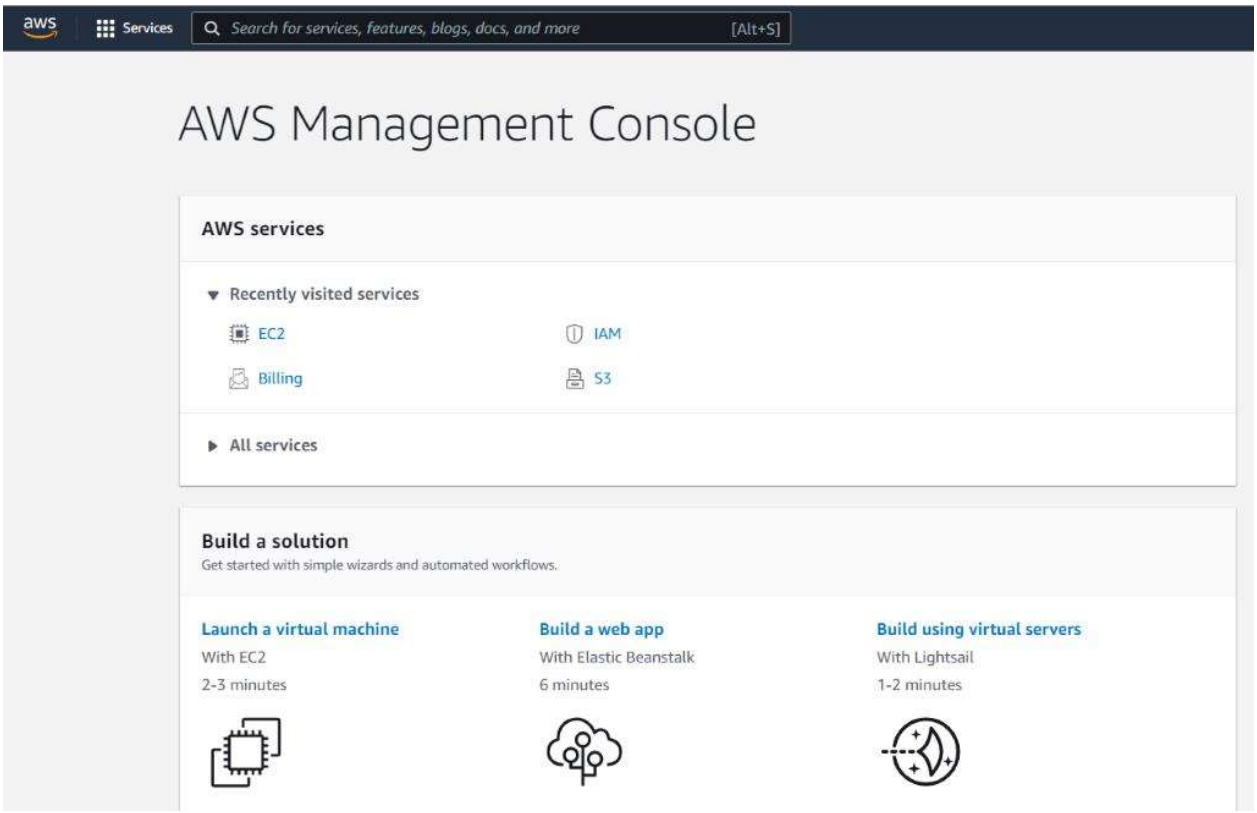
Deploy GitHub Project on Amazon EC2 Using GitHub Actions and AWS Code Deploy

Prerequisite

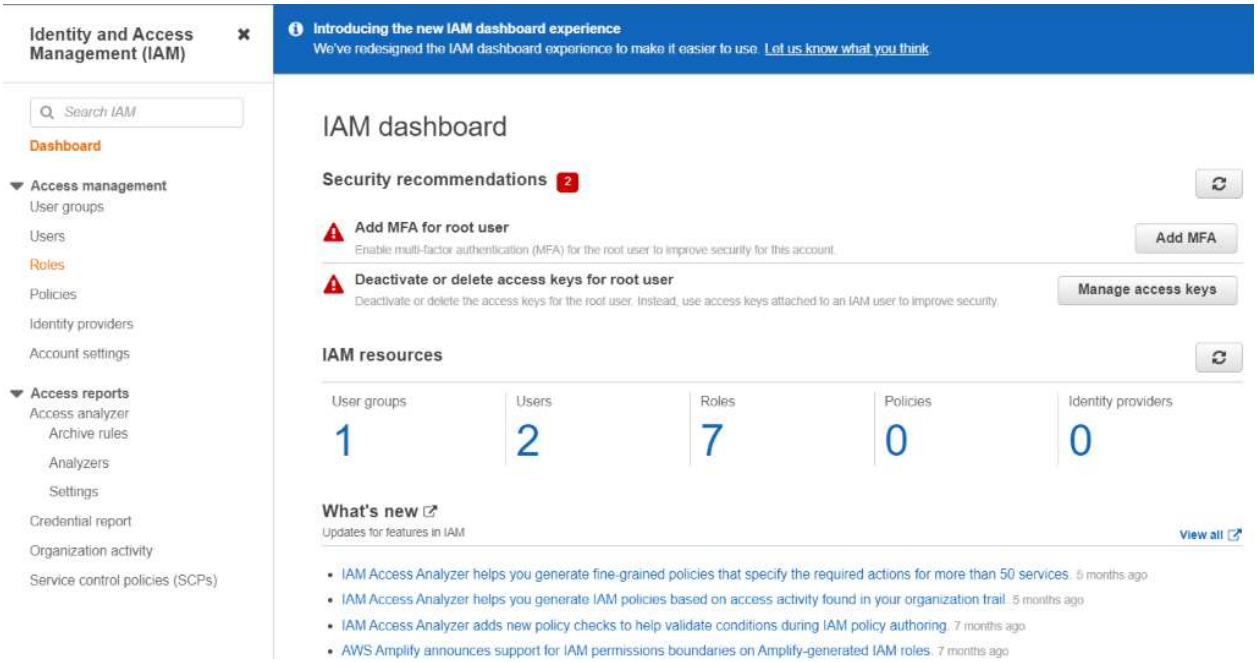
- Create a GitHub account
- Create an AWS Account

Create IAM Role for EC2 and Code Deploy

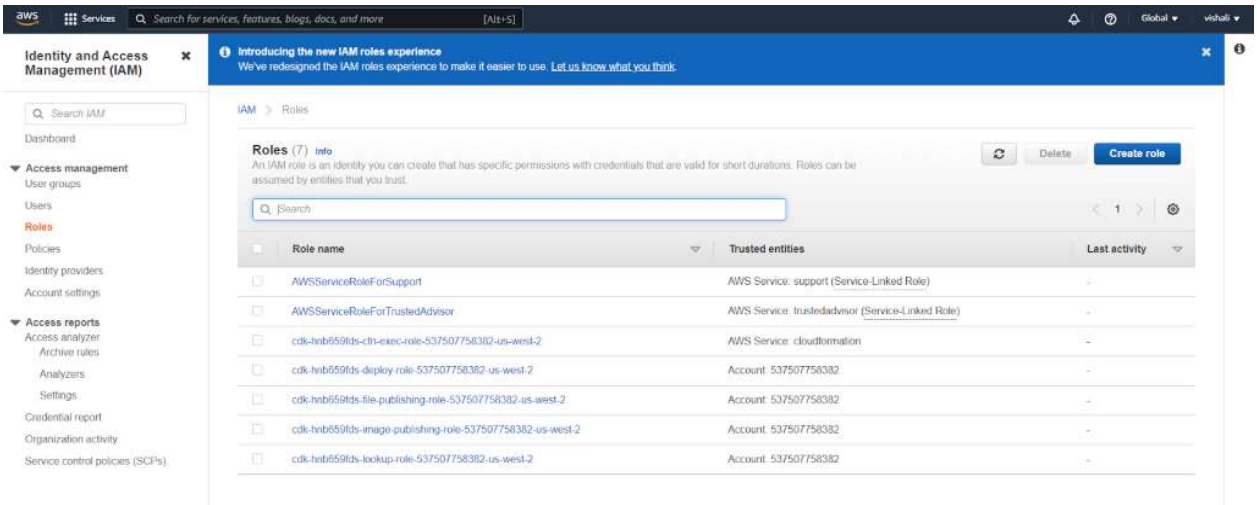
1. Open AWS Management Console and select IAM service



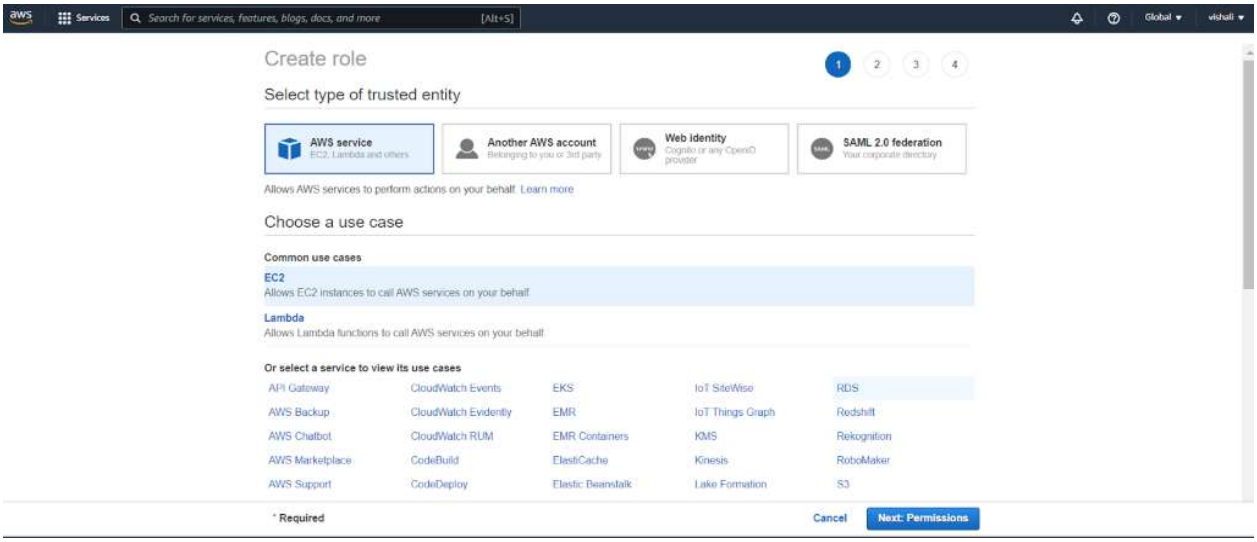
2. Select Roles from Access management



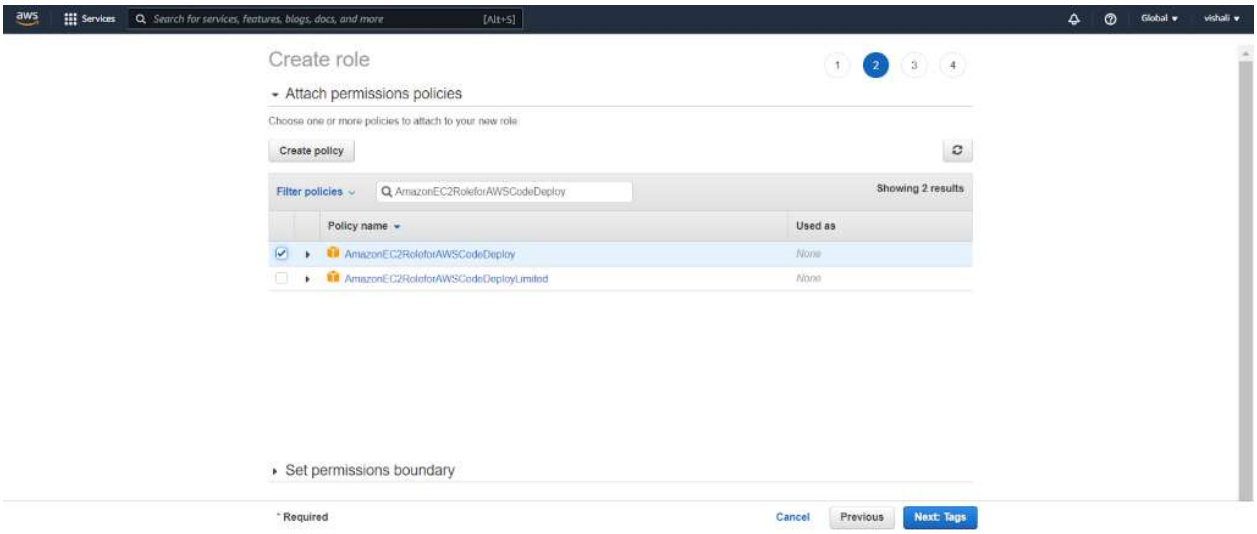
3. Select Create Role



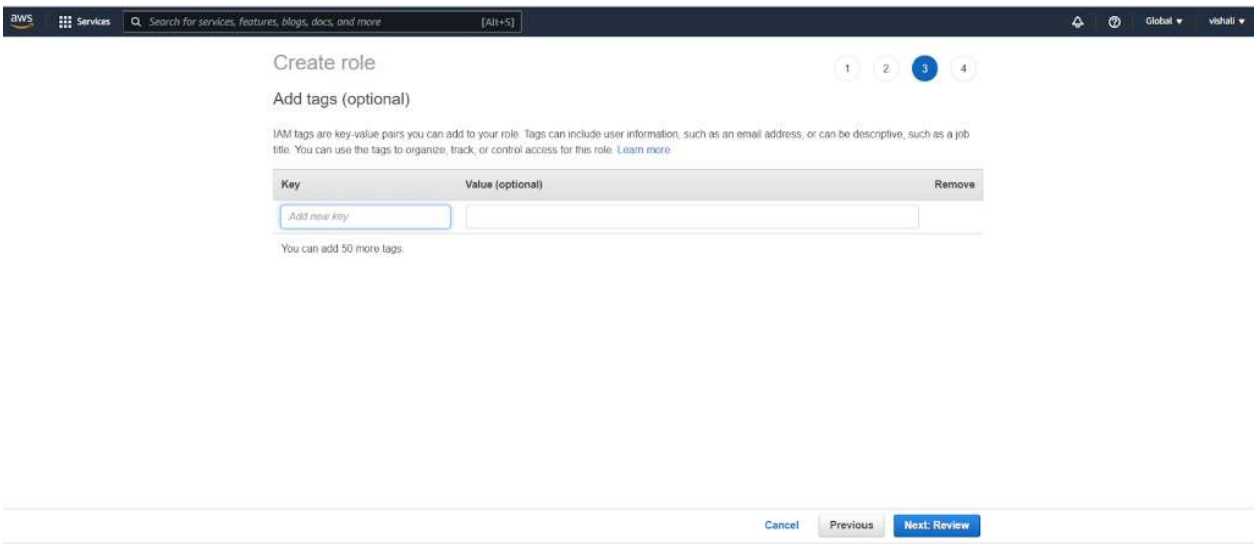
4. Create a role for EC2 Instance. Select AWS Service as *trusted entity* and EC2 as *usecase*, click on *Next:Permissions*.



