**CICD Using ECS S3 CodeCommit CodeBuild & CodePipeline**

**Prerequisite :**

* AWS CodeCommit
* AWS CodeBuild
* AWS CodePipeline
* AWS IAM

## **Create CodeCommit Repository**

* Create Code Commit Repository with name as **ecs-cicd-nginx**
* Create git credentials from IAM Service and make a note of those credentials.
* Clone the git repository from Code Commit to local repository

**Create Two File & Upload It On CodeCommit**

* Dockerfile
* index.html

sudo nano Docker

FROM nginx

COPY index.html /usr/share/nginx/html

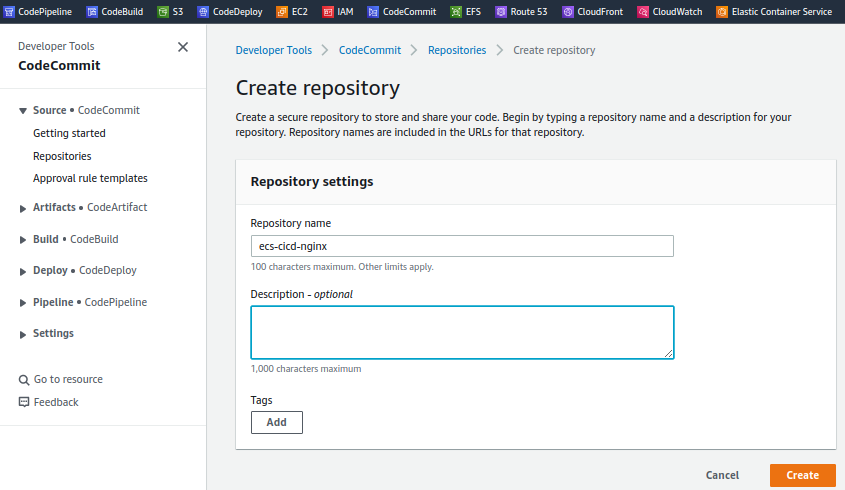
sudo nano index.html

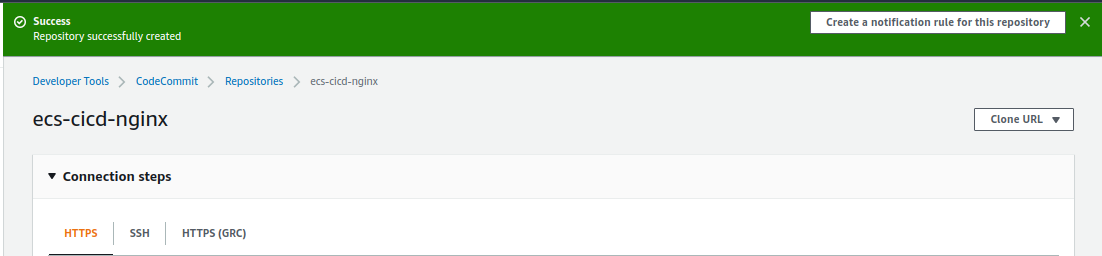
Continuous Integration & Continuous Deployment

* Commit code and Push to CodeCommit Repo

Go to the AWS console search for AWS DeveloperTool - - - > choose **CodeCommit** service & follow the steps, go ahead & create CodeCommit Repository.

Repository Name : **ecs-cicd-nginx**





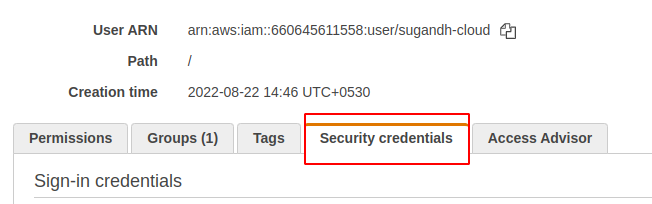
1. **Prerequisites :**

Now once that is done in your local machine, please make sure you have that git client present & then if you can go here to [view](https://git-scm.com/downloads), downloads page & then download for MacOS, Windows, Linux & then install it on your local machine.

1. **Git credentials**

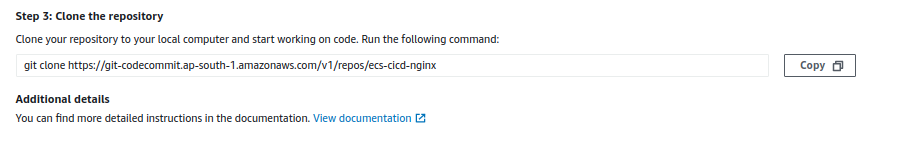
Go ahead & then create Git Credential for our respective IAM user

Go back to IAM service - - - > go to users - - - > & go to sugandh-cloud user - - - > go to security credencial.



Go & generate new credentials then download credencial - - - > now go to the CodeCommit repo.

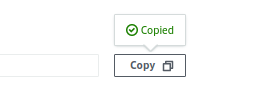
1. **Clone the repository**

****

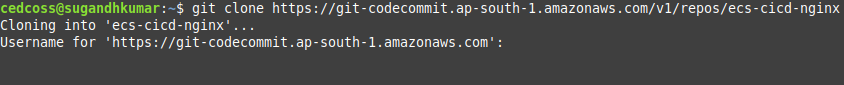
git clone <https://git-codecommit.ap-south-1.amazonaws.com/v1/repos/ecs-cicd-nginx>

Copy this then get a clone & then complete the repository so let's go back to the command line.

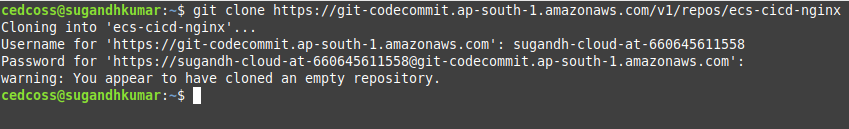
Copy CodeCommit repository link & clone it on your local machine.



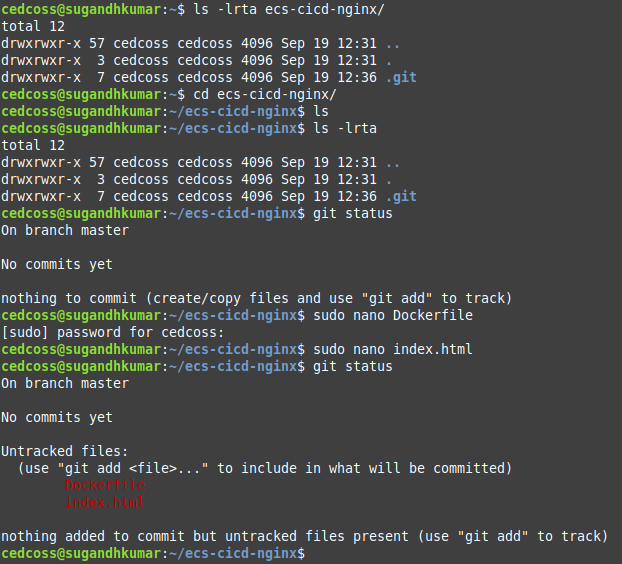
git clone <https://git-codecommit.ap-south-1.amazonaws.com/v1/repos/ecs-cicd-nginx>



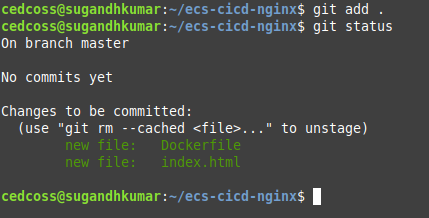
Now we should have to open our CodeCommit IAM credential & enter the username & password, for cloning this.



We successfully cloned an empty git repository, So now what we need to do is let’s go ahead & add some data here.

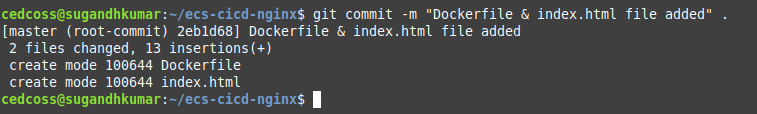


Now track all the files using **git add .**

****

Now commit & push the data to CodeCommit repository.

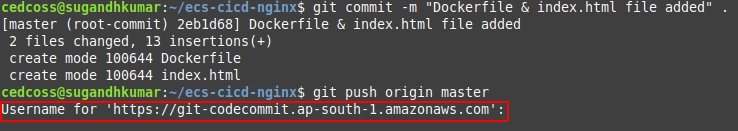
***git commit -m “Dockerfile & index.html file added” .***



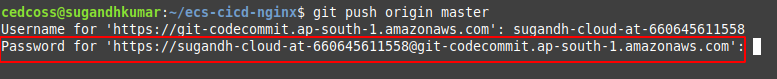
Now push the data to master branch of CodeCommit repository

***git push origin master***

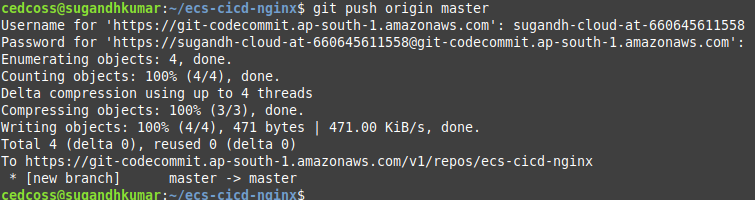
After this command it will again ask you about UserName & Password for security purposes.



