Name: Shivam.R.Prajapati Div : D15C Roll No: 41 Academic Year: 2024-25

Cloud Deployment with Automation

Concepts Used:

- AWS CodePipeline
- EC2
- S3

Problem Statement:

Build a simple web application using AWS CodeBuild and deploy it to an S3 bucket. Then, automate the deployment process using AWS CodePipeline, ensuring the application is deployed on an EC2 instance. A sample index.html page will be used for demonstration.

Tasks:

1. **Set up AWS CodeBuild** for the web app.
2. **Create a pipeline** that deploys the web app to an S3 bucket.
3. **Use AWS CodeDeploy** to push updates to an EC2 instance.

1. Introduction

Case Study Overview:

This case study focuses on building a simple web application and automating its deployment using a combination of AWS services—**AWS CodeBuild, S3, CodePipeline, and CodeDeploy.** The task is to create a basic application, package it, and deploy it to an S3 bucket as the initial step. Afterward, AWS CodePipeline is used to automate the deployment, ensuring updates are automatically pushed to an EC2 instance using AWS CodeDeploy. The goal is to demonstrate a seamless, automated deployment pipeline for a web application, which simplifies continuous integration and delivery (CI/CD) processes.

Key Feature and Application:

The unique feature of this case study is the integration of multiple AWS services to create a fully automated deployment pipeline. By using AWS CodeBuild for compiling and packaging the web app, AWS S3 as the storage for the deployed app, and AWS CodePipeline for automating the process, the deployment becomes efficient and scalable. Additionally, CodeDeploy ensures that the web application can be easily pushed and updated on an EC2 instance, facilitating fast iteration and real-time updates to the deployed application. This automation greatly reduces the manual workload involved in deployment and allows for continuous delivery, making it highly practical for modern web applications that need frequent updates.

2. Step-by-Step Explanation

Step 1: Create an S3 Bucket

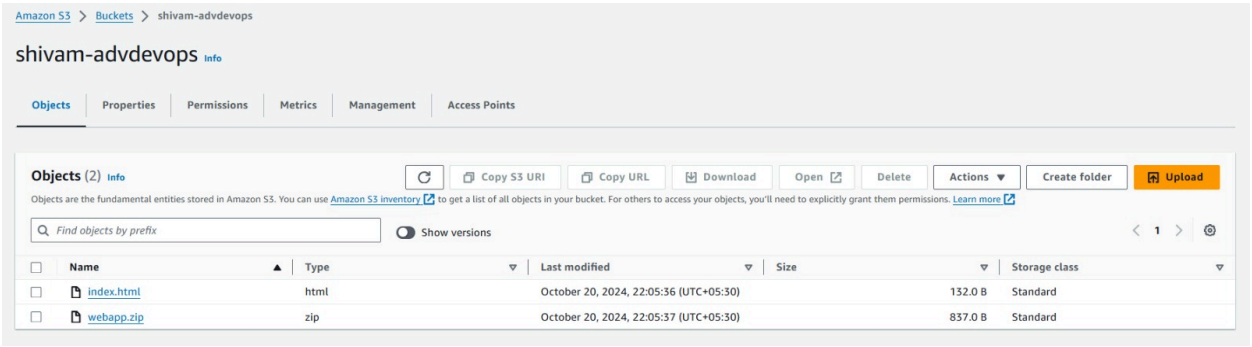
To begin, navigate to the S3 service in the AWS Management Console and click on "**Create bucket.**" Provide a unique name for your bucket, such as "shivam-advdevops" For testing purposes, uncheck the option "**Block all public access**" to allow public access. After creating the bucket, go to the "Properties" tab and enable versioning by selecting "**Enable**" and saving the changes.

Step 2: Create a Simple Web Application

First, create a file named index.html on your local machine and add some basic HTML code. Next, create a configuration file named Appsec.yml for your application's security settings. Then, create a script called install_dependencies.sh to update the system and install Nginx. After that, create another script named start_nginx.sh to start the Nginx service. Finally, zip all these files into a file named webapp.zip to make uploading them easier.

