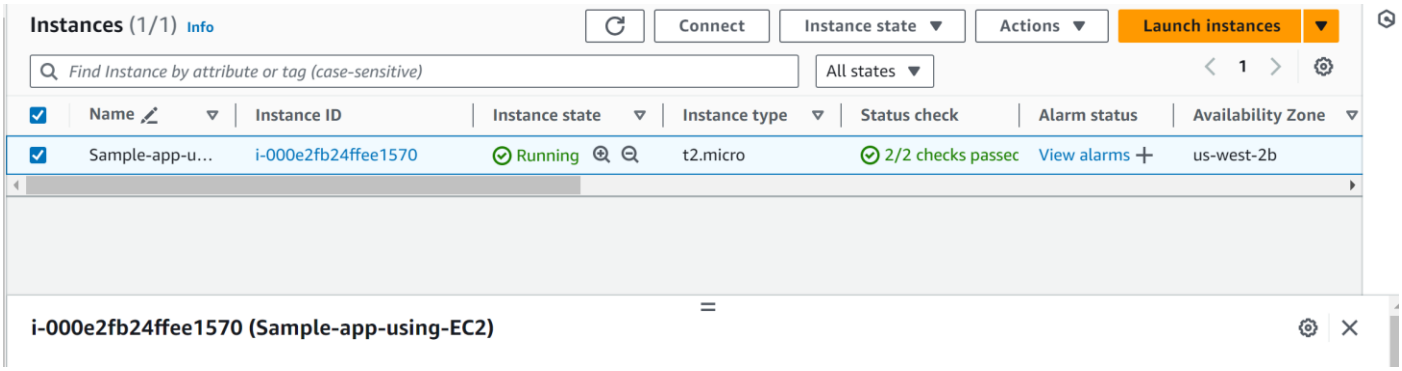
Name: Pranav Chaudhari

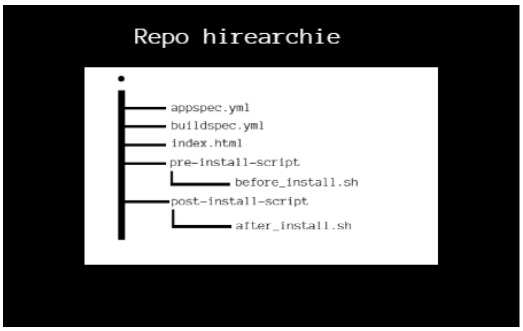
Task: Create a sample app and automate CI/CD pipeline with AWS Services such as CodeBuild, CodeDeploy, CodePipeline and EC2.

Github Repo: https://github.com/techdecipher/using-aws-dev-tools

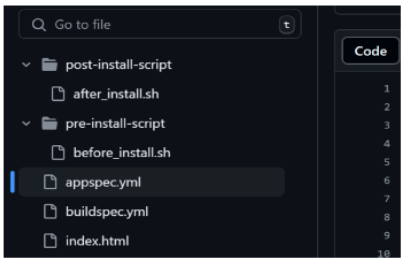
Step 1) Launch an EC2 instance, give proper SG groups and key pairs as needed. I am using AMZ linux 2023



Step 2) Setup necessary files on Git repo with respect to CodeBuild (buildspec.yml), CodeDeploy (appspec.yml), and index file for the sample page. Along with this, we would need scripts like before_install.sh and after_install.sh to automate apache installation and necessary steps that we might need to be done before the deploy goes in. Unfortunately, due to alignment and space allocation issue, cant attach all the scripts, but it can be found on the git repo shared above.