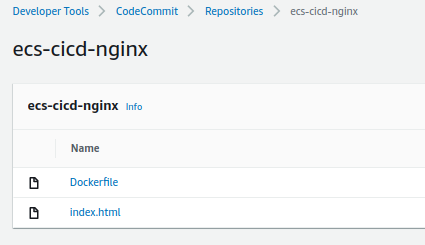
Enter the required things & push the data to your respective repository.



After accepting UserName it will ask you about Password, enter the password.



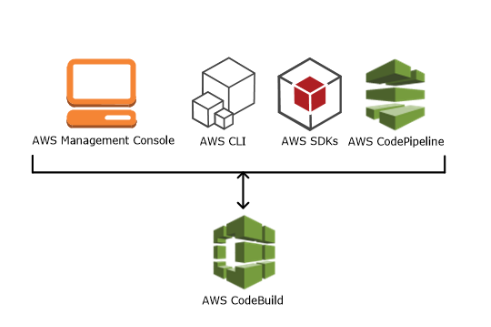
Done! Go to aws console - - - > CodeCommit check your repository.



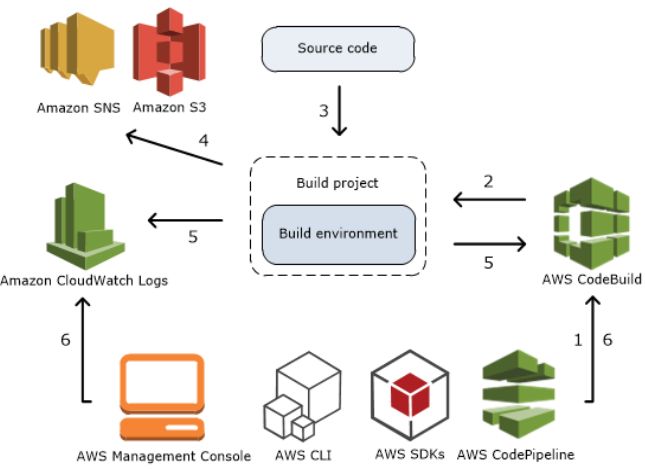
**AWS CodeBuild**

* CodeBuild is a fully managed build service in the cloud.
* Compiles our source code, runs unit tests & produces artifacts that are ready to deploy.
* Eliminates the needs to provision, manage & scale our own build servers.
* It provides pre-packaged build environments for most popular programming languages & build tools such as Apache Maven, Gradle & many more.
* We can also customize build environments in CodeBuild to use our own build tools.
* Scale automatically to meet peak build requests.

**How to run CodeBuild**



**How CodeBuild Works Generally**



From CodeBuild's perspective, the core things are the build project & the build environment, so we create a build project & inside that build environment.

CodeBuild picks up our source code from the version control system, which is nothing but CodeCommit & then that build environment, it will start processing our build based on the **buildspec.yml** If a java package are spring boot application, say you say **clean package , mvn clean package , mvn build** whatever document we run there.

So based on that, whatever is specified in **buildspec.yml** using the build environment, it runs & then generates the build artifacts.

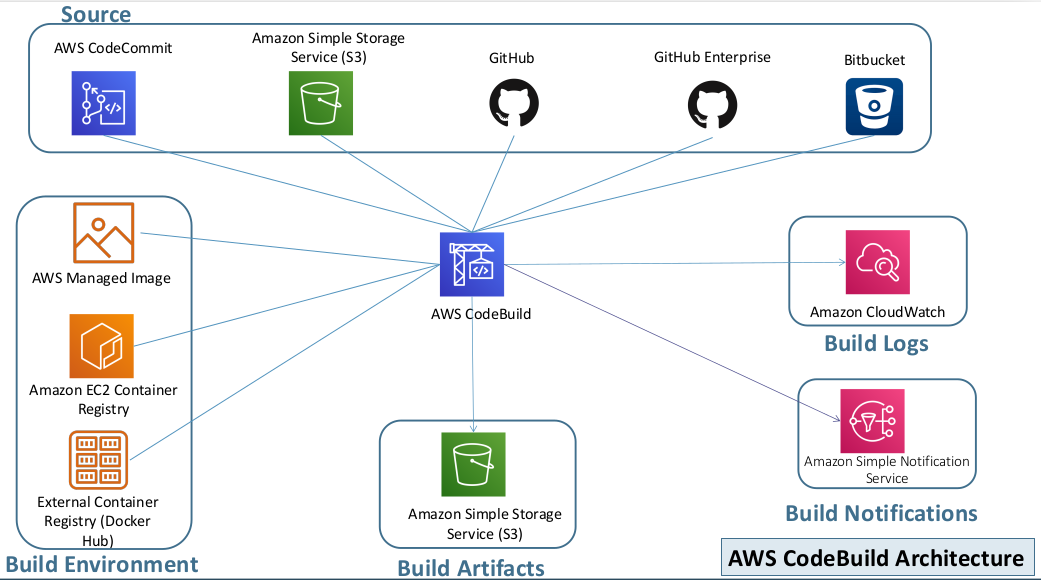
So if the build artifacts are generated, it is going to store those artifacts in AWS S3 Buckets & in addition, it also stored logs during the process, meaning it generates logs during the build process.

So those will be stored in the AWS CloudWatch logs.

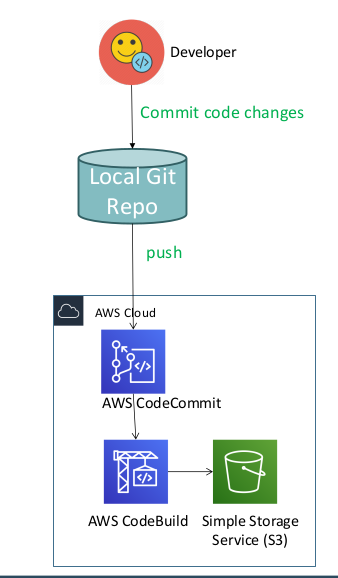
In addition our building fails, if our building is passed if our build has some issues whatever it is, based on the event, respective events if we configure the notification using **AWS** **SNS** (**Simple Notification Service)** it can even send emails for us during the build process.

So we can events extend these events to event do some other activities, like if this event is triggered then & rebuild & then redeployment or whatever, all those things also we can do and then make the things more advanced & then advancer.

**AWS CodeBuild Artitecture**

****

**Build Steps**

****

**Create buildspec.yml for CodeBuild**

* Create a new repository in Elastic Container Registry (ECR) with name as **ecs-cicd-nginx** and make a note of ECR Repository full name.
* Create buildspec.yml file in local desktop folder **ecs-cicd-nginx.**
* Update **buildspec.yml** file
* Update REPOSITORY\_URI value with complete ECR Repository name
* Update the **Container Name** at printf **'[{"name":"ecs-cicd-nginx"** in buildspec.yml
* **Important Note:** In ECS Task Definition also when we are creating it, please ensure we give the container name as **ecs-cicd-nginx**

### **buildspec.yml**

[Click the link to see buildspec.yml](https://docs.google.com/document/d/15P-Bv7T535y1HFqE1wFpGuSFObDnK2aYoHpRnvvbZzE/edit?usp=sharing)

sudo nano buildspec.yml

**Push the updated code to CodeCommit Repository.**

git status

git add .

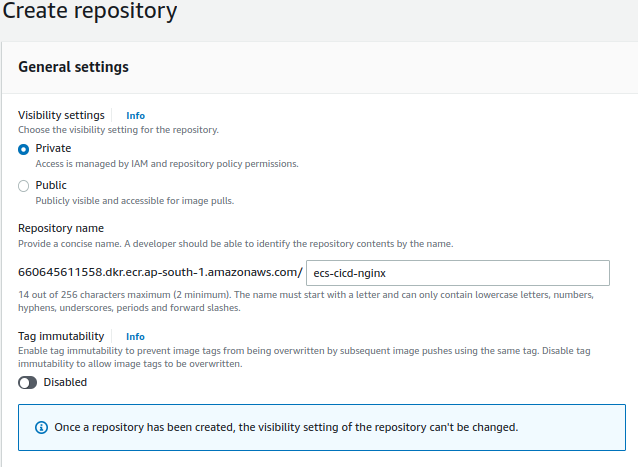
git commit -am "2-Added buildspec.yml"

git push

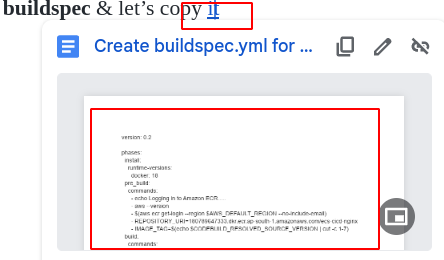
git status

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

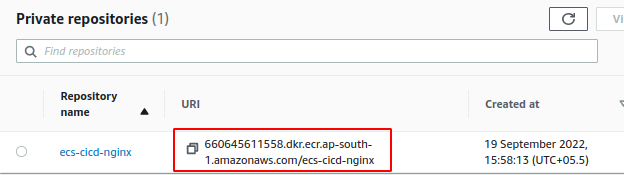
Go to AWS console select AWS ECR - - - > Create Repository - - - - **ecr-cicd-nginx**

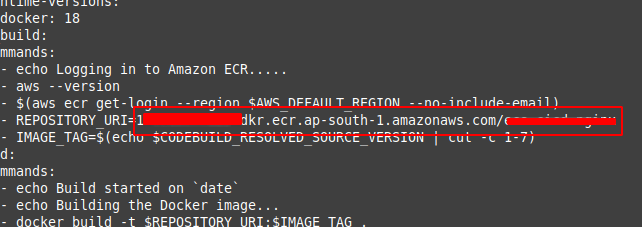


Let’s go back terminal local machine then create **buildspec** & let’s copy [it](https://docs.google.com/document/d/15P-Bv7T535y1HFqE1wFpGuSFObDnK2aYoHpRnvvbZzE/edit?usp=sharing)