Step 3: Step 3: Upload the Web App to S3

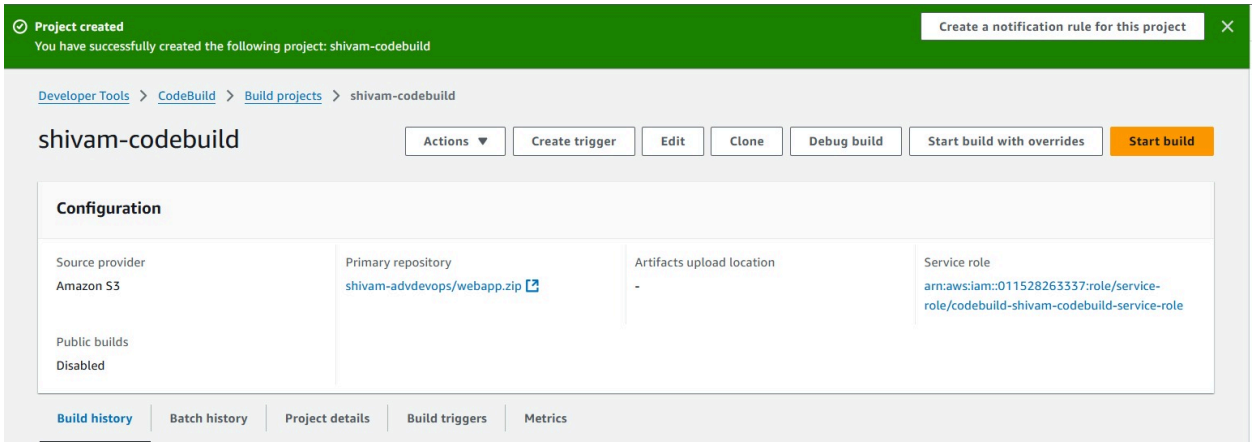
To upload your web app to S3, return to the S3 console and select your bucket. Click on "Upload," add the webapp.zip file, and then click "Upload" to complete the process.



Step 4: Set Up AWS CodeBuild

Click on "Create build project" and set it up with the following configurations:

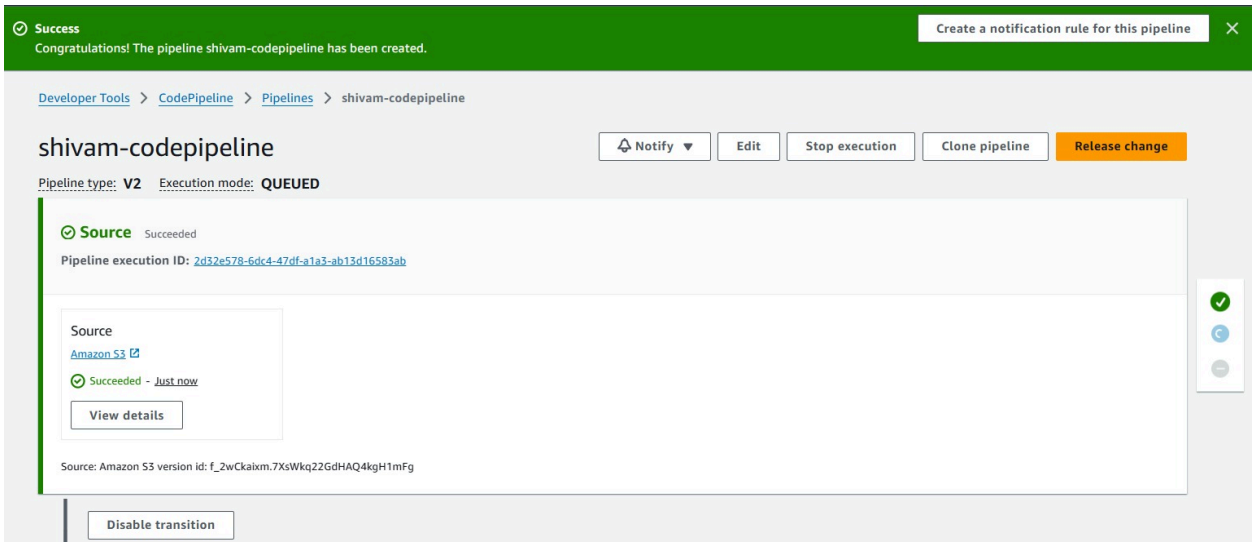
- **Project Name**: Provide a name for your project.
- **Source Provider**: Choose Amazon S3 as the source.
- **Bucket**: Select the S3 bucket you've created.
- **Object Key**: Choose webapp.zip as the file to use.
- **Operating System**: Set it to Ubuntu.
- **Buildspec**: Add the necessary build commands using a YAML script.
- Uncheck the option to upload logs to CloudWatch.