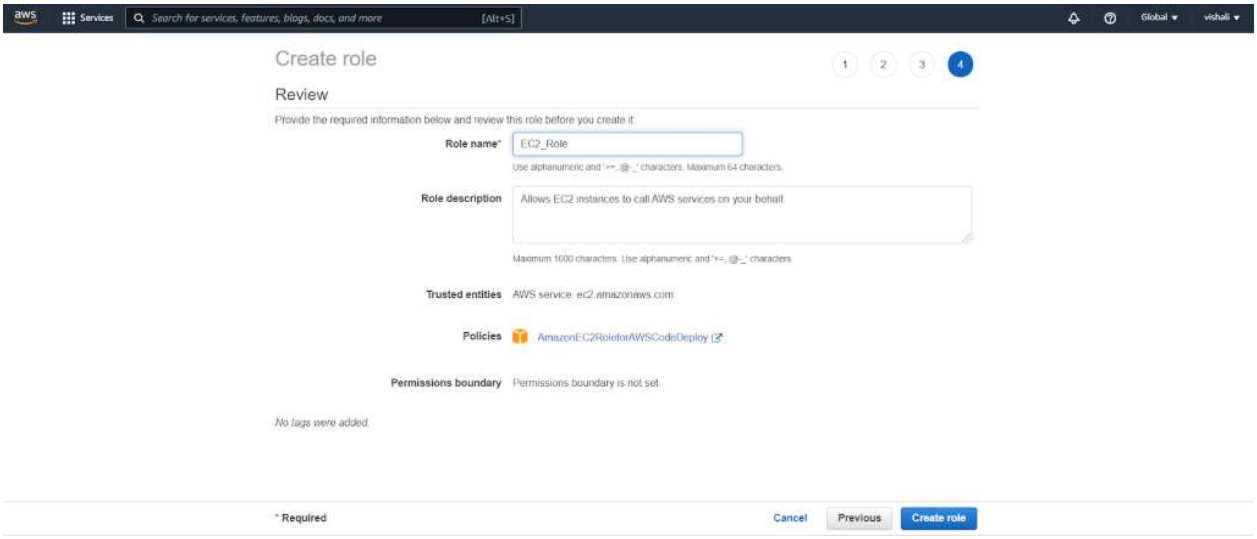
5. On the Permissions page, select AmazonEC2RoleforAWSCodeDeploy Policy and Click on *Next:Tags*



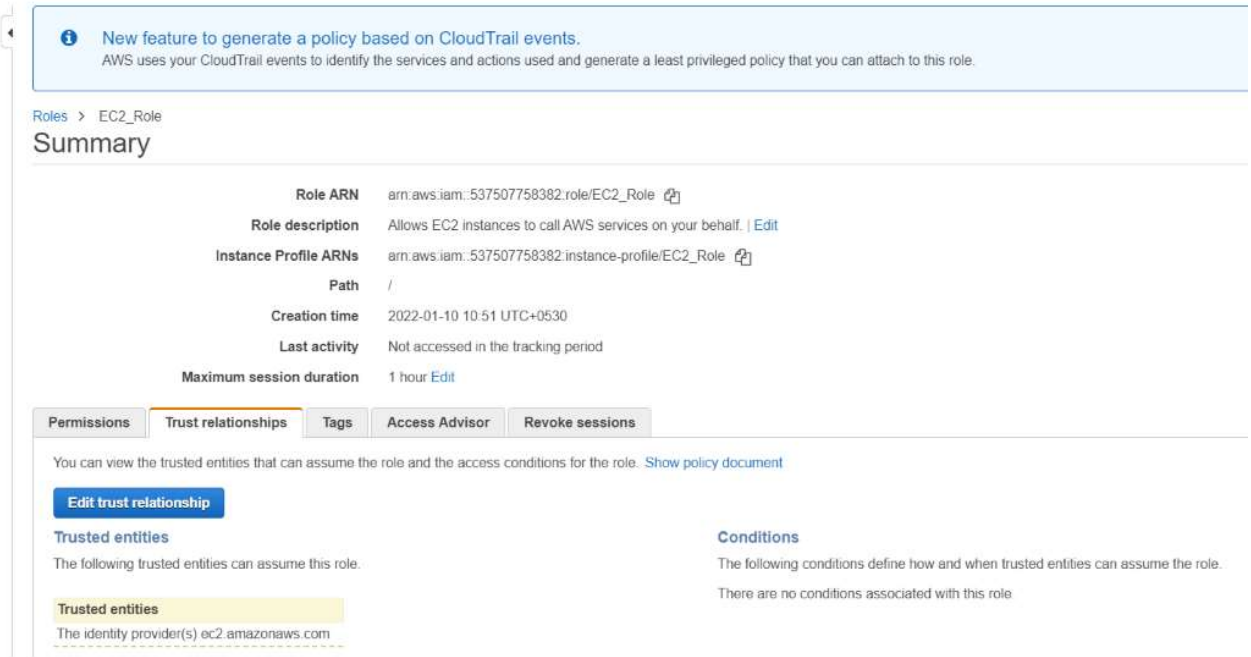
6. Ignore the tags and click *Next:Review*.



7. Provide the role name as EC2_Role on the review page and click create role.



8. Open the EC2_Role and go to *Trust Relationships*.



9. Click on *Edit Trust Relationship* and paste below policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Edit Trust Relationship

You can customize trust relationships by editing the following access control policy document.

Policy Document

```
{
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "ec2.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }
]
}
```

Cancel | Update Trust Policy

10. Now we will create a role for CodeDeploy. Open [AWS Management Console] and select IAM service.Select Roles from Access management and click on Create Role. Select AWS Service as trusted entity and EC2 as usecase, click on Next:Permissions.

Create role

1 2 3 4

Select type of trusted entity

AWS service EC2, Lambda and others

Another AWS account Belonging to you or 3rd party

Web identity Cognito or any OpenID provider

SAML 2.0 federation Your corporate directory

Allows AWS services to perform actions on your behalf. Learn more

Choose a use case

Common use cases

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

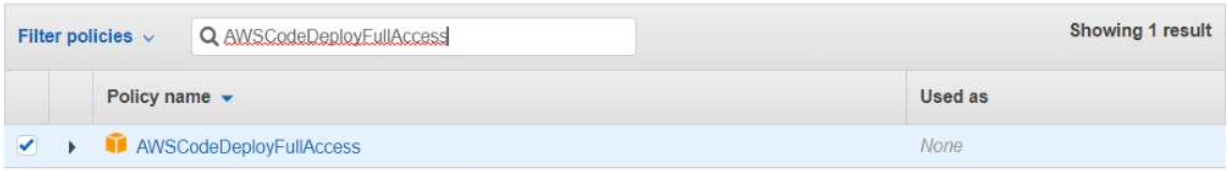
Or select a service to view its use cases

API Gateway	CloudWatch Events	EKS	IoT SiteWise	RDS
AWS Backup	CloudWatch Evidently	EMR	IoT Things Graph	Redshift
AWS Chatbot	CloudWatch RUM	EMR Containers	KMS	Rekognition
AWS Marketplace	CodeBuild	ElastiCache	Kinesis	RoboMaker
AWS Support	CodeDeploy	Elastic Beanstalk	Lake Formation	S3

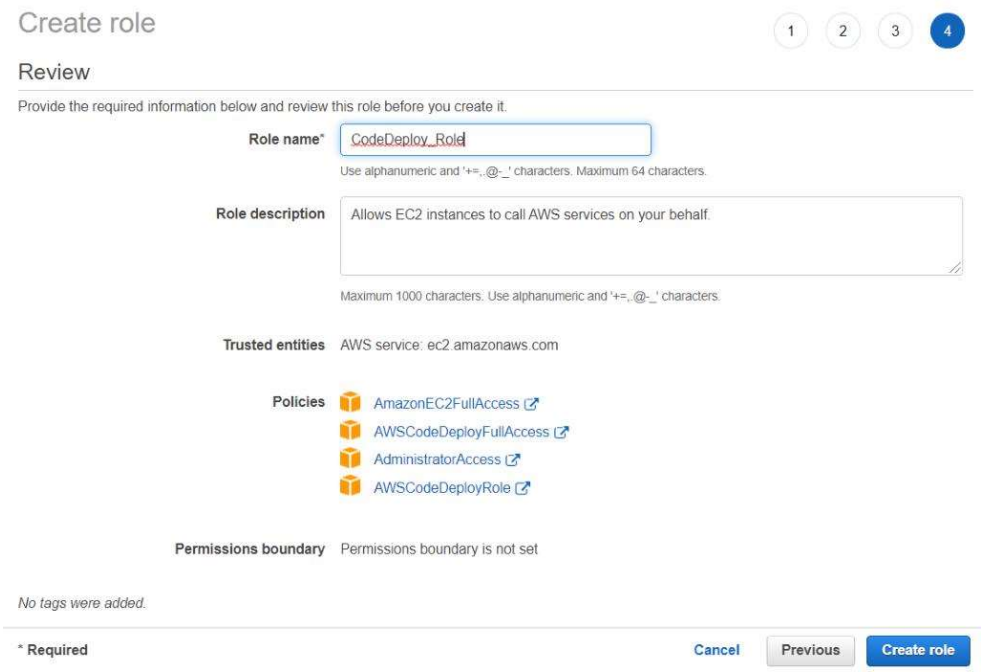
* Required

Cancel | Next: Permissions

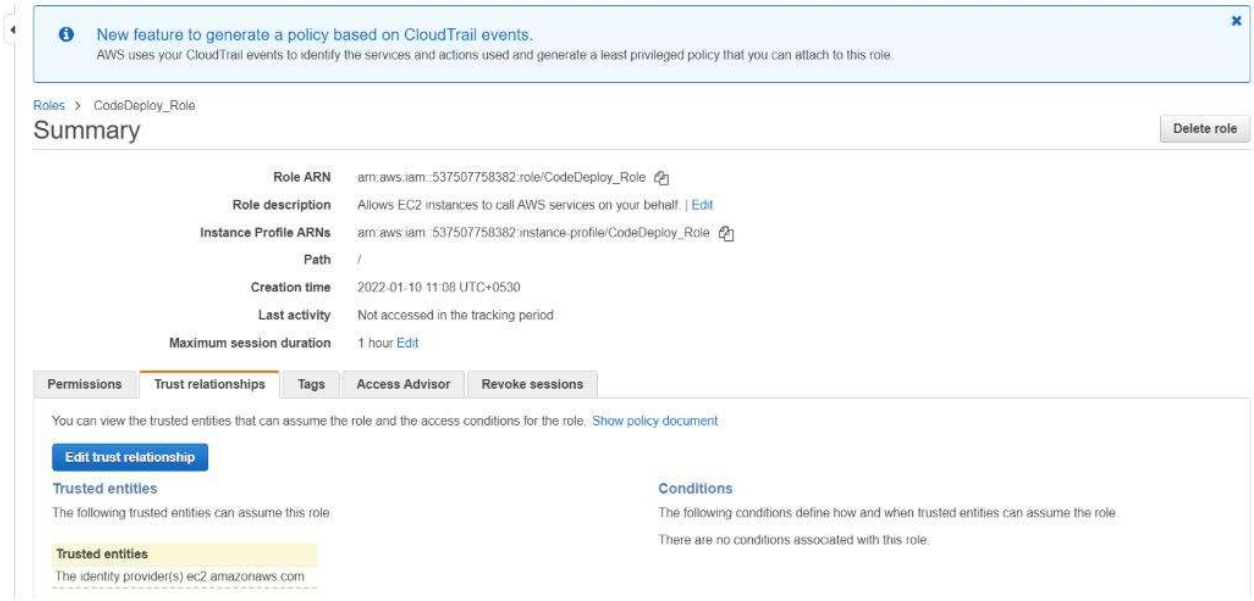
11. On the Permissions page, select the below policy and Click on *Next:Tags*.
    AmazonEC2FullAccess, AWSCodeDeployFullAccess, AdministratorAccess, AWSCodeDeployRole



12. Tags can be ignored, click on *Next:Review*.
13. Provide the role name as CodeDeploy_Role on the review page.



14. Once CodeDeploy Role is created, Open the CodeDeploy_Role.



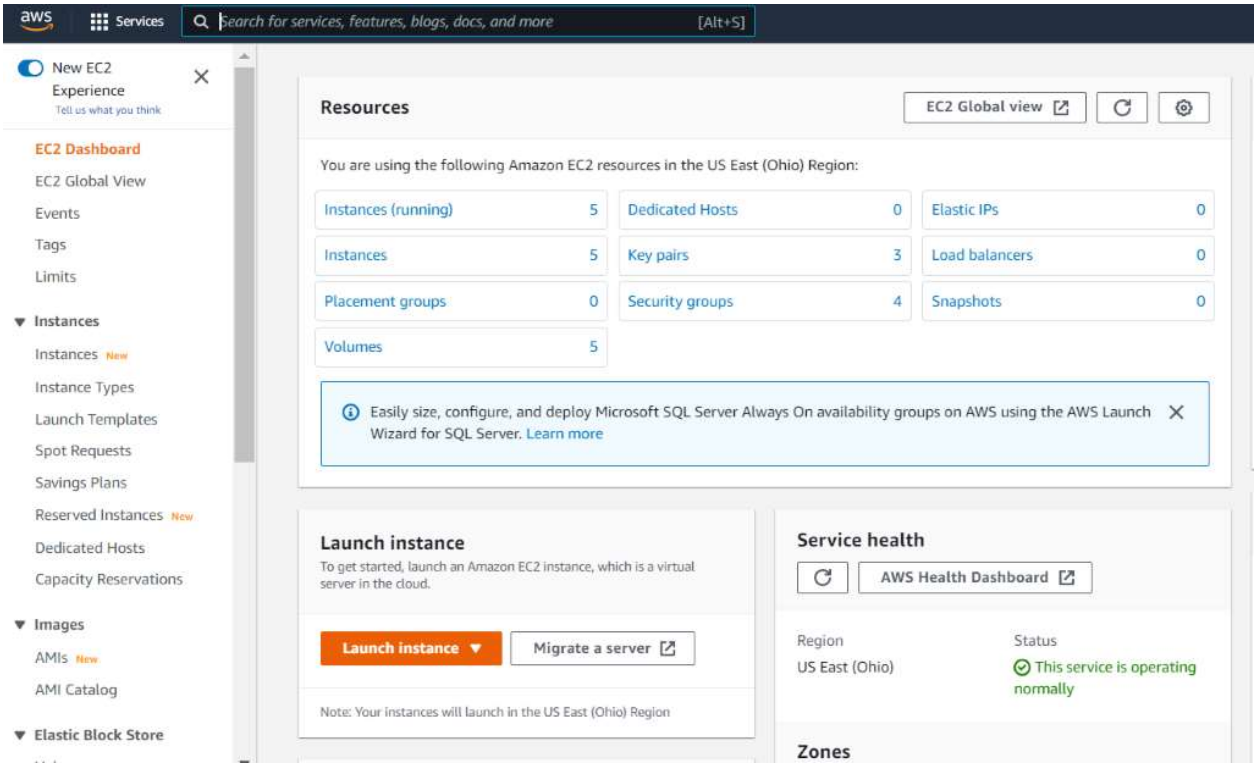
15. Go to *Trust Relationships* then *Edit Trust Relationship* and use below policy

```
{
 "Version": "2012-10-17",
 "Statement": [
```