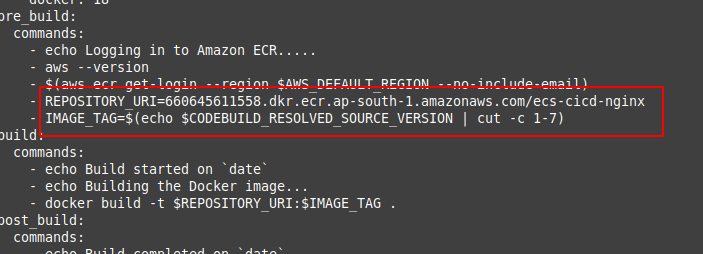


Now I need to update two things, **1 : Repository URI** so whatever we have seen here same URI (go to ECR copy the URI)

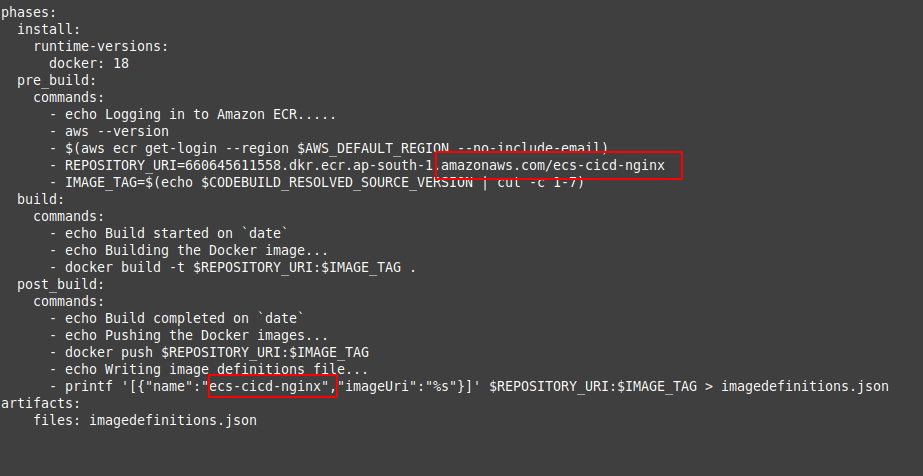




Paste the URI in place of old repository URI

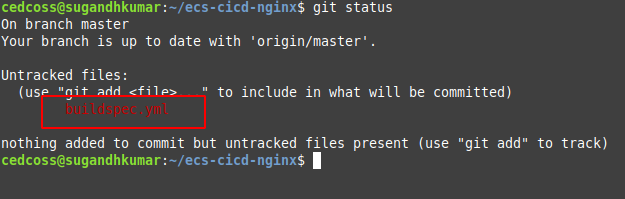


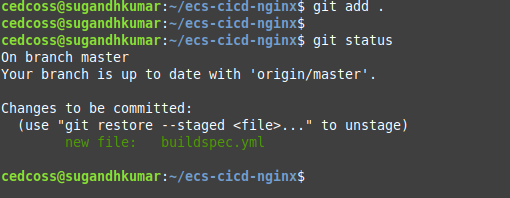
And the 2nd is the name of the container (ensure that the repository & container name both should be the same).



Now we have completed all the changes of our buildspec.yml repository URI & then container name, So let's go CLI push the buildspec file to remote repository.

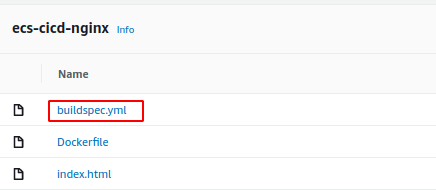
*git status*





Do the same things as we did previously.

Now go to ECR & check our buildspec file successfully pushed, including two others.



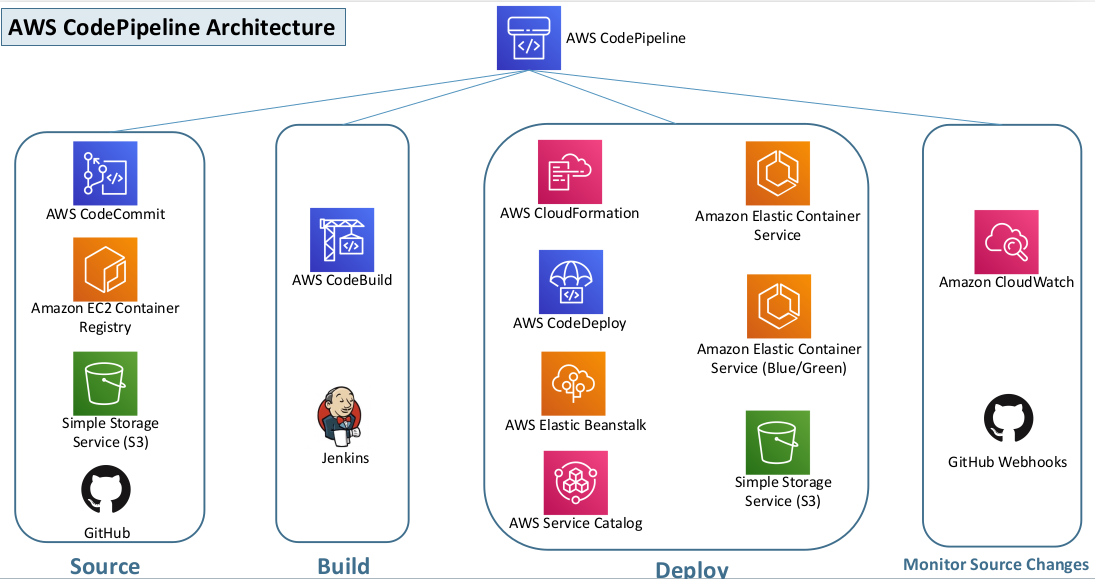
**AWS CodePipeline**

* AWS CodePipeline is a continuous delivery service to model, visualize, & automate the steps required to release your software .

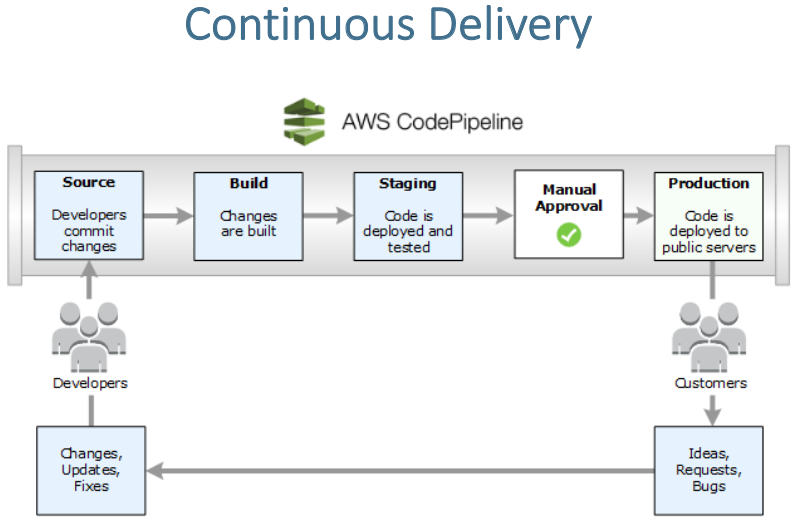
**Benefits**

* We can automate our release processes.
* We can establish a consistent release process.
* We can speed up delivery while improving quality.
* Supports external tools integration for source, build & deploy.
* View progress at a glance
* View pipeline history details.

**AWS CodePipeline Architecture**

****

**Continuous Delivery**

****

From the Continuous Delivery perspective inside CodePipeline what are these stages included? So that is this slide about.

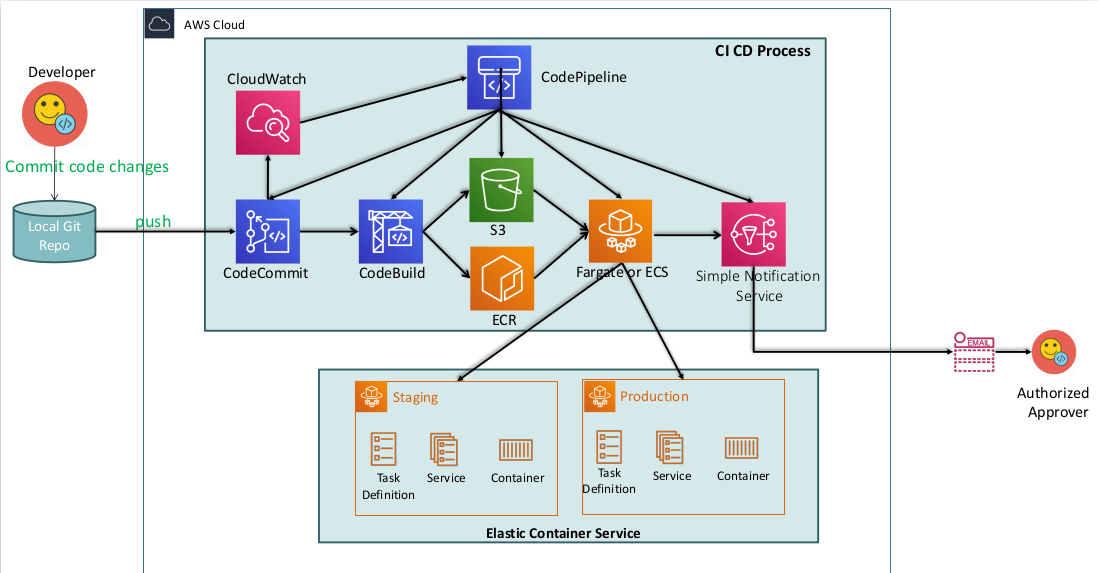
The **Source**  - - - **Build - - - Staging**  - - - **Manual Approval** - - - **Production.**

So these are the five stages which can build for the total production deployment without any issues like sources is get to the code from our CodeCommit repository build using the CodeBuild, we build the artifacts and the stored to S3 & then we will deploy after build will usually do the deployment to the staging environment.

