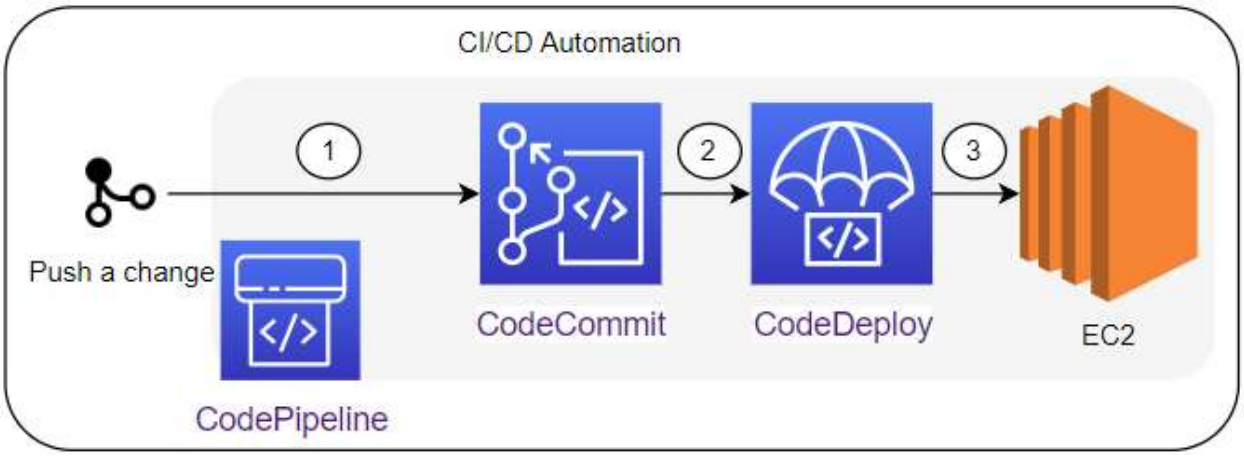
Create a CI/CD pipeline on AWS

The ***CodePipeline*** is triggered when you push a code change to the ***CodeCommit*** repository. After that, the pipeline deploys your changes to an EC2 instance using ***CodeDeploy*** as the deployment service.

