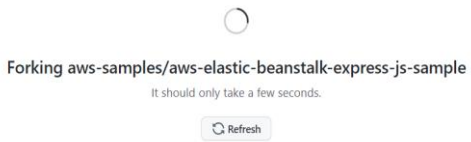
Continuous Delivery Pipeline

Overview

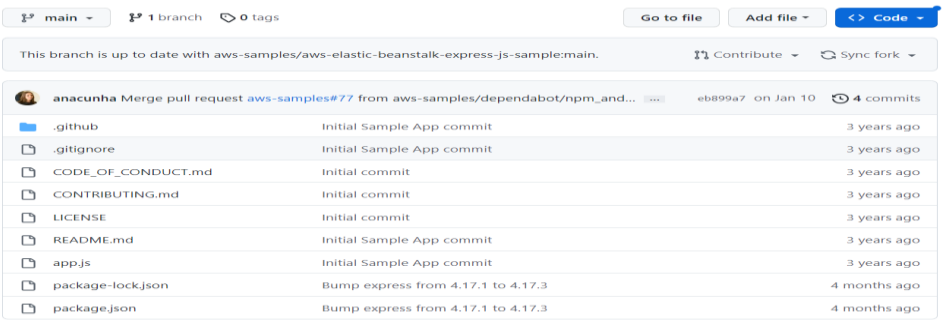
Continuous delivery pipeline for a simple web-based application. First, we use a version control system(GitHub) to store our source code. Then, we create an Elastic Beanstalk environment to deploy our application AWS CodeBuild to build the source code from GitHub and finally, a continuous delivery pipeline that will automatically deploy our web application whenever our source code is updated.

Setting up a Git Repository

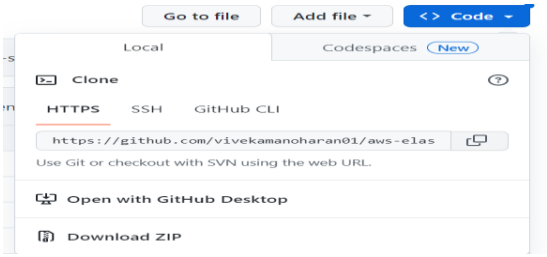
- In this step, I forked an already existing Repository into my GitHub account and created a copy of forked new repository.
- Next, I have cloned the forked repository to my local git terminal.
- And edited the files in my working directory and committed the changes and pushed the changes to my central repository (GitHub).
- Forking the repository means it's just a copy of an already existing repository with public access.
- Forked an already existing repository to my GitHub using this GitHub repo linkhttps://github.com/aws-samples/aws-elastic-beanstalk-express-js-sample



- After fork we can see the files and folders in our central repository(GitHub).



- Next to work in my local terminal, I cloned my GitHub repo via code in HTTPS URL to clone it to my local git terminal.

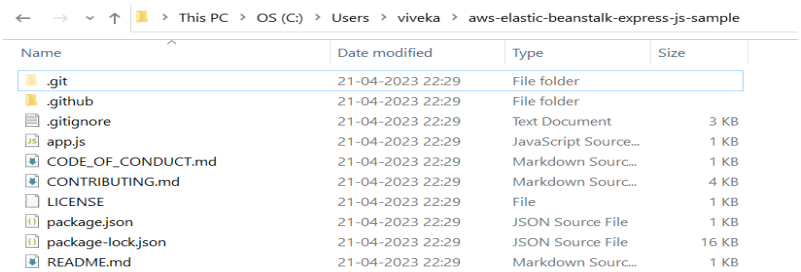


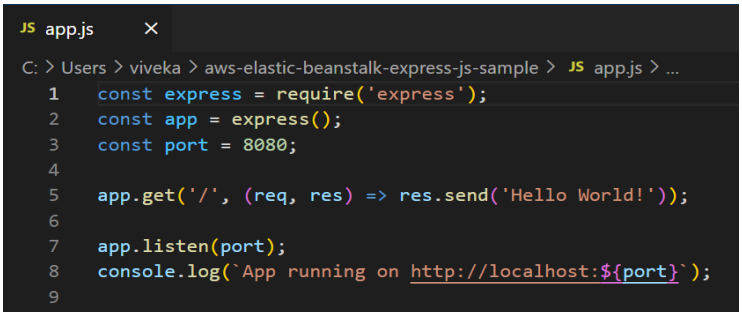
- Cloning means copying the repository from GitHub to our local machine. Cloning a repository pulls down a full copy of all the repository data that GitHub had at that time, including every file and folder for the project.
- With git clone and my repo https url cloned the central repo to the local terminal.

```
viveka@DESKTOP-98HJ7V5 MINGW64 ~ (master)
$ git clone https://github.com/vivekamanoharan01/aws-elastic-beanstalk-express-j
s-sample.git
Cloning into 'aws-elastic-beanstalk-express-js-sample'...
remote: Enumerating objects: 20, done.
remote: Total 20 (delta 0), reused 0 (delta 0), pack-reused 20
Receiving objects: 100% (20/20), 14.65 KiB | 2.09 MiB/s, done.
Resolving deltas: 100% (4/4), done.

viveka@DESKTOP-98HJ7V5 MINGW64 ~ (master)
$
```