Once the deployment staging happens, we need to deploy to production ‘But How?’

We need a manual approval process AWS has define a manual approval process where is its trigger an email to the respective approval & then he/she can click on the link to approve or reject the build, whether he/she want to promote to production or not, If this is approved automatically, it will be deployed to the production environment.

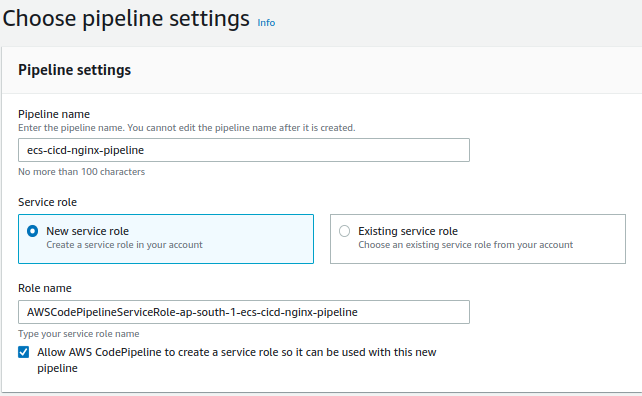
-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

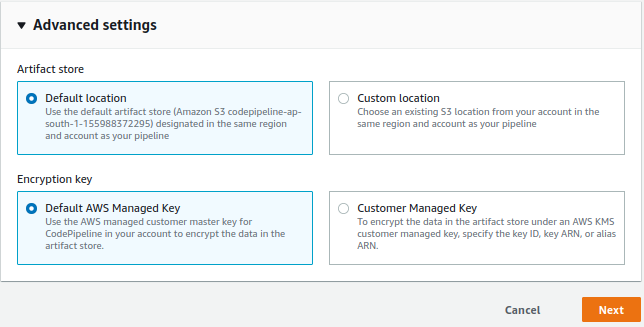


## **Create CodePipeline**

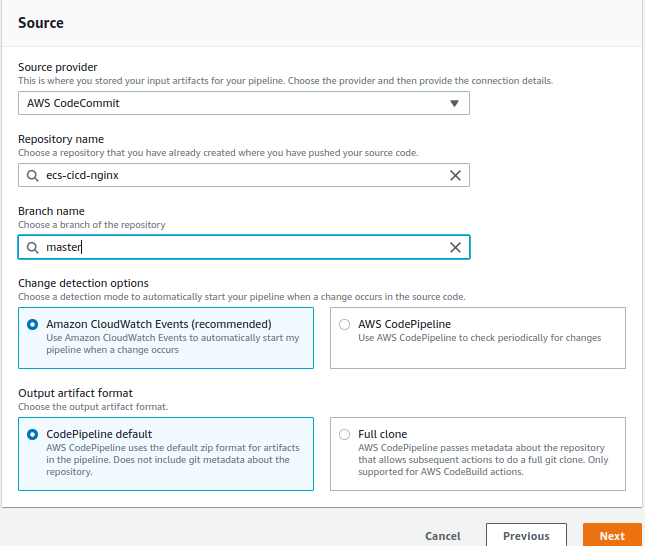
* Create CodePipeline
* Update the CodeBuild Role to have access to ECR to upload images built by codeBuild.
* Policy Name: AmazonEC2ContainerRegistryFullAccess
* Test by accessing the static html page

Go to AWS console - - -> AWS Pipeline - - - > Create Pipeline - - - **ecs-cicd-nginx-pipeline**

****

****

Now the source provider is **CodeCommit**



And our repository name in CodeCommit is **ecs-cicd-nginx** & branch is going to be **master**

Now we have to things **Change detection option :**

Whenever a change happens to overcode a commit repository, whenever we check in the code , there are two ways for the system to identify and select the event needed to identify & then trigger the CodePipeline complete process.

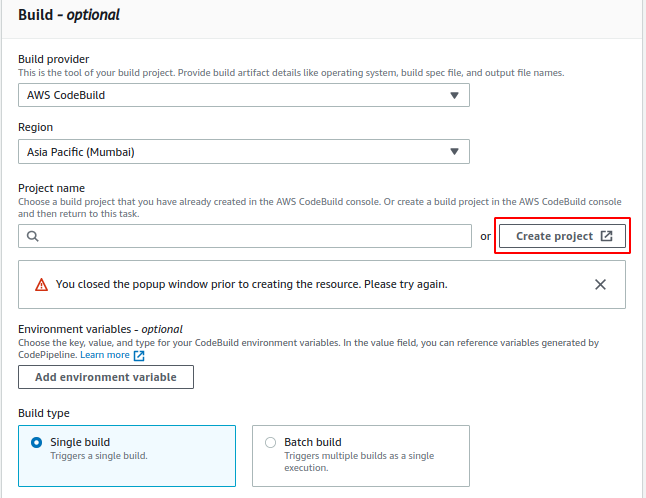
1 : Amazon **CloudWatch** events & other is **CodePipeline**, so using CodePipeline periodically check for the changes which is a bad option,

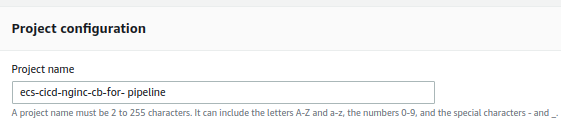
Even when an event is triggered, it’s like whenever the CodeCommit checking event is triggered immediately triggers the pipeline, so this is one of the good options which Amazon recommended also.

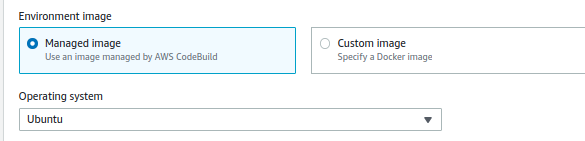
Click - - - Next - - - Select the **Build Provider**

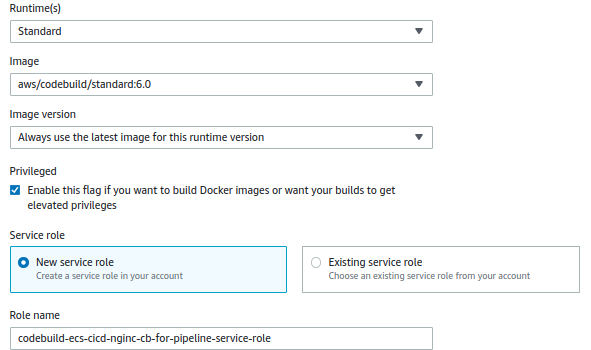
We have to option here **AWS CodeBuild** & **Jenkins** select CodeBuild iam in asia pacific region **ap-south-1.**

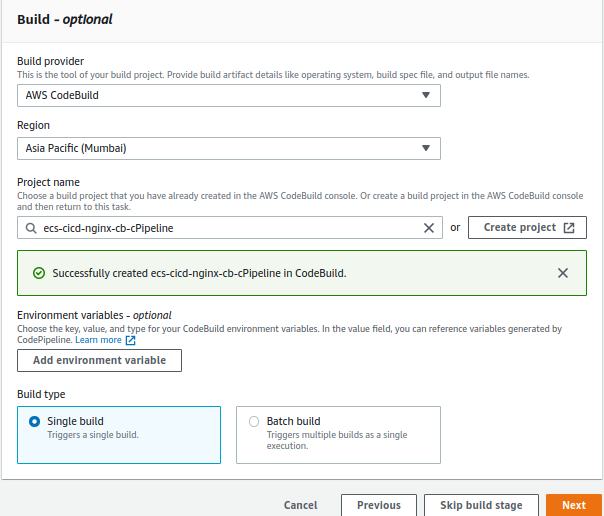
Let’s go ahead and create a **build project** from here







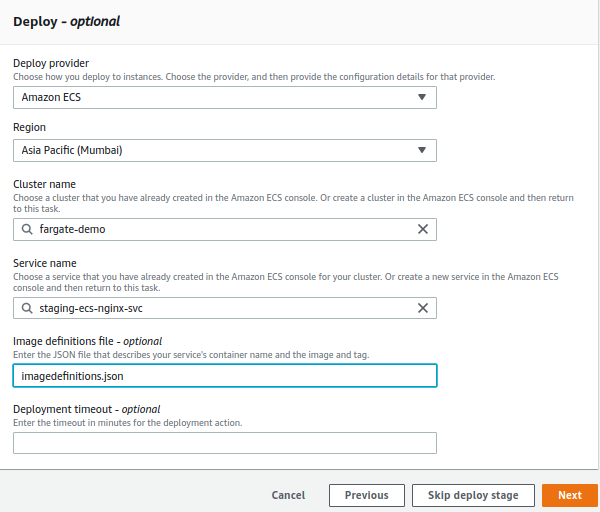


****

CodeBuild project got created & it also populated with project here **Click Next**

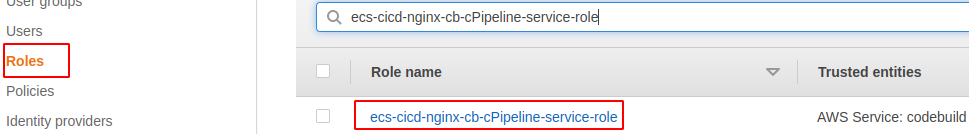
**Note :** Before going to the next step that is **CodeDeploy** provision ECS Cluster Task Definitions & Services [click](https://docs.google.com/document/d/1OfUs6_rSN6sl1yN95vZpKyhZ_WGvHgAoodDMdGSvq7c/edit?usp=sharing) the & follow the steps.

