National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cloud Computing

Dilip Singh
Student ID: x19208073

School of Computing
National College of Ireland

Supervisor: Jitendra Kumar Sharma

Configuration Manual

Dilip Singh
x19208073

1 Introduction

1.1 Purpose of the document

Based on the requirements of the NCI Research project, this Configuration Manual is completed. This document describes the software tools and settings required to optimise the AWS Elastic BeanStalk performance.

1.2 Document structure

Section	Purpose
General Information	The experimental environment setup is explained in this module, which explains the requirements for project.
Setup prerequisites	For development and update of the solution, this module describes steps necessary for setting up the development environment.
Deployment procedure	The deployment procedure for a proposed model is described in this module
Validations	This module describes the requirements for validating the success of the deployment of the solution

2 General Information

2.1 Objective

The objective of this research work is to optimize the application deployment time of AWS Elastic BeanStalk using custom script. The script was written in python 3.8 and utilizes concurrent.features library to enable thread level parallelism. With the help of thread level parallelism the application files will get deployed simultaneously.

2.2 Architecture requirement

AWS services required for building a Composite model are described in this section.

2.2.1 AWS Elastic BeanStalk

AWS Elastic BeanStalk service is required to deploy the application code and optimise the performance of it. The Elastic BeanStalk is used to inject the custom script to enable thread level parallelism [1].

2.2.2 AWS Simple Storage Service(S3)

AWS S3 is used for storing application source during application deployment with custom code [2].

2.2.3 AWS CodePipeline

AWS CodePipeline service is used to establish a pipeline connectivity between GitHub repository where application code is stored and AWS Elastic BeanStalk [3].

2.2.4 AWS CloudWatch

AWS CloudWatch is used for continuous monitoring of deployed application and to record the application deployment without custom code[4].

2.2.5 AWS Elastic Cloud Compute(EC2)

The AWS EC2 instance is used to deploy the application and further inject the custom code inside it[5].

2.2.6 GitHub

GitHub repository is used as application source provider while deploying it without custom code.

2.3 Required Skill

It is assumed that you already have a basic understanding of Amazon Web Services before reading this guide. The user must also be familiar with the Python language to create functions and understand the code.

[1] `https://aws.amazon.com/elasticbeanstalk/`
[2] `https://aws.amazon.com/s3/`
[3] `https://aws.amazon.com/codepipeline/`
[4] `https://aws.amazon.com/cloudwatch/`
[5] `https://aws.amazon.com/ec2`

3 Development Environment Requirement

3.1 Code Repository

Please refer to the zip file I have submitted in the ICT solution.

3.2 Programming language required

- Python Version 3.8
- Boto3
- Shell scripting

3.3 Creating Elastic BeanStalk Environment

Before deploying application we need to create a AWS Elastic BeanStalk environment. Below are the steps required to create an BeanStalk instance.

- Navigate to AWS Elastic BeanStalk service and click on 'create a new environment' button.
- Select the environment tier i.e. Web server environment and click next.
- Now provide the environment details such as application name, environment name, application platform type and application code (default sample application) and click on 'create environment' button as shown in figure 1.

