

Department of Computer Science

University of Bridgeport

Bridgeport, CT-06604

USA

**Name :** Divya Samragni Nadakuditi

**ID#:** 1068743

**Name**: Bhavana Vankhede

**ID#**: 1034353

**Date**: 9th, may 2019

**Project Name**: Project name: DevOps :Continuous Integration(CI) and Continuous Deployment(CD) using AWS CodePipeline & AWS Elastic Beanstalk

**Table of Contents:**

**Abstract ---------------------------------------------- 3**

**Introduction ---------------------------------------- 4**

**implementation ------------------------------- 5**

**Screen shots ---------------------------------------- 6**

**Conclusion ------------------------------------------ 14**

ABSTRACT

In this project, we will build a fully automated Continuous Deployment  pipeline using Amazon web services. We will build a pipeline using AWS Codepipeline which will automatically take changes from a source control repository which in our case is Git repository. It is further deployed into the a production server. We use AWS Elastic Beanstalk to deploy our application on Amazon Cloud Infrastructure.

We use a Basic PHP Web Application, write and run the test , Manage the Data, Deploy this application using AWS Beanstalk, Setup MySQL database on AWS RDS, Manual CI/CD AWS Codepipeline, use AWS Lambda Actions, Setup a fully automated pipeline  and Monitor the pipeline.

INTRODUCTION

Continuous Integration in software development practice is where engineers  integrate the work that work very frequently. Once the code is integrated it then verifies it to ensure that the integrated code has not introduced any errors or bugs. The CI software is degraded with the source code repositories directly so, it allows in detection of changes as developers commit it.

Continuous Delivery or Continuous deployment aims to building, testing and releasing software faster frequently. CI/CD pipeline is basically a set of steps that a code change has to go through.

In this project, we use a PHP web application that uses a  MySQL database. This code is modified using Cloud9 . We deploy the application in Elastic Beanstalk and use Codepipeline which is deployed using Git where we make the changes to the code. Below figure 1 shows the architecture of the project.

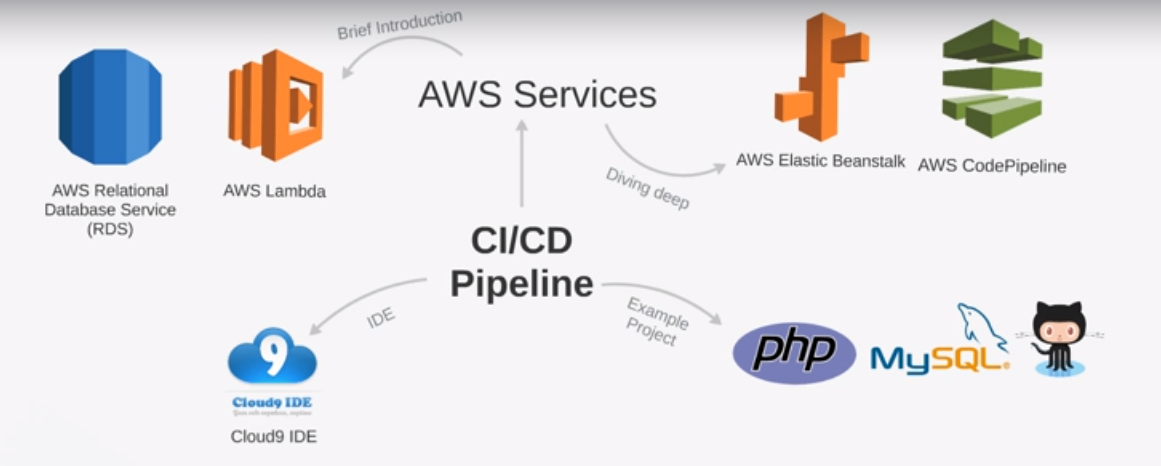


Figure 1: Architecture of the project

We further use AWS CloudWatch to generate the alerts of the applications health.

IMPLEMENTATION

List of All the services used:

* Github.
* AWS ElasticBeanstalk.
* AWS Codepipeline.
* AWS RDS.
* AWS S3.
* AWS Lambda.
* AWS CloudWatch.

PHP Web Application Changes:

Firstly, we downloaded the PHP Application, initialize and pull all the dependencies. We further set environment variables. These are used to configure options. We add them to elastic beanstalk, this variable is used to determine what stage the application is in Development stage, Beta stage, production stage. We also use these variables to store database credentials. We commit all the code into GitHub Repository.

Deploy Project: Setting up AWS Elastic Beanstalk & RDS:

We will now deploy the application into Elastic Beanstalk. We will have builder and production stages. We will create a web application and configure the options before creation.