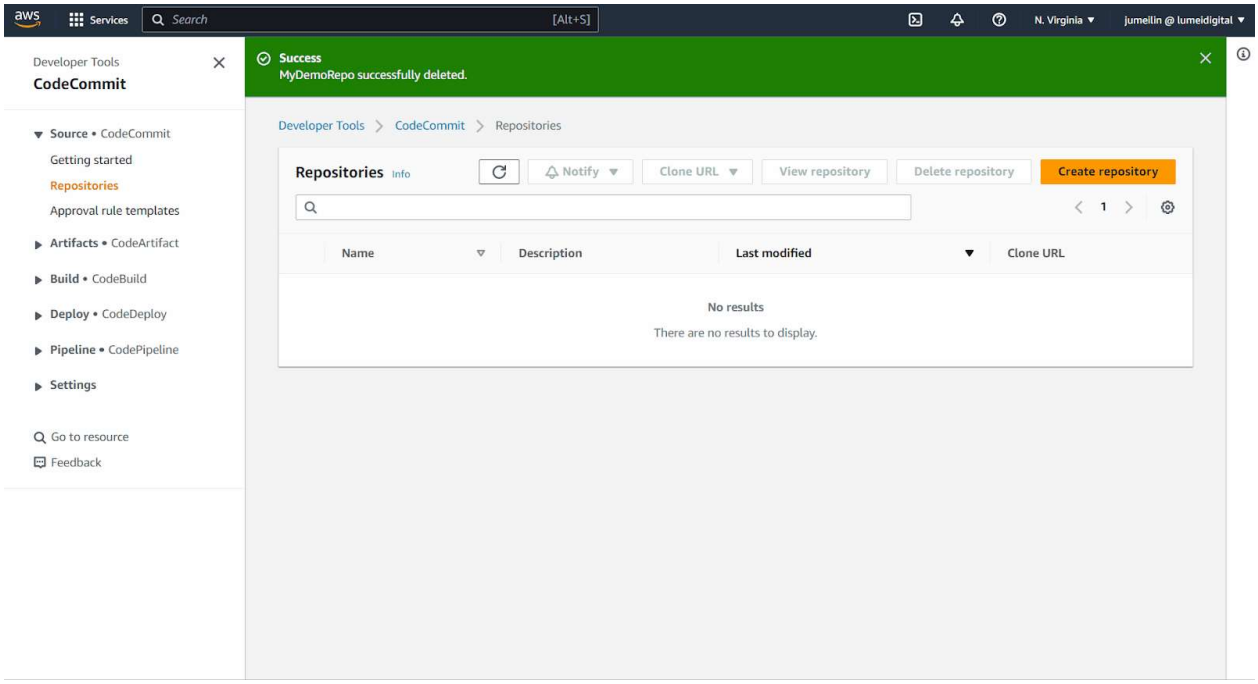


The pipeline has two stages:

- A source stage (**Source**) for your CodeCommit source action.
- A deployment stage (**Deploy**) for your CodeDeploy deployment action.

Step 1: Create a CodeCommit repository

1. Setup for HTTPS users using Git credentials
2. Open the CodeCommit console
3. Choose the AWS Region
4. On the **Repositories** page, choose **Create repository**.
5. On the **Create repository** page, in the **Repository name,** ***MyDemoRepo***, enter a name for your repository
6. Choose **Create**



Step 2: Add sample code to your CodeCommit repository

Step 3: Create an ubuntu22 Linux instance and install the CodeDeploy agent

Create shell scripts
Vi install.sh

```
#!/bin/bash

# This installs the CodeDeploy agent and its prerequisites on Ubuntu 22.04.
```

```
sudo apt-get update
```

```
sudo apt-get install ruby-full ruby-webrick
```
wget -y

cd /tmp

wget
https://aws-codedeploy-us-east-1.s3.us-east-1.amazonaws.com/releases/codedeploy-agent_1.3.2-1902_all.deb

```
mkdir codedeploy-agent_1.3.2-1902_ubuntu22
```

```
dpkg-deb -R codedeploy-agent_1.3.2-1902_all.deb
codedeploy-agent_1.3.2-1902_ubuntu22
```

```
sed 's/Depends:.*/Depends:ruby3.0/' -i
./codedeploy-agent_1.3.2-1902_ubuntu22/DEBIAN/control
```

```
dpkg-deb -b codedeploy-agent_1.3.2-1902_ubuntu22/
```

```
sudo dpkg -i codedeploy-agent_1.3.2-1902_ubuntu22.deb
```

```
systemctl list-units --type=service | grep codedeploy
```

```
sudo service codedeploy-agent status
```

bash install.sh

To create an instance role:

1. Open the IAM console at https://console.aws.amazon.com/iam/).
2. From the console dashboard, choose **Roles**.
3. Choose **Create role**.
4. Under **Select type of trusted entity**, select **AWS service**. Under **Choose a use case**, select **EC2**. Under **Select your use case**, choose **EC2**. Choose **Next: Permissions**.
5. Search for and select the policy named `AmazonEC2RoleforAWSCodeDeploy`.
6. Search for and select the policy named `AmazonSSMManagedInstanceCore`. Choose **Next: Tags**.

7. Choose **Next: Review**. Enter a name for the role (for example, `EC2InstanceRole`)

To launch an instance

1. Open the Amazon EC2 console at
2. From the side navigation, choose **Instances**, and select **Launch instances** from the top of the page.

