Jai Talreja 61 D15A

Cloud Deployment with Automation

Introduction

This case study explores the concepts of AWS CodePipeline, EC2 and S3. In this we build a simple HTML WebApp using AWS CodeBuild, and automatically upload it to an S3 bucket to then automatically deploy it to an EC2 instance using CodeDeploy.

The tools and concepts used for this case study are :-

AWS CodeBuild

Key Features:

Build Automation: Fully managed service that compiles source code, runs tests, and produces software packages.

Custom Build Environments: Supports Docker images for custom build environments.

Scalability: Automatically scales to meet build demand.

Pay-as-you-go Pricing: You only pay for the compute resources you use.

Practical Uses:

Continuous Integration (CI) for automated testing and building of applications.

Integrating with other AWS services for a seamless development pipeline.

Building and packaging applications for deployment.

AWS CodePipeline

Key Features:

Continuous Delivery: Automates the software release process using defined workflows.

Integration with Other AWS Services: Works seamlessly with CodeBuild, CodeDeploy, and third-party tools.

Customizable Workflows: Easily define stages for building, testing, and deploying applications.

Practical Uses:

Automating the release process from code commit to deployment.

Creating pipelines for microservices or multi-environment setups.

Enabling rapid and reliable application delivery.

Amazon S3 (Simple Storage Service)

Key Features:

Scalable Storage: Virtually unlimited storage capacity.

Durability and Availability: Designed for 99.999999999% durability and high availability.

Security Features: Supports access control, encryption, and versioning.

Practical Uses:

Storing build artifacts and deployment packages.

Hosting static websites and serving assets for web applications.

Backup and archival storage.

Amazon EC2 (Elastic Compute Cloud)

Key Features:

Flexible Computing: Provides resizable compute capacity in the cloud.

Variety of Instance Types: Different instance types for various workloads.

Auto Scaling: Automatically adjusts capacity based on demand.

Practical Uses:

Hosting applications and services in a scalable manner.

Running batch processing and data analytics workloads.

Deploying web applications or back-end services.

AWS CodeDeploy

Key Features:

Automated Deployments: Automatically deploys applications to EC2, Lambda, or on-premises servers.

Blue/Green Deployments: Reduces downtime and risks during application updates.

Monitoring and Rollback: Monitors deployment status and can roll back if issues are detected.

Practical Uses:

Managing updates and deployments for applications running on EC2 instances.

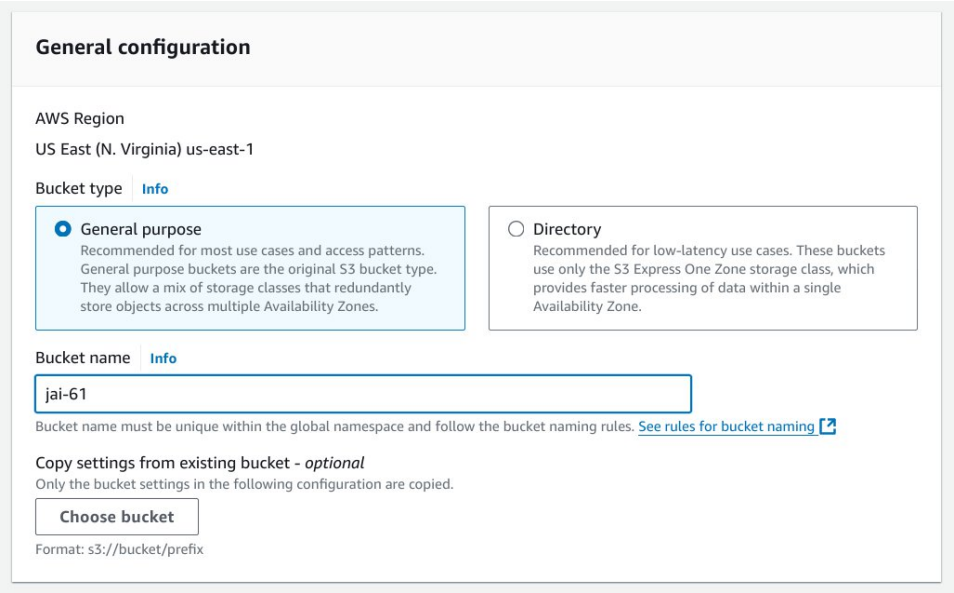
Facilitating microservices deployments with minimal downtime.

Integrating with CI/CD pipelines to automate the deployment process.