**Screenshots:**

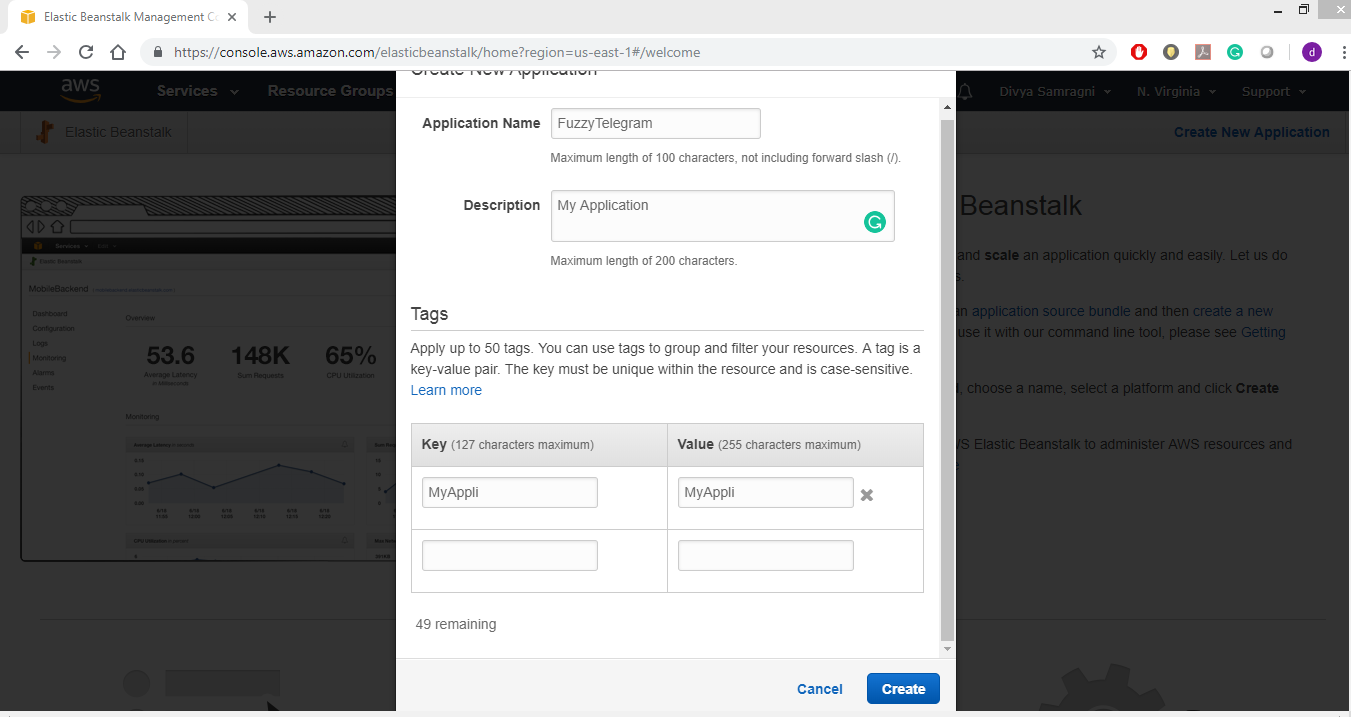


Figure 2:Creating the Elastic Beanstalk Application.

We further add the database configurations as shown in figure 3.

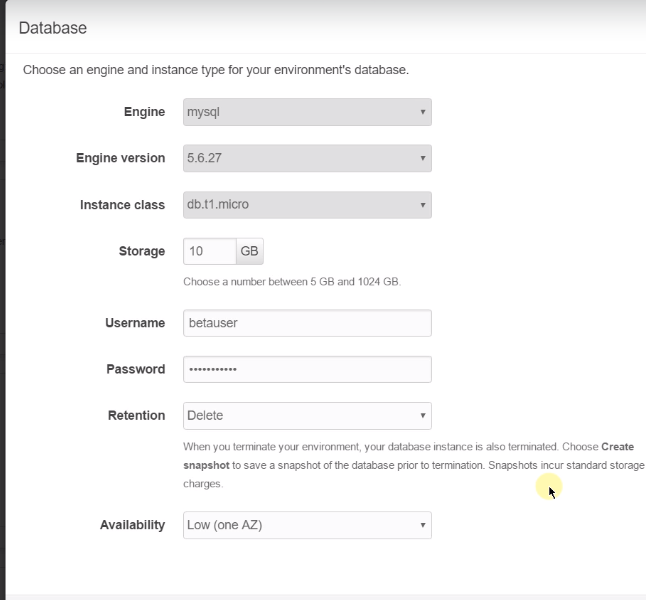


Figure 3: Database configuration while creating an Elastic Beanstalk Application.

This should launch a sample application as we did not deploy our application. Figure 4 is the screenshot of the application launched.

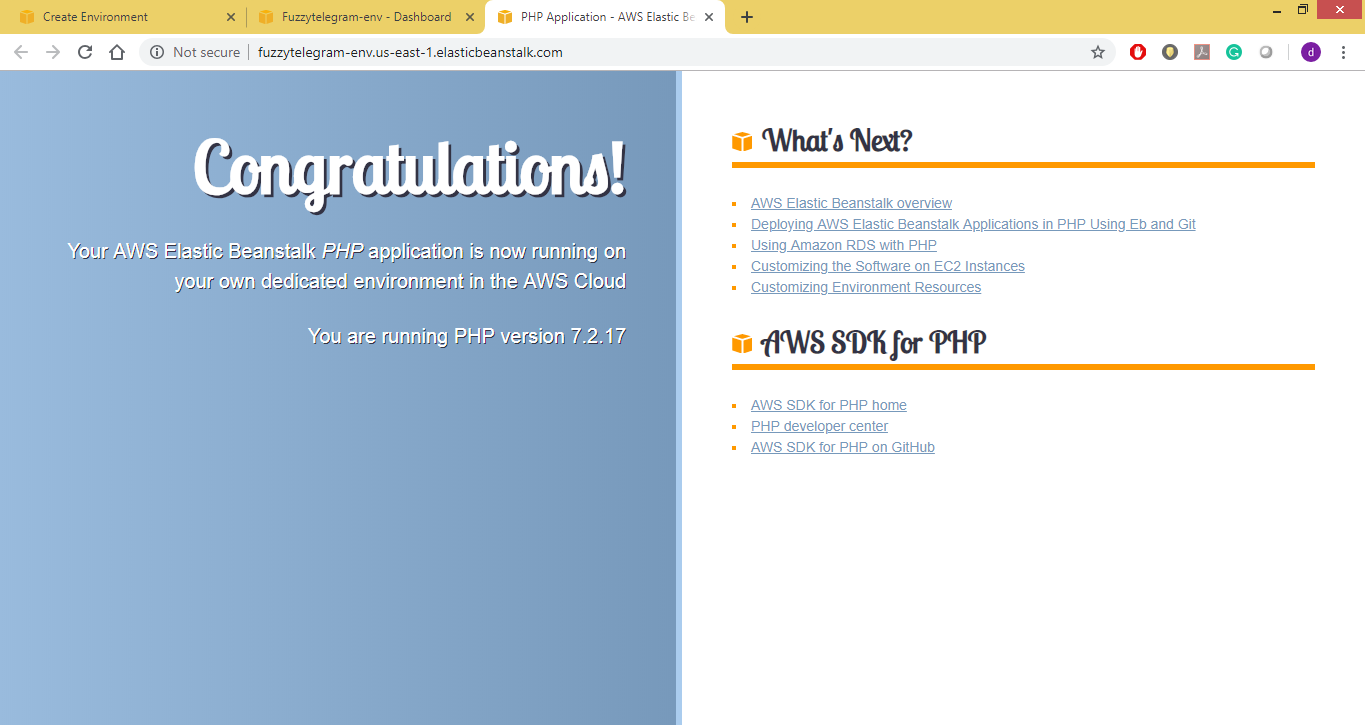


Figure 4: Sample Application Launched in AWS Elastic Beanstalk

Now we will zip the code files and upload it to the Elastic Beanstalk Application. After Uploading, it will automatically deploy the Elastic Beanstalk Application. After the Deployment is completed, it will show the Application Environment as in Figure 5.