3. In **Name**, enter `MyCodePipelineDemo`. This assigns the instance a tag **Key** of `Name` and a tag **Value** of `MyCodePipelineDemo`. Later, you create a CodeDeploy application that deploys the sample application to this instance. CodeDeploy selects instances to deploy based on the tags.
4. Under **Application and OS Images (Amazon Machine Image)**, locate the **Amazon Linux** AMI option with the AWS logo, and make sure it is selected. (This AMI is described as the Amazon Linux 2 AMI (HVM) and is labeled “Free tier eligible”.)
5. Under **Instance type**, choose the free tier eligible `t2.micro` type as the hardware configuration for your instance.
6. Under **Key pair (login)**, choose a key pair or create one. You can also choose **Proceed without a key pair**.

7. Under **Network settings**, do the following. In **Auto-assign Public IP**, make sure the status is **Enable**.
   - Next to **Assign a security group**, choose **Create a new security group**.
   - In the row for **SSH**, under **Source type**, choose **My IP**.
   - Choose **Add security group**, choose **HTTP**, and then under **Source type**, choose **My IP**.

8. Expand **Advanced details**. In **IAM instance profile**, choose the IAM role you created in the previous procedure (for example, `EC2InstanceRole`)

9. Under **Summary**, under **Number of instances**, enter `1`..

10. Choose **Launch instance**.

Step 4: Create an application in CodeDeploy

Step 4: Create an application in CodeDeploy

In CodeDeploy, an *application* is a resource that contains the software application you want to deploy.

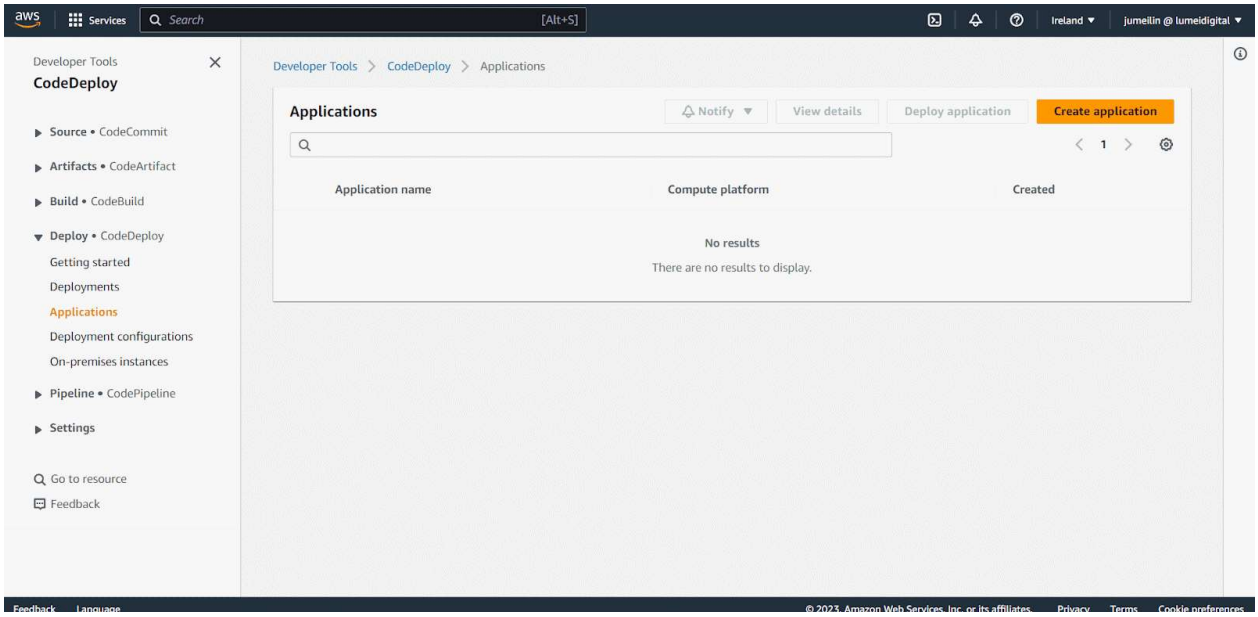
To create a CodeDeploy service role:

1. Open the IAM console at https://console.aws.amazon.com/iam/).
2. From the console dashboard, choose **Roles**.
3. Choose **Create role**.
4. Under **Select trusted entity**, choose **AWS service**. Under **Use case**, choose **CodeDeploy**. Choose **CodeDeploy** from the options listed. Choose **Next**. The `AWSCodeDeployRole` managed policy is already attached to the role.
5. Choose **Next**.