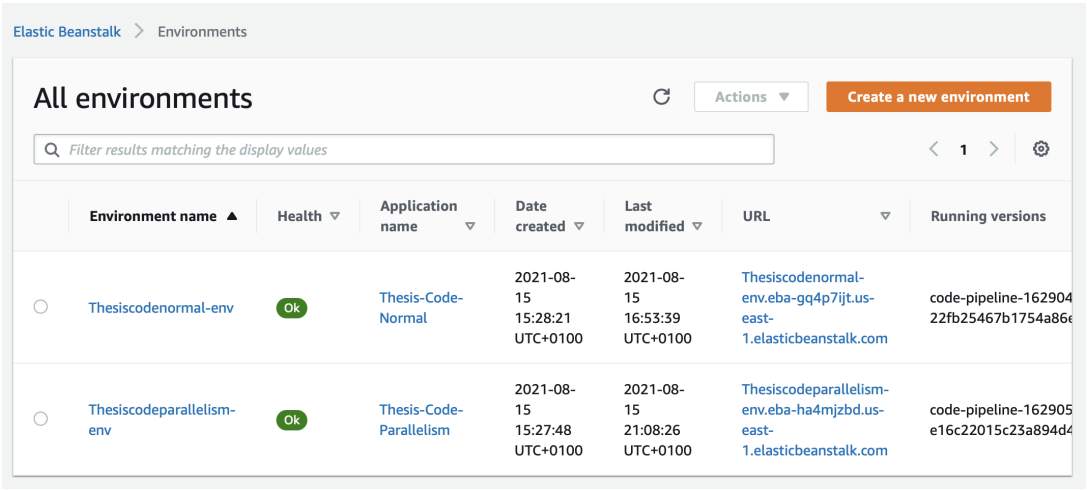

Figure 1: Elastic BeanStalk environment creation

3.4 Configuring security group

On successful creation of AWS Elastic BeanStalk we will locate the BeanStalk instance and modify its security group. Since, we don't get any SSH key from Elastic BeanStalk to access its instance. Hence, we will modify the security assign to the instance to get web based SSH access. Steps are as follows:

- Navigate to EC2 service and search for instance having suffixed named of Elastic BeanStalk environment name.
- Now, click on instance id and navigate to 'Security' tab. Click on security group assigned to instance and edit the inbound traffic rule.
- Add SSH rule and source address as 0.0.0.0/0 (it will allow access to instance from anywhere) as shown in figure 2
- click on 'save rule' button and try connecting with instance using EC2 instance connect option as shown in figure 3.

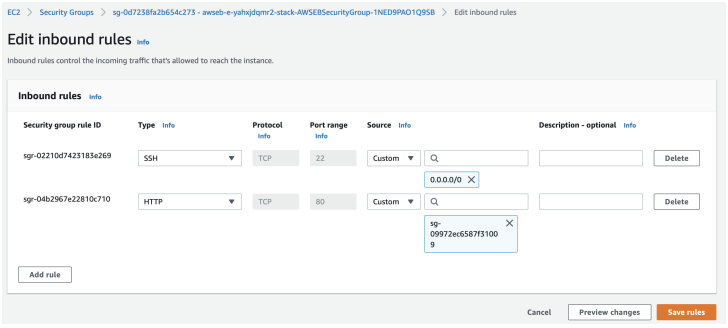

Figure 2: Security group configuration

Figure 3: Elastic BeanStalk instance interface

3.5 Uploading application source code to AWS S3

Due to resource restriction by AWS we cannot modify the AWS Elastic BeanStalk environment directly. Therefore, we will upload the application source code in AWS S3 bucket and manually check the application deployment time. Below are the steps for uploading application file to AWS S3:

- Navigate to AWS S3 service and click on 'Create Bucket' button.
- Now enter the bucket name(must be unique), allow all public access and click on 'Create Bucket' button.
- After successful creation of bucket we will upload the application code inside it as shown in figure 4.