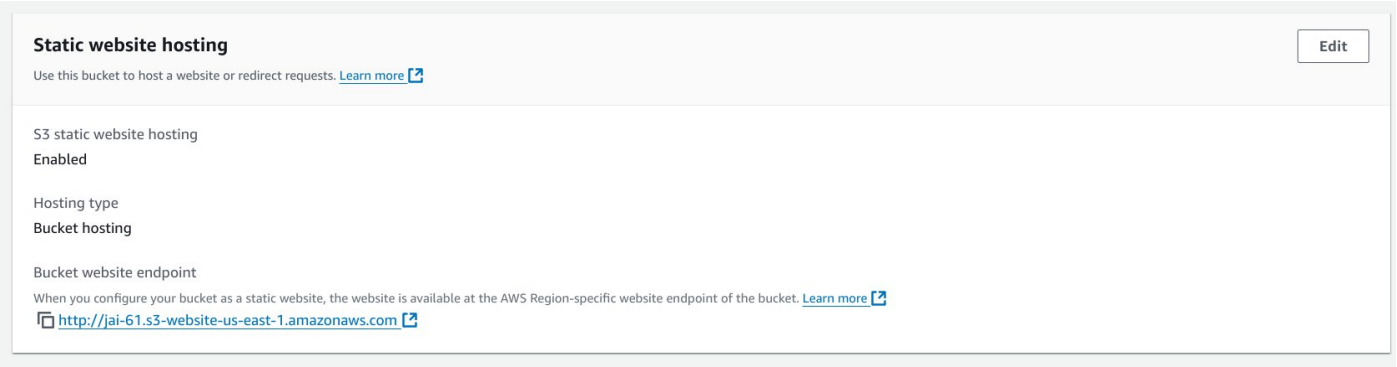
Jai Talreja 61 D15A

Steps

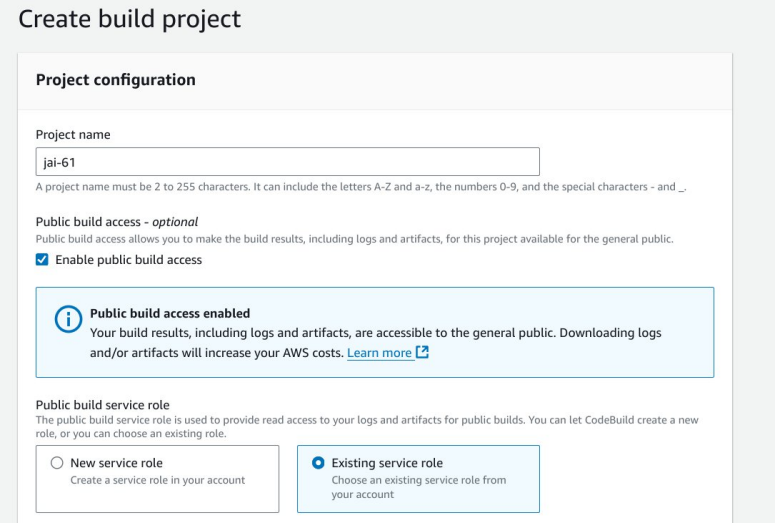
Creating an S3 Bucket



Enabling static website hosting

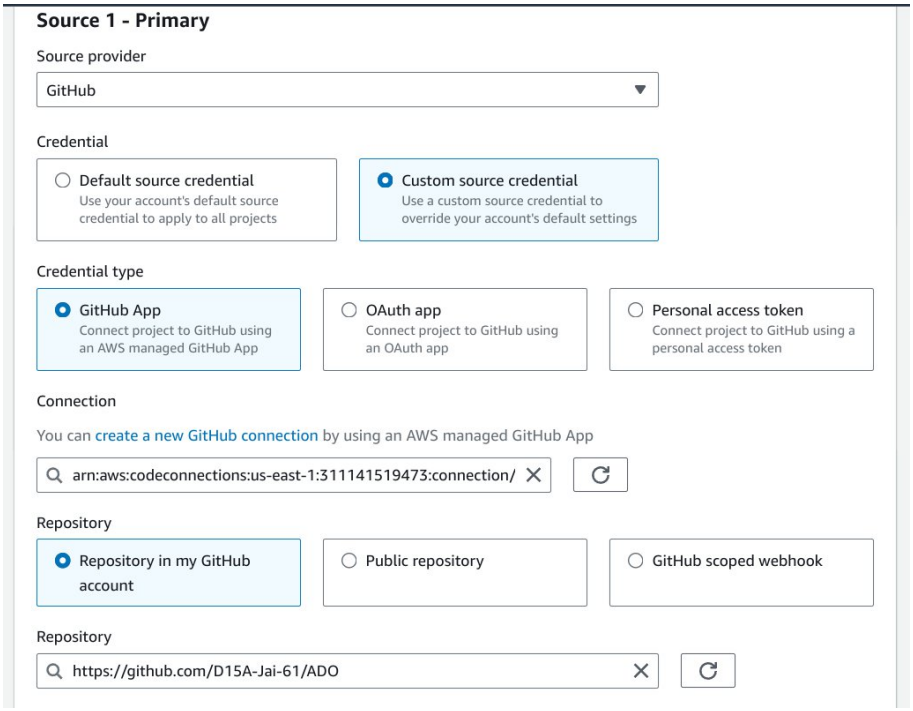


Creating a Build project