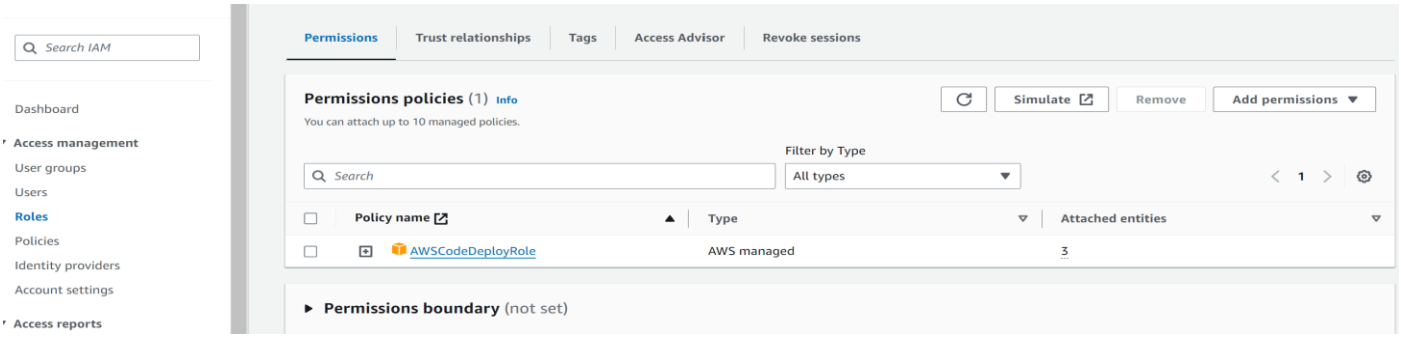


Repo on git

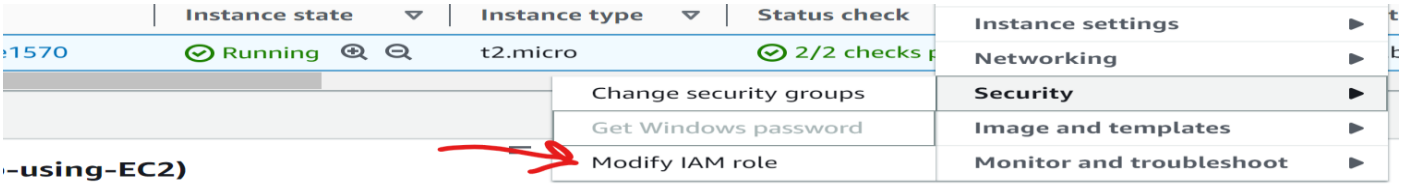


Step 3) Create IAM roles for CodeDeploy, just before going over to further steps, ensure a new role with respect to CodeDeploy is created and attach it to the EC2 created in Step 1.

IAM > Roles > Create Role > CodeDeploy > Next > Create



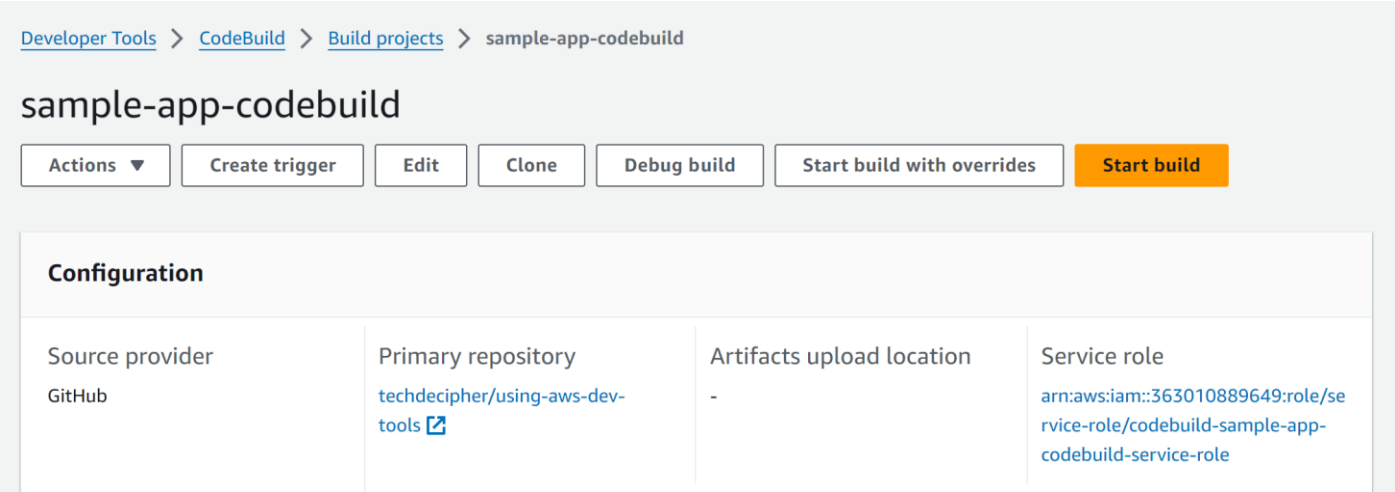
EC2 > Action > Security > Modify IAM role