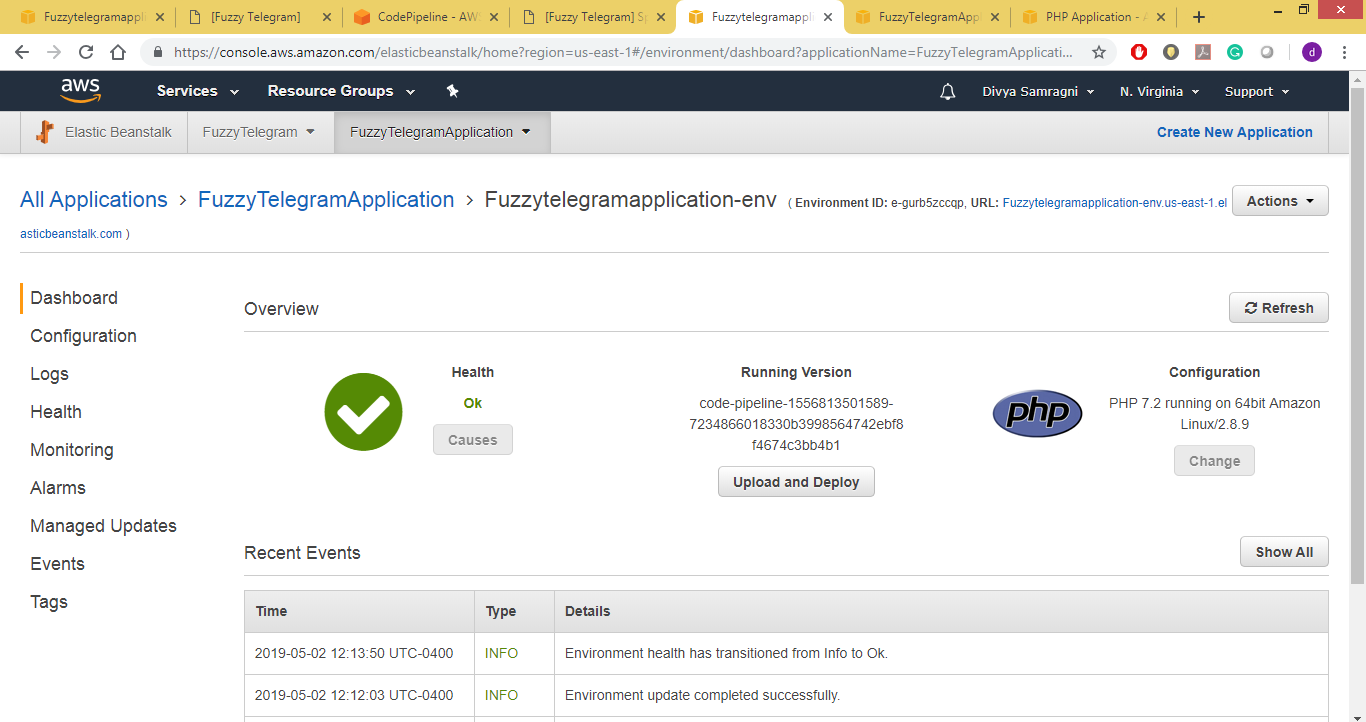


Figure 5: Health Status of Elastic Beanstalk environment after deploying the PHP Web Application.

When you click on the url available in the Elastic Beanstalk Environment, the following application Page should open as shown in Figure 6.

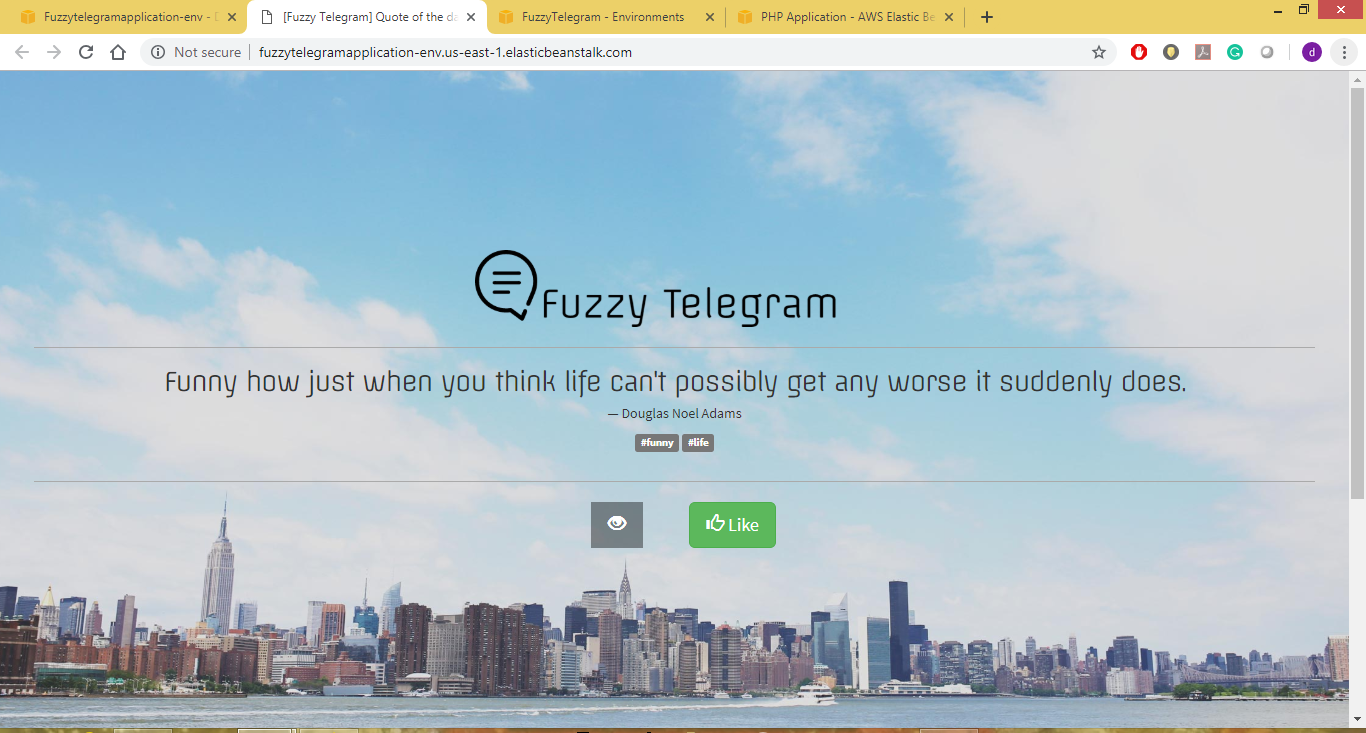


Figure 6: PHP Web Application launched on AWS ELastic Beanstalk.

We use Elastic Beanstalk extensions to run custom scripts. We then use the same procedure to add a new environment called Production to the Application created in Beanstalk. We use the same procedure like we created the Beta Application. In the versions section of Elastic Beanstalk, you can check the versions of code deployed into the Beanstalk. Figure 7 shows the different versions of code deployed.