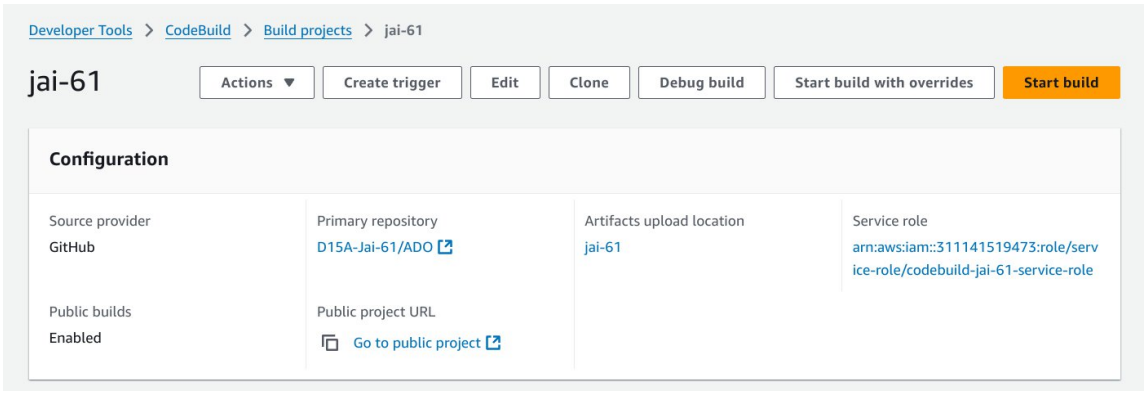


Jai Talreja 61 D15A

Configuring source for CodeBuild project as a GitHub repository



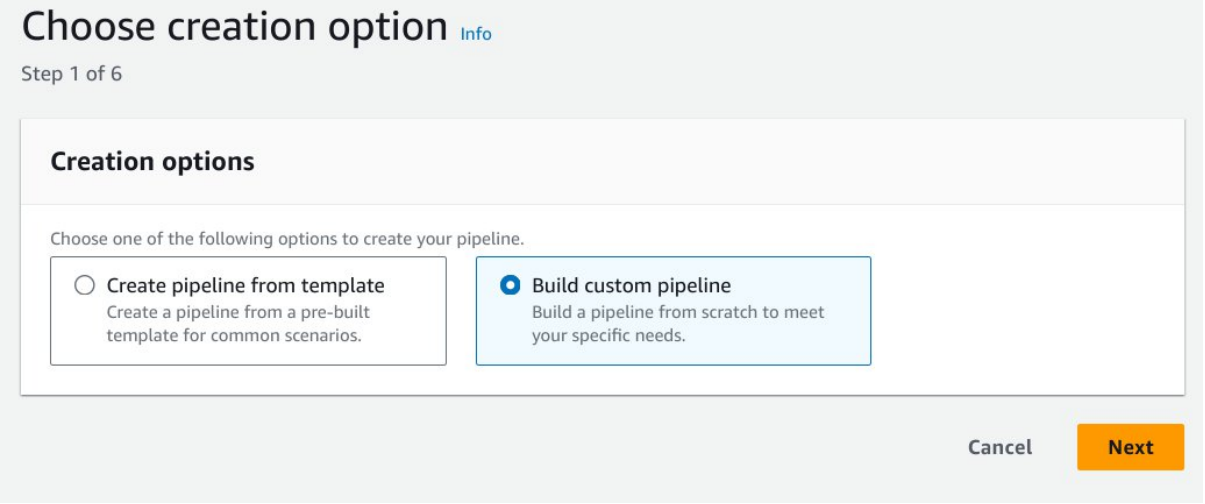
Build project created