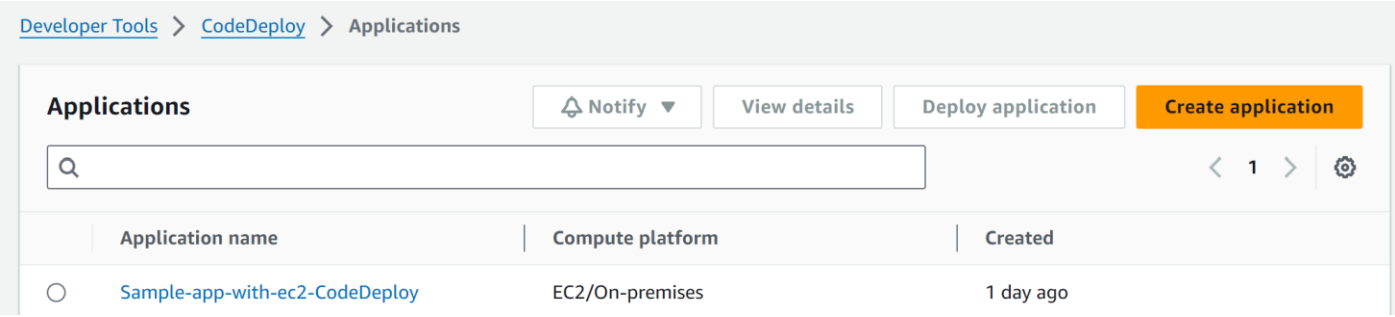
Step 4) Configure CodeBuild as It automates the build process, ensuring that code is compiled and packaged consistently.

CodeBuild > Create Project >

Project name: sample-app-codebuild
Source: GitHub
GitHub repository: using-aws-dev-tools
>>Create Build Project



Step 5) Configure CodeDeploy to automate the deployment process, ensuring application is deployed reliably and consistently. Going over to CodeDeploy and creating an application. Application name: Sample-app-with-ec2-CodeDeploy and Compute Platform: EC2/On-premises.



Setup the Deployment group as it is needed and it specifies the target instances for deployment and manages the deployment settings. CodeDeploy > Create Deployment Group >