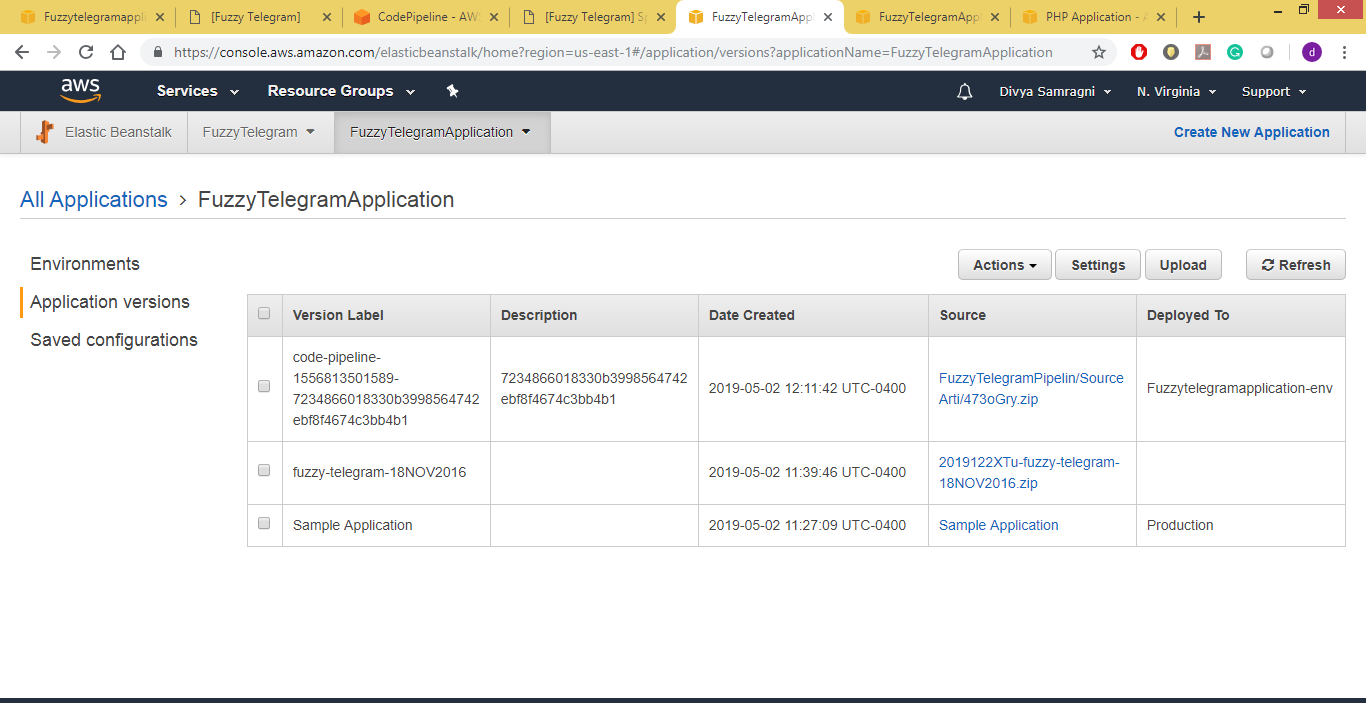


Figure 7: Application Versions of Code deployed in Elastic Beanstalk.

Setting a CI/CD pipeline using AWS CodePipeline:

We first commit all the code in Github. Below is the link of the code in Github.

<https://github.com/DivyaSamragniNadakuditi/fuzzy-telegram>

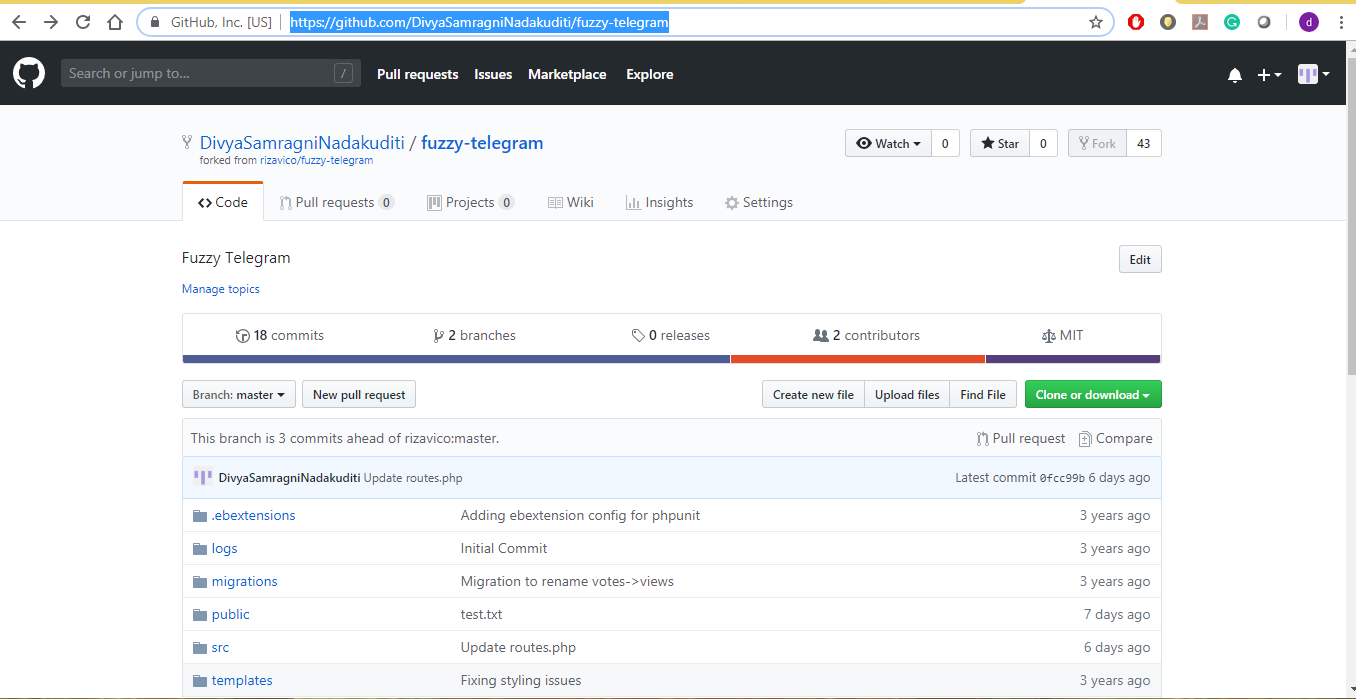


Figure 8: Code in Github.

Now Create a new Codepipeline. Figure 9 shows the creation of new pipeline for our Application.