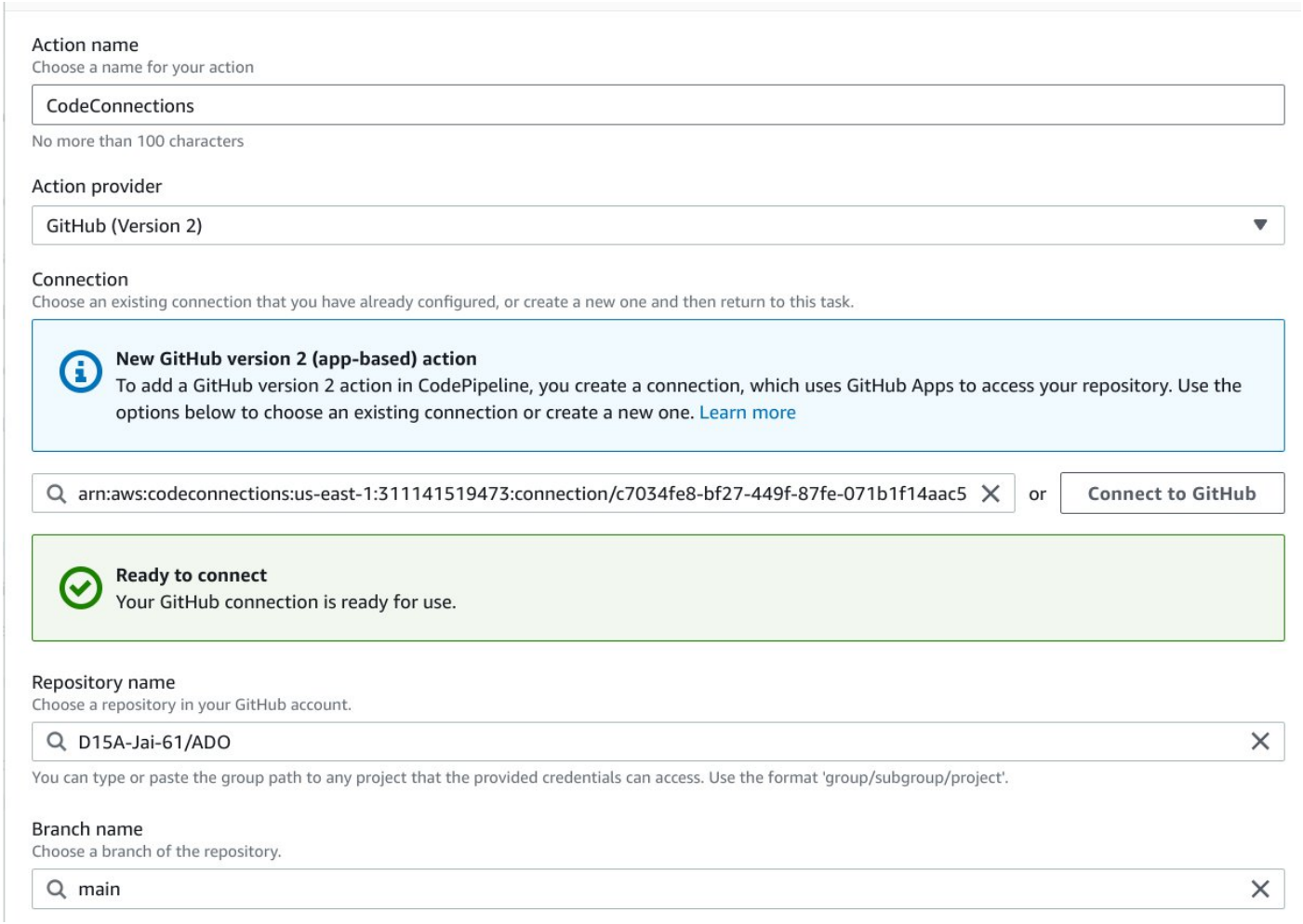
Build process succeeded and finished

Jai Talreja 61 D15A

Creating a custom Pipeline to deploy the built project

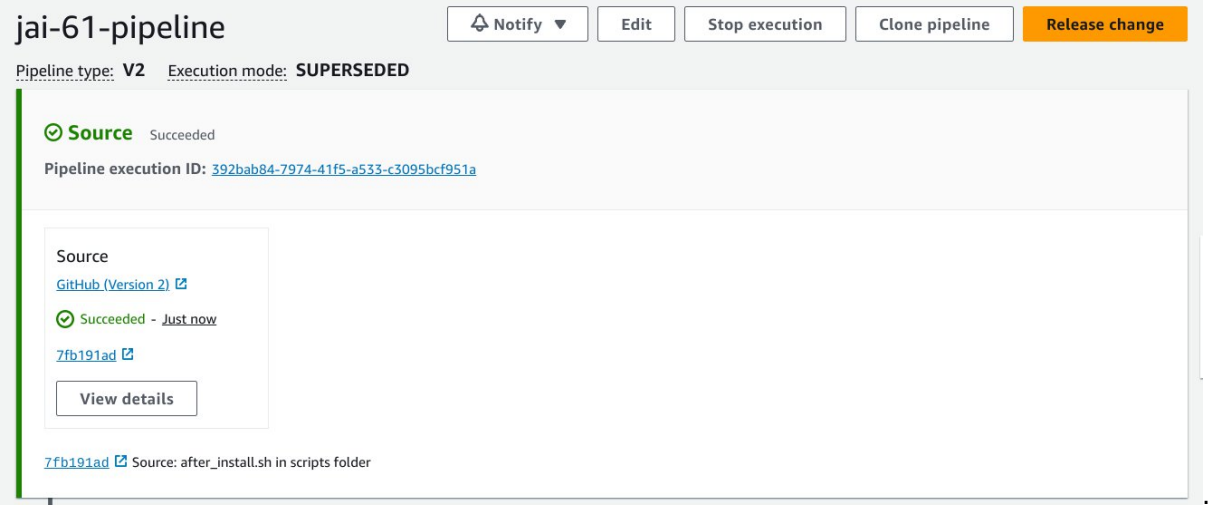


Pipeline settings

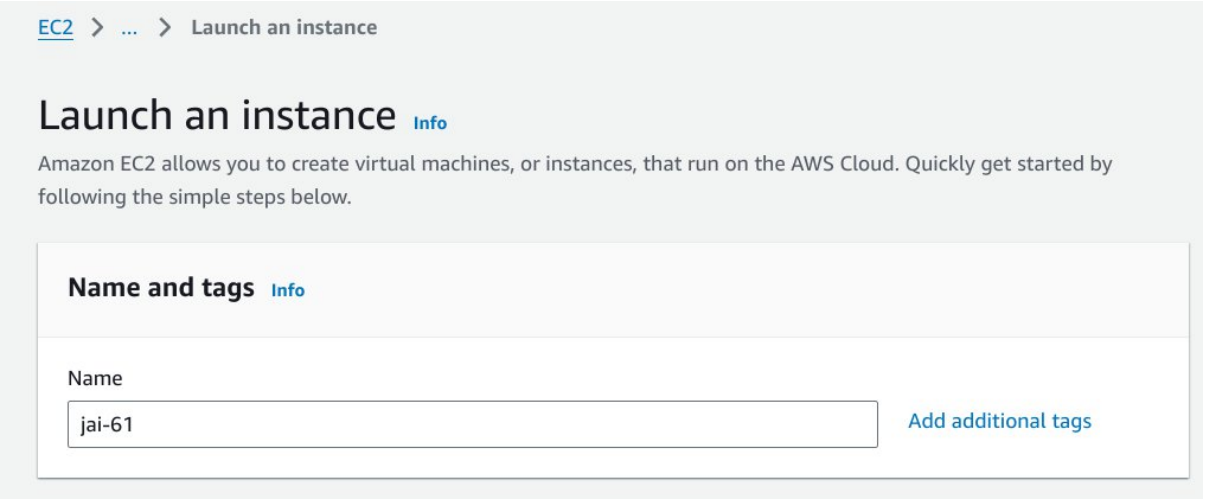