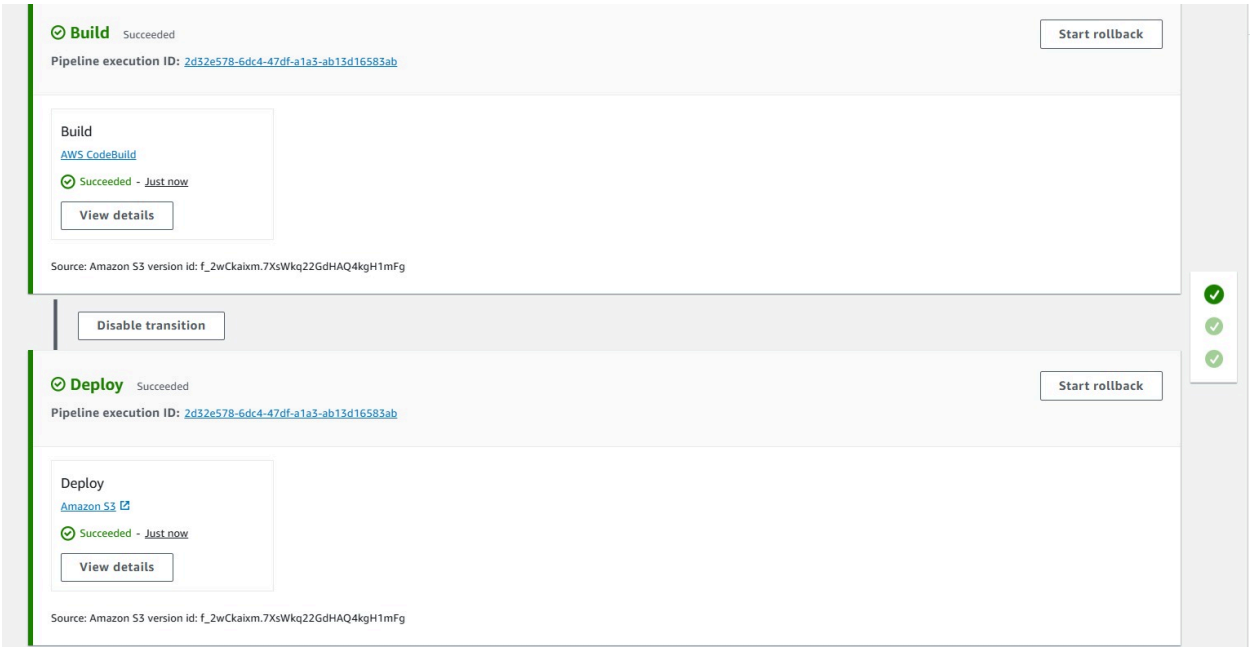


Step 5: Set Up AWS CodePipeline

In the AWS Management Console, search for **CodePipeline** and click on **Create Pipeline**. Configure the following settings:

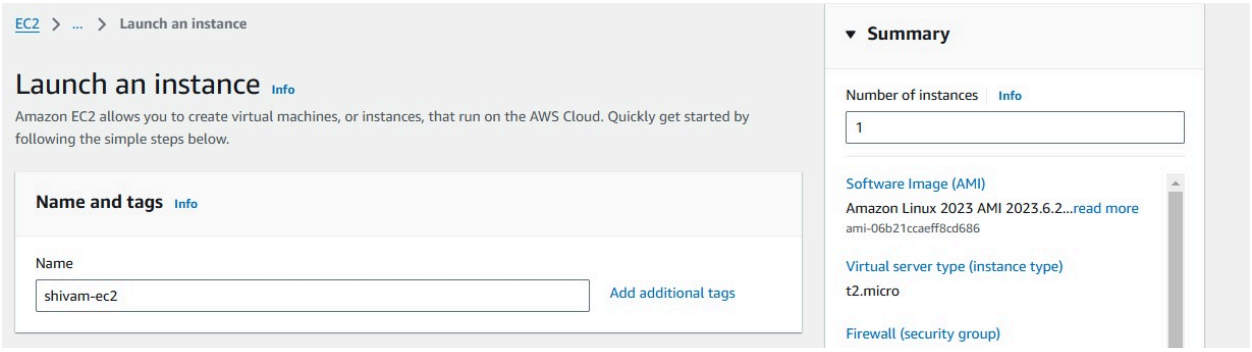
- **Pipeline Name**: Enter a name for your pipeline.
- **Service Role**: Choose to create a new role.
- **Source Provider**: Select Amazon S3.
- **Bucket**: Choose the S3 bucket where you uploaded webapp.zip.
- **S3 Object Key**: Specify webapp.zip.
- **Build Provider**: Select AWS CodeBuild.
- **Deploy Provider**: Choose Amazon S3 and enable file extraction before deployment.

