step 5: Connecting the RTOS Device to AWS IoT Core

1. **MQTT Communication**:
    - After flashing, the device should automatically try to connect to AWS IoT Core using MQTT protocol.
    - Use the FreeRTOS demo application to publish messages to the AWS IoT

MQTT broker.

2. **Verify Connection**:
    - Go to the AWS IoT Core Console.
    - Under **Test** in the left navigation, subscribe to the topic that your device is publishing to.
    - You should see messages coming in from your RTOS device if everything is set up correctly.

Step 6: Monitor and Manage Device from AWS IoT

1. **Monitor Device Activity**:
    - You can monitor your device's connectivity and data published via the **AWS IoT Monitor** section.
2. **Create IoT Rules and Actions**:
    - Create rules to process the data your device sends, such as sending it to AWS Lambda, storing it in an S3 bucket, or sending alerts to SNS.
3. **Device Shadow**:
    - Optionally, implement the AWS IoT Device Shadow feature to maintain a virtual representation of your device's state in AWS. This allows you to interact with and manage device states without a direct connection.

Step 7: Deploy Custom Applications on the Edge

- With the RTOS successfully connected to AWS IoT Core, you can now deploy additional functionality to your edge device:
    - Implement local data processing using FreeRTOS tasks.
    - Run real-time actions based on MQTT messages received.
    - Use AWS Greengrass for running AWS Lambda functions directly on your edge device if further integration is needed.

Step 8: Securing the Deployment

1. **Enable TLS**: Ensure that all communication between the RTOS device and AWS IoT Core is secured using TLS encryption.
2. **Monitor Security**: Use AWS IoT Device Defender to monitor and audit security configurations and device behavior.

By following these steps, you can deploy an RTOS on your edge device and connect it securely to AWS IoT Core for real-time data handling and device management

RESULT :

EX NO 13 MANAGE AND SECURE RTOS USING AWS IOT FOR EDGE.

AIM :

The aim is to enhance the security, scalability, and operational efficiency of IoT deployments by leveraging AWS IoT's robust security features, device management capabilities, and edge computing support.

PROCEDURE :

Part 1: Onboarding and Connecting an RTOS Device to AWS IoT

1. **Download FreeRTOS and AWS IoT Libraries**:
    - Download the FreeRTOS source code from the FreeRTOS website.
    - Include the AWS IoT libraries (aws_iot_mqtt, aws_clientcredential.h, etc.) into your RTOS project.
2. **Configure AWS IoT Credentials on RTOS**:
    - Obtain your AWS IoT Core **endpoint**, **device certificate**, **private key**, and **Root CA** certificate.
    - Open aws_clientcredential.h in the FreeRTOS code and insert:
        - Your AWS IoT endpoint.
        - Wi-Fi SSID and password.
        - Device certificate, private key, and Root CA.
3. **Register Device on AWS IoT Core**:
    - Log into AWS IoT Console and create a new **IoT Thing** representing your RTOS device.
    - Download the certificates and keys, and note the endpoint URL.
4. **Attach IoT Policy to the Device**:
    - In AWS IoT, attach an **IoT Policy** to the device certificate, granting it permissions such as iot:Publish, iot:Subscribe, etc.
5. **Compile and Flash RTOS Firmware**:
    - Use a compatible IDE (e.g., Eclipse or VS Code) to compile the FreeRTOS firmware.
    - Flash the firmware to your microcontroller using a USB or JTAG programmer.
6. **Monitor Serial Logs**:
    - Open a serial monitor (e.g., PuTTY or Arduino Serial Monitor) to verify that the device is connecting to AWS IoT Core over MQTT.
7. **Test Connection in AWS IoT Core**:
    - In AWS IoT Core, go to the **Test** section and subscribe to the MQTT topic the device publishes to.
    - Verify that messages from the RTOS device are being received.

Part 2: Securing Communication with TLS and Certificates

1. **Verify TLS Configuration**:
    - Ensure that the RTOS device is using **TLS 1.2** for communication with AWS IoT.
    - Check the serial monitor logs to confirm that the device is establishing a secure connection.
2. **Set Up Fine-Grained IoT Policies**:
    - Update the IoT Policy to restrict device actions, following the principle of least privilege.
    - For example, allow the device only to publish to specific topics and read its own shadow.
3. **Test Communication Security**:
    - Perform a test where an invalid certificate is loaded on the RTOS device, and verify that AWS IoT rejects the connection.
    - Check for error logs on the device and AWS IoT Core.

Part 3: Managing the Device with AWS IoT Device Management

1. **Organize Devices in Thing Groups**:
    - Create a **Thing Group** in AWS IoT to organize multiple RTOS devices based on location, function, or firmware version.
    - Add the current device to the group.
2. **Create an AWS IoT Job for Remote Actions**:
    - Create an **AWS IoT Job** to execute a remote action (e.g., reboot the device, send configuration changes).
    - Deploy the job to your device and verify its completion through the job status.
3. **Monitor Device Activity**:
    - Enable **CloudWatch Logs** for your IoT device to track its activity in real-time.
    - Verify that you can see logs related to device connection, disconnection, and data transfer.

Part 4: OTA Firmware Updates

1. **Prepare OTA Update**:
    - Modify your RTOS firmware (e.g., change a sensor reading or LED behavior) and recompile the new version.
    - Upload the new firmware binary to an **S3 bucket**.
2. **Create an OTA Update Job**:
    - In AWS IoT Core, create a **Firmware OTA Job**.
    - Point to the S3 bucket containing the new firmware and specify the target device or group.
    - Monitor the job status as the firmware is downloaded and applied to the device.
3. **Verify the Update**:
    - Check the serial logs on the RTOS device to ensure the firmware was updated successfully.
    - In case of failure, implement rollback mechanisms to revert to the previous firmware.

Part 5: Monitoring Security with AWS IoT Device Defender

1. **Enable Device Defender**:
    - Enable **AWS IoT Device Defender** to continuously audit your device fleet for potential security vulnerabilities.
    - Set up audit checks for TLS configurations, device certificates, IoT policies, and more.
2. **Monitor Device Anomalies**:
    - Configure anomaly detection based on device behavior, such as the number of MQTT messages or connection attempts.
    - Simulate an anomaly by sending unexpected data or triggering repetitive connections from the device.
3. **Automate Alerts**:
    - Use **CloudWatch Alarms** or **SNS** to send notifications when anomalies or security breaches are detected.
    - Perform security audits using the Device Defender audit reports and address any issues.

RESULT :

44