Jai Talreja 61 D15A

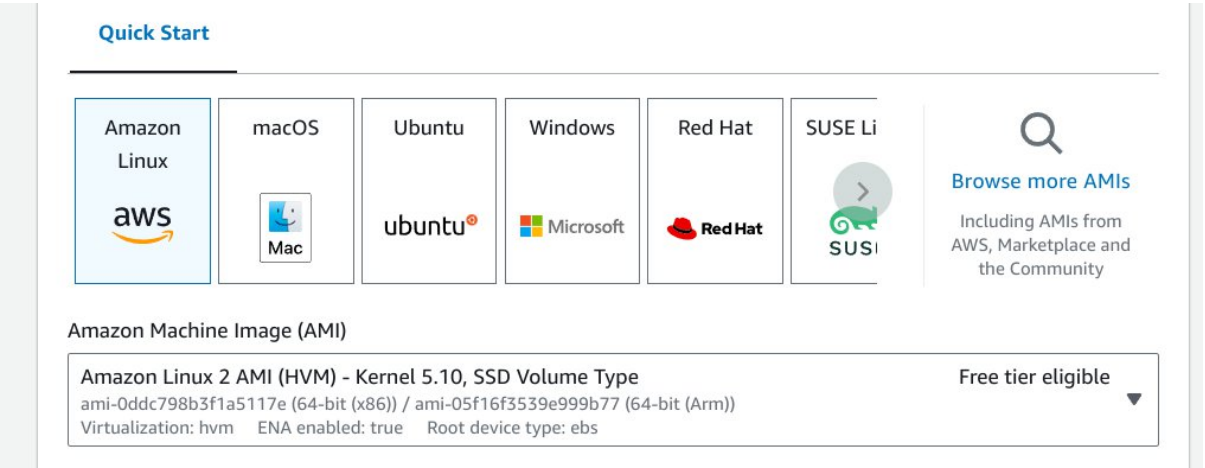
Pipeline successfully created



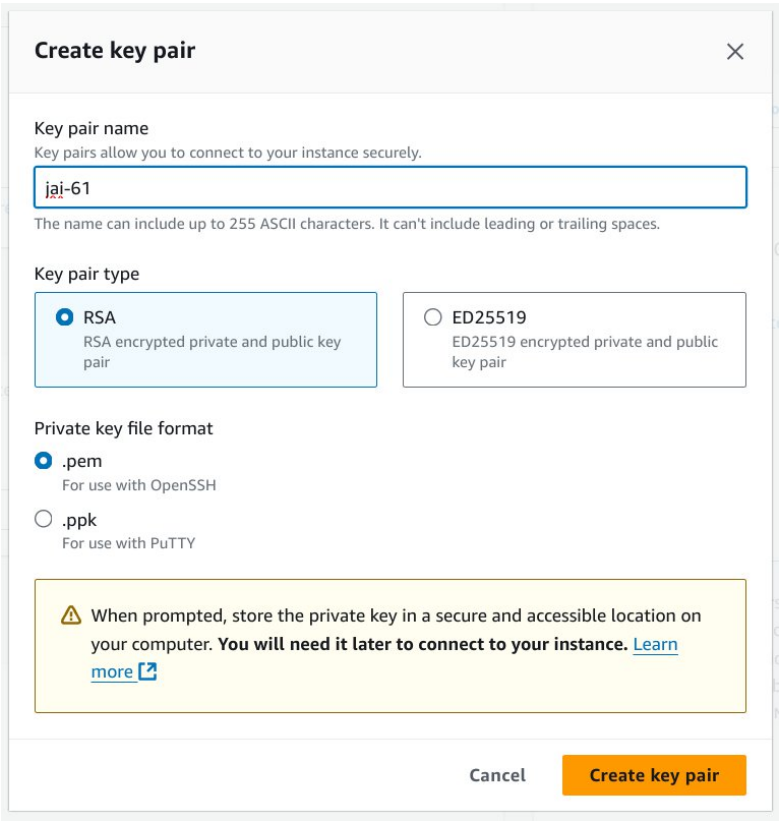
Launching an EC2 instance to deploy to and host the built WebPage