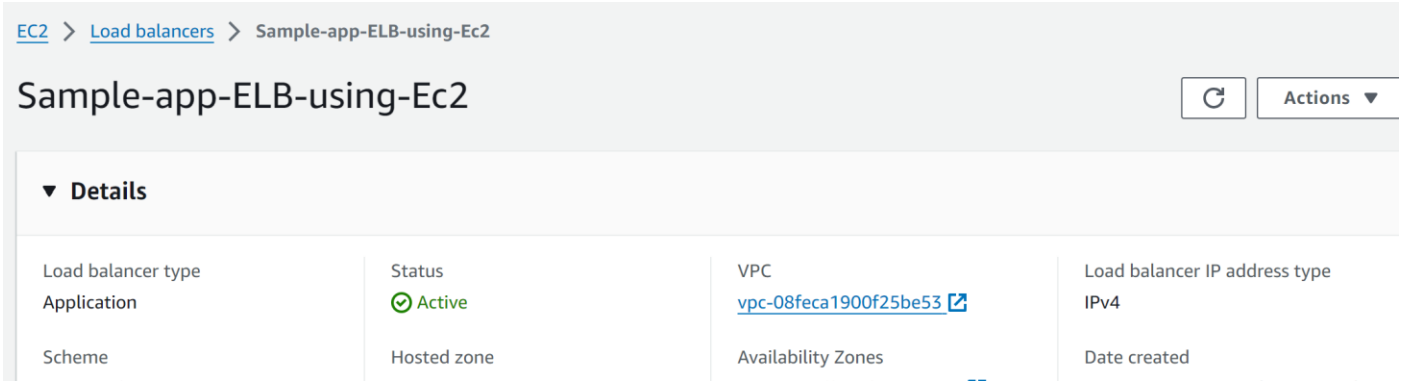
Deployment group name: Sample-app-deployment-group
Service role: role name
Deployment type: in-place
Environment configuration: Amazon EC2 instances
Add the necessary instance with respect to the instance Id in Env Configuration.
Load balancer: Application Load Balancer or Network Load Balancer

As needed in our sample project, while stage of configuring Deployment group we will setup ALB for it.

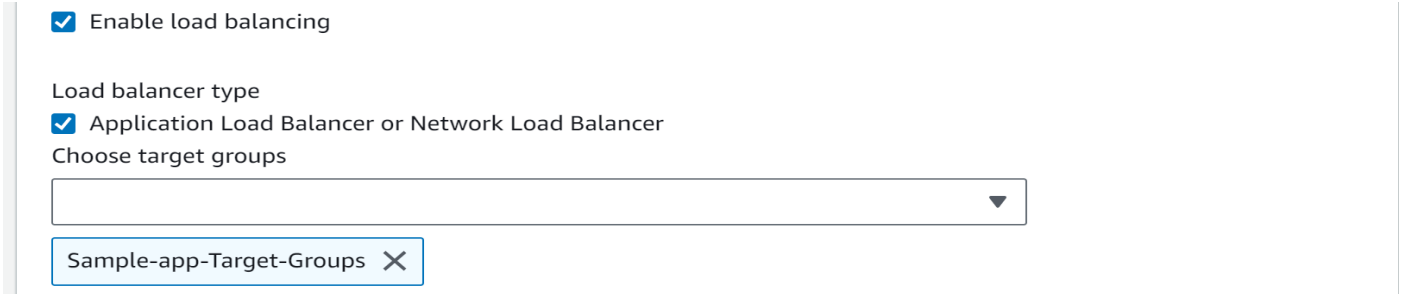
Step 5.1) Configure ALB for finishing CodeDeploy's Deployment Group completion as we needed it with ALB.

Load Balancers > create it with following attributes.