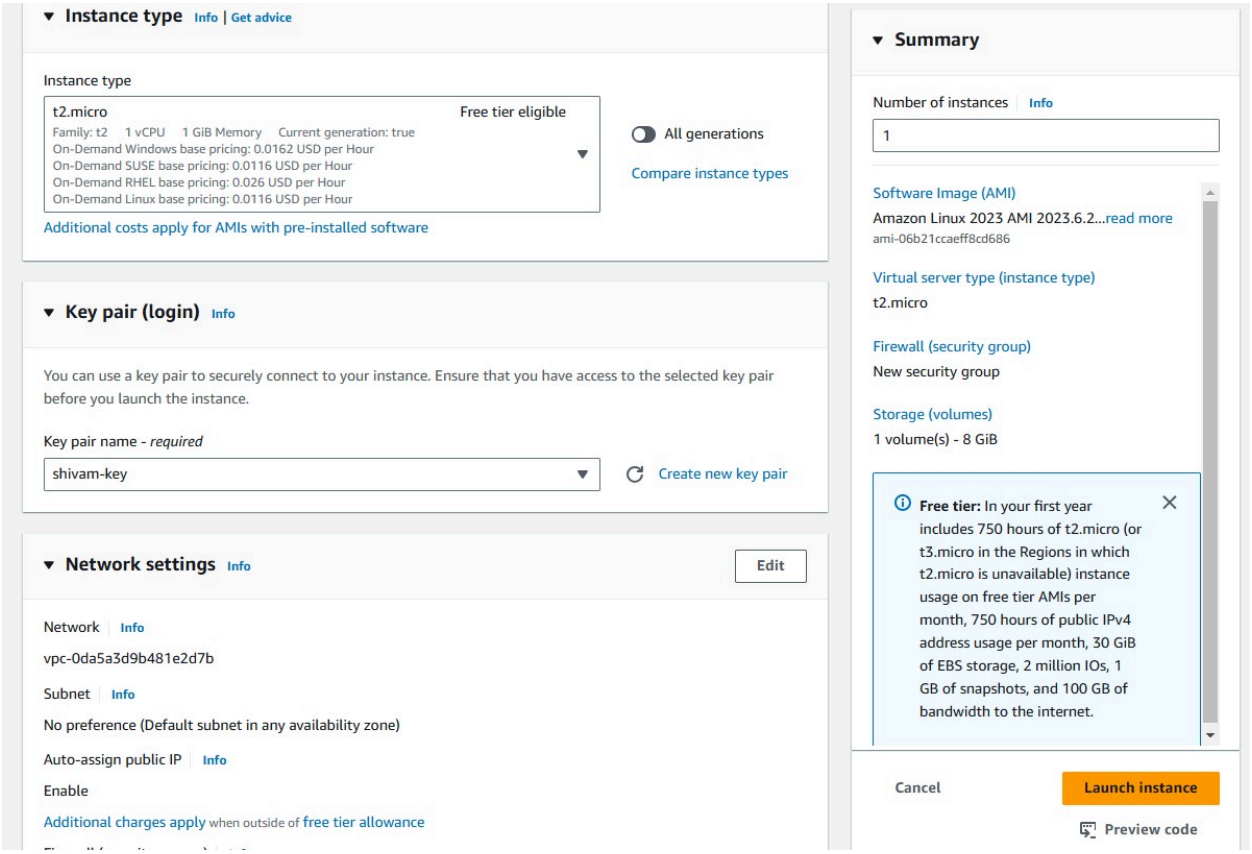




Step 6: Create an EC2 Instance

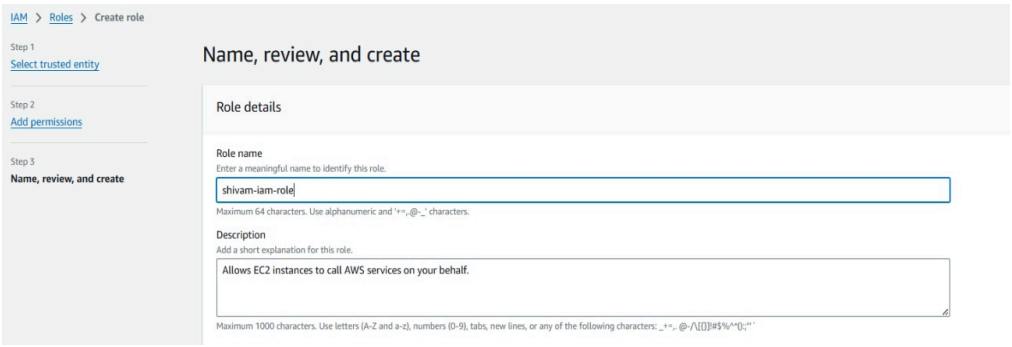
Click on **Launch Instance** and select an instance type, such as **t3.micro**. Next, create a new RSA key pair to enable SSH access to your instance. Make sure to configure the security settings to allow HTTP and HTTPS traffic from anywhere, ensuring that your instance can handle web traffic.