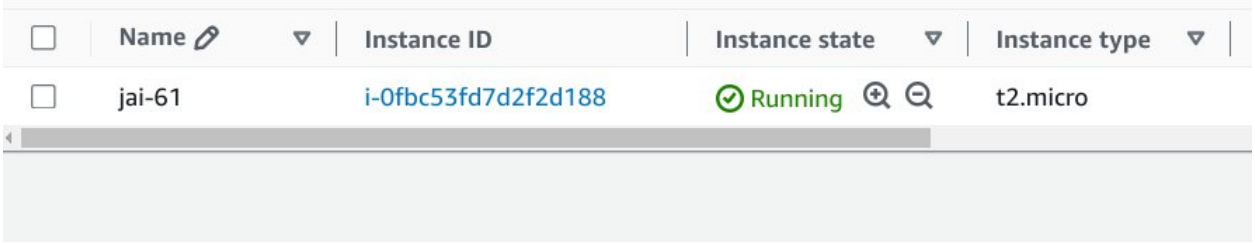
Amazon Linux as the Linux distribution of the EC2 instance



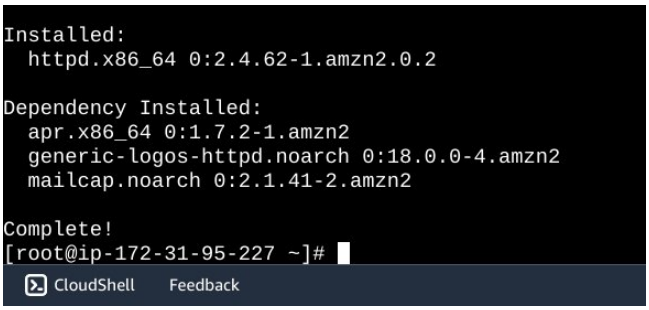
Jai Talreja 61 D15A
Key pair for EC2 instance



Instance created



Installed HTTP Daemon



Installed Ruby

Jai Talreja 61 D15A

HTTP Daemon status

```
[root@ip-172-31-95-227 ~]# systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2024-10-24 13:09:50 UTC; 39s ago
     Docs: man:httpd.service(8)
 Main PID: 3825 (httpd)
   Status: "Total requests: 0; Idle/Busy workers 100/0;Requests/sec: 0; Bytes served/sec:   0 B
   CGroup: /system.slice/httpd.service
           ├─3825 /usr/sbin/httpd -DFOREGROUND
           ├─3826 /usr/sbin/httpd -DFOREGROUND
           ├─3827 /usr/sbin/httpd -DFOREGROUND
           ├─3828 /usr/sbin/httpd -DFOREGROUND
           ├─3829 /usr/sbin/httpd -DFOREGROUND
           └─3830 /usr/sbin/httpd -DFOREGROUND

Oct 24 13:09:50 ip-172-31-95-227.ec2.internal systemd[1]: Starting The Apache HTTP Server...
Oct 24 13:09:50 ip-172-31-95-227.ec2.internal systemd[1]: Started The Apache HTTP Server.
[root@ip-172-31-95-227 ~]#
```