So now we are in CodeDeploy from this perspective iam select **AWS ECS**

****

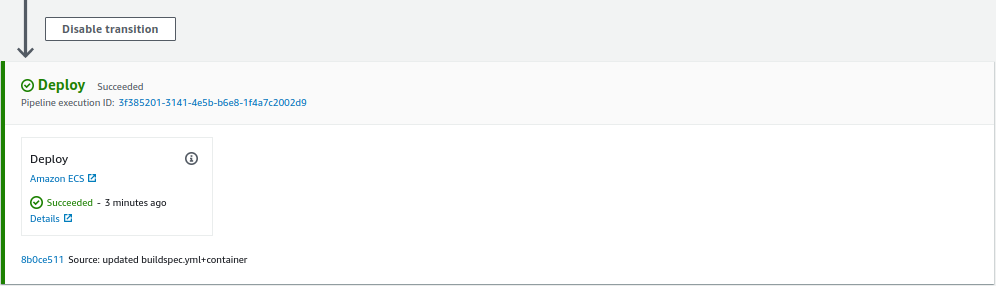
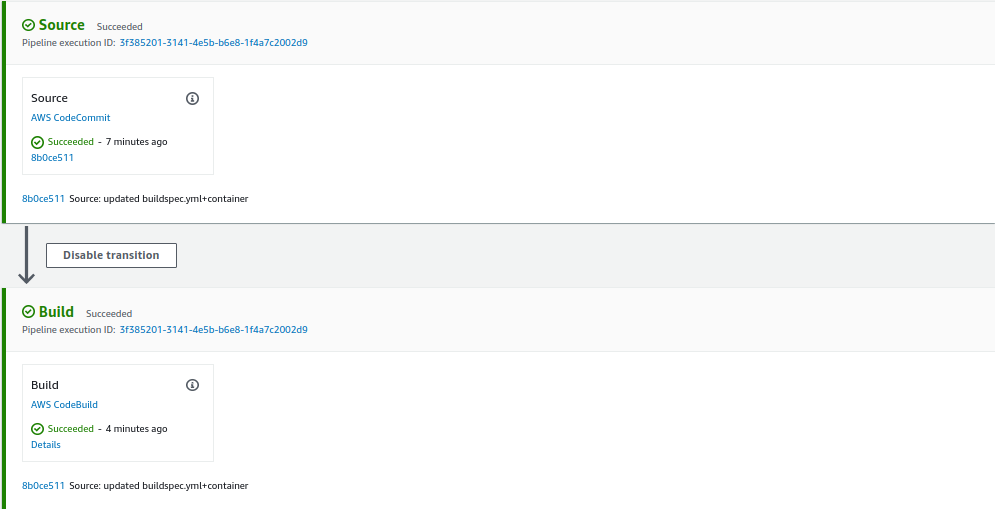
Then select Cluster name that is **fargate-demo** - - - > select the service name, so the file name which i going to use is **image-definition.json** , which present our **buildspec.yml**

Now go **IAM** & search for the role whatever we have created just now

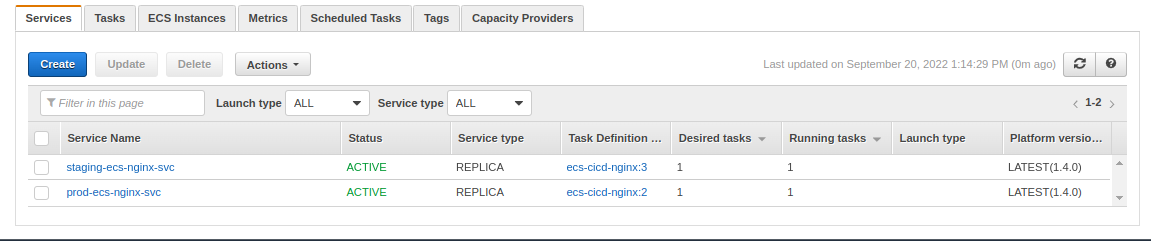


We need to add an additional policy & policy is to contain a registry **AmzonEC2-ContainerRegistryFullAccess**

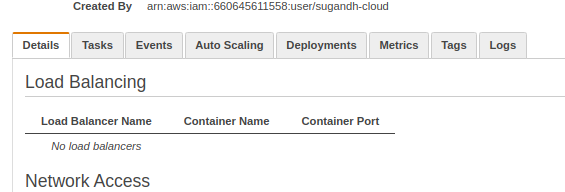
Now create Pipeline & it will automatically build testing deploying the stages



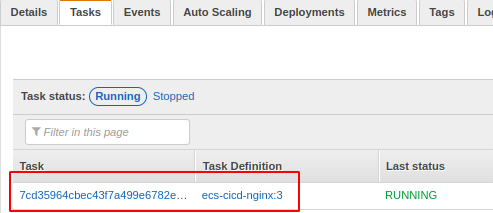
Now go to ECS - - - > Clusters - - - > Service ([staging-ecs-nginx-svc](https://ap-south-1.console.aws.amazon.com/ecs/home?region=ap-south-1#/clusters/fargate-demo/services/staging-ecs-nginx-svc/details)) - - - Inside Task - - - > Copy the public IP paste it on the browser.



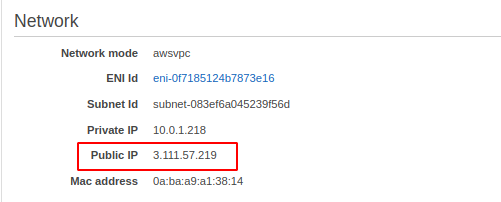
Click on **staging-ecs-nginx-svc** - - - > Go inside task - - ->



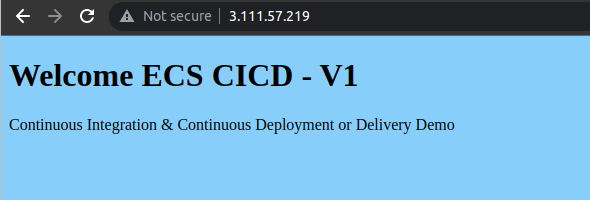
Scroll Down - - - > Click on task status



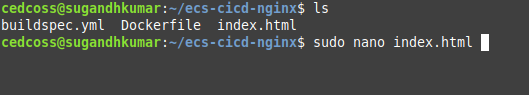
Copy the public IP



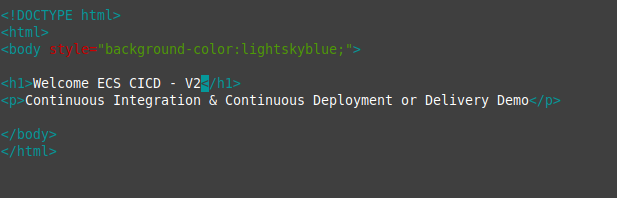
Paste it on browser, Successfully Deploy CICD v1



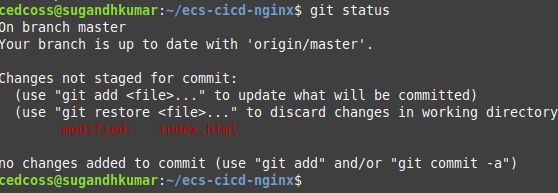
Now go back to index.html & make a change then save it.



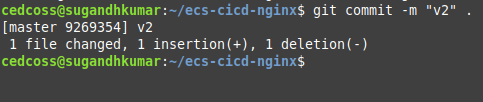
I did little bit change in my index.html just replace **v1** to **v2**

****

***git status***

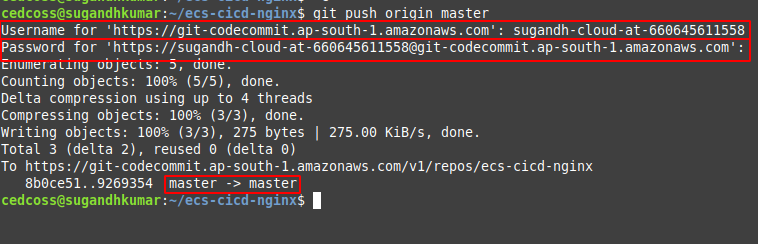


Now commit the changes using the command **git commit -m “v2” .**



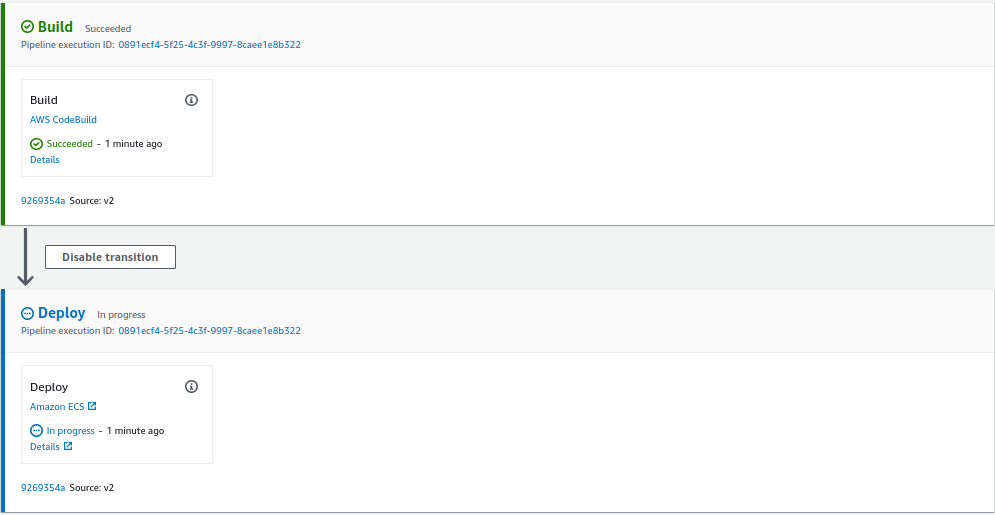
Now push the commit using command **git push origin master**