After cloning my working directory folder which it created after cloning the central repo on that app.js file, I edited line 5 using VS Code and saved the file.





```
C: > Users > viveka > aws-elastic-beanstalk-express-js-sample > JS app.js > ...
1  const express = require('express');
2  const app = express();
3  const port = 8080;
4
5  app.get('/', (req, res) => res.send('continuous delivery pipeline'));
6
7  app.listen(port);
8  console.log(`App running on http://localhost:${port}`);
9
```

- Whenever we are making changes to files and are adding new files to our working directory, we should add the file into the staging area and commit the changes to our local git terminal.
- Git add . cmd changes the working directory to the staging area.
- Git commit –m cmd is used to save our changes in the repository

```
viveka@DESKTOP-98HJ7V5 MINGW64 ~/aws-elastic-beanstalk-express-js-sample (main)
$ git add app.js

viveka@DESKTOP-98HJ7V5 MINGW64 ~/aws-elastic-beanstalk-express-js-sample (main)
$ git commit -m "just changed line 5"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

viveka@DESKTOP-98HJ7V5 MINGW64 ~/aws-elastic-beanstalk-express-js-sample (main)
$
```

- After editing the app.js file in my local git terminal, I pushed the changes to the central repo using git push cmd.

```
iveka@DESKTOP-98HJ7V5 MINGW64 ~/aws-elastic-beanstalk-express-js-sample (main)
 git push
numerating objects: 5, done.
ounting objects: 100% (5/5), done.
elta compression using up to 8 threads
ompressing objects: 100% (3/3), done.
riting objects: 100% (3/3), 324 bytes | 324.00 KiB/s, done.
otal 3 (delta 2), reused 0 (delta 0), pack-reused 0
emote: Resolving deltas: 100% (2/2), completed with 2 local objects.
o https://github.com/vivekamanoharan01/aws-elastic-beanstalk-express-js-sample.git
   eb899a7..c14498e  main -> main

iveka@DESKTOP-98HJ7V5 MINGW64 ~/aws-elastic-beanstalk-express-js-sample (main)
```

- We can see the changes in our central repository. The changes we made on line 5 in the local git terminal it will be there.

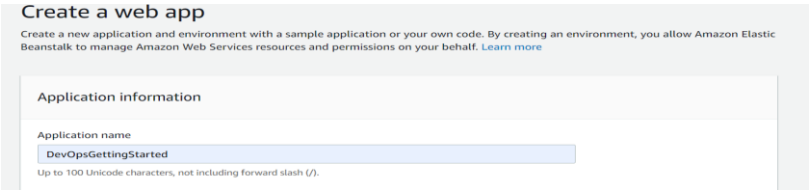
```
8 lines (6 sloc) | 224 Bytes
1  const express = require('express');
2  const app = express();
3  const port = 8080;
4
5  app.get('/', (req, res) => res.send('continuous delivery pipeline'));
6
7  app.listen(port);
8  console.log(`App running on http://localhost:${port}`);
```

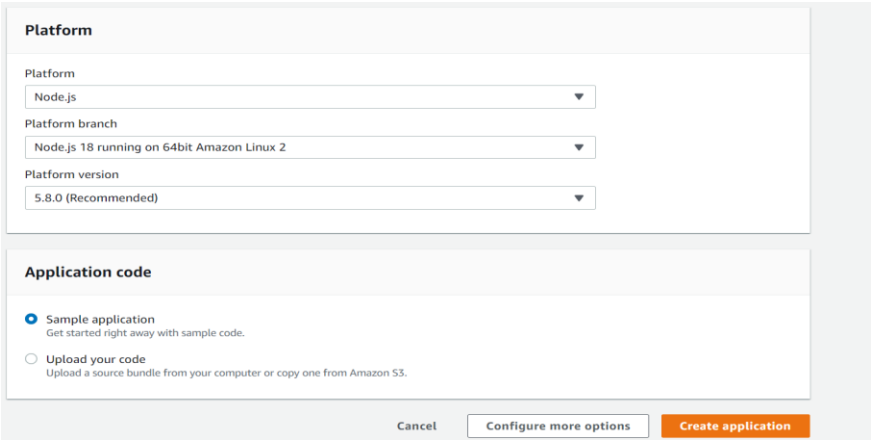
Deploying Web Application

- In this, I have created an elastic Beanstalk environment and deployed a sample web application.
- With Elastic Beanstalk, we can quickly deploy and manage applications in the AWS Cloud without having to learn about the infrastructure that runs those applications.

We simply upload our application, and Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

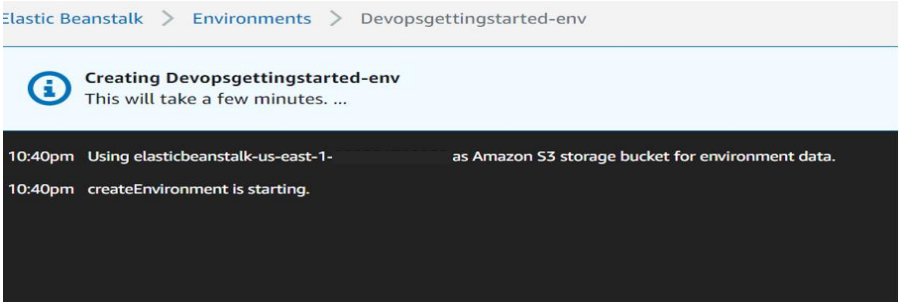
- I used the Node.js platform version and provisions one Amazon EC2 instances, to run my application.
- I configured Elastic Beanstalk application information , application platform and application code.
- Application – collection of Elastic Beanstalk components, including environments, versions, and environment configurations.