Installing AWS CLI on my personal machine running Fedora Linux

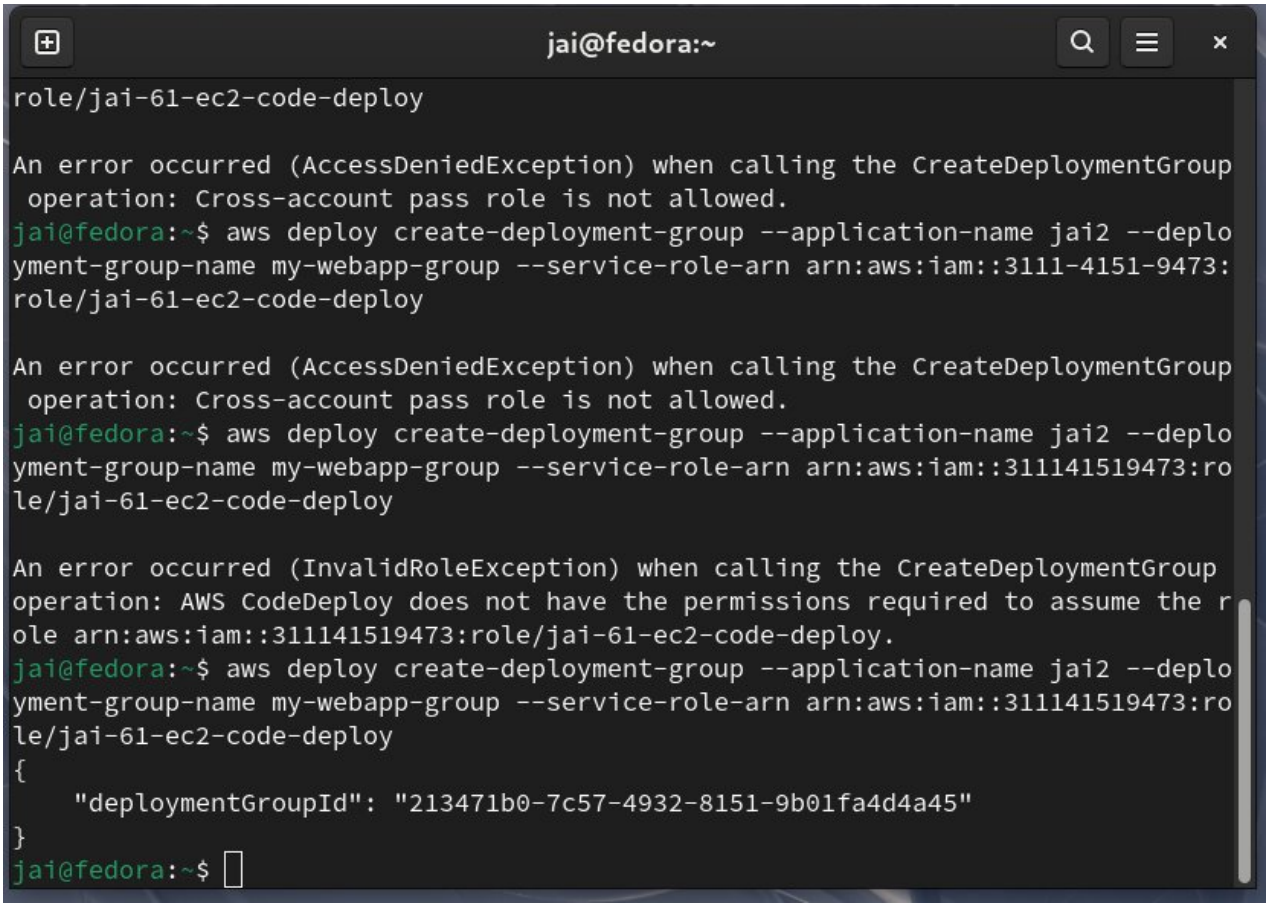
```
jai@fedora:~
jai@fedora:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

AWS CLI configured with Deploy command error that would be fixed later on

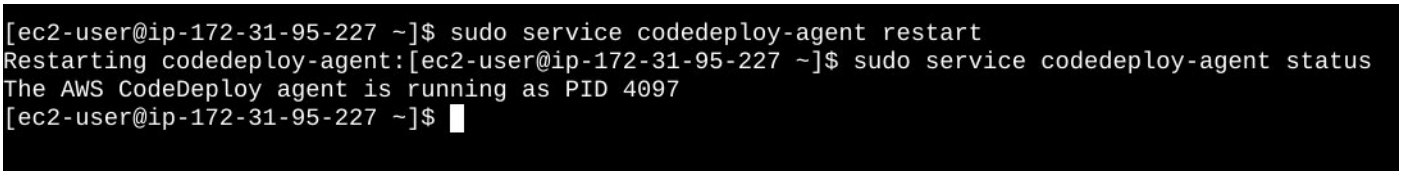
```
jai@fedora:~
jai@fedora:~$ aws configure
AWS Access Key ID [None]: AKIAUQ4L22BYUTTUFM5S
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-1
Default output format [None]: json
jai@fedora:~$ aws deploy create-application --application-name jai-61