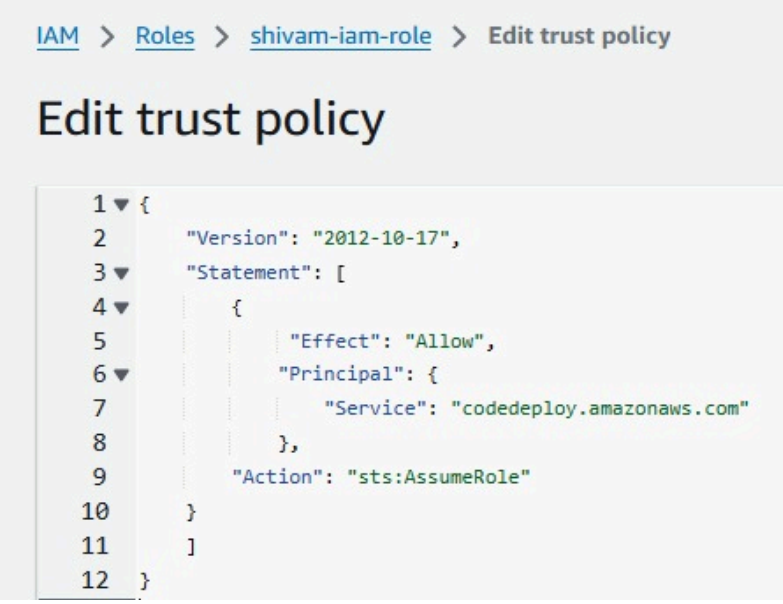




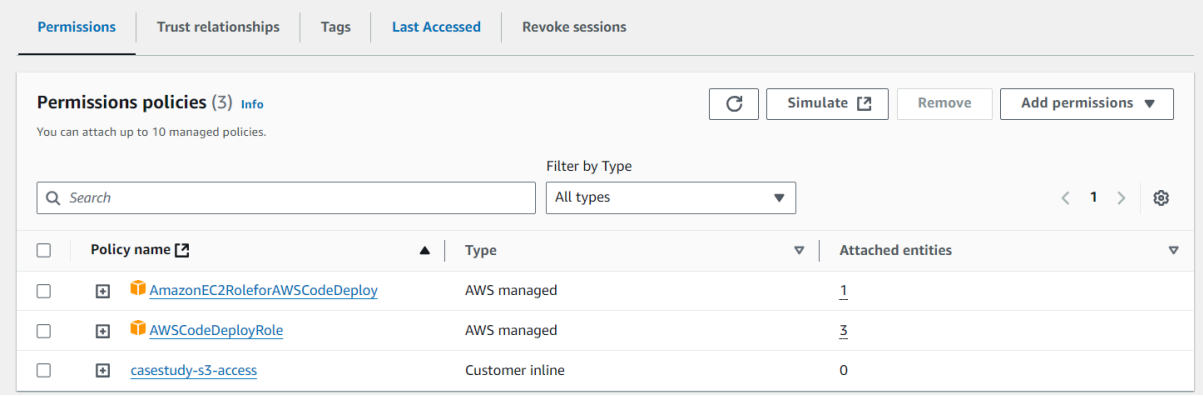
Step 7: Create an IAM Role for EC2 with CodeDeploy

Navigate to **IAM**, click on **Roles**, and create a new role for **EC2**. Attach the **AWSCodeDeployRole** and **AmazonEC2RoleforAWSCodeDeploy** policies to the role. Then, edit the trust policy to allow **CodeDeploy** to assume the role. Additionally, add an inline policy that grants the role access to S3 objects.





Created the IAM Role



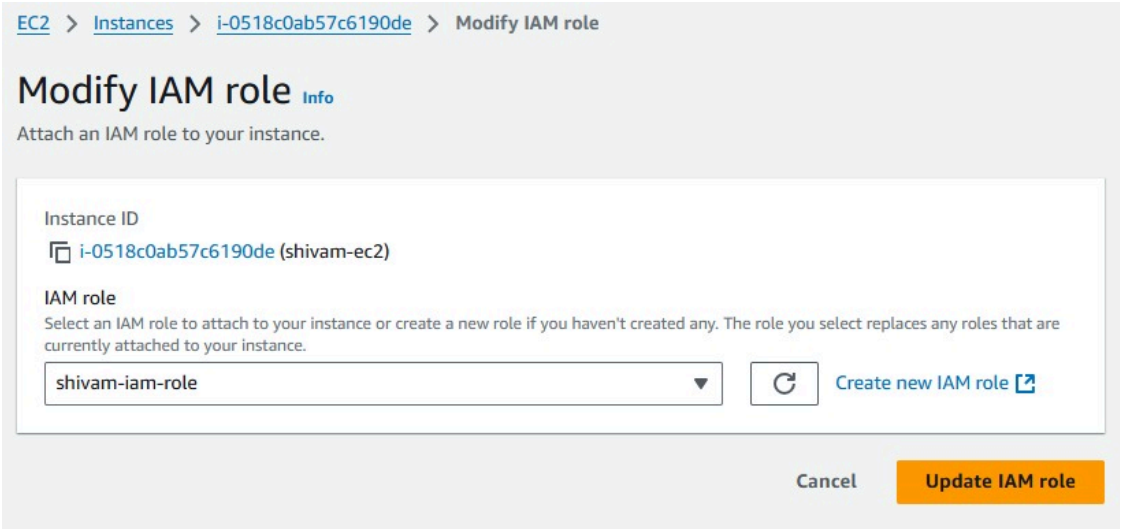
Step 8: Install CodeDeploy Agent and Nginx on EC2