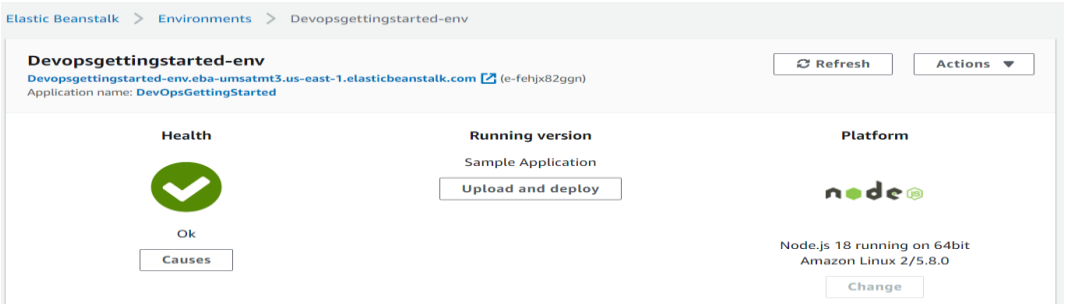


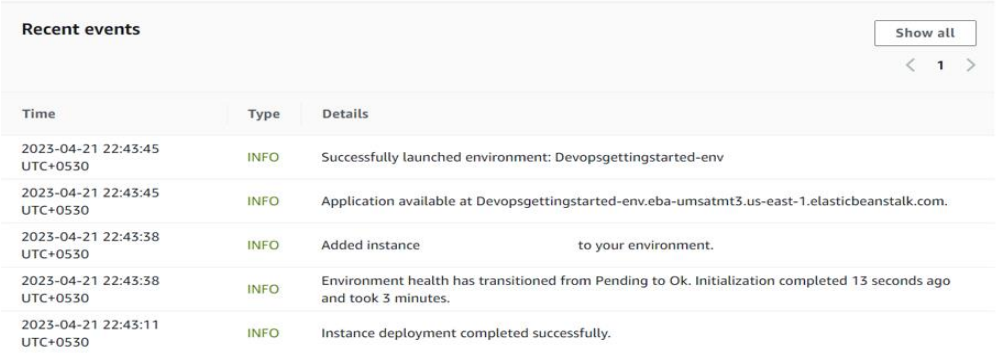
And then created the Elastic Beanstalk application .

- After we clicked the create application button, a small black window with white text .This screen will display messages for our environment

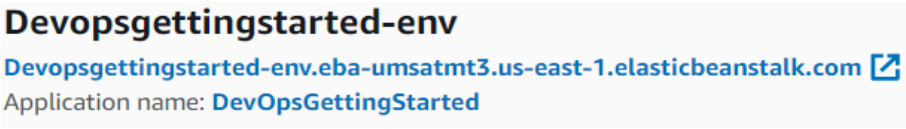


After the creation of the environment, we will see a green checkmark on our screen.





- AWS ElasticBeanstalk environment has created EC2 instances ,AutoScalling , LoadBalancer and s3 bucket.
- After the creation, if we click this, the URL below which is available on the AWS environment page.



It will take us to new page like congratulating as

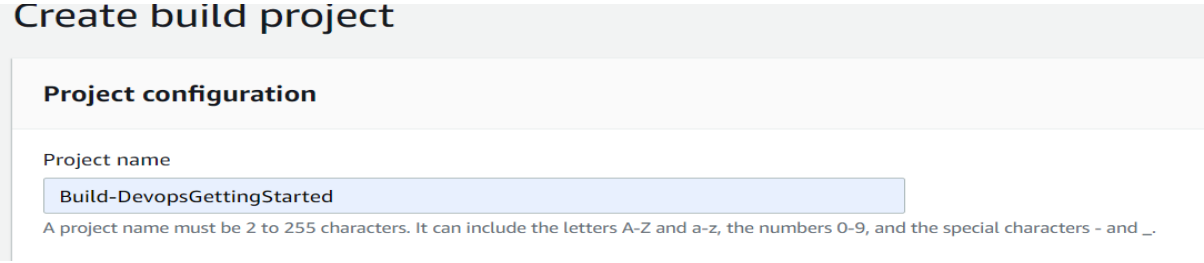
- We have created an AWS Elastic Beanstalk environment and a sample application. We will be using this environment and our continuous delivery pipeline to deploy the Hello World! web app which I created earlier.

Build Project

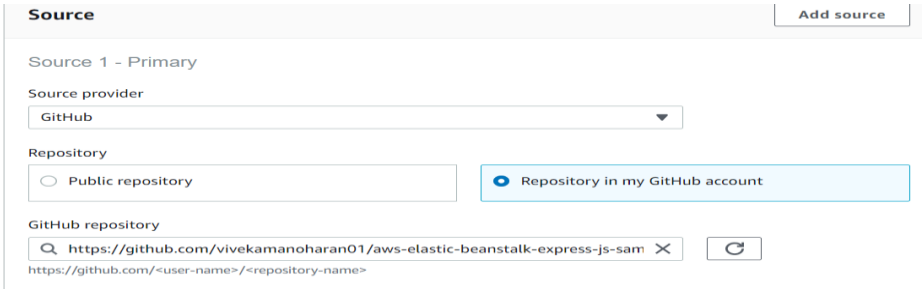
- We will use AWS CodeBuild to build the source code stored in our GitHub repository.
- AWS CodeBuild is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy.