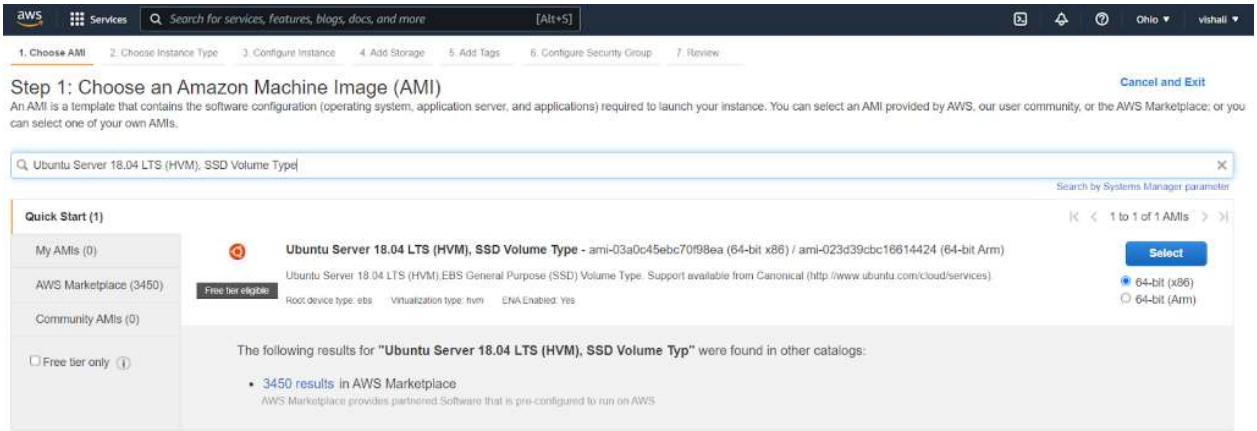
```
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "codedeploy.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Create EC2 Instance

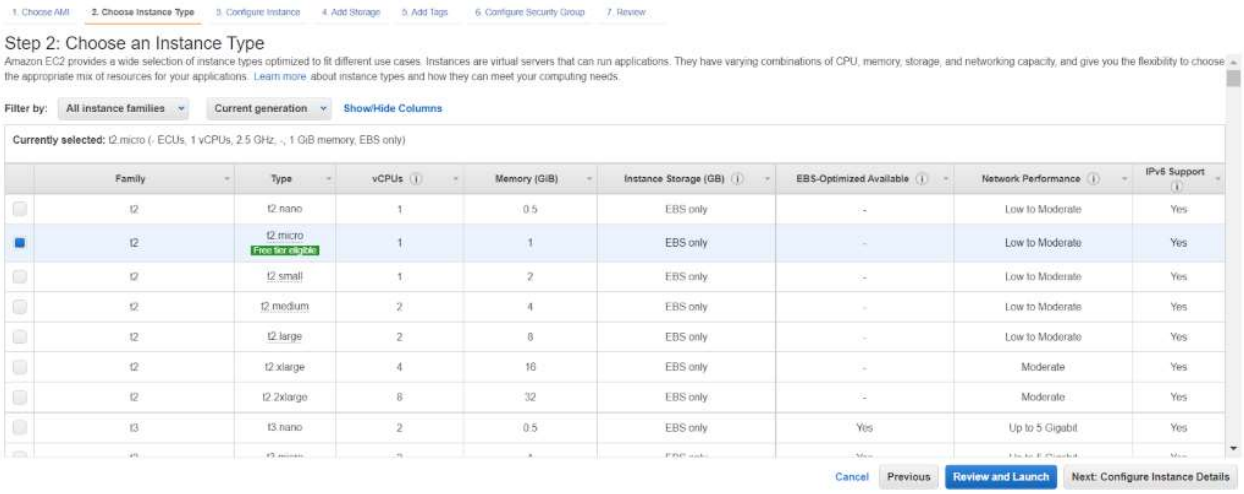
16. To create an EC2 instance, Open [AWS Management Console]() and select EC2 service. Go to EC2 Dashboard and click on Launch Instance.



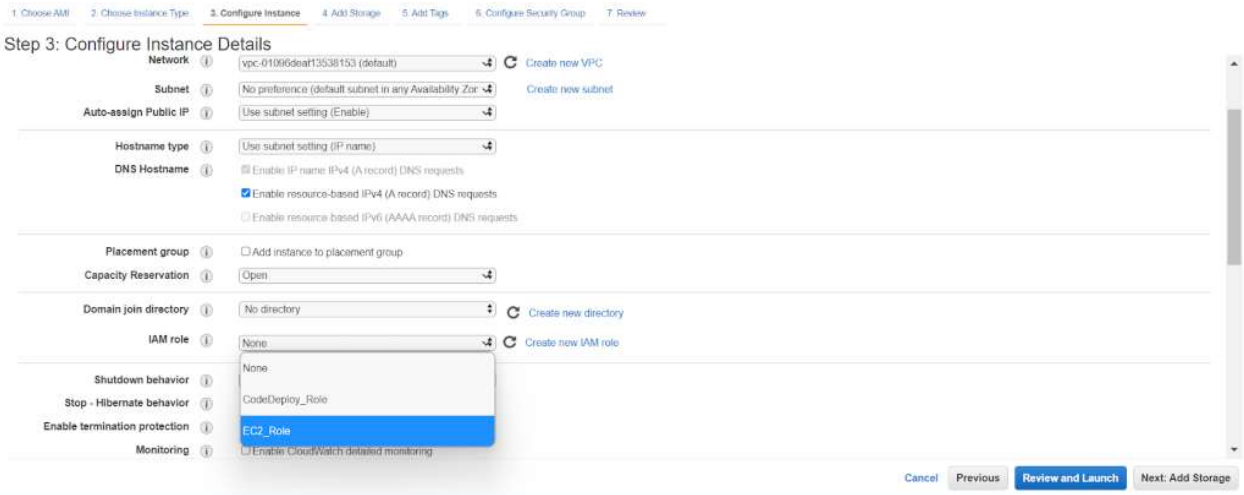
17. Choose an Amazon Machine Image(AMI)
    In the free tier option,
    Ubuntu Server 18.04 LTS (HVM), SSD Volume Type — ami-0a313d6098716f372 (64-bit x86)



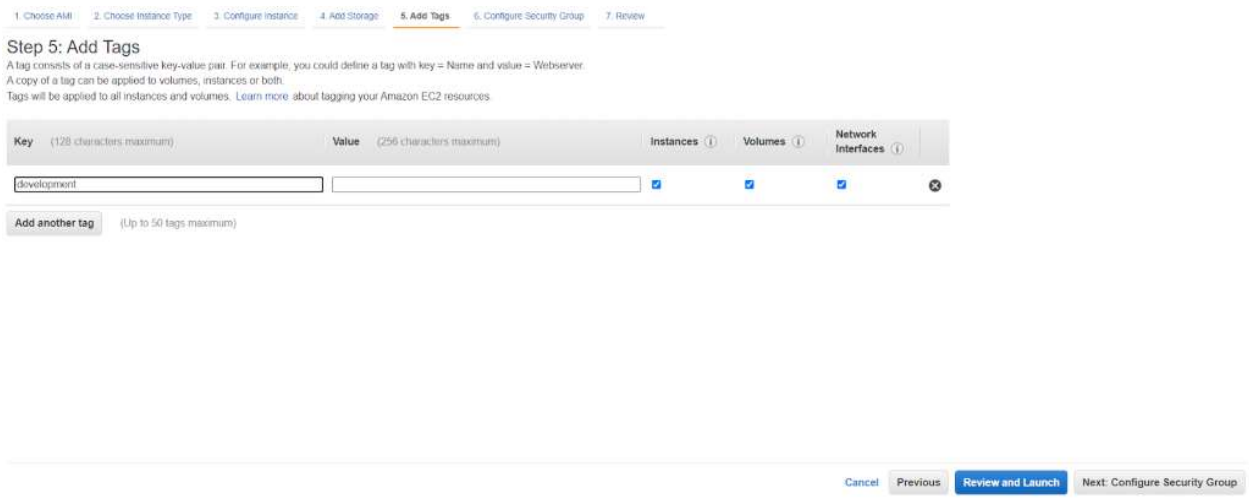
18. Select t2.micro in *Choose Instance Type* page and proceed to *Configure Instance* page.



19. To establish the connection between EC2 instance and codeDeploy, Select EC2_Role, which we created before.

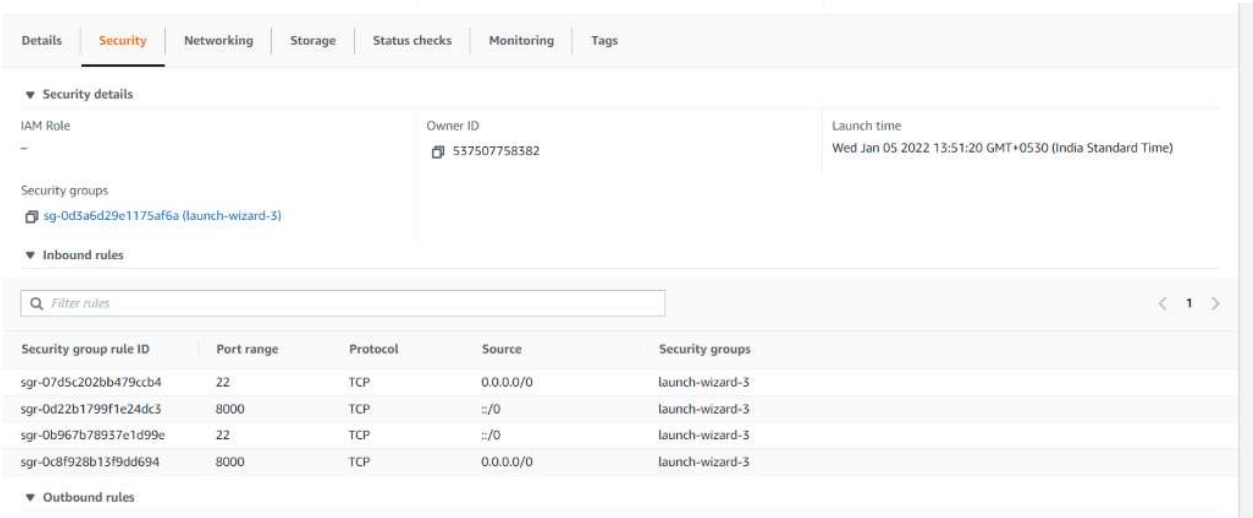


20. On the *Tag page*, add a tag called development. The tag will require creating a *codeDeploy* service.

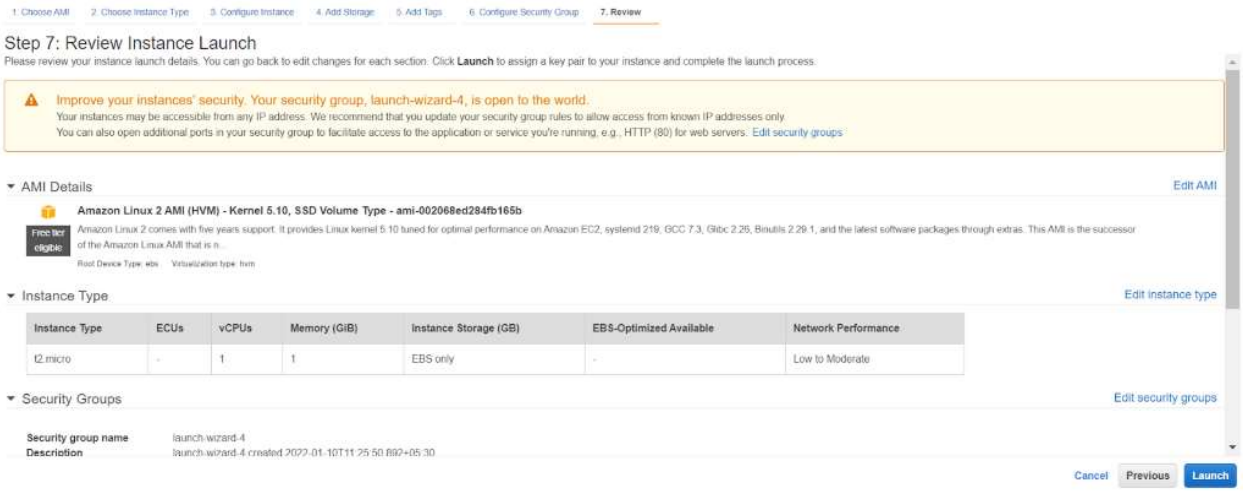


21. In the Configure *Security Group* page *Add Rule* called All traffic, select *source* called anywhere. This rule will enable you to connect the Instance from anywhere. NOTE - This is not advisable in the Production environment. Add the following rules:

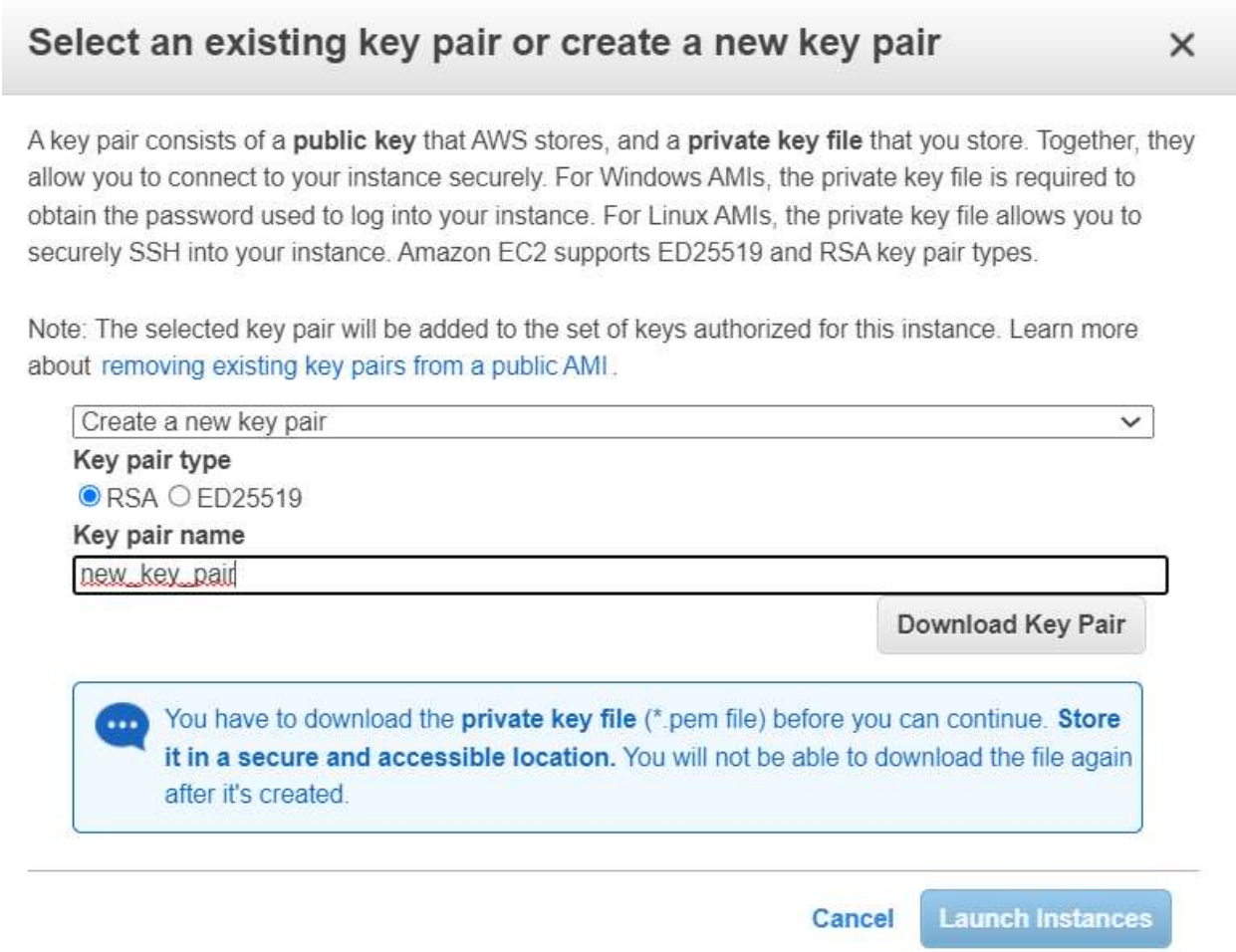
Type	Port Range	Description.
SSH	22	Port for SSH'ing into your server
HTTP	80	Port for HTTP requests to your web server
HTTPS	443	Port for HTTPS requests to your web server
Custom TCP	8000	Port which Django will run
Custom TCP	5432	Port at which to connect to PostgreSQL



22. Select the review page, then Launch the Instance. Wait for a few minutes to start the EC2 Instance.



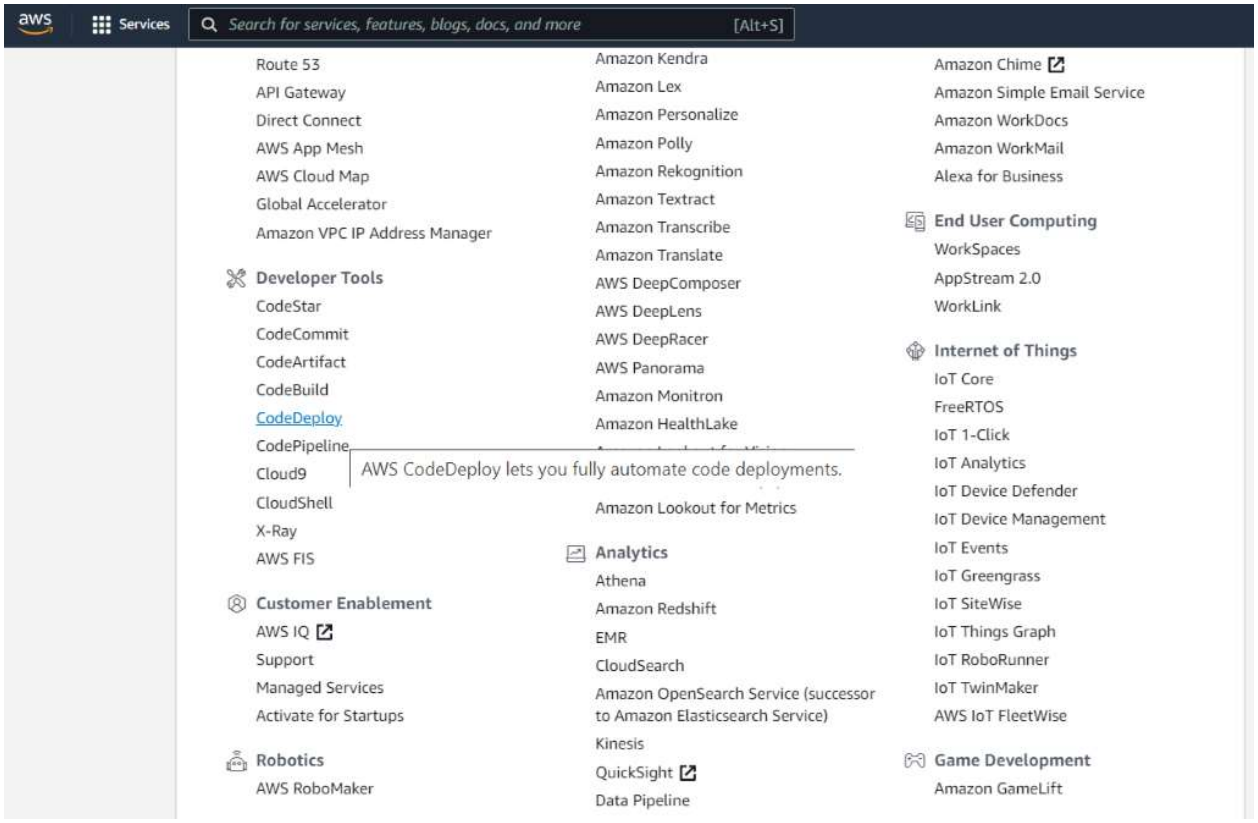
23. If you want to access the Instance (ssh) from your local system, create a new Key Pair and download the key.



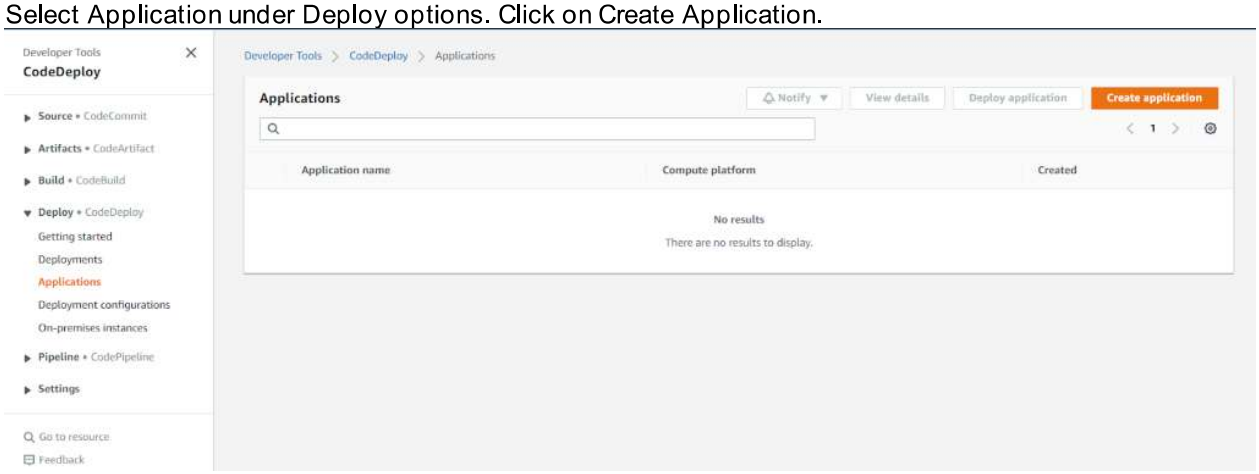
CodeDeploy Service Configuration

AWS CodeDeploy Service will automate the GitHub application deployment to EC2.

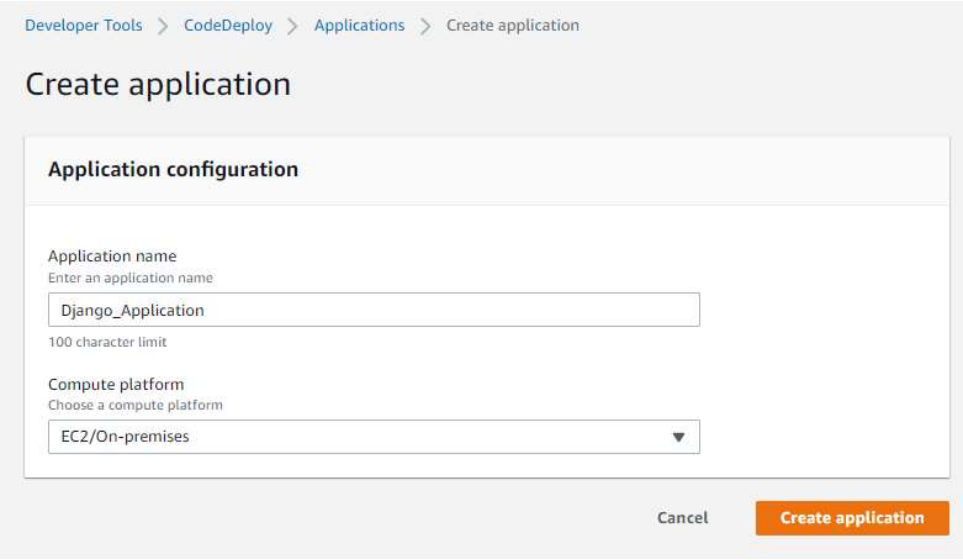
1. Open AWS Management Console and select CodeDeploy under Developer tools.

2. 

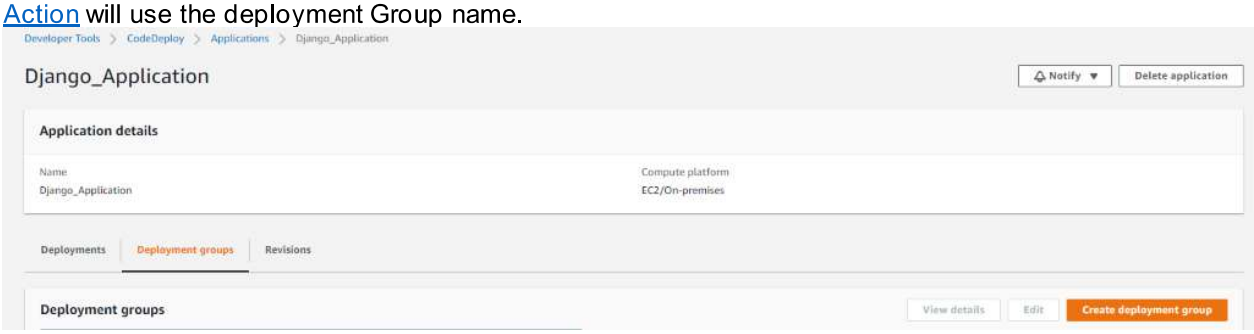
3. Select Application under Deploy options. Click on Create Application.

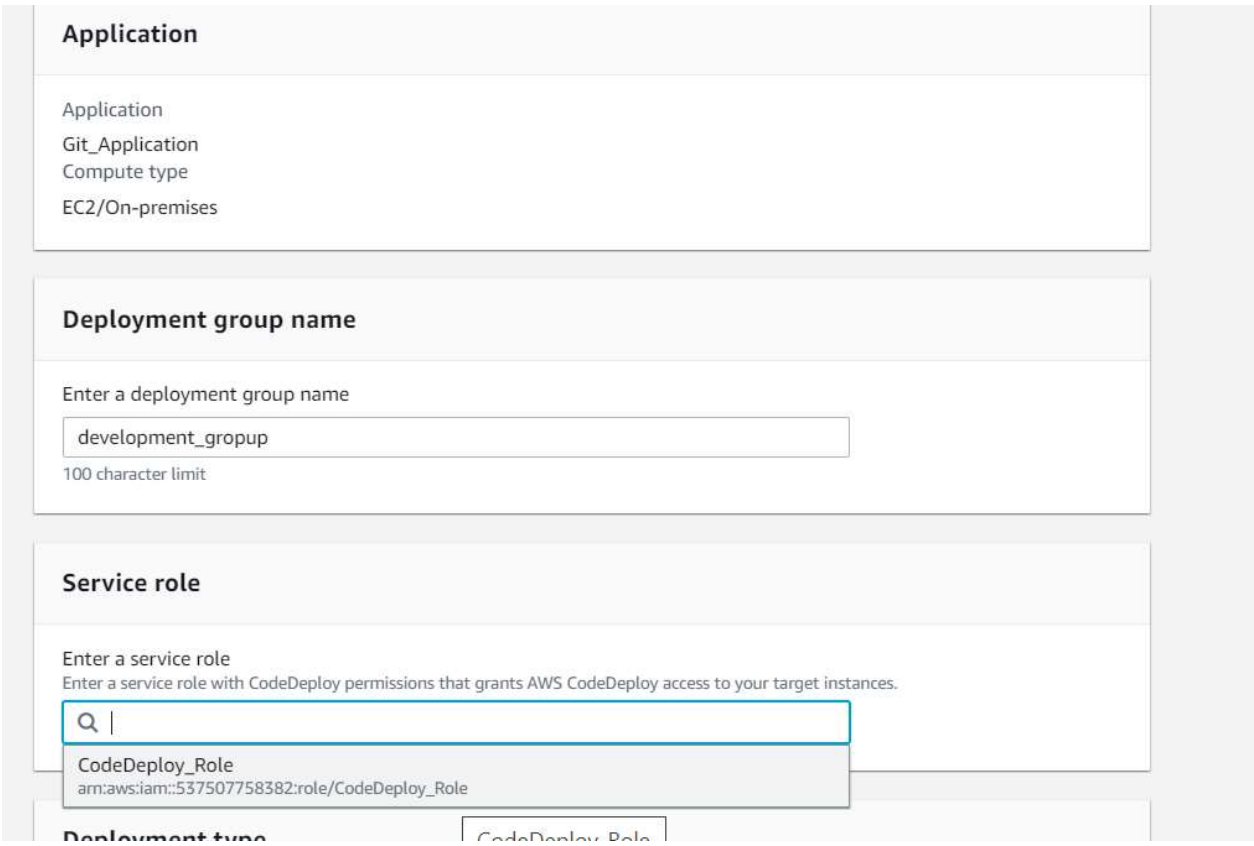
4. 

5. Create an Application name called Django_Application with *compute platform* EC2/On-premises. GitHub Action will use the application name.

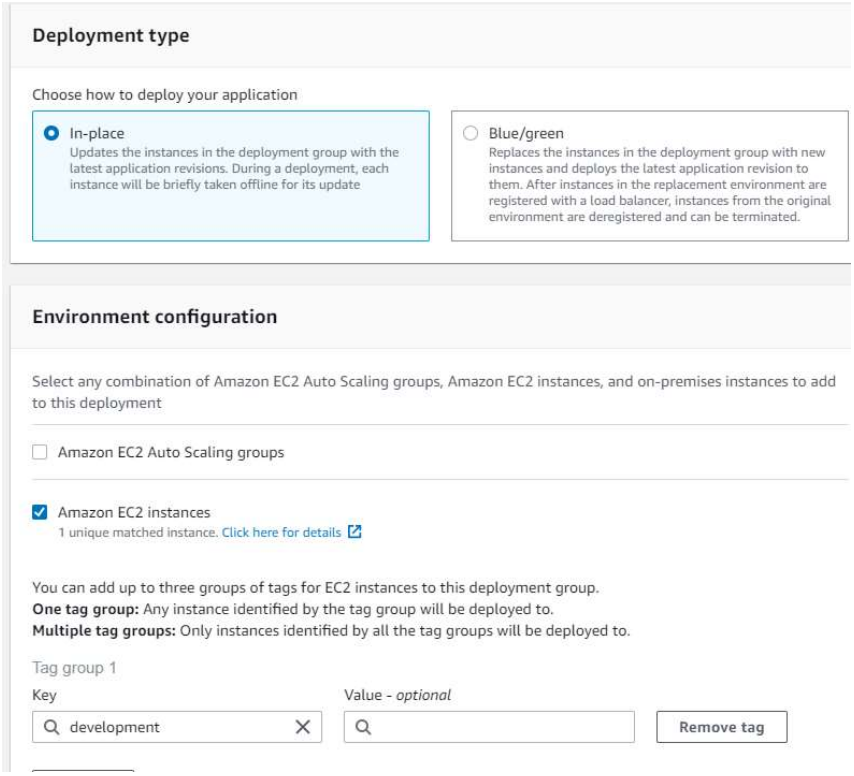
6. 

7. Once Application Created, Create a *Deployment Group* and name development_gropup. Get the *Role ARN* from CodeDeploy_Role, which we created before and put in the service role. GitHub Action will use the deployment Group name.

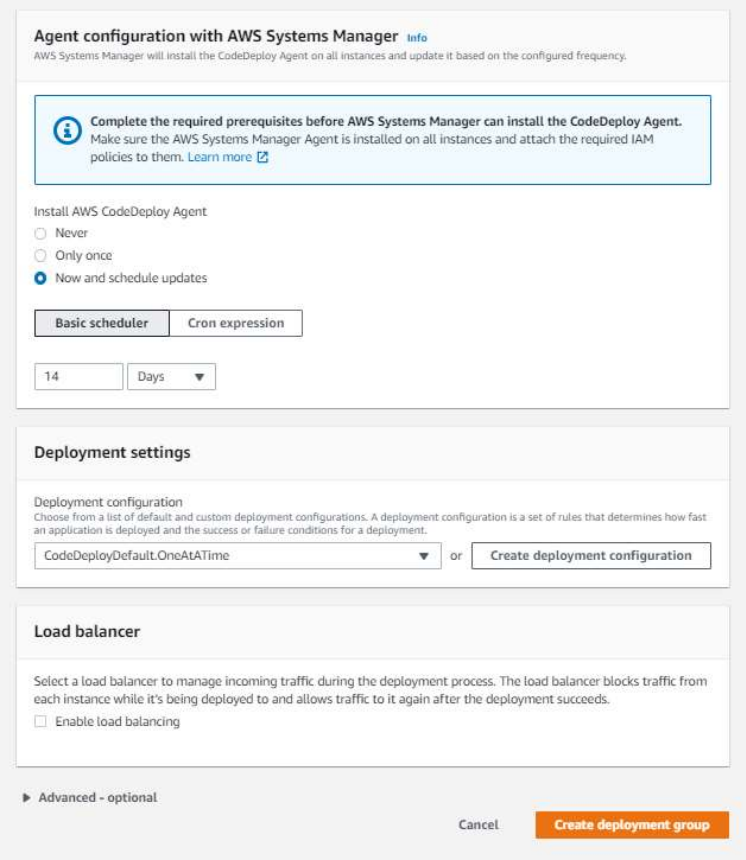
8. 

9. 

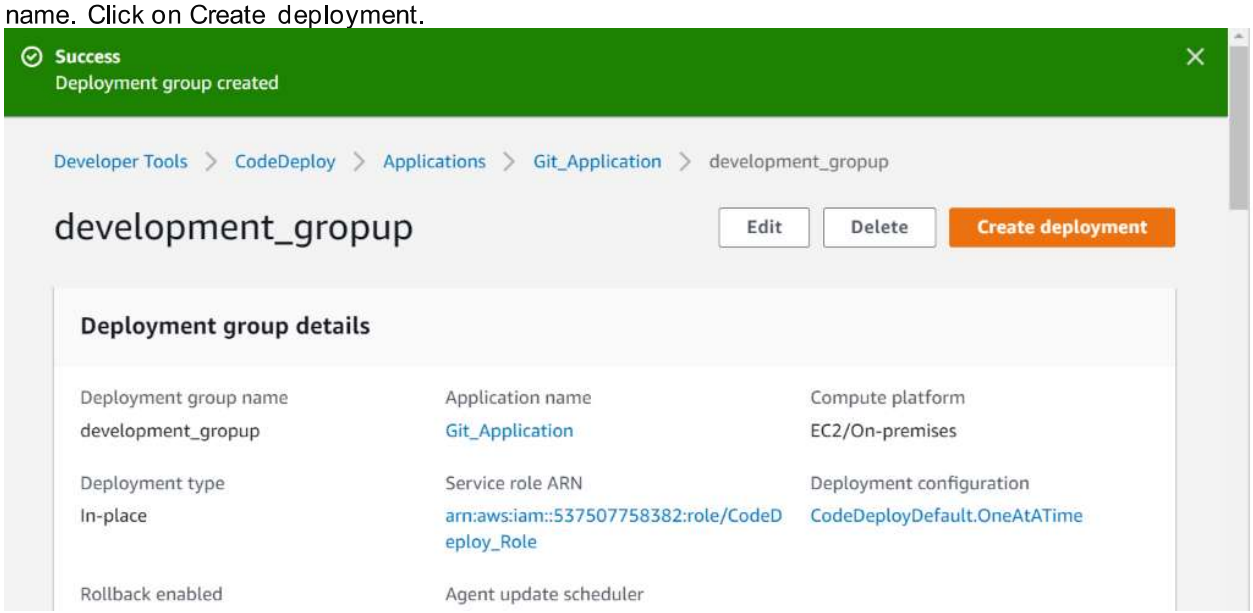
10. Choose In-place Deployment type. Select Amazon Ec2 Instances environment configuration and Tag key development to create AWS EC2 instance.

11. 

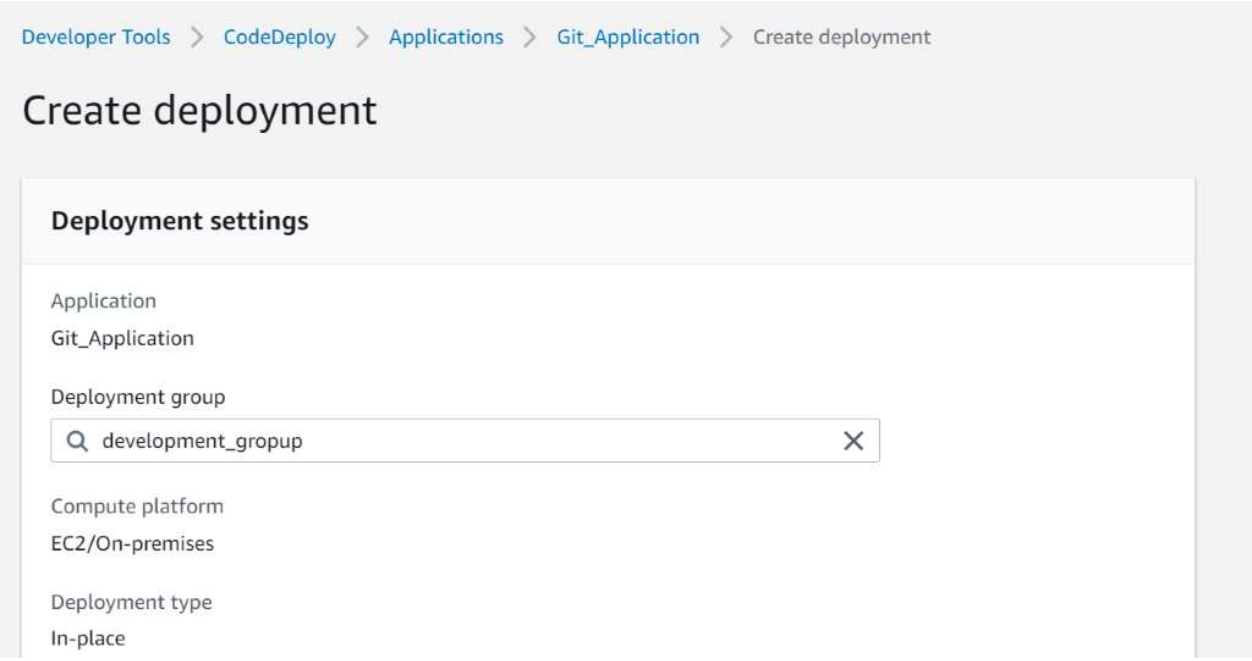
12. Select a schedule manager to install the CodeDeploy agent. Set *OneAtATime* deployment setting and Create Deployment Group without a load balancer.