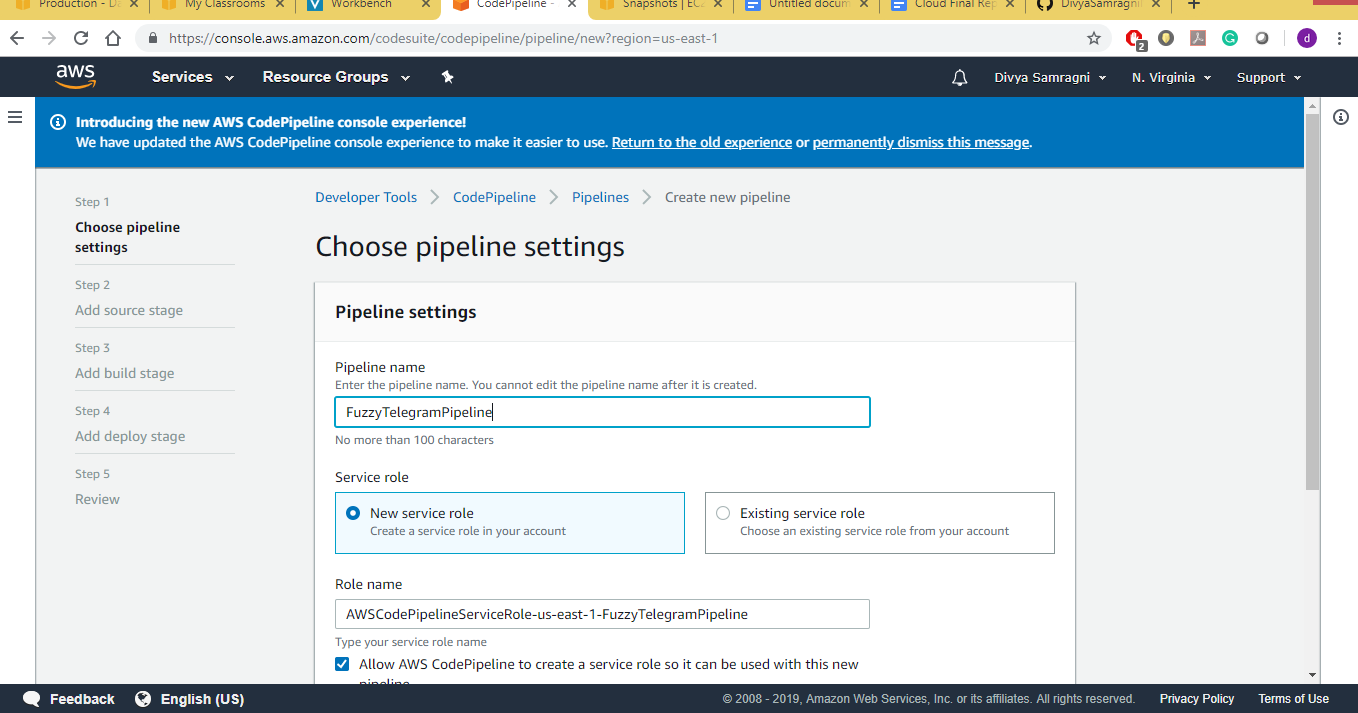


Figure 9 : Pipeline Creation for our Application.

Also set the Deployment to Elastic Beanstalk.

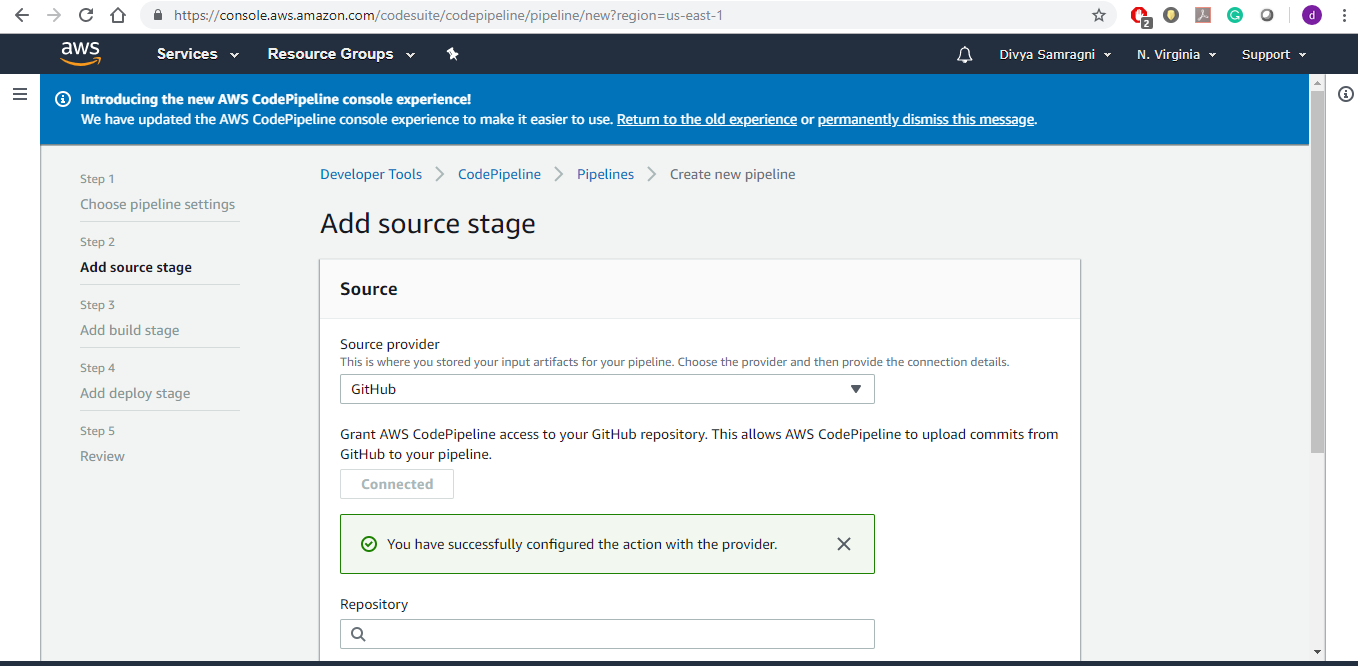
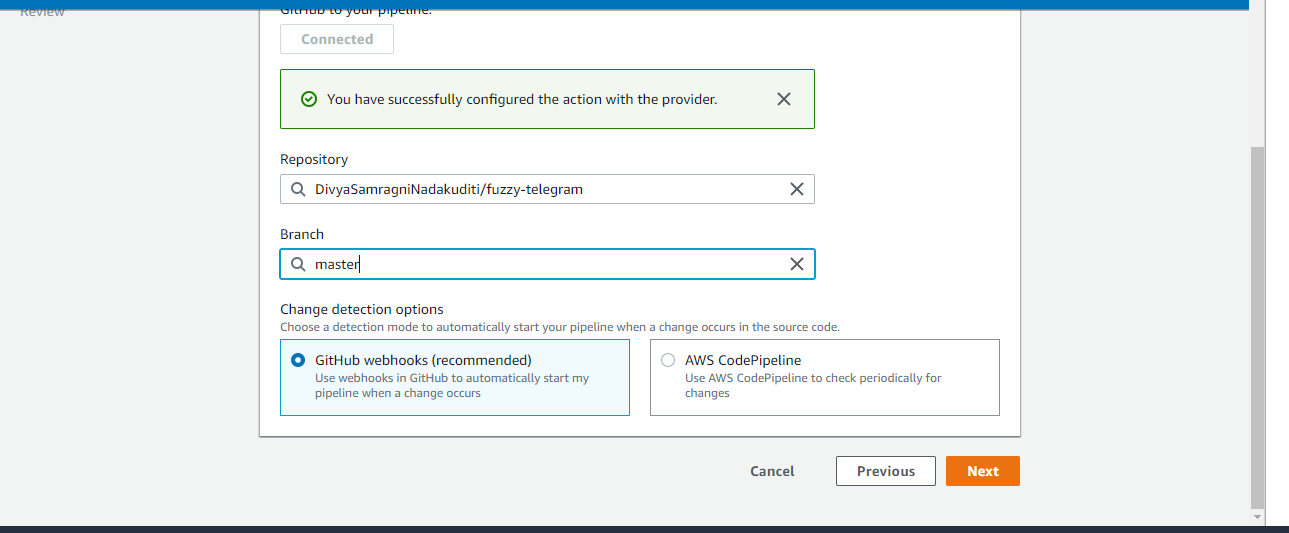