SSH into your EC2 instance and run the following commands

- sudo yum install -y ruby
- cd /tmp
- wget https://aws-codedeploy-us-east-1.s3.us-east-1.amazonaws.com/latest/install
- chmod +x ./install
- sudo ./install auto
- sudo service codedeploy-agent start
- sudo yum install -y nginx
- sudo service nginx start

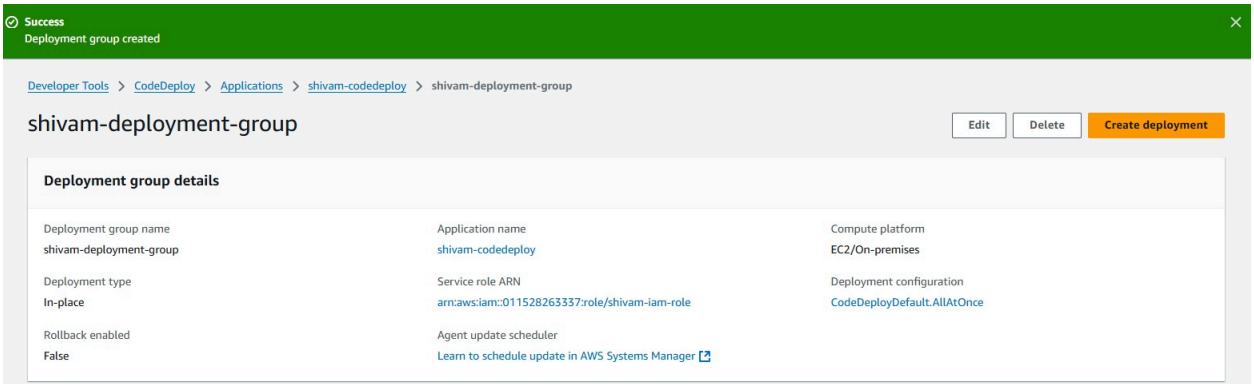
Step 9: Attach the IAM Role to the EC2 Instance

In the **EC2 Dashboard**, locate and select your instance. Click on **Actions**, then go to **Security** and choose **Modify IAM Role**. From there, attach the IAM role you previously created to the instance.



Step 10: Set Up AWS CodeDeploy

Open the **CodeDeploy Console** and create a new application. Next, create a deployment group by entering the ARN of the service role you created for CodeDeploy. For the environment configuration, select **EC2 instances** and provide the tag key-value pair to specify the instances you want to deploy to.



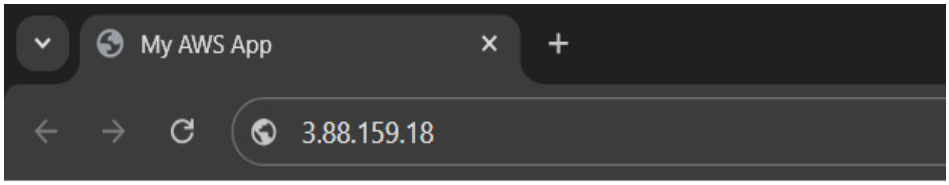
Step 11: Give S3 Access to CodeDeploy

Navigate to your S3 bucket and click on the **Permissions** tab. Check the bucket policy to ensure that it allows access for CodeDeploy. Make any necessary adjustments to the policy to grant CodeDeploy the required permissions.

```
{ "Version": "2012-10-17",
 "Statement": [
  { "Effect": "Allow",
   "Principal": {
    "Service": "codedeploy.amazonaws.com"
   },
   "Action": "s3:GetObject",
   "Resource": "arn:aws:s3:::shivam-advdevops/*"
  } ]
}
```

Step 12: Create a Deployment for Your Application

In the **Deployments** tab, click on **Create deployment**. Once the deployment is complete, verify that your application is accessible by using the public IPv4 address of your EC2 instance.



Best Practices and Guidelines

Ensure Unique Names for S3 Buckets:

- **Global Uniqueness:** S3 bucket names need to be unique across all AWS accounts, so always verify name availability before creating a new bucket.
- **Meaningful Names:** Choose clear, descriptive names that reflect the bucket's intended use
- **Versioning:** Incorporate timestamps or version numbers in bucket names to facilitate tracking of changes.