Configuring the AWS CodeBuild project

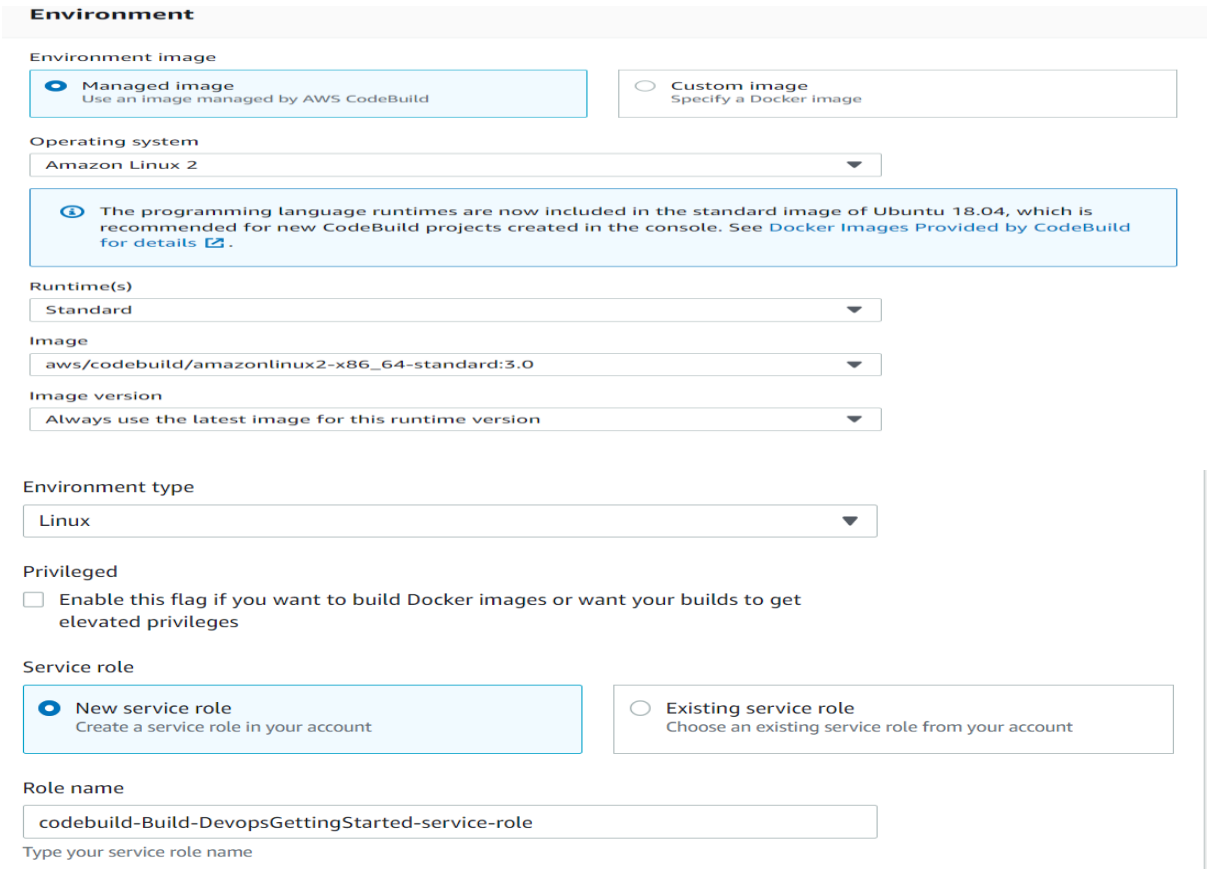
- CodeBuild uses the build project to create the build environment.
- A build project includes information about how to run a build, including where to get the source code, which build environment to use, which build commands to run, and where to store the build output.



- In the project configuration, I gave the project name as DevopsGettingStarted.
- At the source stage, I configured Github as a source provider where codebuild will take code and start build project.

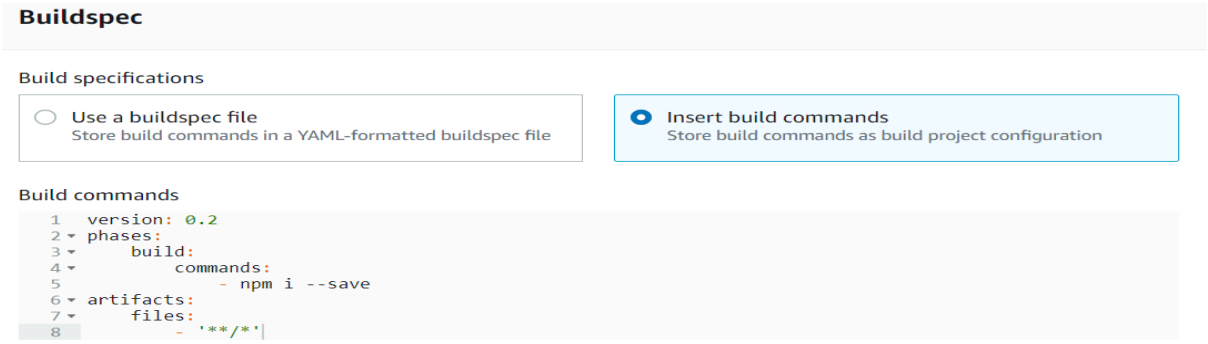


In environmental settings, a built environment represents a combination of operating systems. I chose Amazon Linux and the programming language runtime is standard , and created a new service role.