Figure 10: Source Provider details



**Webhooks to detect deployment changes: -**

For Github, we have used default system which is webhooks to recognize the changes made in Github code. Webhooks is one of the powerful ways to analyses and import the changes in cloud integrations. Web is event system

Webhooks are tightly integrated with Particle's event system. Devices have the ability to both [publish events](https://docs.particle.io/reference/device-os/firmware/#particle-publish-) to the Particle cloud, as well as [subscribe to events](https://docs.particle.io/reference/device-os/firmware/#particle-subscribe-) from the cloud.

A webhook listens for a specific event published by a device. When this event is published, the webhook triggers a [web request](http://rve.org.uk/dumprequest) to a URL on the web. The request sent by the webhook can include information about the event, such as its name as well as any data included when the event was published.

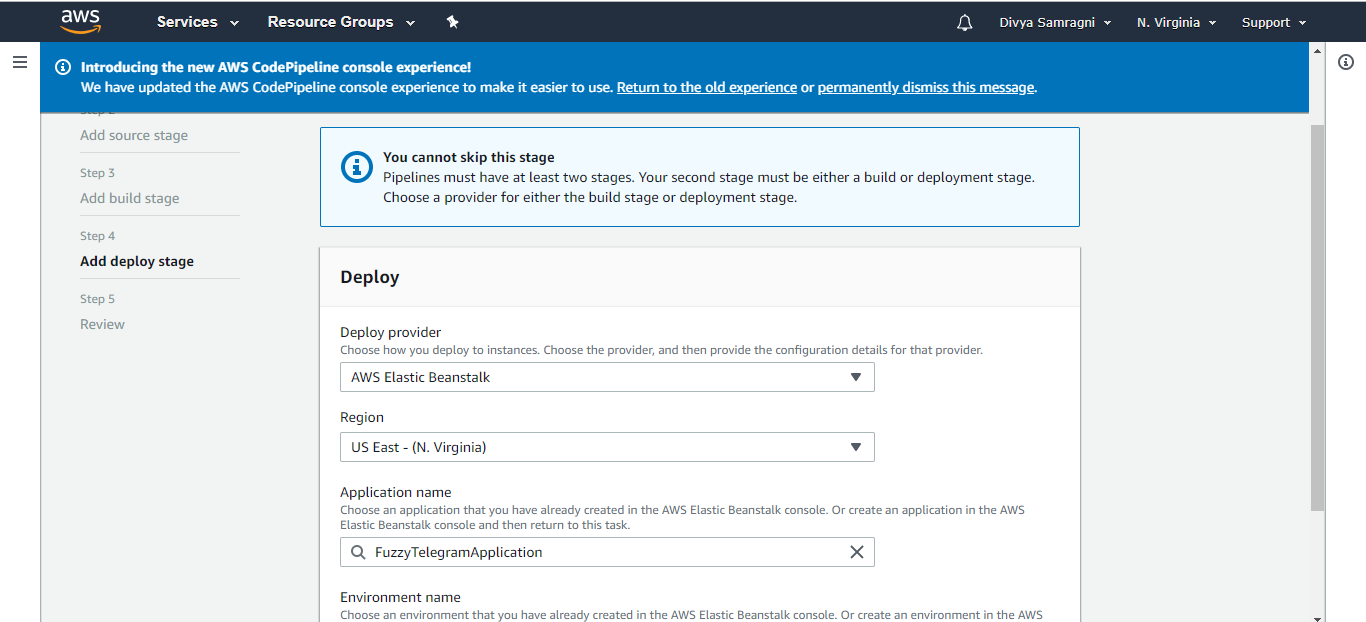


Figure 11: Deployment details

Below are the Figures of the Codepipeline Created Build and Deployed.