Regularly Monitor Your Deployments and Logs:

- **Use CloudWatch:** Monitor application performance and resource usage in real-time, and set up alerts and dashboards for critical metrics.
- **Log Reviews:** Periodically examine AWS CodeDeploy and CodePipeline logs to identify and resolve issues, improving the deployment process.
- **Enable CloudTrail:** Keep a detailed log of API calls to assist with security audits.
- **Set Up Alerts:** Create notifications for significant issues to ensure application stability.

Adopt Strong Security Practices:

- **Control Public Access:** Implement S3 bucket policies to limit public access, allowing only designated users or IP addresses.
- **Apply Least Privilege Principle:** Assign only the permissions necessary for tasks, ensuring that EC2 instances have restricted resource access.
- **Utilize Resource-Based Policies:** Control S3 bucket access based on specific conditions with these policies.
- **Conduct Regular IAM Audits:** Review and eliminate any unused IAM roles and policies periodically.
- **Enable Data Encryption:** Activate encryption for data in S3 and utilize AWS Key Management Service (KMS) for managing encryption keys.

Demonstration Preparation

Key Points:

- **Introduction to AWS Services:** Begin with a brief overview of AWS services such as CodePipeline, CodeBuild, CodeDeploy, S3, and EC2, emphasizing how they collaborate to streamline the deployment process.
- **Deployment Process Breakdown:** Highlight the key steps involved in deployment:
  - Create an S3 bucket and enable versioning to manage file updates.
  - Develop the web application and prepare the deployment package for AWS.
  - Configure AWS CodeBuild and CodePipeline to automate the deployment workflow.
  - Launch an EC2 instance and set it up for application deployment.
  - Set up IAM roles and permissions to ensure secure access.
  - Use AWS CodeDeploy to confirm the deployment was successful.
- **Security Considerations:** Underline the necessity of security practices, including the use of S3 bucket policies and restrictions on IAM roles to safeguard your deployment.
- **Ongoing Monitoring and Maintenance:** Discuss the importance of regular monitoring through CloudWatch and the need to review logs to proactively identify and resolve any issues.

Practice:

- **Rehearse the Presentation:** Go through your material multiple times to ensure you can present it clearly and confidently. Familiarize yourself with each part of the process to maintain a natural flow in your speech.
- **Manage Your Time:** Practice timing each section to make sure you stay within the allotted time during the demonstration.
- **Get Comfortable with the AWS Console:** Make sure you are proficient in navigating the AWS console and executing commands, so you can smoothly handle any live demonstrations or troubleshooting during your talk.

Questions:

- **Prepare for Common Inquiries:**
  - **What advantages does AWS offer for deployment?**
    AWS provides scalability, reliability, and seamless integration of its various services, enhancing the efficiency and flexibility of deployment.

- **How do you manage errors that occur during deployment?**
  Effective monitoring of logs and setting up automated alerts are vital for quickly identifying and addressing errors.
- **What key security measures should be implemented for this deployment?**
  Important security practices include enforcing IAM role restrictions, setting clear bucket policies, and ensuring data encryption.
- **What distinguishes CodeDeploy from CodePipeline?**
  CodeDeploy is focused on the application deployment itself, while CodePipeline orchestrates the entire workflow from source code to production deployment.

Conclusion:

This case study explored the automation of cloud deployment using AWS services, focusing on AWS CodePipeline, EC2, and S3 for building and deploying a simple web application. We illustrated how these tools integrate to facilitate a scalable and efficient deployment process, supporting continuous integration and deployment through AWS CodeBuild and CodeDeploy. We also emphasized the importance of security measures, such as unique naming conventions for S3 buckets, strict IAM role restrictions, and robust monitoring practices to ensure application integrity and performance. By leveraging the AWS ecosystem, organizations can streamline development processes, boost productivity, and deliver high-quality applications rapidly, helping them remain competitive in the fast-evolving digital landscape.