13. 

14. Once the Deployment Group is created, test the deployment by creating a Deployment with any name. Click on Create deployment.

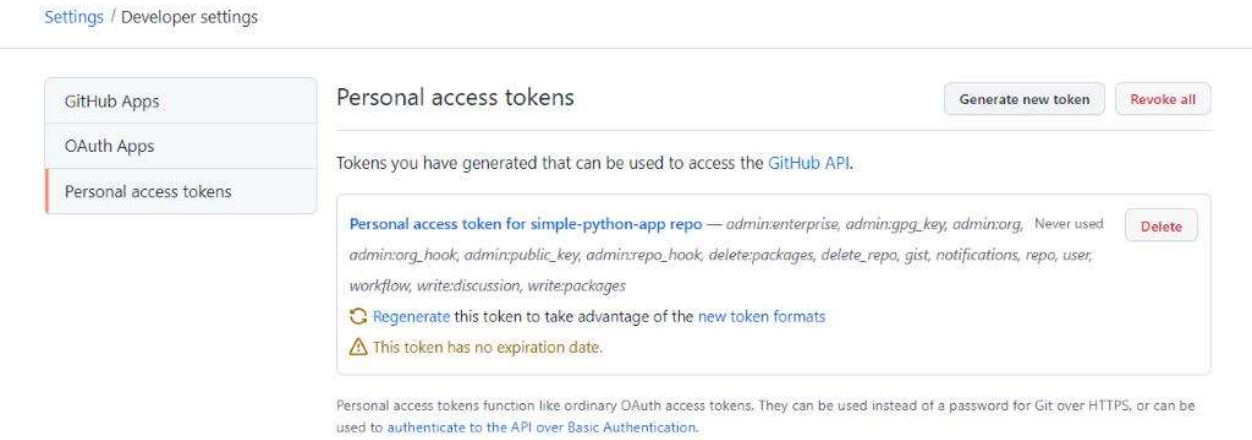
15. 

16. You will see the create deployment settings.

17. 

18.

19. Go to the GitHub settings page. Under Developer settings, click on Generate new token

20. 

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Note

Used for deployment purpose in EC2 instance

What's this token for?

Expiration *

30 days — The token will expire on Wed, Feb 9 2022

Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

☑ **repo**	Full control of private repositories
☑ repo:status	Access commit status
☑ repo_deployment	Access deployment status
☑ public_repo	Access public repositories
☑ repo:invite	Access repository invitations
☑ security_events	Read and write security events
☑ **workflow**	Update GitHub Action workflows
☑ **write:packages**	Upload packages to GitHub Package Registry
☑ read:packages	Download packages from GitHub Package Registry
☑ **delete:packages**	Delete packages from GitHub Package Registry
☑ **admin:org**	Full control of orgs and teams, read and write org projects

21.

☑ **notifications**	Access notifications
☑ **user**	Update ALL user data
☑ read:user	Read ALL user profile data
☑ user:email	Access user email addresses (read-only)
☑ user:follow	Follow and unfollow users
☑ **delete_repo**	Delete repositories
☑ **write:discussion**	Read and write team discussions
☑ read:discussion	Read team discussions
☑ **admin:enterprise**	Full control of enterprises
☑ manage_runners:enterprise	Manage enterprise runners and runner-groups
☑ manage_billing:enterprise	Read and write enterprise billing data
☑ read:enterprise	Read enterprise profile data
☑ **admin:gpg_key**	Full control of public user GPG keys (Developer Preview)
☑ write:gpg_key	Write public user GPG keys
☑ read:gpg_key	Read public user GPG keys

Generate token Cancel

22.

Personal access tokens Generate new token Revoke all

Tokens you have generated that can be used to access the GitHub API.

Make sure to copy your personal access token now. You won't be able to see it again!

✓ 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 Delete

Personal access token for simple-python-app repo — *admin:enterprise, admin:gpg_key, admin:org, admin:org_hook, admin:public_key, admin:repo_hook, delete:packages, delete_repo, gist, notifications, repo, user, workflow, write:discussion, write:packages* Never used Delete

Regenerate this token to take advantage of the new token formats

This token has no expiration date.

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

23.

24. **GitHub Token:** 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
25. Select Revision Type My application is stored in GitHub, and select Connect to GitHub by providing the GitHub token.

Revision type

My application is stored in Amazon S3

My application is stored in GitHub

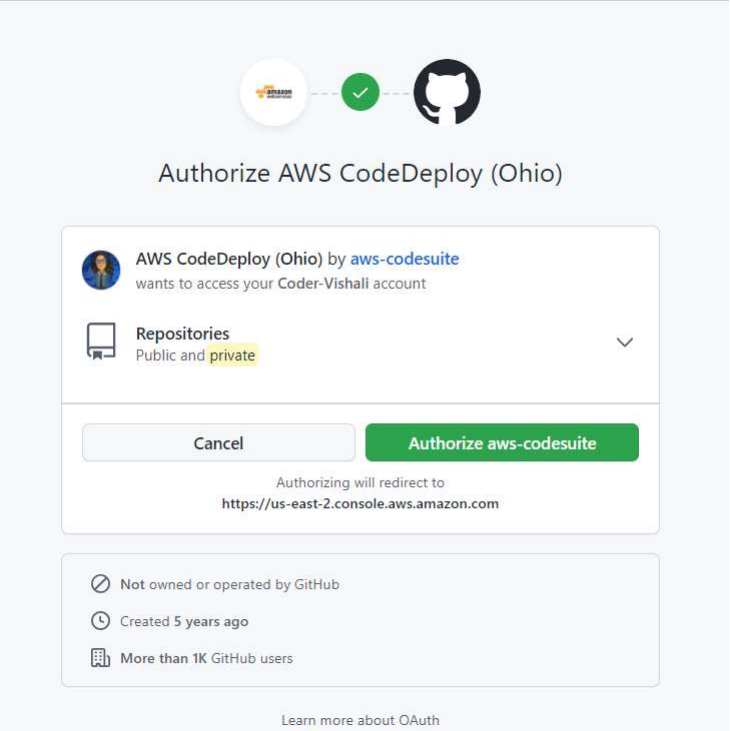
GitHub token name