**‘**It will ask you Username & Password**’**

****

So another changes were pushed to this, let’s come back to Pipeline & then start monitoring here,

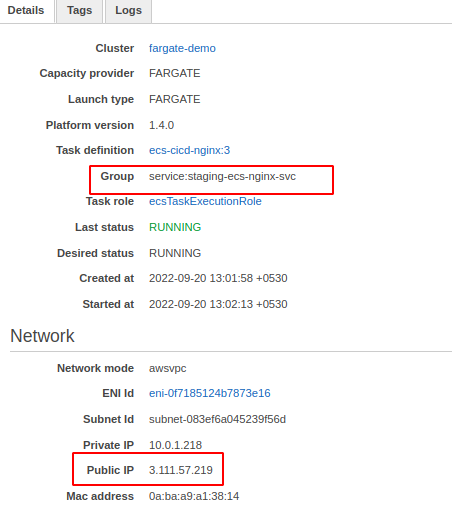
Now it will automatically trigger the pipeline



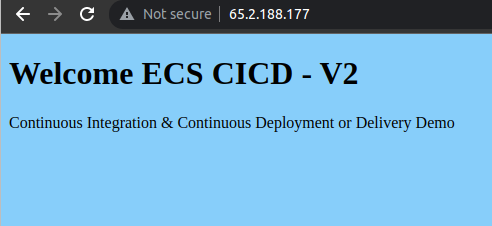
Once the build is completed again it will going to deploy to our staging ECS service, & when we access the staging experience, we should get the value **v2** instead of **v1**

Using that staging ECS new task, public IP, we need to go ahead & then access it.

Go to ECS - - - > fargate-demo - - - > and go to staging-ecs-cicd-nginx - - - > inside the tasks - - -> open this tasks - - - > scroll down and take the public IP paste it on the browser.



you can see **v2**, that’s the way we can complete our CICD process



During the build process, what all it has generated, It has generated the image & then push it to the ECR (elastic container registry)

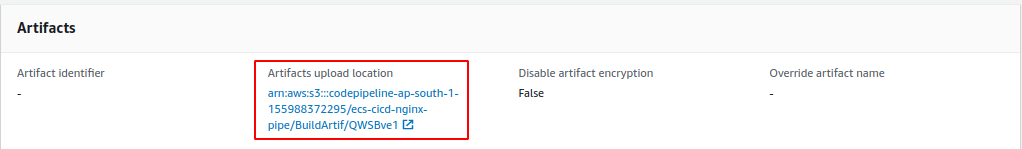
ImageDefinitions.json :

Lets go back to CodeBuild - - - > & goto build details - - -> insides that - - - > Artifacts location , so in addition to the image, whatever it uploaded to the easier, it also generated an artifacts called ‘imagedefinition.json’ so where it got stored.

It stored in the S3 bucket

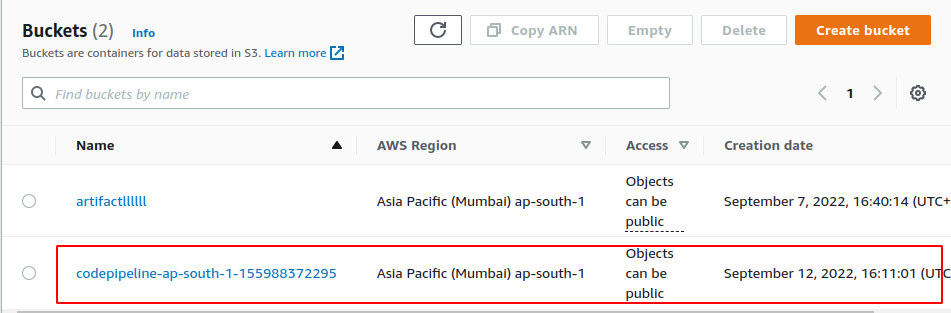
Go to CodeBuild - - - > Build Project - - -> click on your build project - - - > scroll down

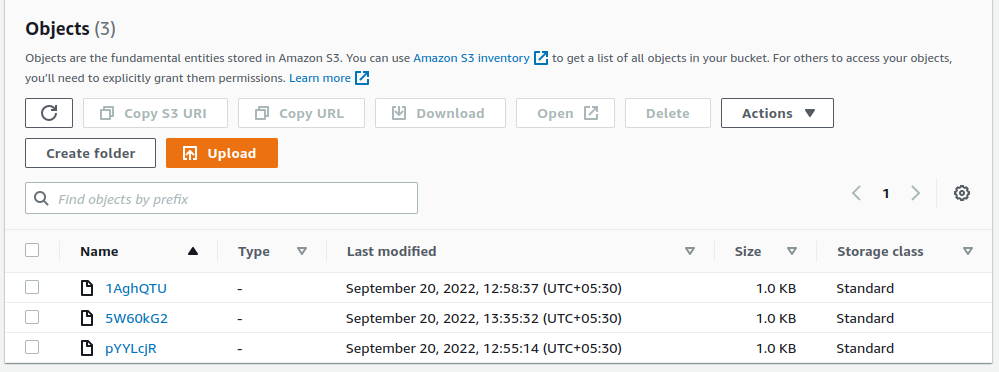
Inside - - - Artifacts



**&**

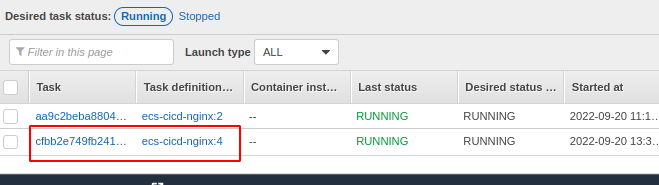
Go to S3 Bucket





This is imagedefinition.json, file is the codefile which is making our ECS definition to version it.

Now if you see inside service the 4rth version should be there for our ecs-cicd-nginx-4 version



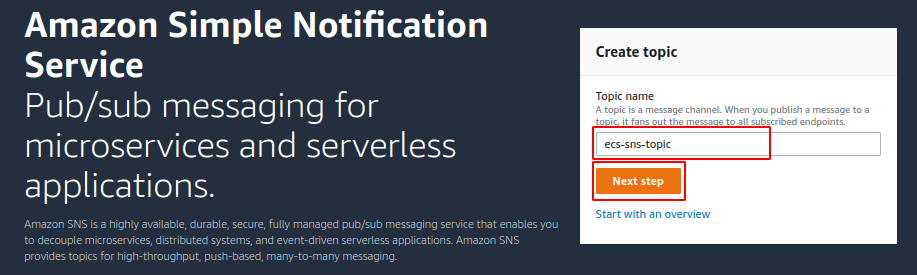
**Manual Approval Stage In CodePipeline**

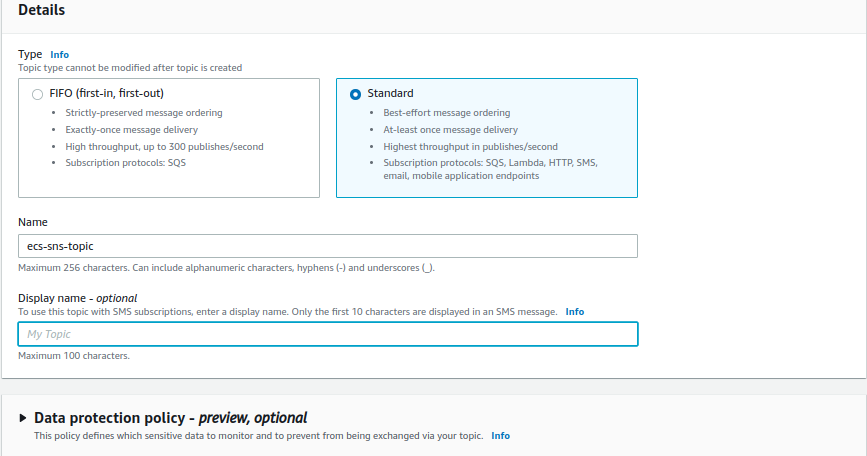
* Create SNS Topic and confirm the email subscription for sending notifications.
* Edit Pipeline and Create Manual Approval Stage

**Why Manual Approval Is Required ?**

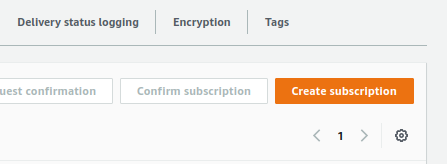
* Whenever we want to plan to do the deployment to the production whenever we are planning to to add production deployment things also as a part of **CodePipeline.**
* So we need some approval, process to be put in place, we can add the manual approval stage here and then configure the **SNS** topic so that email approval will be likes email confirmation will be send to the authorized approval & then once he comes & then approve, then only the deployment to the production will happen.