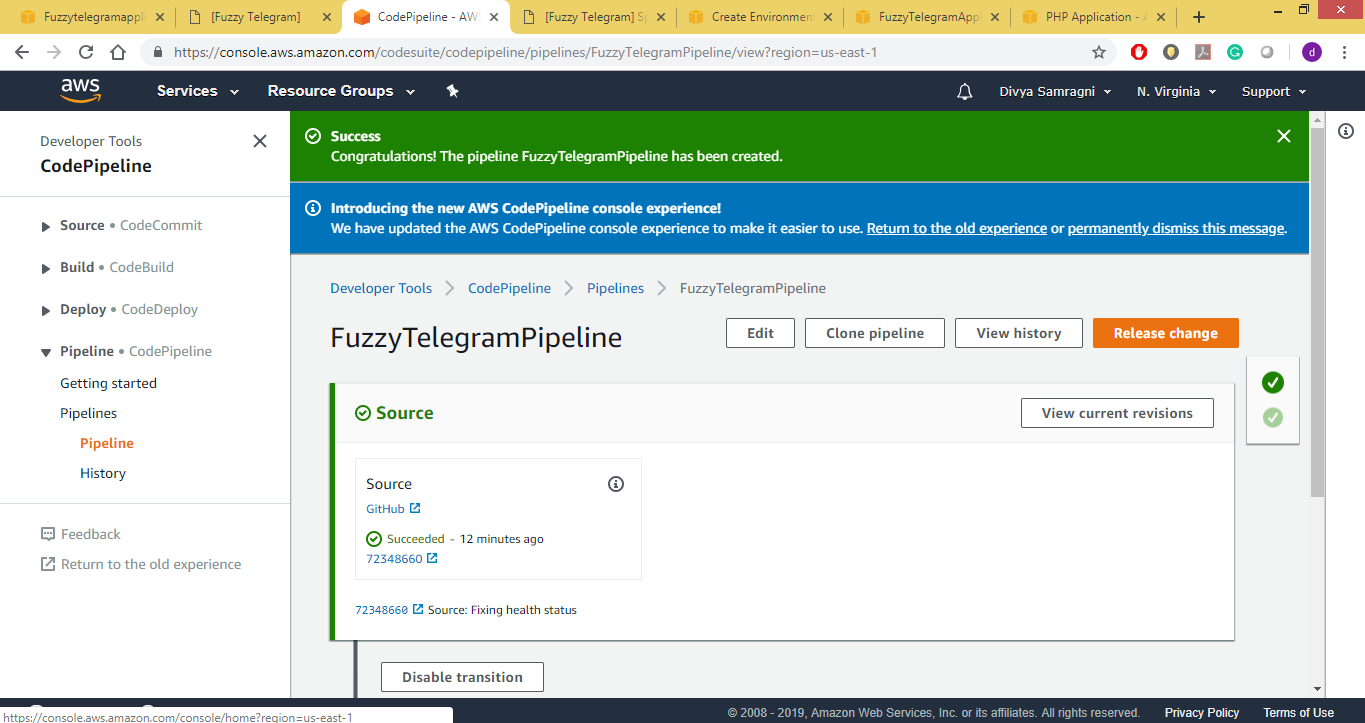


Figure 12: Codepipeline Successfully Build.

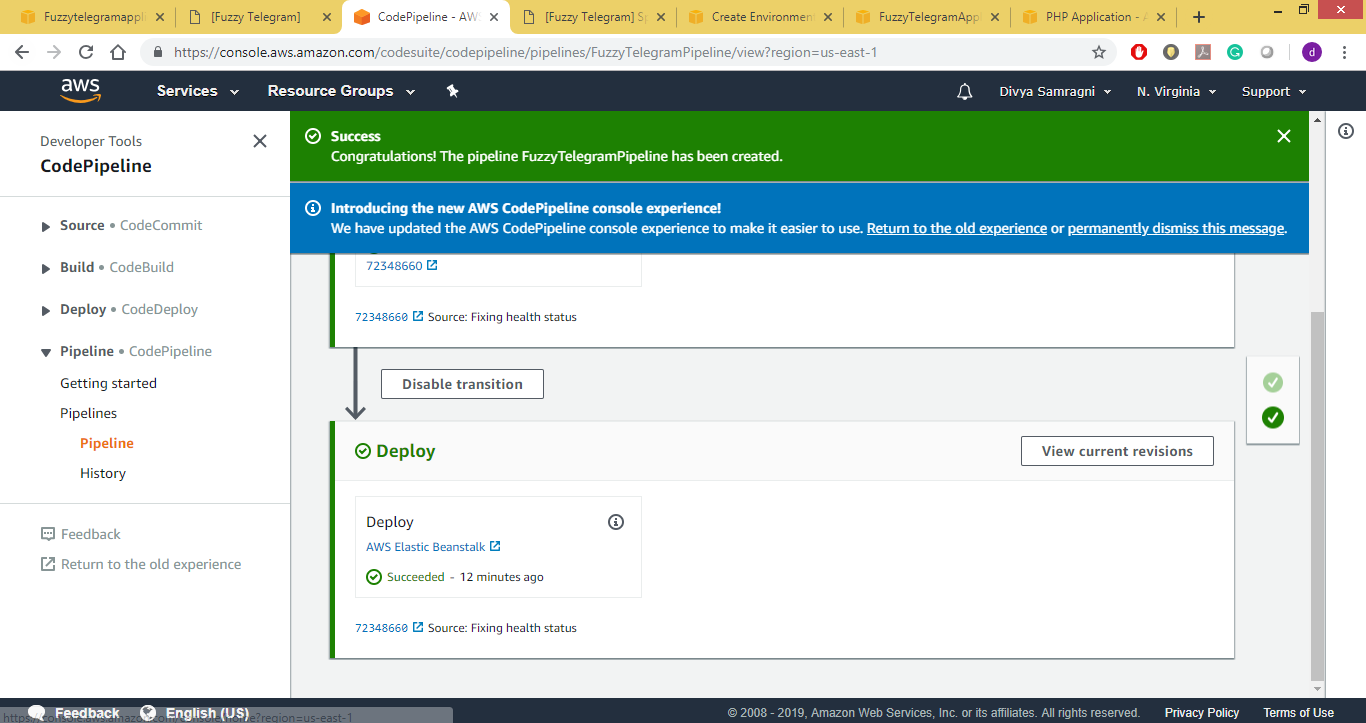


Figure 13: Deployment Successful.

We further add a stage called Production.