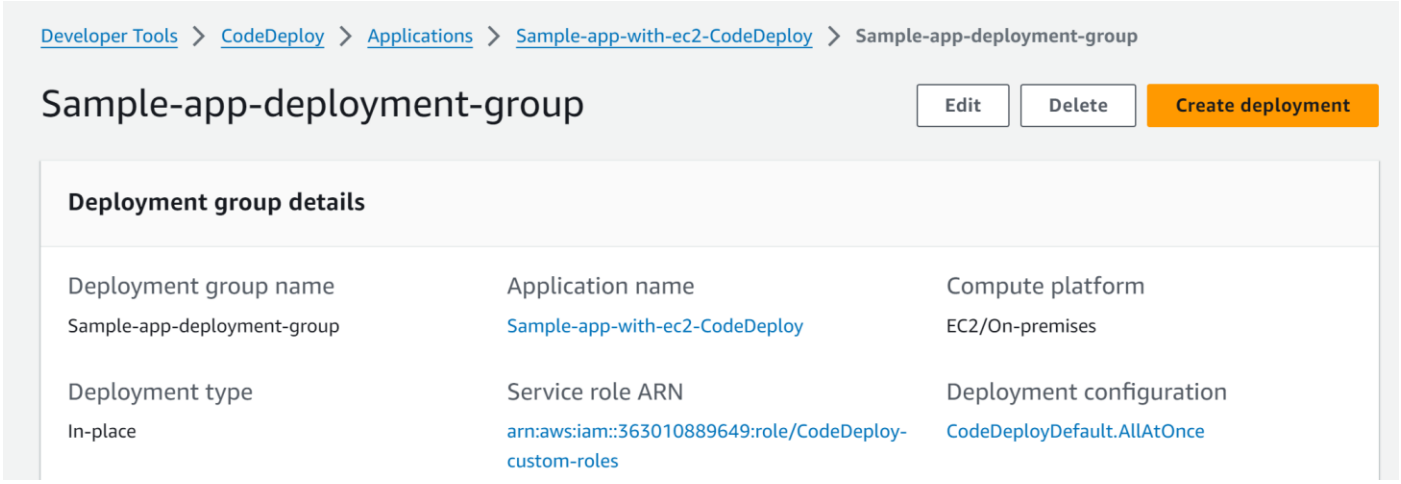
Load balancer types: Application Load Balancer
Load balancer name: Sample-app-ELB-using-Ec2
Scheme: Internet-facing
Load balancer IP address type: IPv4
Network mapping: check all
SG: Default
Listeners and routing: Create the Target Group and add it here > create Load balancer



Step 5.2) Continue Deployment Group configuration after creating ALB, goto CodeDeploy and resume Deployment Group configuration.



Final look of CodeDeploy with Deployment group created.