Buildspec file

So far, I have configured source and environment settings in the codeBuild project. In this step I have inserted build commands in buildspec, Without a build spec, CodeBuild cannot successfully convert build input into build output. A buildspec is a collection of build commands and related settings, in YAML format, that CodeBuild uses to run a build.



Build Commands Declaration :

version

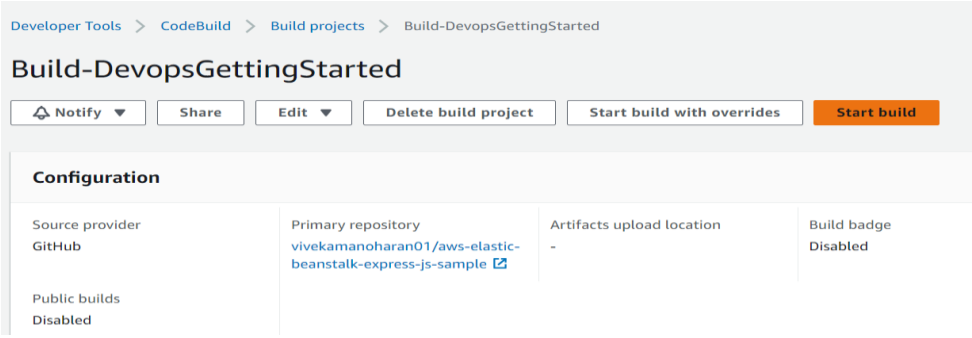
Required mapping. Represents the buildspec version that we use 0.2

- Phases represents the build phases during which we instruct CodeBuild to run commands.

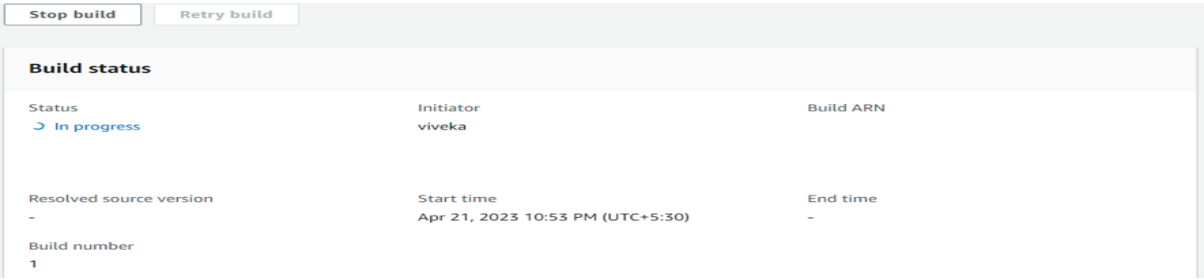
  In this example, during the build phase, we didn't include a build phase to run any commands during this phase.

  - NPM provides the --save option while installing the packages. If we use the --save option after installing the package, it will be saved in package. json inside dependencies.

    After the configuration created codebuild project



  - To test the codebuild we have to click the start build icon and wait for the build to be completed. We can see a green bar at the top of the page with the message Build started. And build status will be changed from in progress to succeeding state.



Create Delivery Pipeline

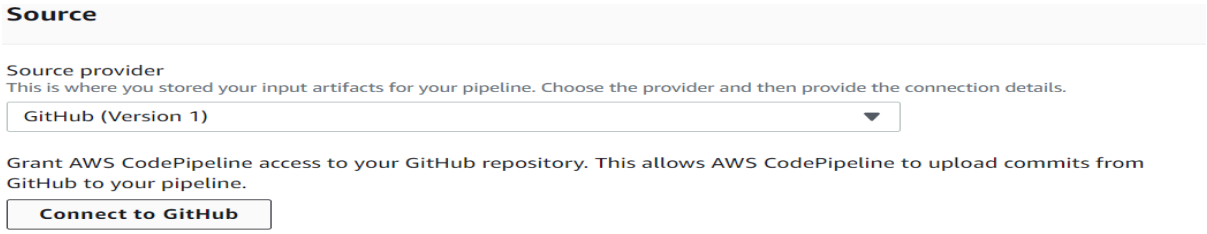
- We will use AWS CodePipeline to set up a continuous delivery pipeline with source, build, and deploy stages. The pipeline will detect changes in the code stored in our GitHub repository, build the source code using AWS CodeBuild, and then deploy our application to AWS Elastic Beanstalk.

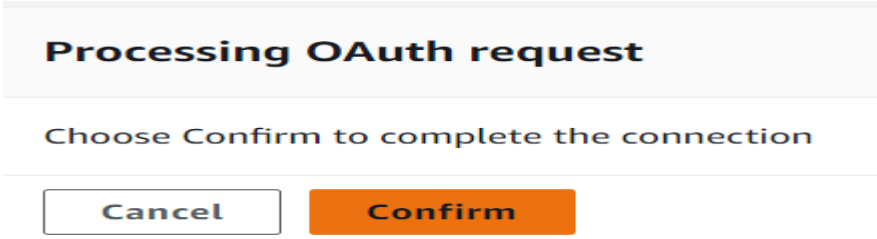
AWS CodePipeline

- AWS CodePipeline is a continuous delivery pipeline that automates the steps required to release our software.