Select the name of the token associated to an account you have already connected, or grant AWS CodeDeploy permission to access a different account. To connect to a GitHub account for the first time, type an alias for the account, and then choose Connect to GitHub

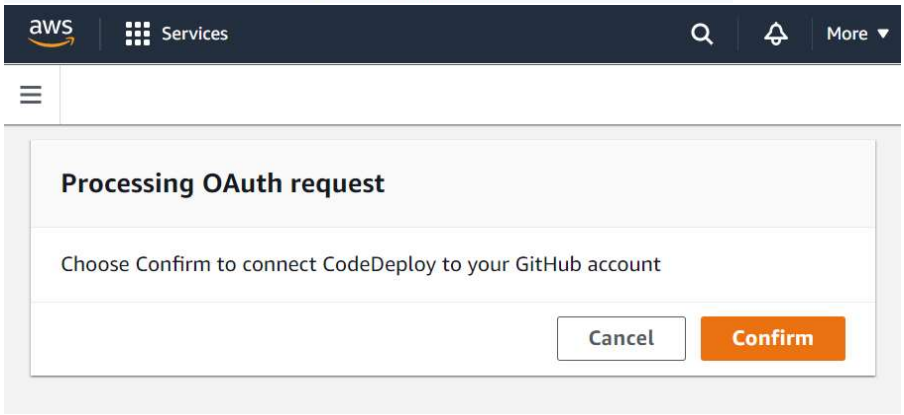
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

Connecting

26.



27.



28.

GitHub token name

Select the name of the token associated to an account you have already connected, or grant AWS CodeDeploy permission to access a different account. To connect to a GitHub account for the first time, type an alias for the account, and then choose Connect to GitHub

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

Connected

Application Git_Application successfully bound to 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 GitHub token

29.

30. Once connected to GitHub, Provide the repository name and last *Commit ID*. Select *Overwrite the content* and Create Deployment.

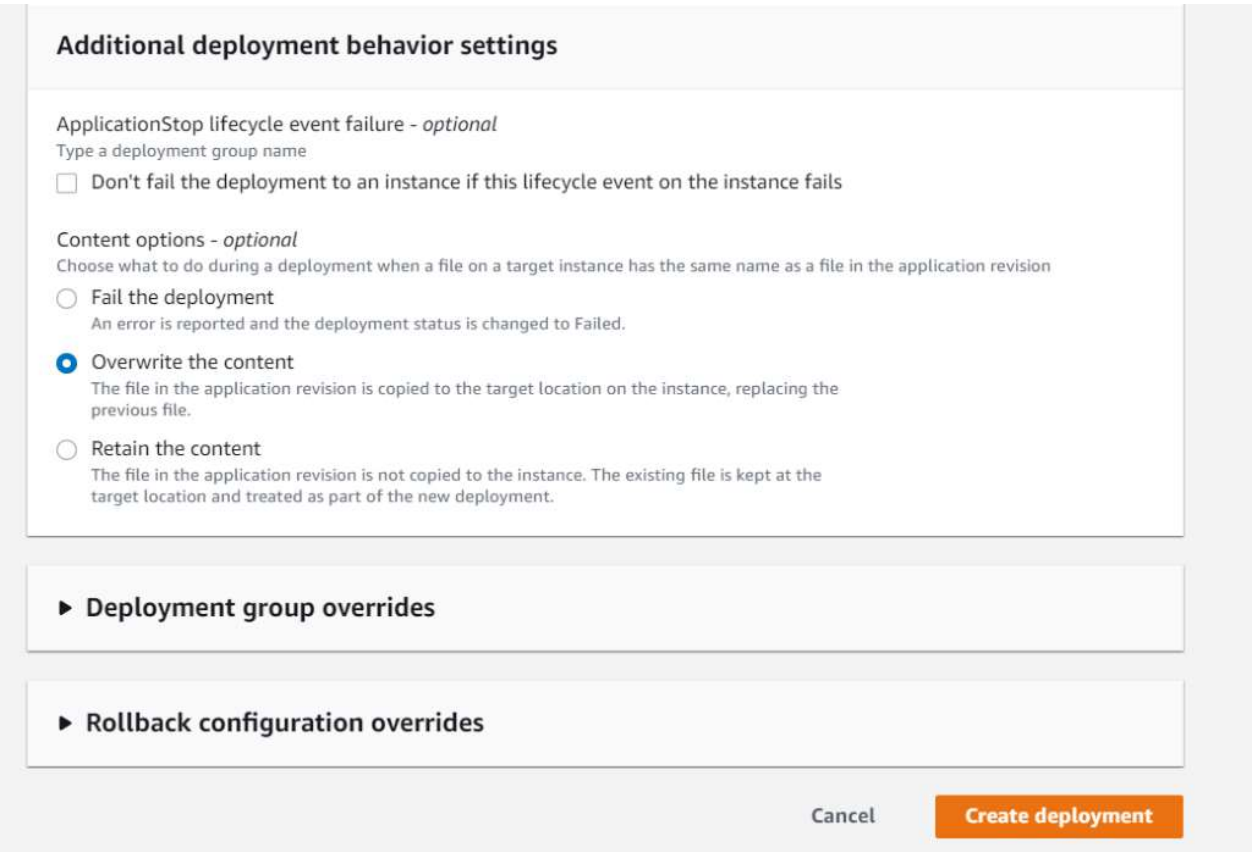
Repository name

Coder-Vishali/spring-boot-mongo

Commit ID

990cb615755ec9e2bca9f9ec3bcd881be4bc3503

31.

32. 

33. Successfully created deployment

GitHub Project

To push the files into GitHub:

git init

Open the Git bash:

git add .

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/Coder-Vishali/django_app.git

git push -u origin main

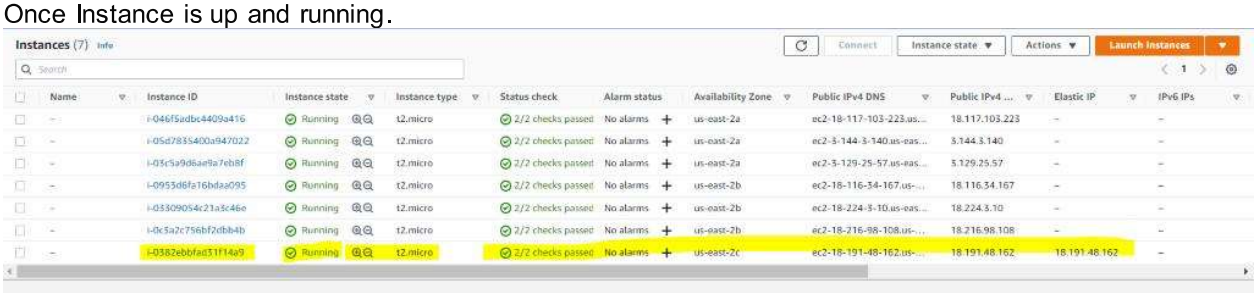
To push local files into ec2 instance:

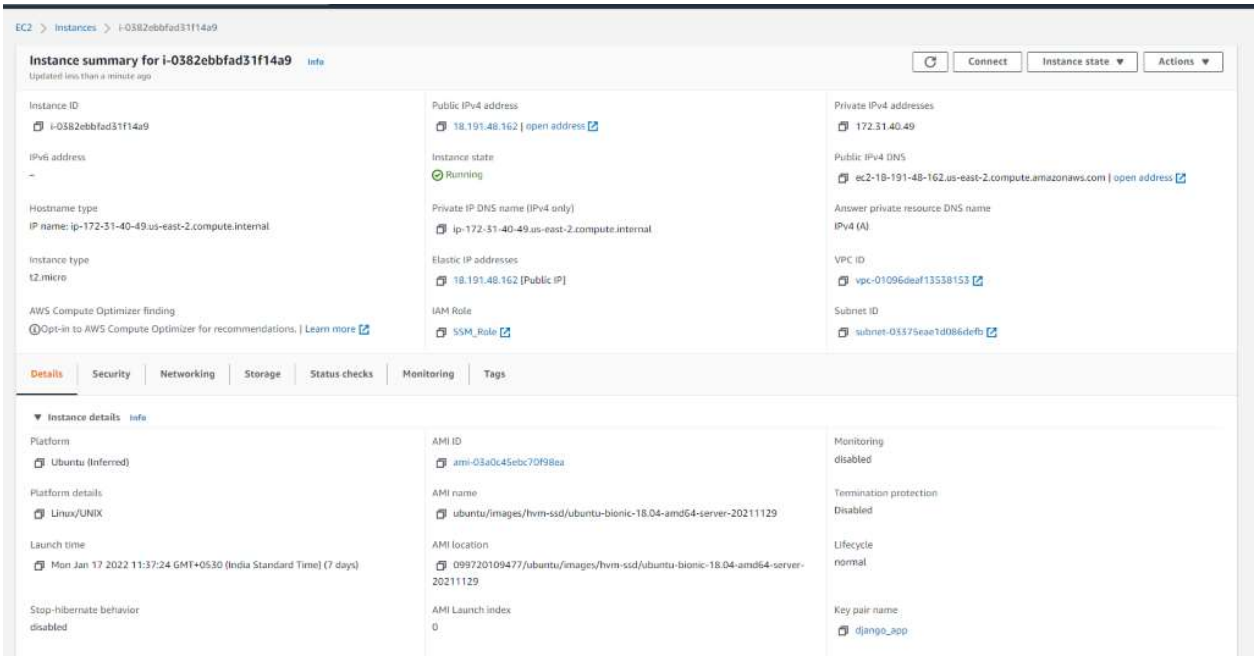
pscp -i <.pkk file> <filename> <ec2 user>@<ip address>:/home/ec2-user

pscp -i C:\Users\VISHALI\Downloads\new_key_pair.ppk C:\Users\VISHALI\Django_app ec2-user@18.224.3.10:/home/ec2-user

Launch EC2 Instance

1. Once Instance is up and running.

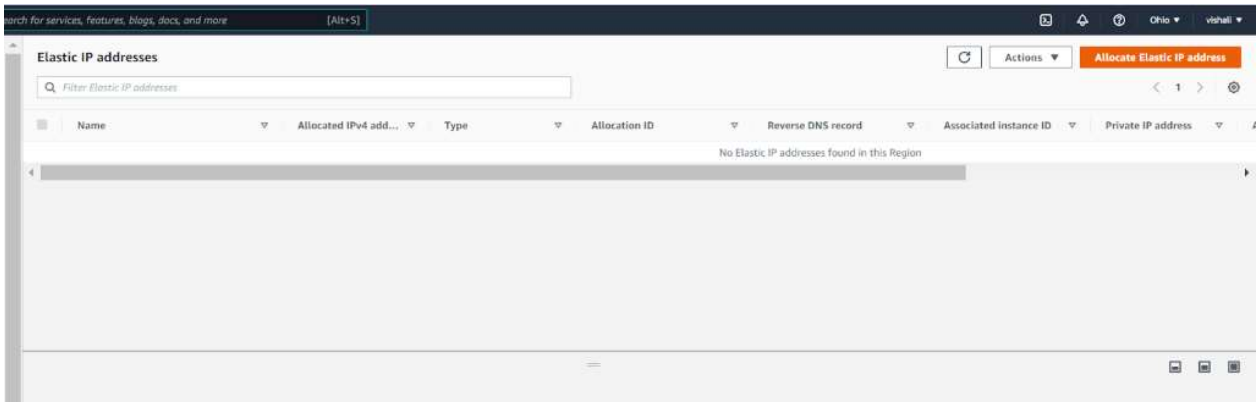
2. 

3. 

4. You can add elastic IPS by following the below instructions. You have to go under network and security. Select Elastic IPs.

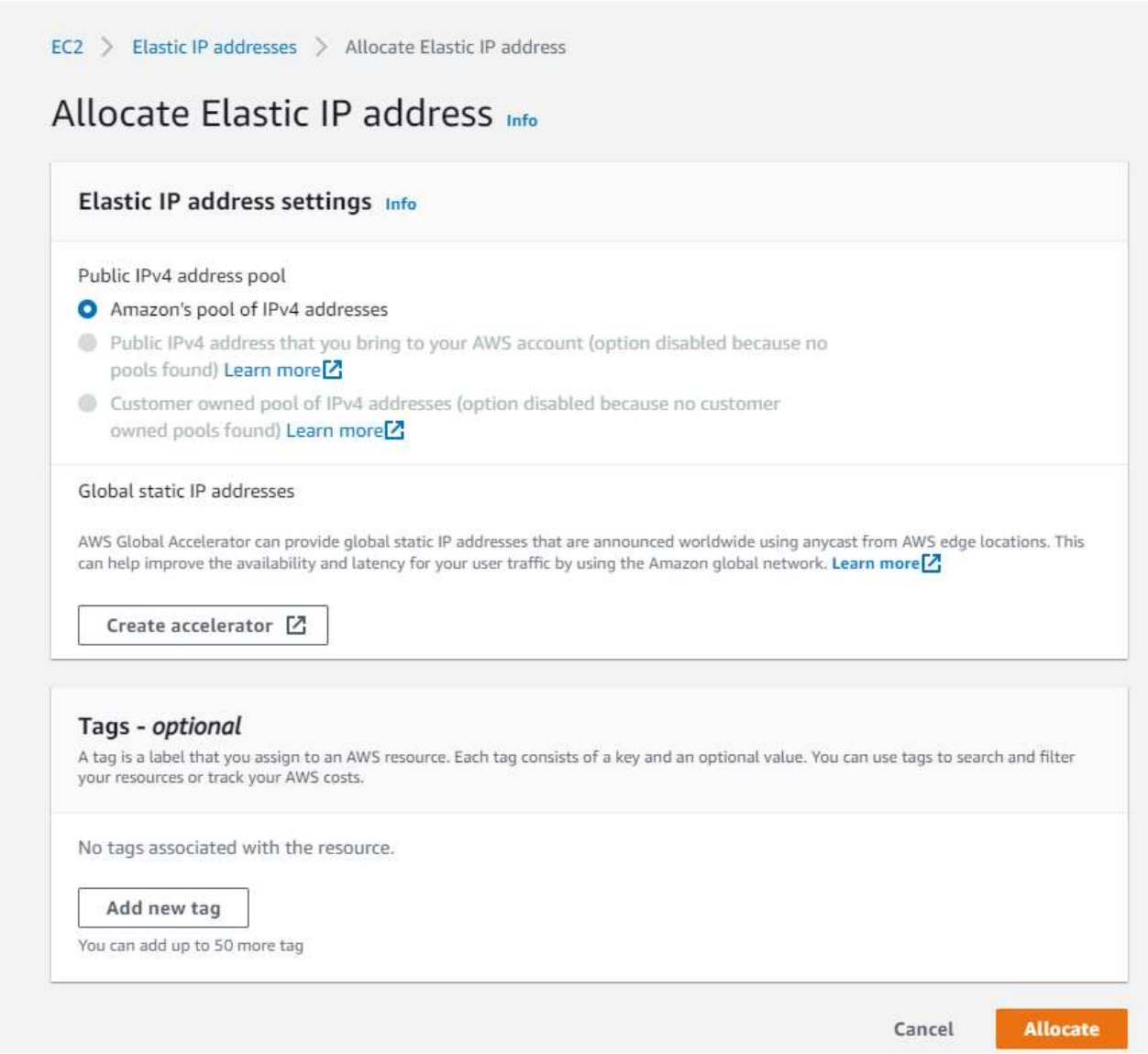
5.

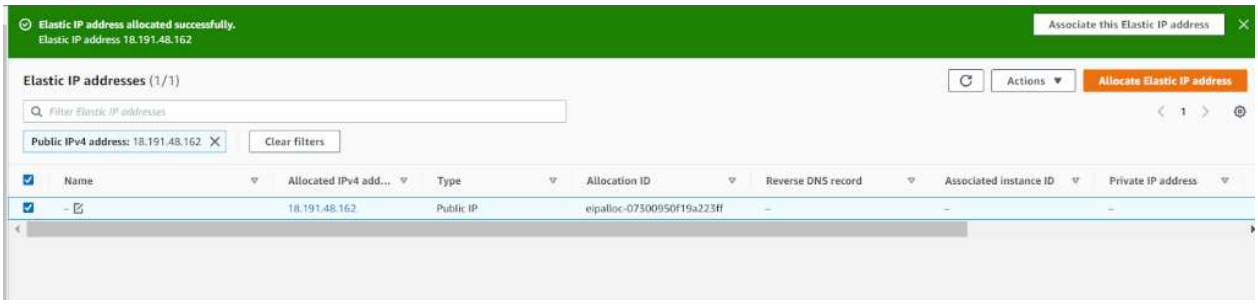
6. Select allocate Elastic IP address



7.

8. Press Allocate

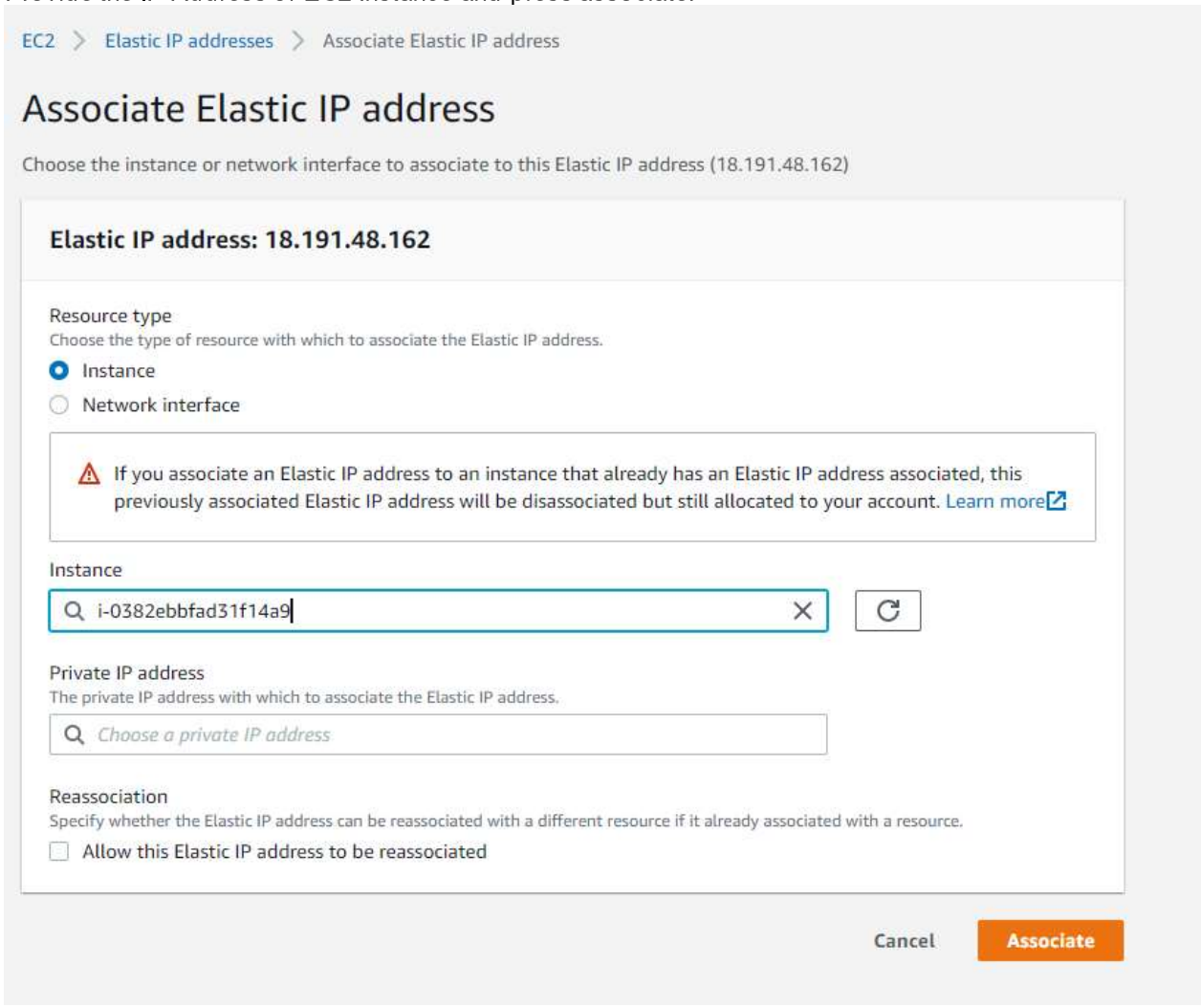
9. 

10. 

11. Go to actions -> Associate Elastic IP Address:

12. 

13. Provide the IP Address of EC2 instance and press associate:

14. 

15. You can use Putty terminal to connect to EC2 instance. Provide username as ubuntu.

Install CodeDeploy Agent on EC2 Instance

To deploy the git repo by using CodeDeploy Service, codeDeploy-agent must install in the EC2 instance. Use the below commands to install codedeploy-agent.

Use the below commands to install codedeploy-agent.

sudo yum update

sudo yum install -y ruby

sudo yum install wget

wget https://bucket-name.s3.region-identifier.amazonaws.com/latest/install

bucket-name is the Amazon S3 bucket containing the CodeDeploy Resource Kit files for your region. region-identifier is the identifier for your region.

You can find the list of bucket names and region identifiers here.

For example:

wget https://aws-codedeploy-us-east-2.s3.us-east-2.amazonaws.com/latest/install

chmod +x ./install

sudo ./install auto

sudo service codedeploy-agent start