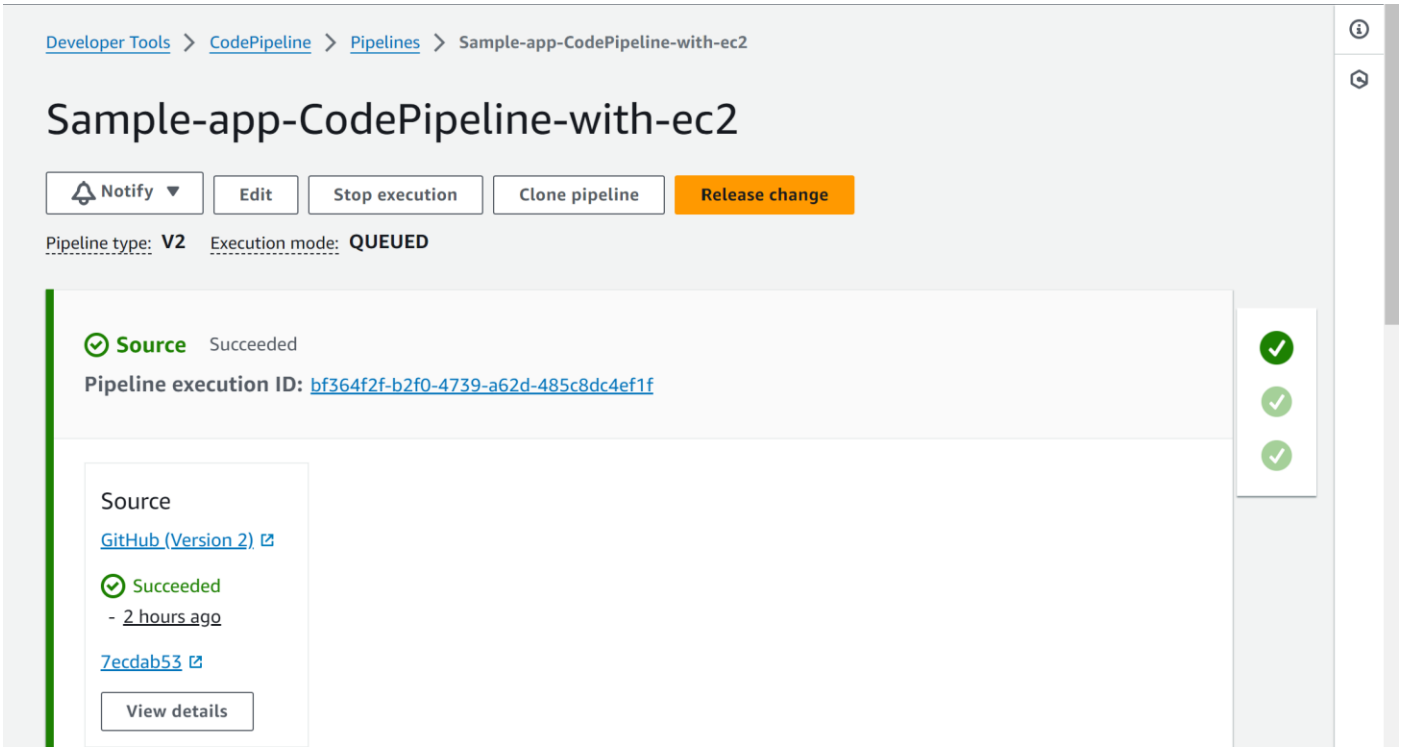


Step 6) Configure CodePipeline as it helps in integrating all the components to automate the release of new code from commit to deployment.

CodePipeline>

Pipeline name: Sample-app-CodePipeline-with-ec2
Service role: New service role >> next >>
Source provider: GitHub v2
Repository name: using-aws-dev-tools
Default branch : main
>> next >>
Build provider: CodeBuild
Project name: CodeBuilds project name
Deploy provider: CodeDeploy
Application name: CodeDeploy's app name
Deployment group: Deployment group created on CodeDeploy
>> Create Pipeline >>

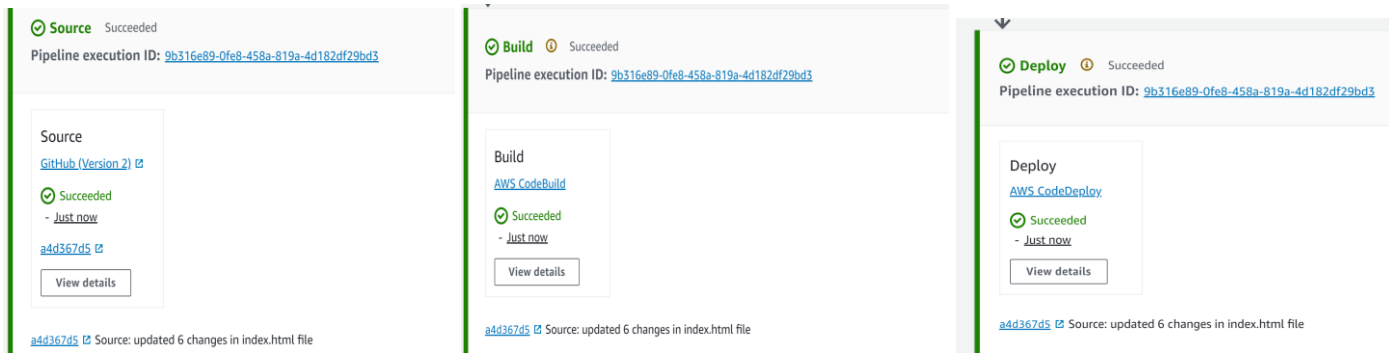
Final look of CodePipeline



Step 7) Making changes to our repository triggers the Source fetch, Build, Deploy, for example updating the text over the index.html web page and pushing changes over.

```
[root@ip-172-31-29-10 using-aws-dev-tools]# vim index.html
[root@ip-172-31-29-10 using-aws-dev-tools]# git add *
[root@ip-172-31-29-10 using-aws-dev-tools]# git commit -m "updated 6 changes in index.html file"
[main a4d367d] updated 6 changes in index.html file
 1 file changed, 1 insertion(+), 1 deletion(-)
[root@ip-172-31-29-10 using-aws-dev-tools]# git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 312 bytes | 312.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/techdecipher/using-aws-dev-tools
   7ecdab5..a4d367d  main -> main
[root@ip-172-31-29-10 using-aws-dev-tools]#
```