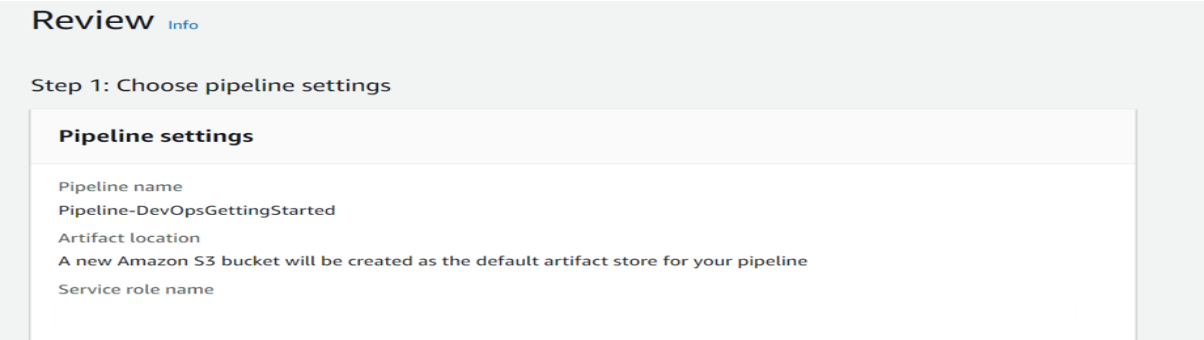
Creating a new pipeline

- On the AWS management console under developer tools in AWS CodePipeline, we can see code pipeline.
- At the source stage, I configured GitHub as a source provider and granted permission to access my GitHub in CodePipeline and connect to my GitHub.





- And configured build stage using codeBuild
- And in deploy stage configured elasticBeanstalk as deploy provider
- And finally Review and created pipeline



Step 2: Add source stage

Source action provider

Source action provider
GitHub (Version 1)

PollForSourceChanges
false

Repo
aws-elastic-beanstalk-express-js-sample

Owner
vivekamanoharan01

Branch
main

Step 3: Add build stage

Build action provider

Build action provider
AWS CodeBuild

ProjectName
Build-DevopsGettingStarted

Step 4: Add deploy stage

Deploy action provider

Deploy action provider
AWS Elastic Beanstalk

ApplicationName
DevOpsGettingStarted

EnvironmentName
Devopsgettingstarted-env

- After creating the pipeline we can see the Pipeline execution it will be in in progress state and within minutes it will be in succeeded state,