**Create a SNS (Simple Notification Service) Topic**

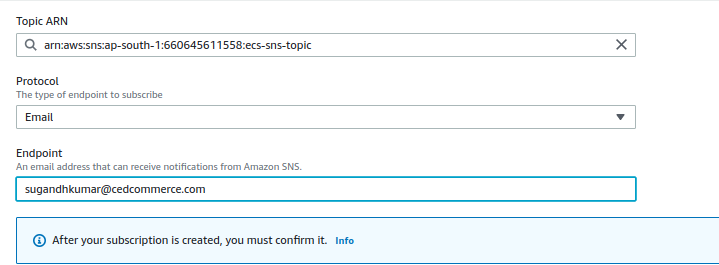
****

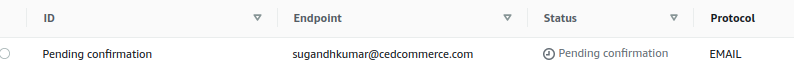
****

Rest all things default & then create,

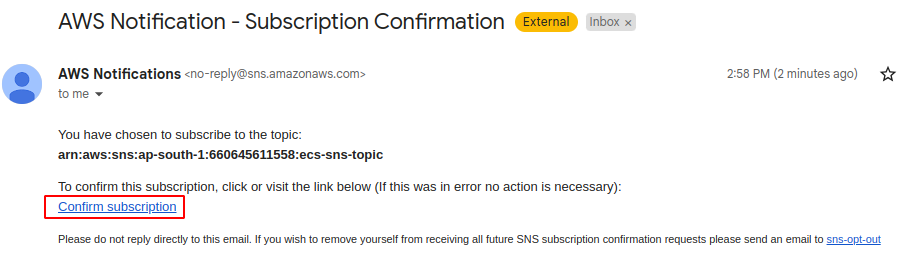


Now create a subscription - - - > select protocol type - - - Email Under Endpoint - - -> Enter your respective Email ID

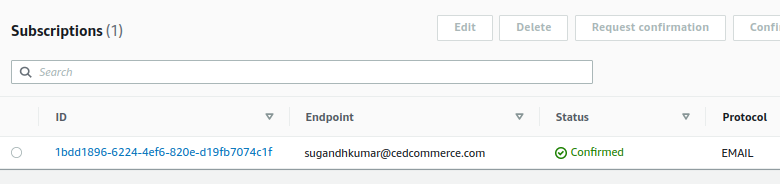




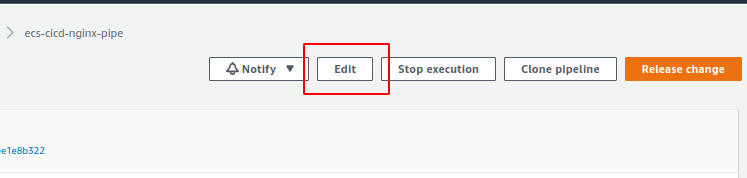
In the subscription pending confirmation, Go ahead your gmail account & confirm the sns topic



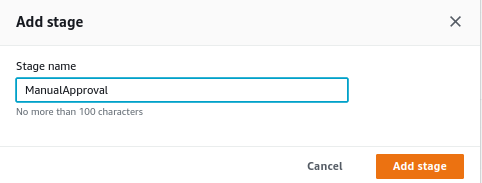
Let’s go ahead **SNS** & refresh the page

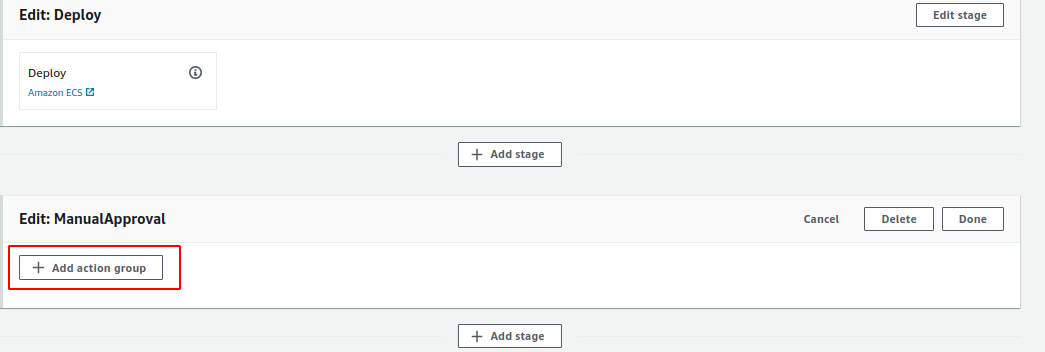


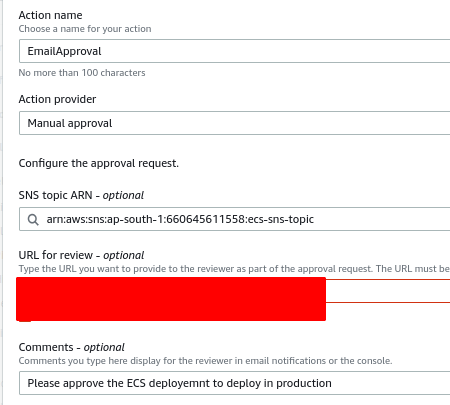
Now my subscription got confirmed, go back to **CodePipeline** then edit the **CodePipeline**

****

Add a new stage here StageName : Manual Approval







Click on **Done** - - > click on **Save**

Create one more Approval here , means like one more stage which is related to production deployment, but before doing that, we can event cross-check & then test

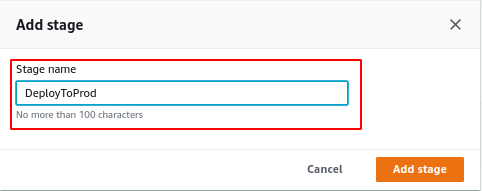
## **Create Deploy to Prod ECS Service stage in CodePipeline**

* Edit the pipeline and create Deploy to prod ECS Service

Goto **CodePipeline -** Edit the stages - Create - **DeployToProd**

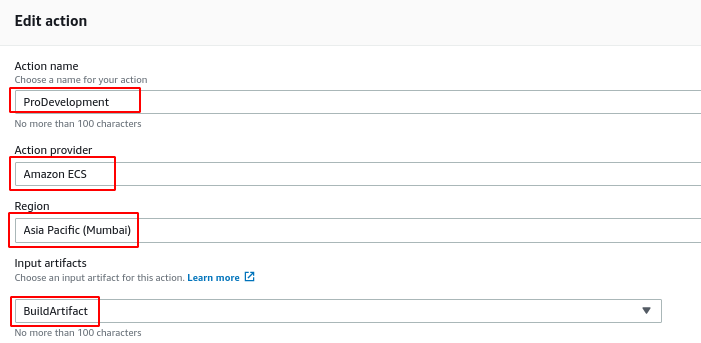
This is last stage we are creating & the action group, i am going to define a **ProdDeeployment** &action provider going to be **AWS ECS** & input artifacts which means after the build process, we are going to do deploy, In general whatever the build artifacts came using that we are going to do that deployment, So input artifact are build artifacts, …… Done!

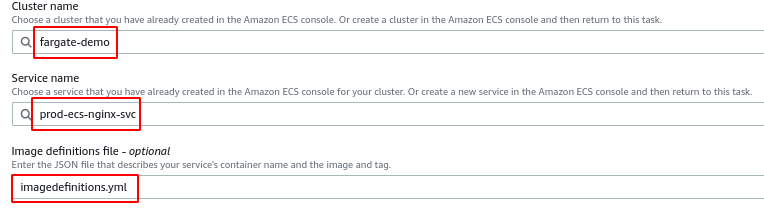
Go back to the **CodePipeline** - - - > Edit the pipeline - - - > **Add Stages**

****

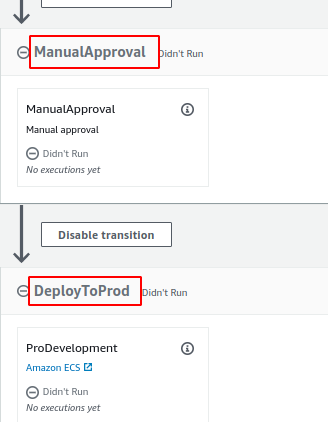
**Now Add Action Group**

****

****

****

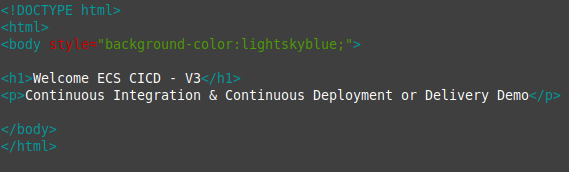
Now we also deployed to the **prod** then click on the **save**

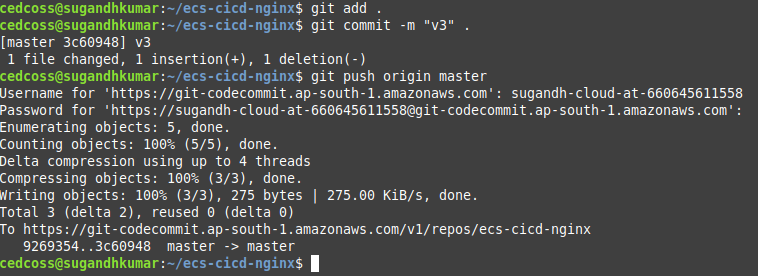


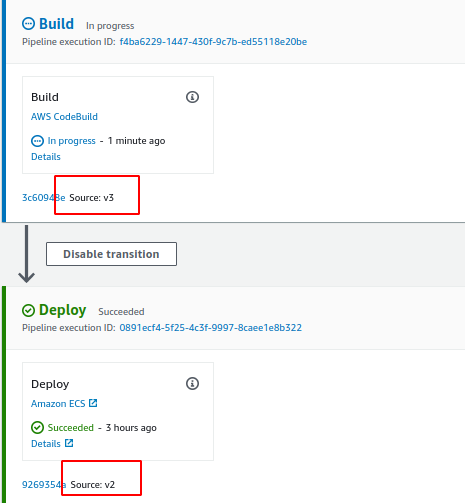
Now this complete the source to the Prod deployment stages creation & then completed , So now what