```
Starting codedeploy-agent: [ec2-user@ip-172-31-29-19 ~]$
```

crtl + c

Firewall Setup:

Enable Firewall and allow OpenSSH

sudo ufw enable

sudo ufw allow OpenSSH

sudo ufw allow 'Nginx Full'

Check to make sure we are allowing OpenSSH:

sudo ufw status

Expected output:

Install the python ngnix and git:

sudo apt-get update

sudo apt-get install python-pip python-dev nginx git

sudo pip install virtualenv

git clone https://github.com/Coder-Vishali/django_app.git

cd django_app

Setup the virtual environment & install necessary packages

python -m venv myvenv

source myvenv/bin/activate

pip install -r requirements.txt

pip install django bcrypt django-extensions

pip install gunicorn

Add the EC2 instance IP in the Allowed host of django settings.py file:

cd django_app

sudo vim settings.py

 # Inside settings.py modify these lines allowed host public IP address

ALLOWED_HOSTS = ['13.59.206.93']

 # add the line below to the bottom of the file

STATIC_ROOT = os.path.join(BASE_DIR, "static/")

Save your changes and quit. ESC :wq

cd ..

Bind the django with gunicorn:

python manage.py collectstatic

gunicorn --bind 0.0.0.0:8000 django_app.wsgi:application

ctrl+c

Edit the gunicorn service:

sudo vim /etc/systemd/system/gunicorn.service