6. Enter a name for the role (for example, `CodeDeployRole`), and then choose **Create role**.

To create an application in CodeDeploy

1. Open the CodeDeploy console at https://console.aws.amazon.com/codedeploy.
2. If the **Applications** page does not appear, on the menu, choose **Applications**.
3. Choose **Create application**.
4. In **Application name**, enter `MyDemoApplication`.
5. In **Compute Platform**, choose **EC2/On-premises**.
6. Choose **Create application**.
7. To create a deployment group in CodeDeploy.



Create application

A *deployment group* is a resource that defines deployment-related settings like which instances to deploy to and how fast to deploy them.

1. On the page that displays your application, choose **Create deployment group**.
2. In **Deployment group name**, enter `MyDemoDeploymentGroup`.
3. In **Service role**, choose the service role you created earlier (for example, `CodeDeployRole`).

4. Under **Deployment type**, choose **In-place**.
5. Under **Environment configuration**, choose **Amazon EC2 Instances**. In the **Key** field, enter `Name`. In the **Value** field, enter the name you used to tag the instance (for example, `MyCodePipelineDemo`).
6. Under **Agent configuration with AWS Systems Manager**, choose **Now and schedule updates**. This installs the agent on the instance. The Linux instance is already configured with the SSM agent and will now be updated with the CodeDeploy agent.
7. Under **Deployment configuration**, choose `CodeDeployDefault.OneAtaTime`.
8. Under **Load Balancer**, make sure **Enable load balancing** is not selected. You do not need to set up a load balancer or choose a target group for this example.
9. Choose **Create deployment group**.

Step 5: Create your first pipeline in CodePipeline

You're now ready to create and run your first pipeline. In this step, you create a pipeline that runs automatically when code is pushed to your CodeCommit repository.

To create a CodePipeline pipeline

1. Sign in to the AWS Management Console and open the CodePipeline console at

http://console.aws.amazon.com/codesuite/codepipeline/home.

2. Open the CodePipeline console at https://console.aws.amazon.com/codepipeline/.
3. Choose **Create pipeline**.
4. In **Step 1: Choose pipeline settings**, in **Pipeline name**, enter `MyFirstPipeline`.
5. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
6. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
7. In **Step 2: Add source stage**, in **Source provider**, choose **CodeCommit**. In **Repository name**, choose the name of the CodeCommit repository you created in Step 1: Create a CodeCommit repository. In **Branch name**, choose `main`, and then choose **Next step**.

8. After you select the repository name and branch, a message displays the Amazon CloudWatch Events rule to be created for this pipeline.
9. Under **Change detection options**, leave the defaults. This allows CodePipeline to use Amazon CloudWatch Events to detect changes in your source repository.
10. Choose **Next**.
11. In **Step 3: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.
12. In **Step 4: Add deploy stage**, in **Deploy provider**, choose **CodeDeploy**. In **Application name**, choose `MyDemoApplication`. In **Deployment group**, choose `MyDemoDeploymentGroup`, and then choose **Next step**.
13. In **Step 5: Review**, review the information, and then choose **Create pipeline**.

```
version: 0.2

phases:
  install:
    runtime-versions:
      nodejs: 16
    commands:
      - npm install -g typescript
      - npm install
  pre_build:
    commands:
      - echo Installing source NPM dependencies...
  build:
    commands:
      - echo Build started on `date`
```

```
      - tsc
      - npm prune --production
    post_build:
      commands:
        - echo Build completed on `date`
artifacts:
  type: zip
  files:
    - package.json
    - package-lock.json
    - "build/**/*"
    - .ebextensions/**/*
```