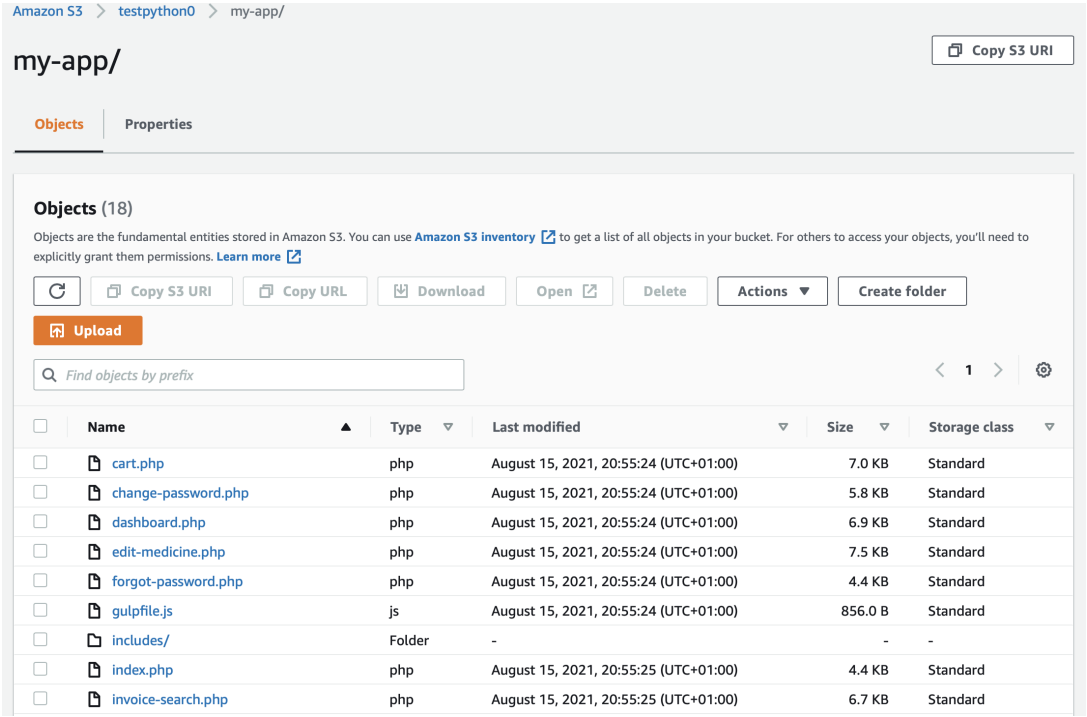

Figure 4: AWS S3 bucket interface

3.6 Creating AWS CodePipeline

Following are the steps required for creating AWS CodePipeline.

- Navigate to AWS CodePipeline service and click 'Create Pipeline' button and enter the pipeline name.
- In next interface select the source provide i.e. GutHub Version 1 and select the application repository without containing .ebextension directory.
- Now skip the build phase step and select the deployment provider i.e. AWS Elastic BeanStalk and select the deployment environment.

- Review the pipeline configuration and wait for pipeline to be created. As shown in figure5 the pipeline is created

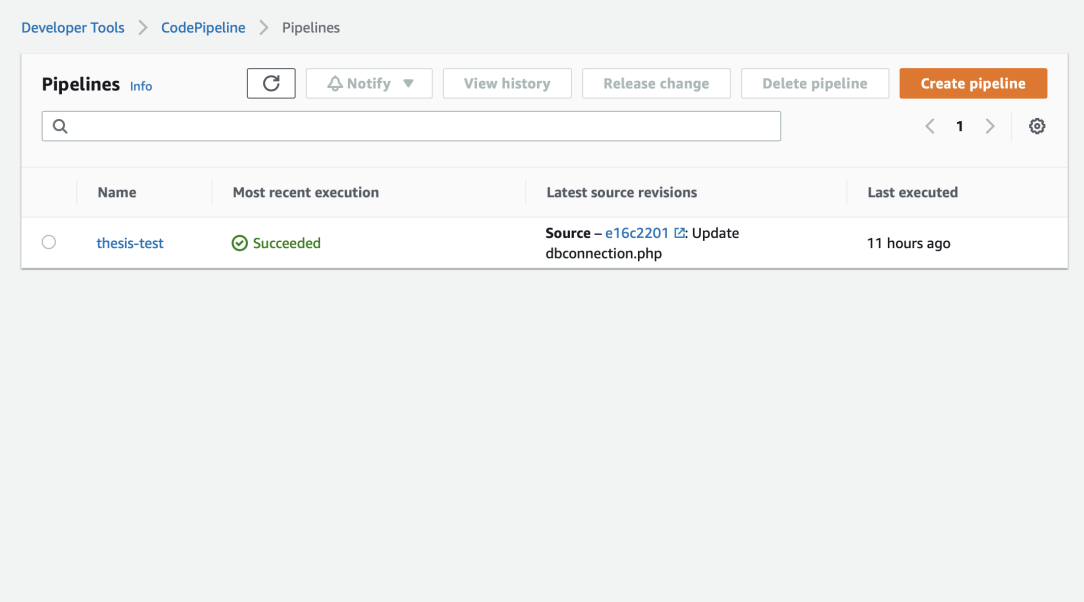

Figure 5: AWS Elastic BeanStalk environment interface

3.7 Configuring and injecting custom code inside instance

After successful SSH into instance we will inject our custom code stored in GitHub repository. Below are the steps for configuring:

- Inside instance navigate to root directory by typing 'cd /'
- Now git clone the custom code repository using command 'git clone url'
- On successful git clone, move the custom code file at / directory. The code snippet is shown in figure 6
- Now create an executable script file named 'run.sh' and type the script shown in figure 7
- Finally install the Boto3 libaray using command 'sudo pip3 install boto3'.

main-code-thesis.py

```python
import boto3
import os
import time
import concurrent.futures

def downloadDirectoryFroms3(directory):
    s3_resource = boto3.resource('s3')
    bucket = s3_resource.Bucket('testpython0') # name of our bucket
    for obj in bucket.objects.filter(Prefix = directory): # for-loop will iterate all files in
        if not os.path.exists(os.path.dirname(obj.key)): # check if there is directory or not
            os.makedirs(os.path.dirname(obj.key)) # make directory if not exist
        bucket.download_file(obj.key, obj.key) # save all files to ngnix server path


def main():
    t1 = time.perf_counter()

    size = (1200, 1200)

    direct = [
        'my-app'
    ]
    with concurrent.futures.ProcessPoolExecutor() as executor:
        executor.map(downloadDirectoryFroms3, direct)

    t2 = time.perf_counter()

    print(f'Finished in {t2-t1} seconds')

if __name__ == '__main__':
 main()
```

Figure 6: Custom code snippet

run.sh

```
#!/bin/sh

python3 code-normal.py && mv -v /my-app/* /var/www/html && rm -rf /my-app
```

Figure 7: Executable script snippet

4 Validation

- To run the experiment simulation without custom code. Navigate to GitHub and edit/update any application file and click 'commit' button. On successful code commit AWS CodePipeline will automatically start deploying the application in Elastic BeanStalk environment as shown in figure 8.

- Now to run the experiment simulation with custom code SSH into BeanStalk instance as explained in section 3.4. Type './run.sh' to execute the run.sh script. The script will trigger the custom code and start application deployment process as shown in figure 9.

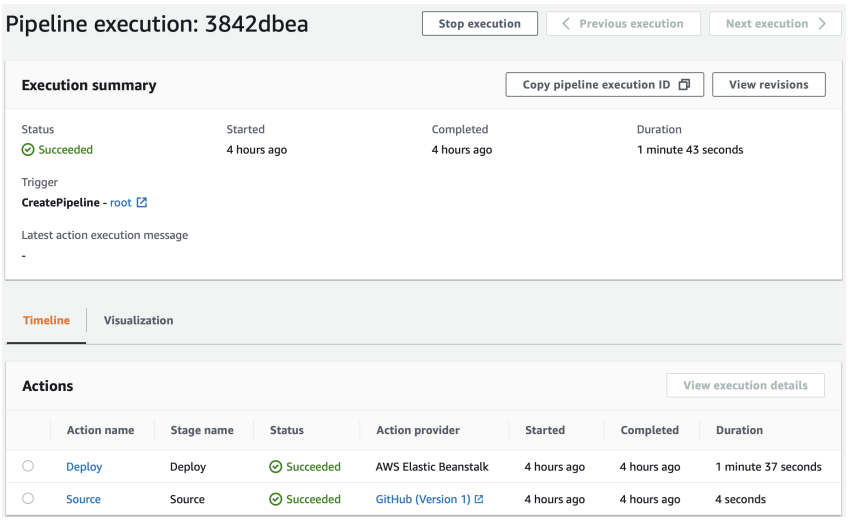

Figure 8: Application deployment without custom code

Figure 9: Application deployment with custom code