```
[Unit]
Description=gunicorn daemon
After=network.target
[Service]
User=ubuntu
Group=www-data
WorkingDirectory=/home/ubuntu/django_app
ExecStart=/home/ubuntu/django_app/venv/bin/gunicorn --workers 3 --bind
unix:/home/ubuntu/django_app/django_app.sock django_app.wsgi:application
[Install]
WantedBy=multi-user.target
```

ESC :wq

Start and Enable the gunicorn service:

sudo chown ubuntu:www-data /home/ubuntu/django_app/

pkill gunicorn

sudo systemctl daemon-reload

sudo systemctl start gunicorn

sudo systemctl enable gunicorn

sudo systemctl status gunicorn.service

Edit the nginx configurations file:

sudo vim /etc/nginx/sites-available/django_app

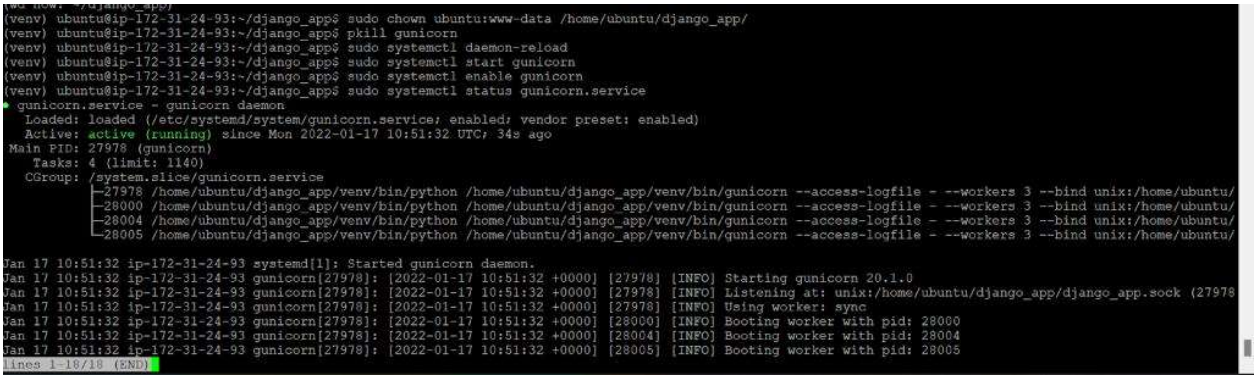
```
server {
 listen 80;
 server_name 18.191.48.162;
```

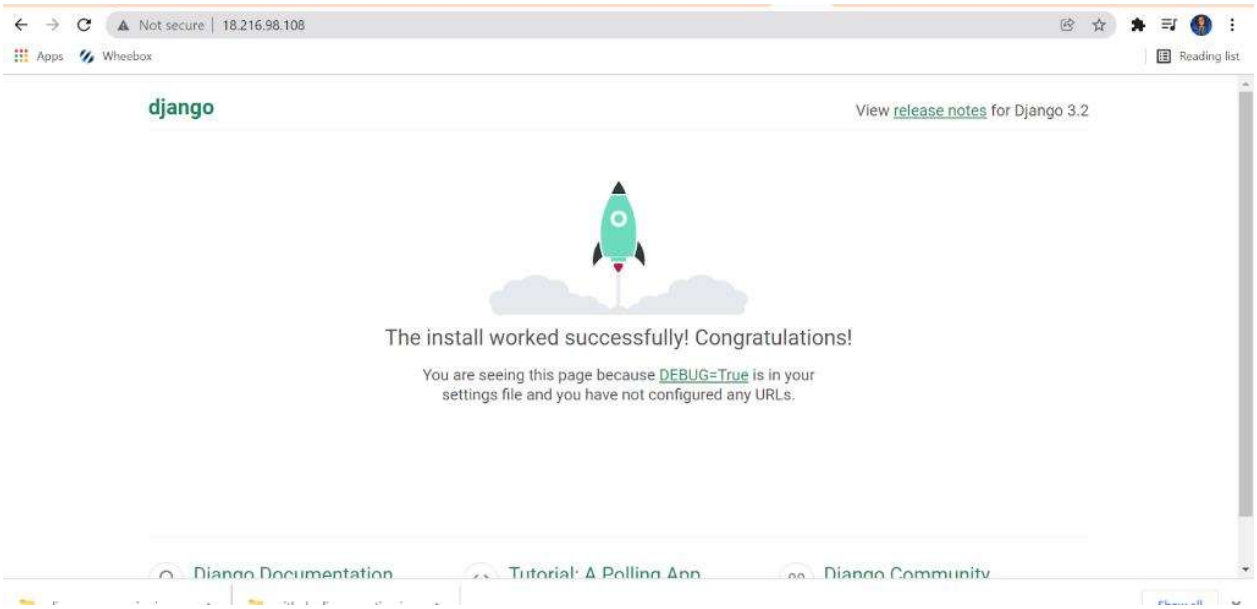
```
  location = /favicon.ico { access_log off; log_not_found off; }
  location /static/ {
    root /home/ubuntu/django_app;
  }
  location /media/ {
    root /home/ubuntu/django_app;
  }
  location / {
    include proxy_params;
    proxy_pass http://unix:/home/ubuntu/django_app/django_app.sock;
  }
}
```

ESC :wq

Start the nginx service:

```
sudo ln -sf /etc/nginx/sites-available/django_app  /etc/nginx/sites-enabled
sudo nginx -t
sudo rm /etc/nginx/sites-enabled/default
sudo service nginx restart
sudo systemctl status nginx
```

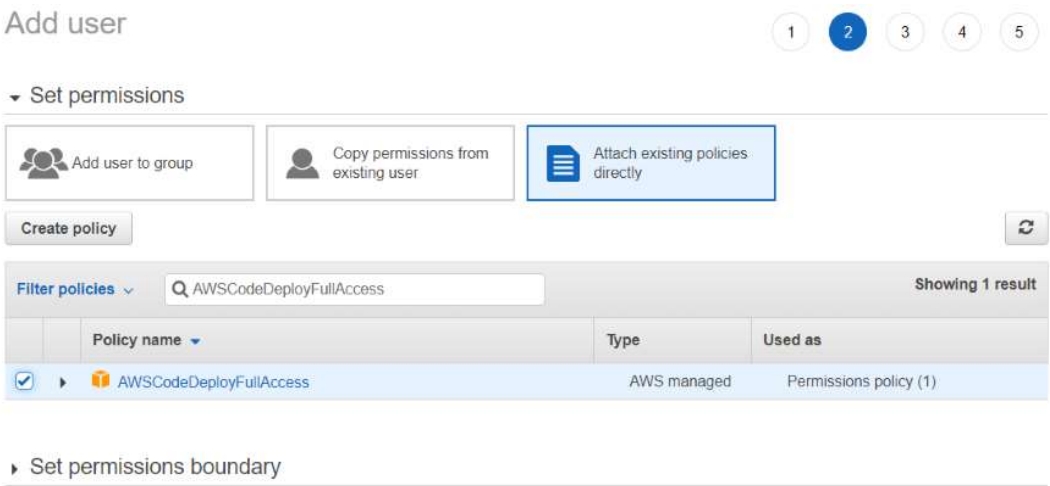


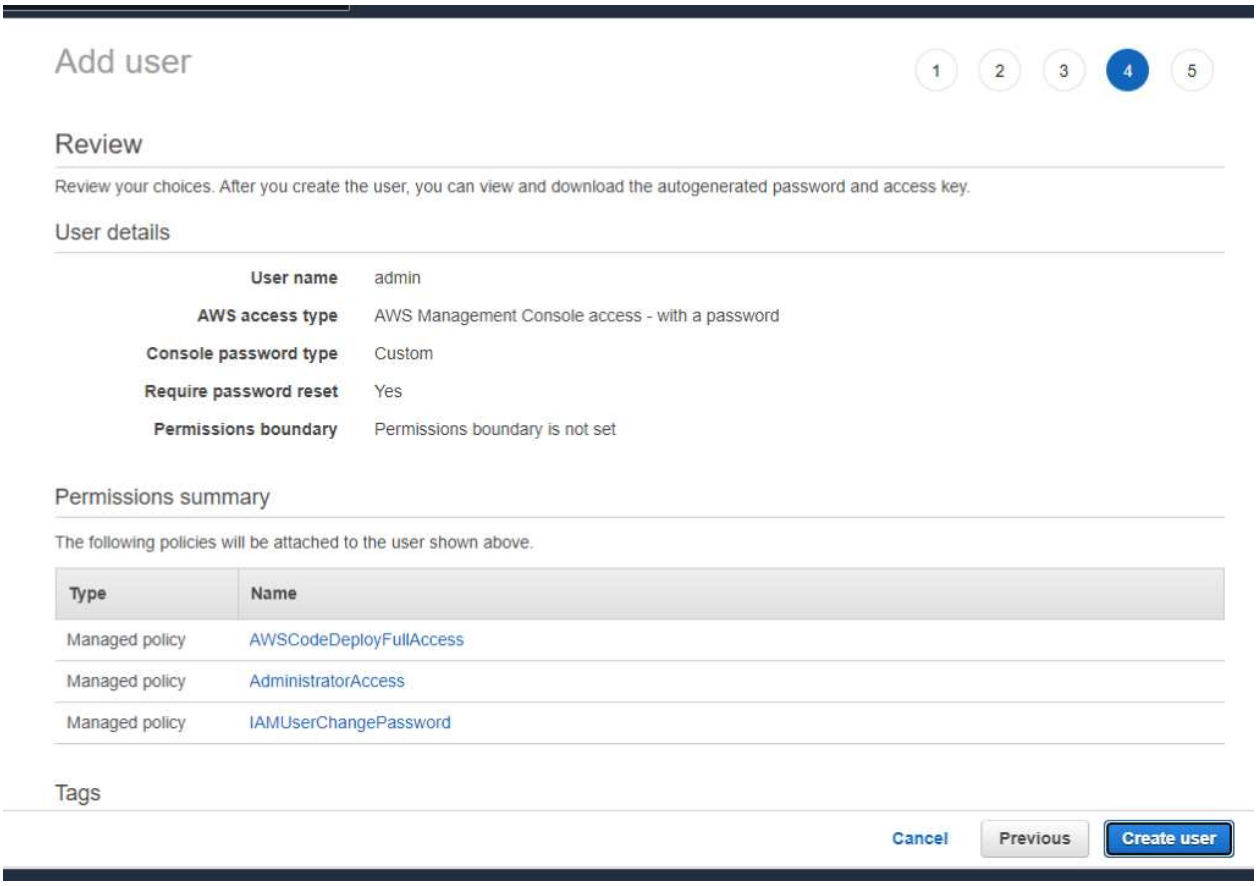


GitHub Action

First, create an IAM user with full AWSCodeDeployFullAccess policy and generate an access key and secret access for the user to configure GitHub Action.

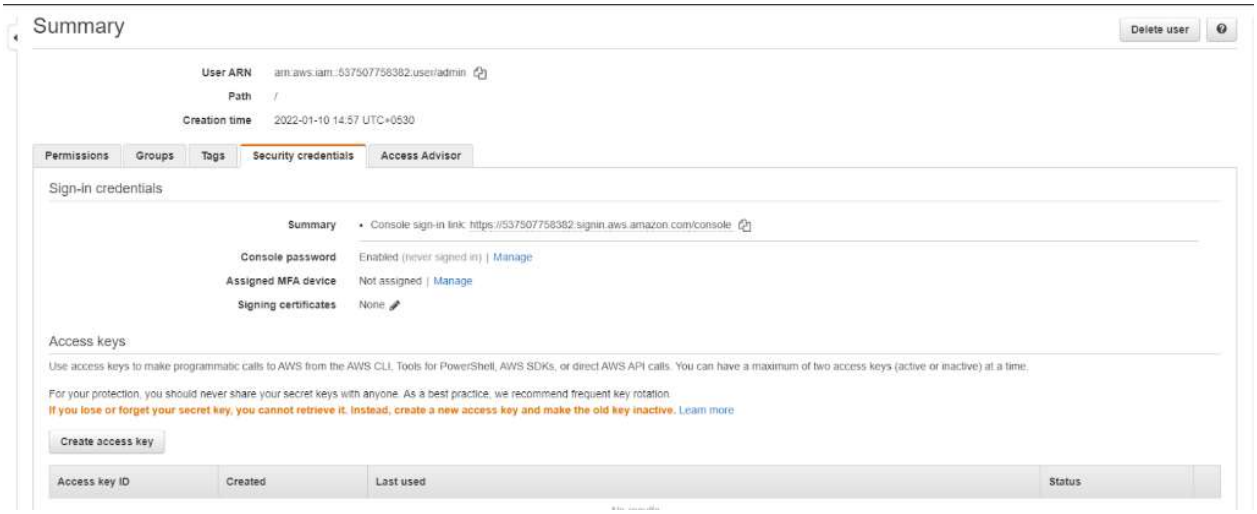
Before configuring Action, set the environment in the GitHub repository.



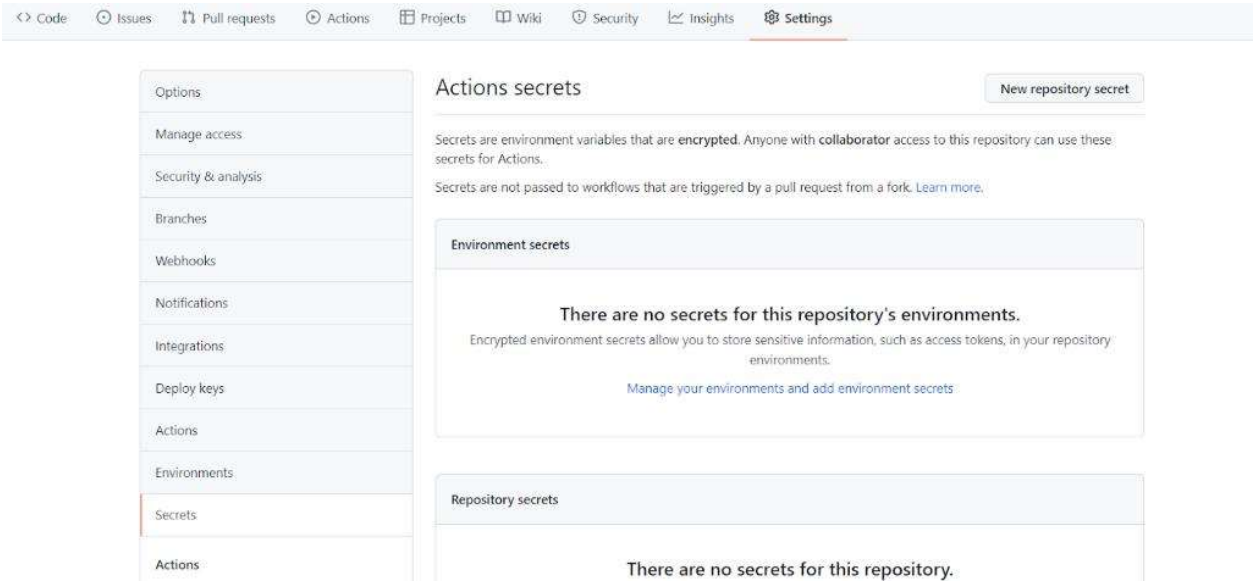


Click on Create user. IAM User will be created.

To generate the access key, click on **Create access key:**

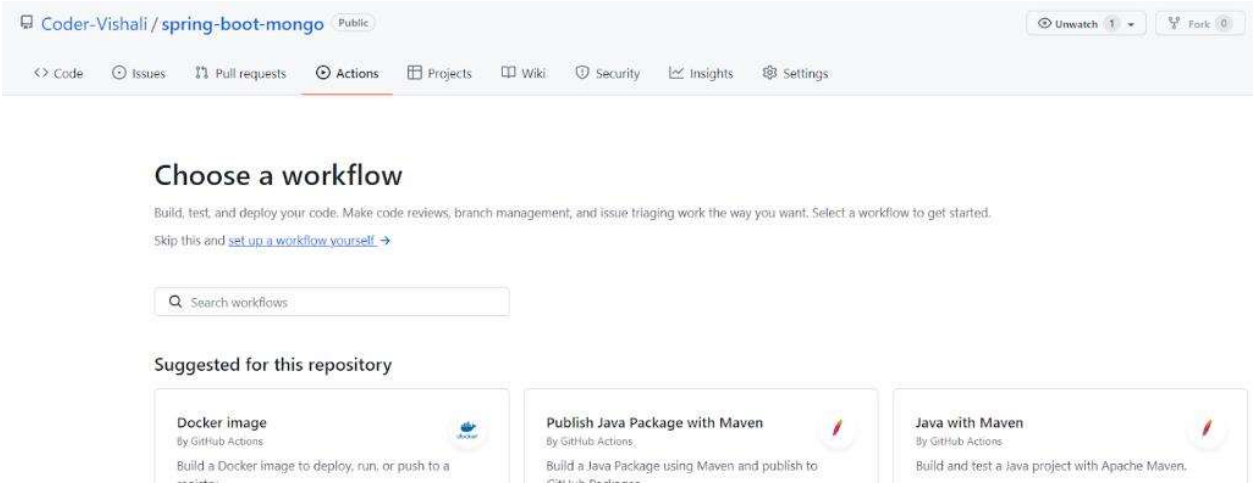


Before configuring Action, set the environment in the GitHub repository.

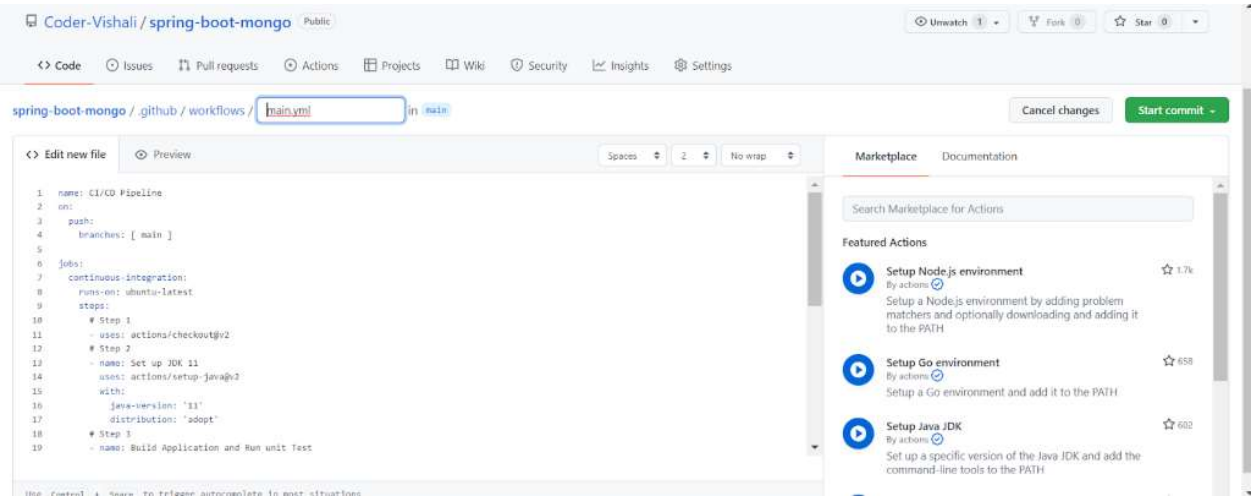


Click on New repository secret and add the following secrets:

GitHub repository changes will trigger GitHub Action,



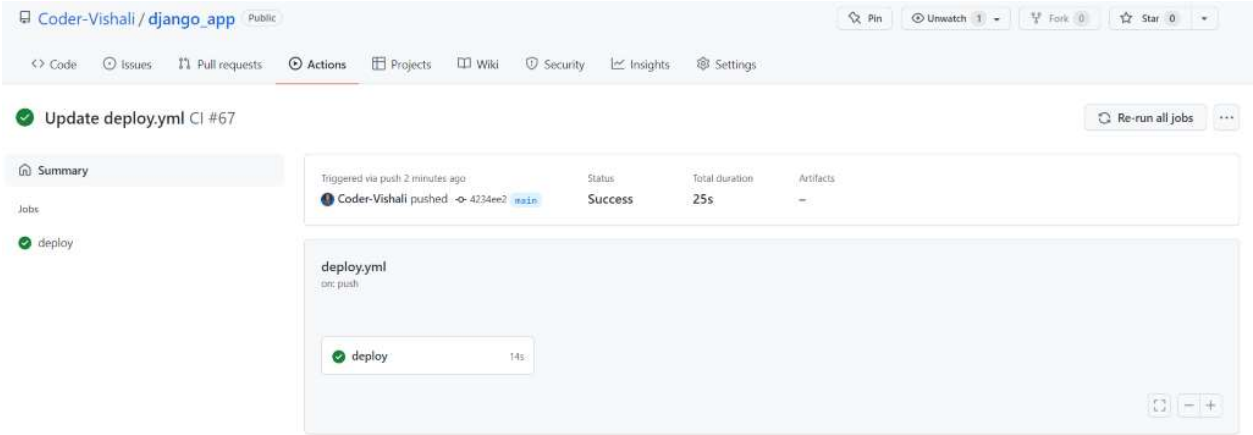
Here, you can configure your actions:



Now make a change to your repository. Your changes should automatically deploy to your EC2 server. Access the application from a web browser or postman.

curl --location --request GET 'http://{{[ec2_public_ip]}}:8000'

Look into GitHub Actions:



Deployment status:

LifecycleEvent - ApplicationStart

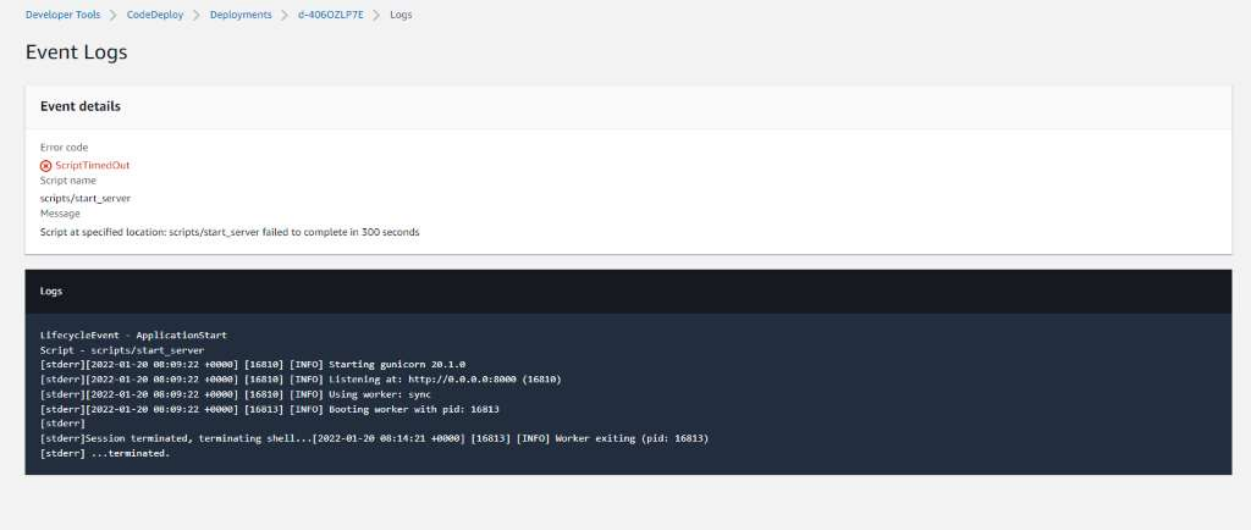
Script - scripts/start_server

[stderr][2022-01-20 08:09:22 +0000] [16810] [INFO] Starting gunicorn 20.1.0
[stderr][2022-01-20 08:09:22 +0000] [16810] [INFO] Listening at: http://0.0.0.0:8000 (16810)
[stderr][2022-01-20 08:09:22 +0000] [16810] [INFO] Using worker: sync

[stderr][2022-01-20 08:09:22 +0000] [16813] [INFO] Booting worker with pid: 16813
[stderr]

[stderr]Session terminated, terminating shell...[2022-01-20 08:14:21 +0000] [16813] [INFO] Worker exiting (pid: 16813)

[stderr] ...terminated.



Reference:

- Django app on aws: https://dev.to/rmiyazaki6499/deploying-a-production-ready-django-app-on-aws-1pk3
- Guicorn and ngnix setup: https://www.youtube.com/watch?v=QjrfUO91wfc
- Github code: https://github.com/Coder-Vishali/django_app