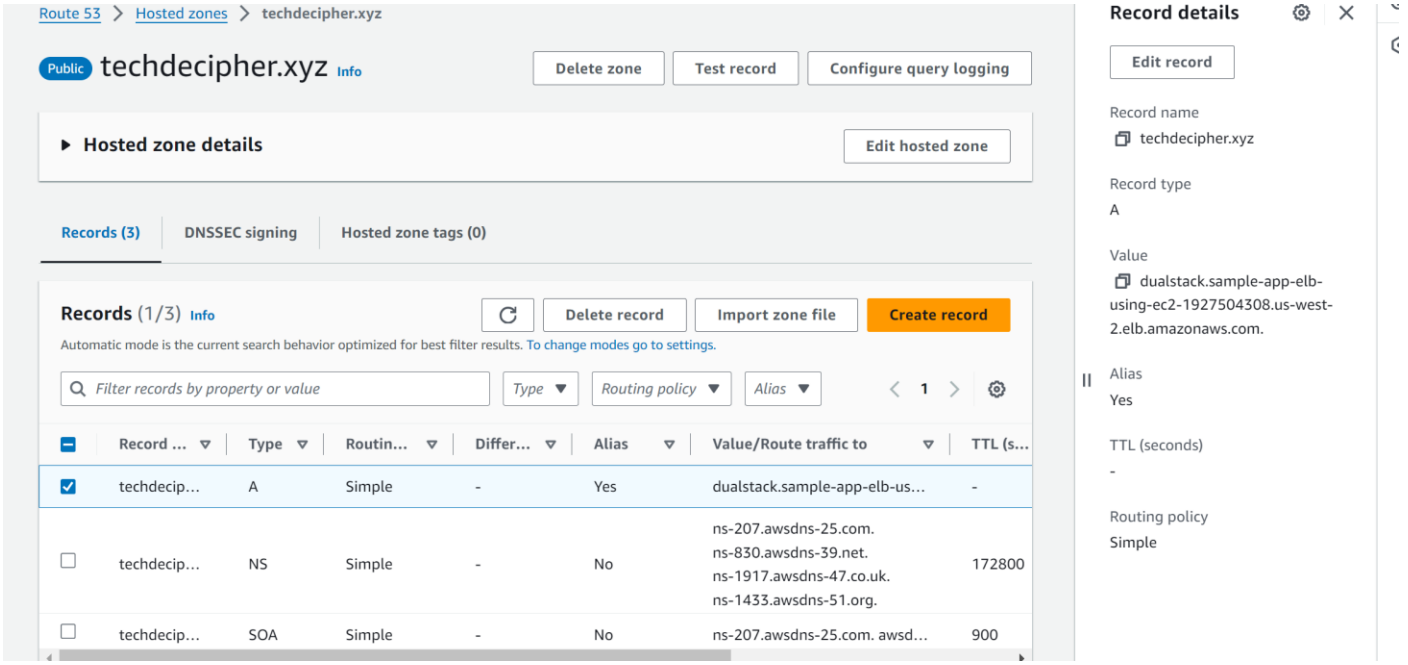
>> Source >> Build >> Deploy >>



This pipeline is working, I already setup my custom domain http://techdecipher.xyz/ on Route53 so took LoadBalancers DNS name and updated in the records so I can access it on more user friendly domain name.

Optional step > Route 53> Hosted zones> techdecipher.xyz > Create record

Route 53> Hosted zones> techdecipher.xyz > Create record Alias: enabled Route traffic to: ALB > Region: (from the list) the region where it is created. Choose Load Balancer (from the list) > Simple Routing >> Save



The final web page looks like this.

So now, anytime any change is required, update the necessary files, and then just commit, and push changes, it should automatically fetch changes, build and deploy the changes on EC2 instance.