An error occurred (AccessDeniedException) when calling the CreateApplication ope
ration: User: arn:aws:iam::311141519473:user/jai-61 is not authorized to perform
: codedeploy:CreateApplication on resource: arn:aws:codedeploy:us-east-1:3111415
19473:application:jai-61 because no identity-based policy allows the codedeploy:
CreateApplication action
jai@fedora:~$
```

Jai Talreja 61 D15A

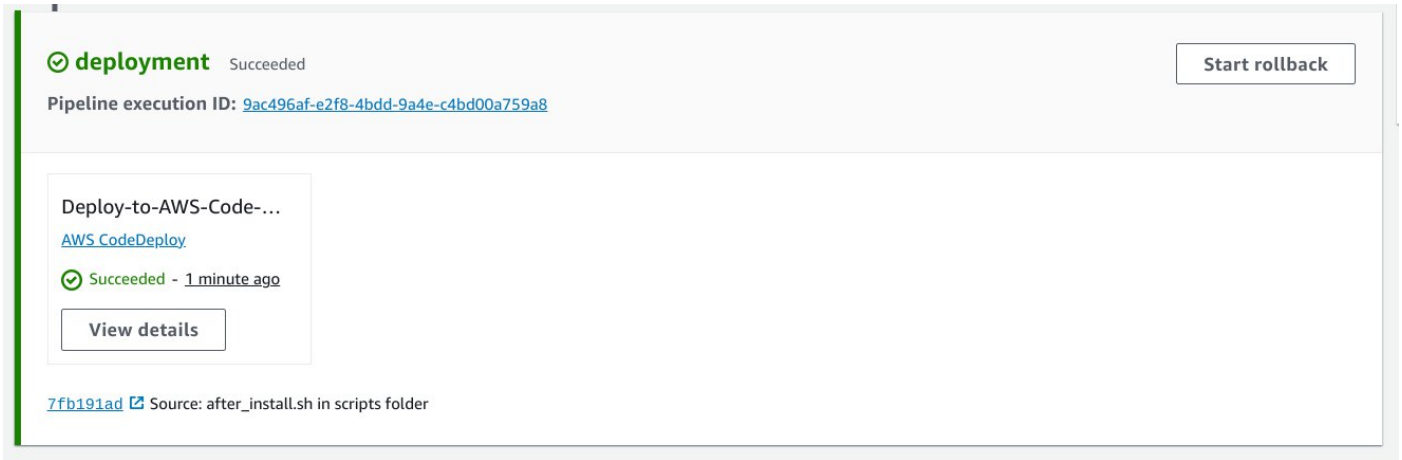
Deploy command now working



Amazon Deploy agent restarted on EC2 to reflect new changes in settings and permissions

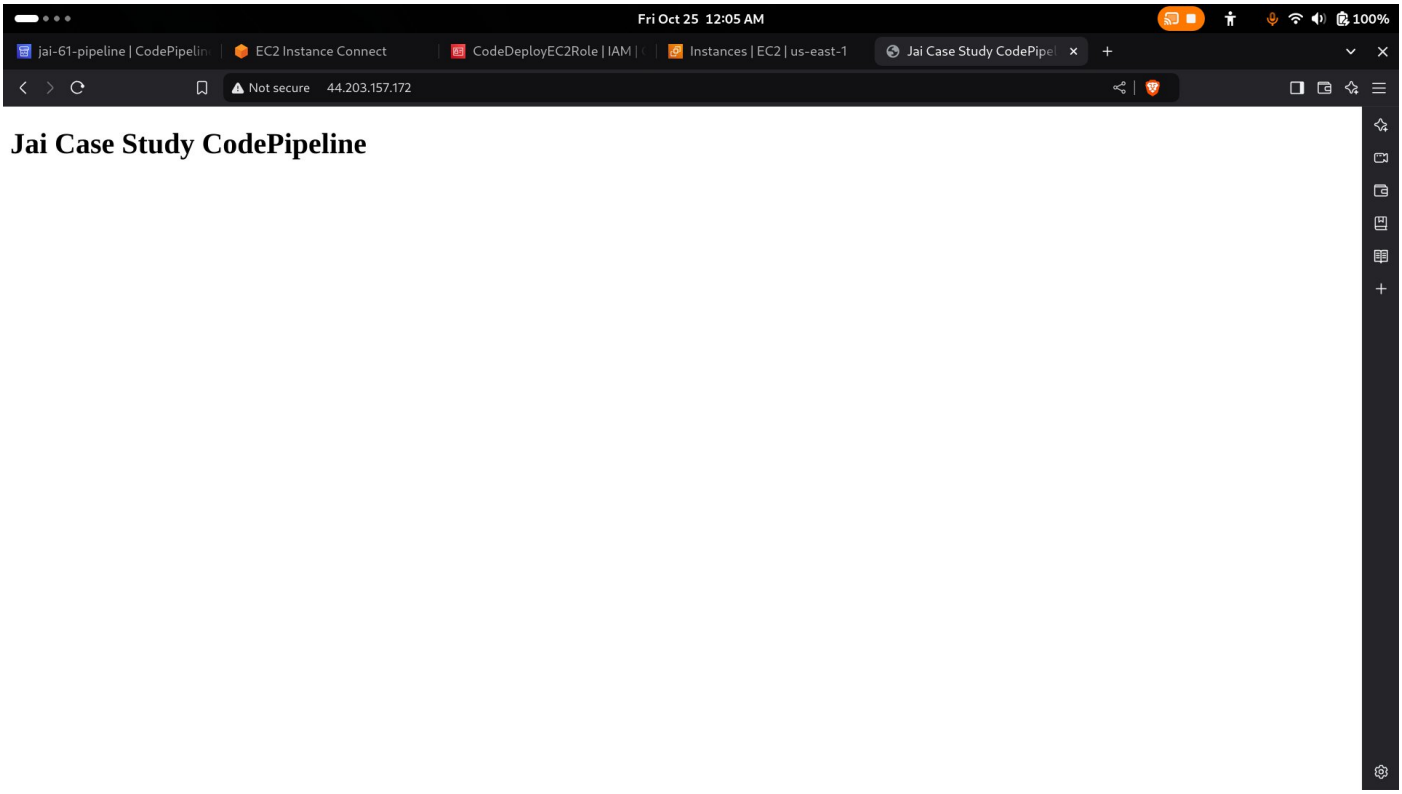


Deployment (final stage of pipeline) is successful



Jai Talreja 61 D15A

Connecting to the Public IP address of the EC2 instance via HTTP to see the result of the deployed WebApp



Learning

One of the things I have learned and has stuck with me is that the number of attached services besides permission policies in IAM roles does not refer to or define the number of services that can access it or are actually using it.

Other than that, this case study played an important role in widening the understanding of how Amazon Web Services works.

Conclusion

This case study was lengthy and tedious, some of the problems were caused due to the incomplete understanding of Amazon Web Services and how it works, with many possible variations in configurations for each and every step.

However, the case study was completed successfully, the issues faced were fixed in time, and the entire process is automated from start to end, starting from source file gathering to building, to deployment, and finally ending at hosting, everything is achieved without further user input.