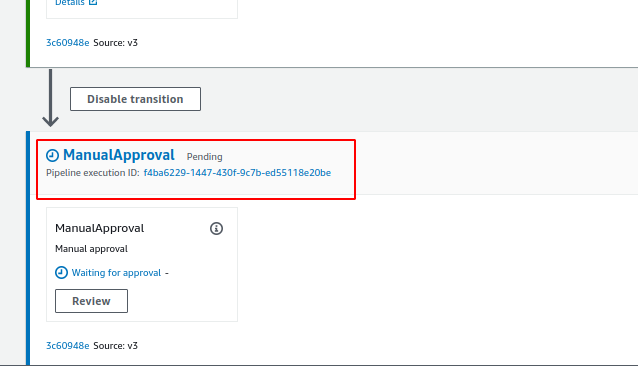
Go to your **index.html** because you need a change to **v3**, then we will verify that it should be deployed in **staging** and also in **production**.

It need to trigger a manual approval, Once we approve, then the deployment should be happen

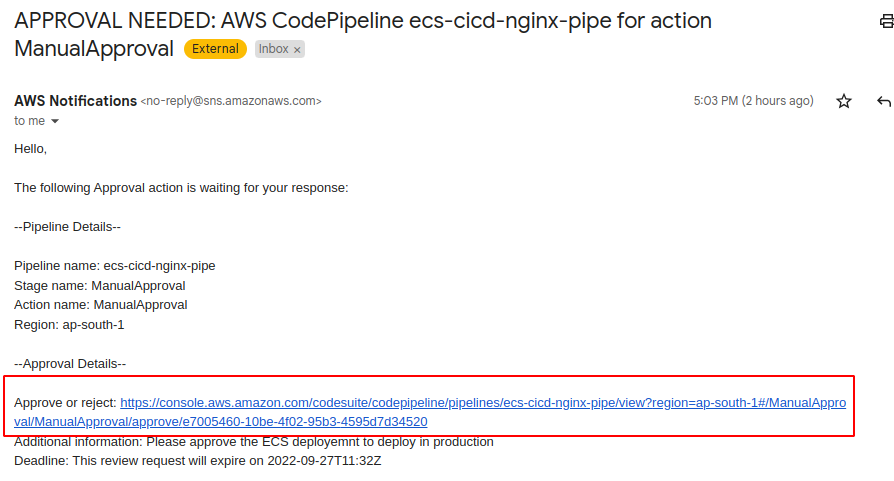




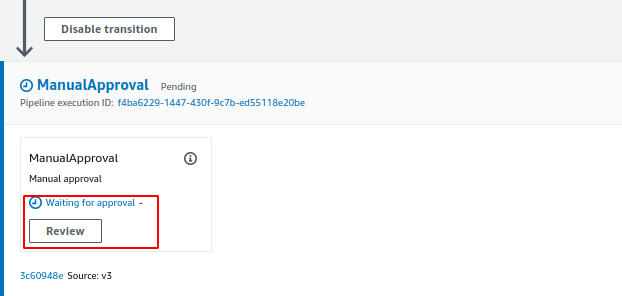


****

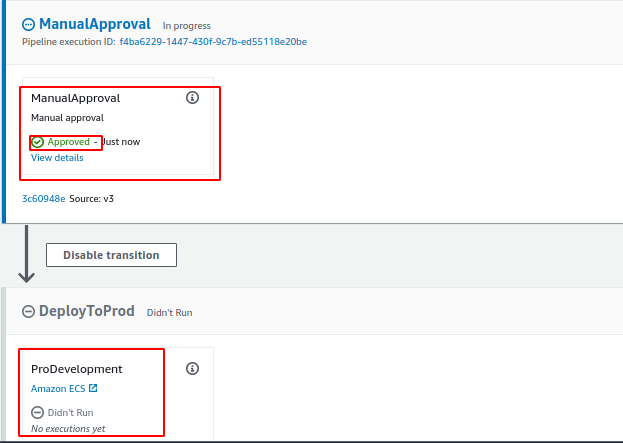
**Now we got the Approval mail**

****

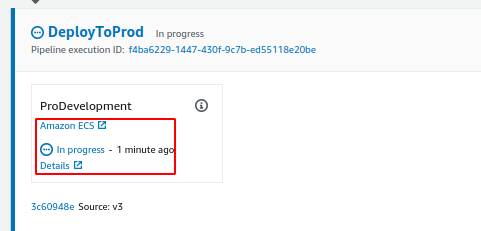
**Click on the link it will redirect you to the Approval stage review location**

****

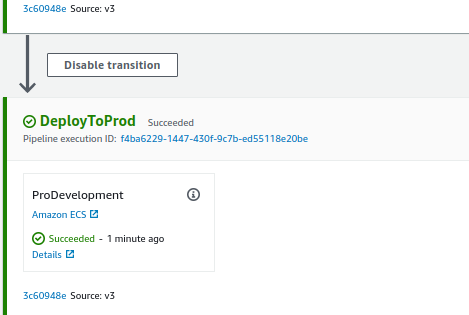
This is the location so we can review & then seek approval whatever we want



Now the manual Approval stage is completed it is going move to the deployed to the prod

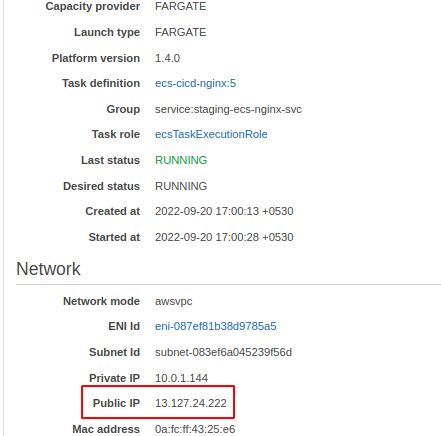


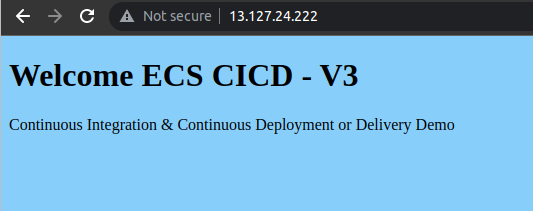
Once the production deployment is completed, we can test both staging and then production at a time

****

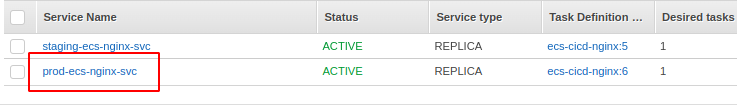
So we can see here another deployed prod also got completed

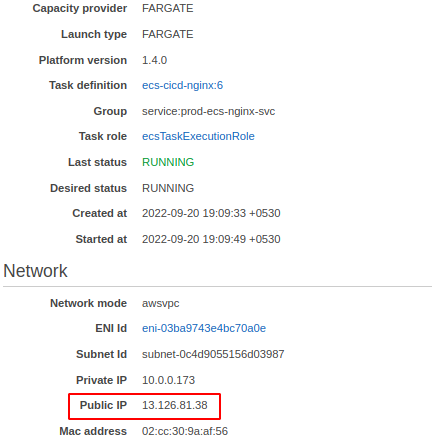
Now go to cluster - - - Service ……. Take the public IP

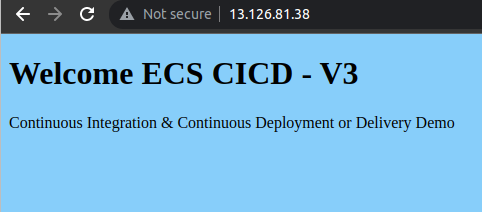




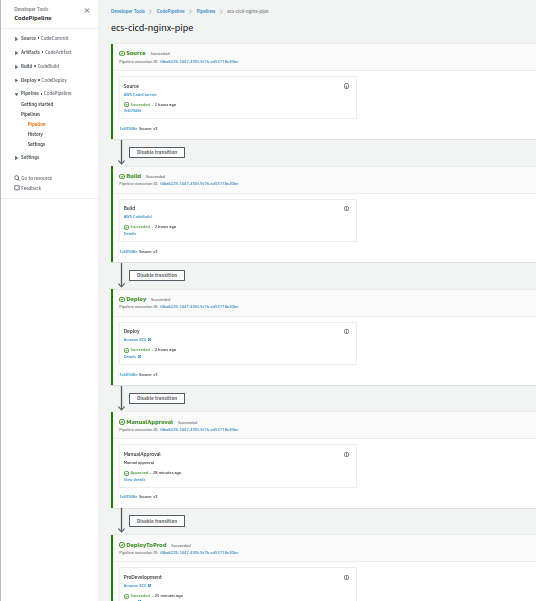
Go back to production service ….. Take the public IP and paste it on the browser.







We are successfully completed the **CodePipeline** both **staging** & then **production** with **production** with **manual** **approval**

****

Done!!