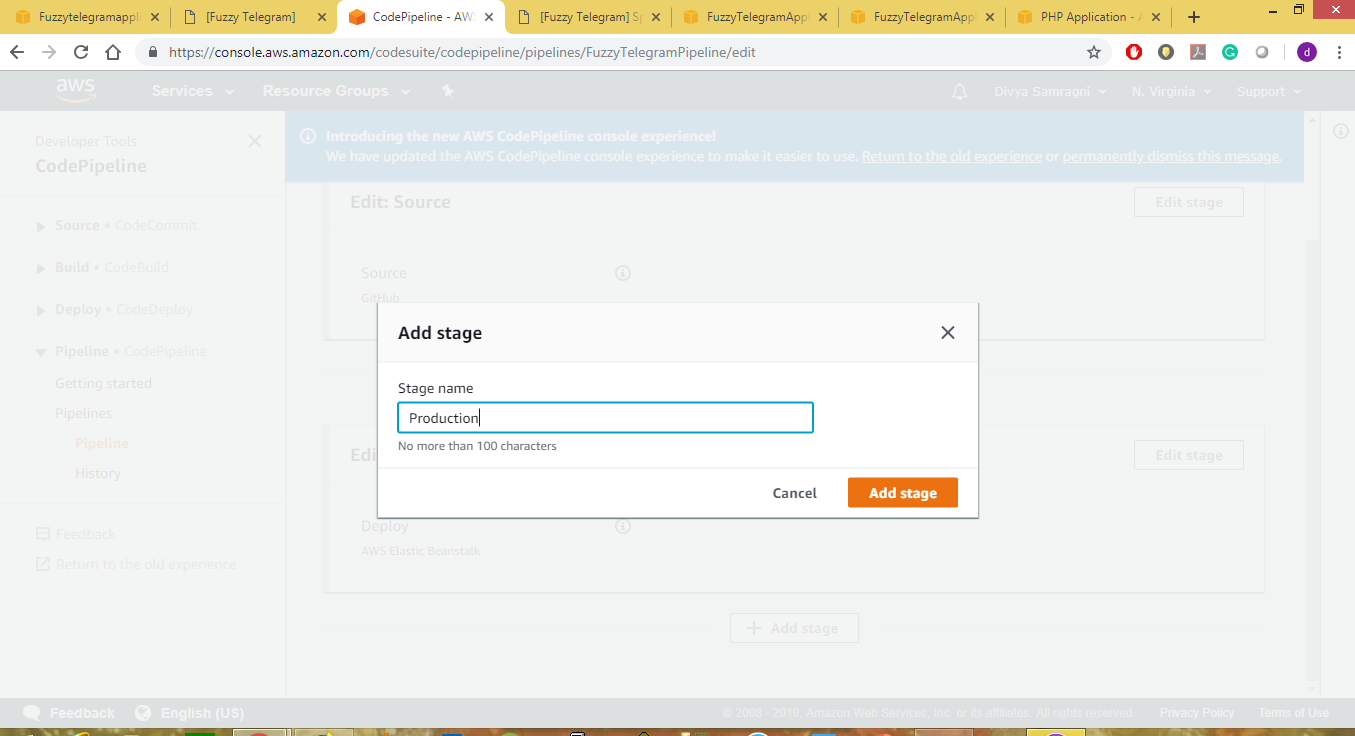


Figure 14: Adding new stage called Production

We have added cloud watch, to analyses the health of deployed Elastic Beans stalk environments and SNS notifications. Also this logs are stored in S3 bucket.

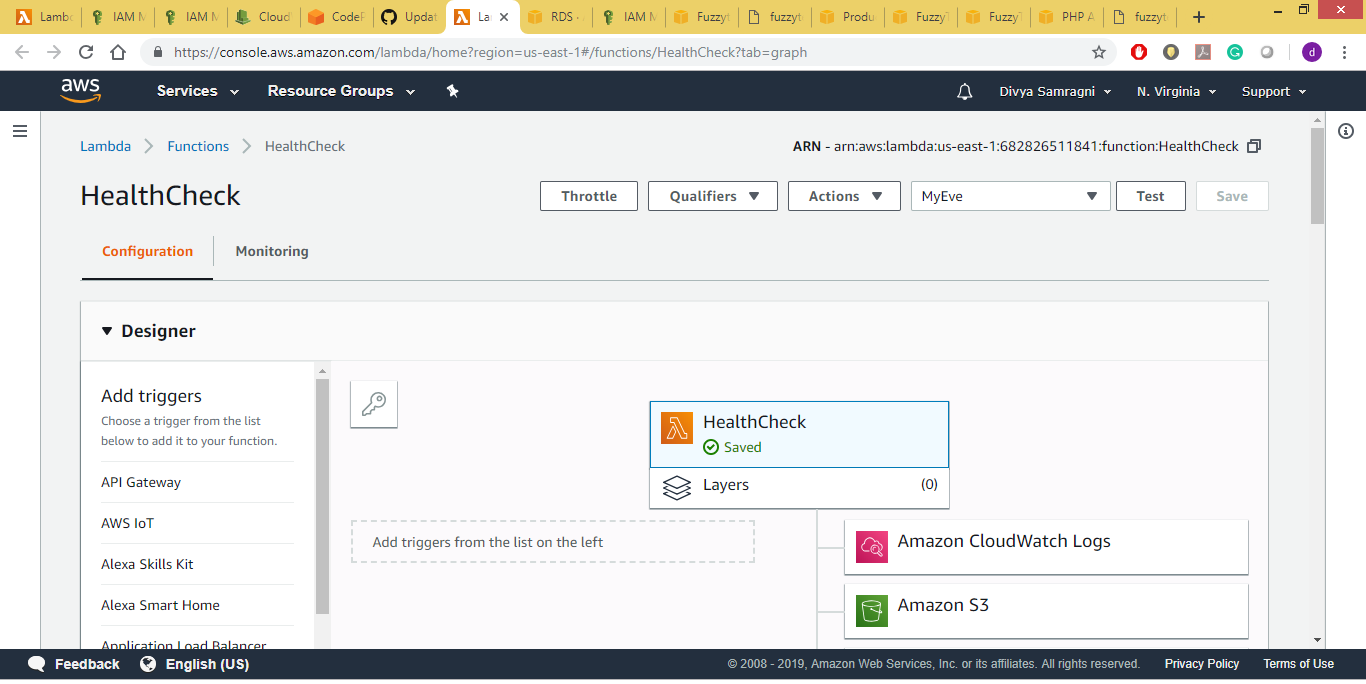


Fig 14. Adding Lambda function to check the health status of ElasticBeanstalk.

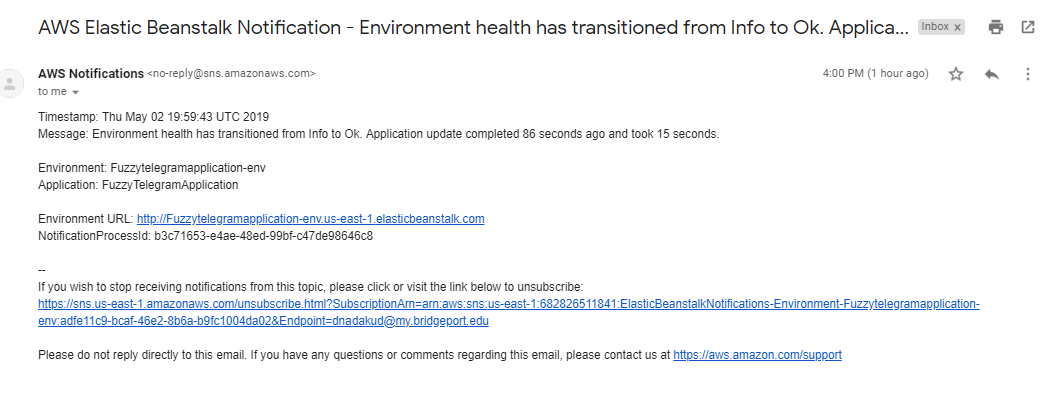


Fig 15. This is an email notification sent to client.

Conclusion: we have successfully implemented DevOps: Continuous Integration(CI) and Continuous Deployment(CD) using AWS CodePipeline & AWS Elastic Beanstalk