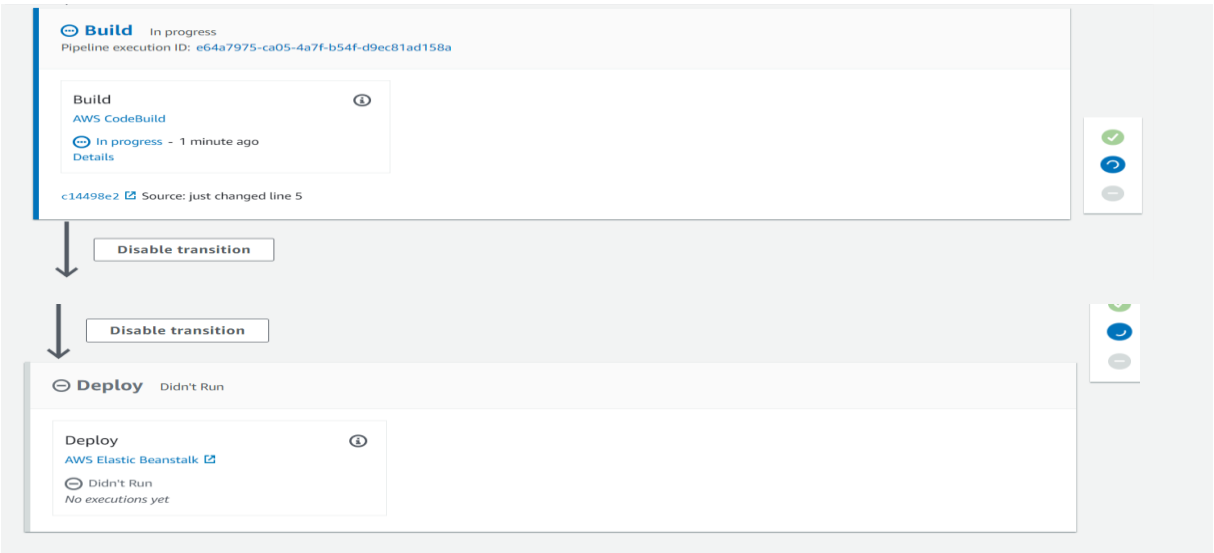
Pipeline-DevOpsGettingStarted

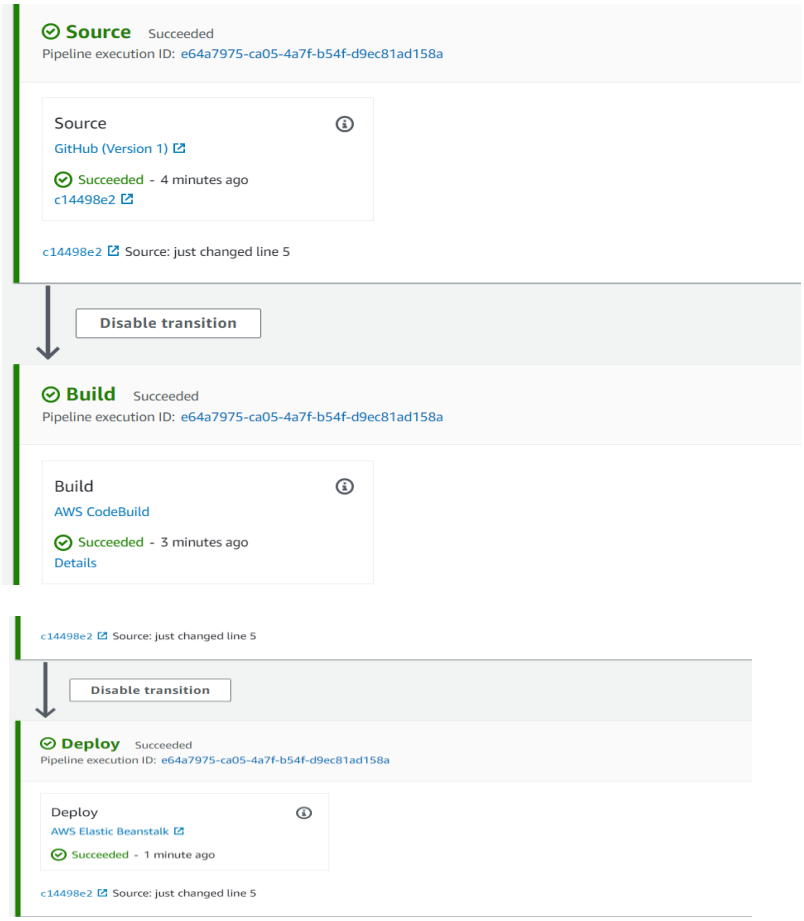
Notify | Edit | Stop execution | Clone pipeline | Release change

Source In progress
Pipeline execution ID: e64a7975-ca05-4a7f-b54f-d9ec81ad158a

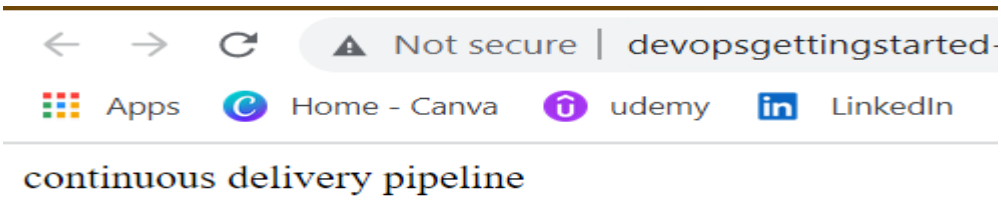
Source
GitHub (Version 1)
In progress - 1 minute ago

Disable transition





- After succeeding state in CodeDeploy in AWS ElasticBeanstalk on if we select the URL be DevopsGettingStarted-. we can see the white Page with text background.

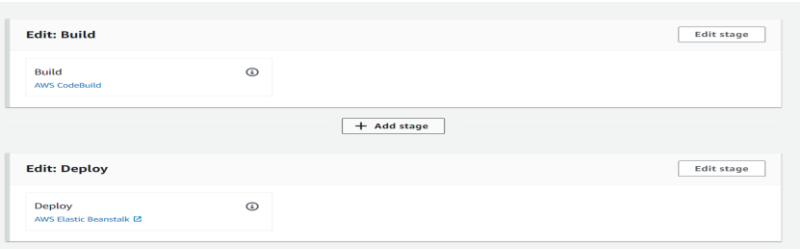


Finalize the Pipeline and test

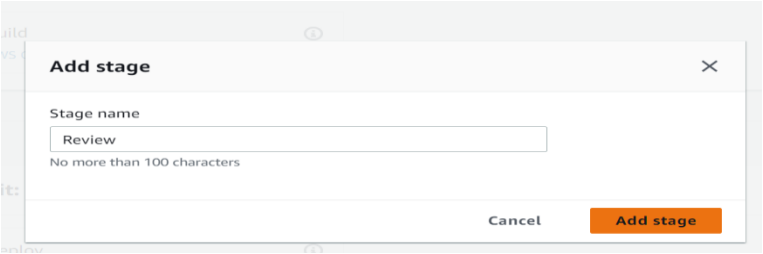
- We will use AWS CodePipeline to add a review stage to our continuous delivery pipeline.
- In this step, we can add an approval action at the point where we want the pipeline execution to stop so we can manually approve or reject the action.If the action is approved, the pipeline execution resumes. If the action is rejected, the pipeline execution does not continue.

Review stage in pipeline

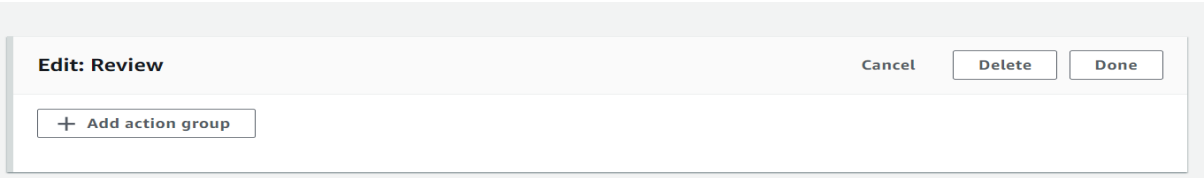
- In AWS CodePipeline, the pipeline I created between Build stage and deploy stage, I added an ADD stage button.

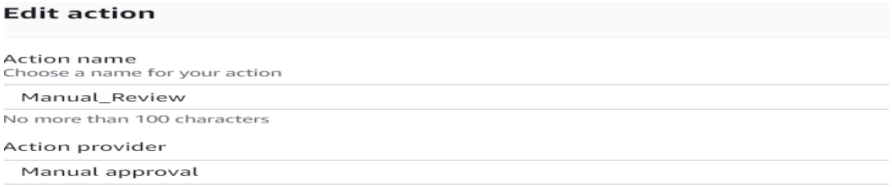


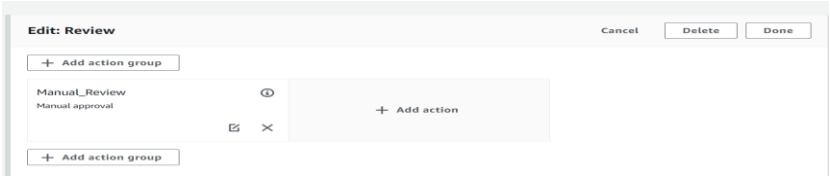
In that added stage, I configured a review as the stage name and created the stage.



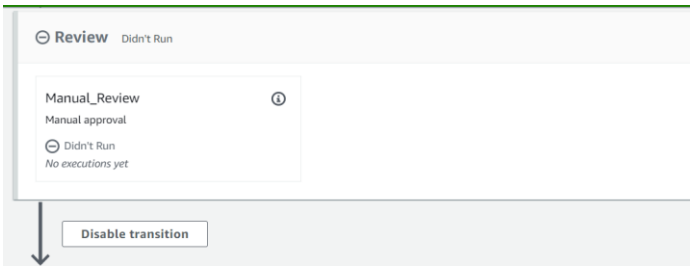
In the review stage, the configured action name as manual review and action provider as manual approval.



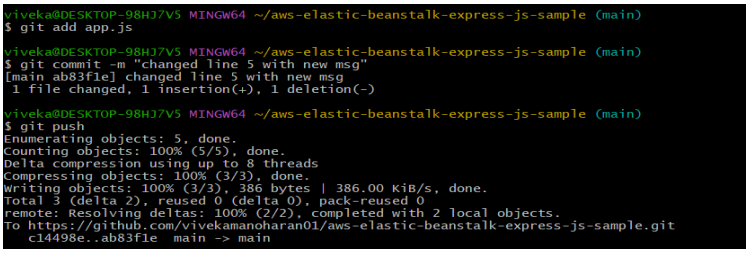




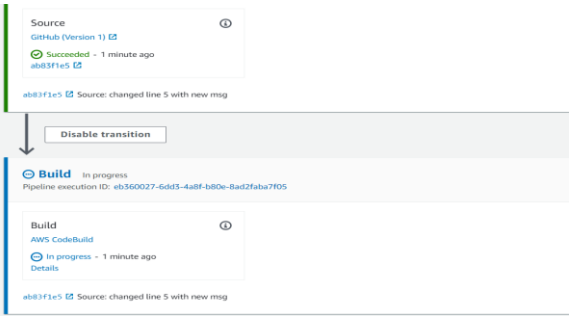
- And saved the changes
- We can see the pipeline with four stages



In my local working directory in app.js file I have changed line 5, adding new text msg and commit and pushed changes to central repo

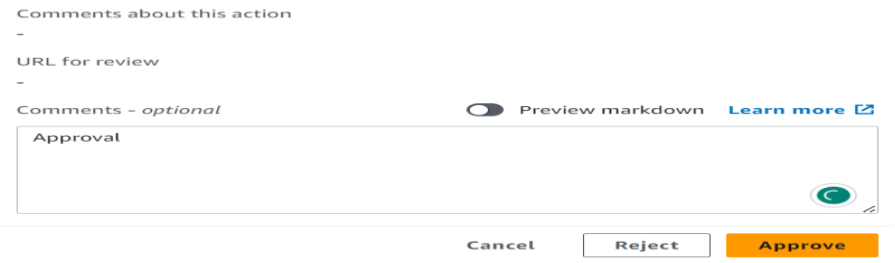


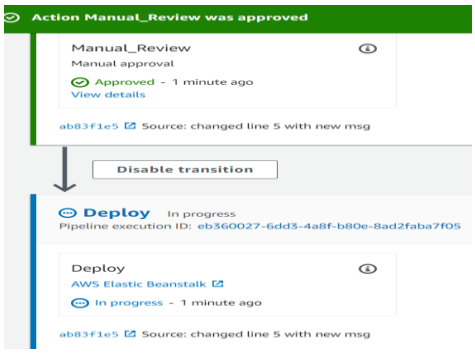
- After these changes, we can monitor the pipeline and manually approve the pipeline to deploy the web application.
- We have used AWS CodePipeline to add a review stage with manual approval to our continuous delivery pipeline.



- We can see the review stage has been waiting for approval
- Now, our code changes will have to be reviewed and approved before they are deployed to AWS Elastic Beanstalk.

With that, if we click add enter Approval cmd, the pipeline will go to the next stage of deploy stage.





After progressing to a successful state, if we click the ElasticBeanstalk url, we can see new changes in our web browser.

continuous delivery pipeline is an automated software release process everytime there is code change it automatically deploy the changes