Optimizing Continuous Deployment Performance Using Multi-Level Thread Parallelism

MSc Research Project
Cloud Computing

Dilip Singh
Student ID: x19208073

School of Computing
National College of Ireland

Supervisor: Jitendra Kumar Sharma

Optimizing Continuous Deployment Performance Using Multi-Level Thread Parallelism

Dilip Singh
x19208073

Abstract

With the rapid growth of Agile software delivery trends Continuous Integration and Continuous Deployment(CICD) has gained a wide adoption across IT organizations. Continuous Integration and Continuous Deployment(CICD) are the basic pillars of DevOps which enable rapid software deployment with quick feedback. It also bridges the gap between operations and the developer team. Recent studies show that adopting DevOps practices for software development sometimes creates additional overhead for developers due to overly long build time. It becomes even worse when waiting for a build to successfully finish gets fail. To overcome this barrier, the paper emphasis improving Continuous Deployment(CD) performance to reduce the application deployment makespan. The motive is to optimize the performance of AWS Elastic BeanStalk using multi-level thread parallelism in Python. I have used a custom script written in python 3.7 to invoke multiple threads for deploying application files from the AWS S3 bucket to the AWS EC2 instance. The entire software delivery can be improved while reducing the costs by lowering the execution time for Continuous Deployment (CD). For the experiment, we have used AWS cloud and evaluated various matrices such as execution time, CPU utilization, a cost that occurred before and after optimization.

1 Introduction

Traditional software development techniques are insufficient to meet today's corporate needs Riungu-Kalliosaari et al. (2016). As the software market becomes more competitive, companies devote more attention and resources to developing and delivering high-quality software at a faster speed. Currently, software development has progressed to the point where regular software upgrades are supplied concurrently without hampering software use. All this was feasible due to DevOps Continuous Integration and Continuous Deployment(CICD) practices which enable efficiency, flexibility and speed Software Development Life Cycle(SDLC). Previously the development team pushes new software versions into production regularly, but the operations team blocks modifications to ensure software stability and other problems. This method ensures greater software production stability. In practice, however, this strategy results in a considerable wait between a software update and deployment Leite et al. (2019). To overcome the problem of long delays DevOps practices were introduced. **?** proposed the notion of Continuous Integration (CI) in the year 2010 expanded this concept into Continuous Delivery (CD) as a deployment pipeline concept. The main advantages of CI methods are that they reduce risk and ensure that software is bug-free and dependable, removing the barrier to frequent delivery. **?**, defined DevOps as "a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production while ensuring high quality".

Continuous practices provide various benefits such as (1) Obtaining more frequent and timely input from customers and the software development process; (2) releasing software updates on a regular and consistent basis, resulting in higher customer satisfaction and product quality; (3) CD strengthens the link between development and operations teams, and many manual activities can be removed Shahin et al. (2017). Traditionally in software

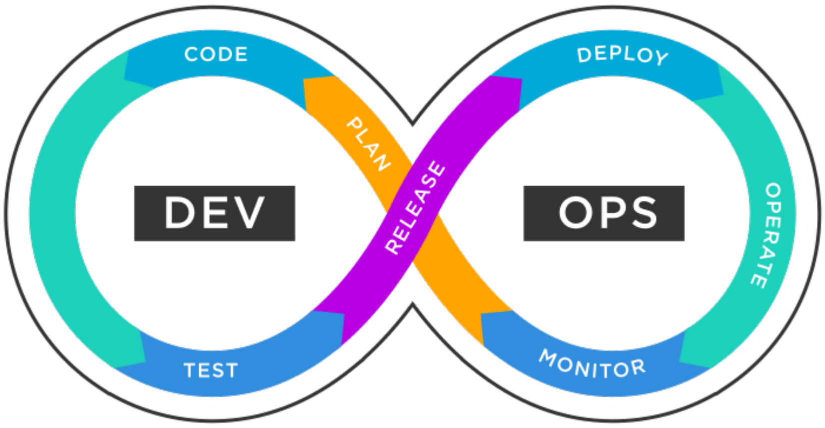


Figure 1: Phases of DevOps

companies, two teams were formed (Developer team and Operation team) to developed and monitor the performance of the software. But with the rising conflicts between two teams during software development organization came up with the idea of DevOps. Where only one team holds the accountability of both development and operations. In the figure, various phases involved in DevOps are shown. The phases are as follows (1)Plan, (2)Code, (3)Build, (4)Test, (5)Release, (6)Deploy, (7)Operate and (8)Monitor. During the planning phase, the developer team along with various stack holders analyses the market need and evaluates the functional and non-functional requirements. After that, the next phase i.e code phase starts where the development team starts coding and code integration takes place considering functional and non-functional requirements discussed during the planning phase. On successful integration, the building is performed on designed code. If the code successfully qualifies the build process then the various test cases were formed to check the correct working of specified modules. After the test phase, the software versioning is performed and a unique label is assigned to the qualified software bundle. After which labeled software version is pushed into the deployment phase to make it available for end-users. Even after the successful deployment of the software into the production environment, its functionality is continuously monitored by the same team (but now as an operation team) for any kind of run-time malfunction. If any kind of malfunction is identified during time-time, the alert is the gene, rated and again the whole cycle iterates.

Nowadays various automation tools were introduced to ease the DevOps practices such as Jenkins, but with every, tool there are some limitations. In the proposed study, I will analyze the advantages and disadvantages of employing DevOps in software development. DevOps is used by many IT organizations for continuous software development. The proposed research focuses on reducing the time required to deploy an application during the continuous development (CD) phase. The AWS Elastic BeanStalk will be utilized to reduce the time required for continuous development (CD). The design of AWS Elastic BeanStalk is outlined in this paper, and the factor responsible for application deployment delays is addressed. Task parallelism is offered to reduce application deployment execution time. AWS Elastic BeanStalk incorporates the custom code. Threads are used in the custom code to task-level parallelism. The suggested study is assessed using AWS CloudWatch measurements for CPU use and execution time while deploying applications with and without custom code.

1.1 Motivation

Speed up the deployment time of AWS Elastic BeanStalk using custom code written in Python 3.8 which multi-level level parallelism to deploy the application. The main aim is to provide fine-grained performance to Elastic BeanStalk to enhance the overall CI/CD experience during software development.

1.2 Research Question

The main objective is to reduce the time required for deploying and application in cloud infrastructure. Analyzing and finding the existing limitations of AWS Elastic BeanStalk and optimize it to achieve fine-grained performance. Hereby this research proposal solves the following question:

Can makespan of Continuous Deployment(CD) be improved by implementing multi-level thread-based parallelism using python for different cloud providers e.g. Amazon Web Services(AWS)?

1.3 Report Structure

Section 2 describes the related work conducted in the area of DevOps and agile software development. It describes the limitations and solutions of each area for various researchers. Section 3 describes the steps and required configuration to optimize the performance. Section 4 elaborates the existing model, proposed model, code sample and environment setup steps. Section 5 and 6 show the implementation and evaluation matrices used to compare the performance of Elastic BeanStalk with and without integration of custom code. Finally, section 7 describes the conclusion and future scope of the proposed model.

2 Related Work

This section elaborates various research related work in Continuous Deployment(CD). This section is divided into four parts 2.1 Review on DevOps, 2.2 Review on Continuous Deployment, 2.3 Analysis of related work.

2.1 Review on DevOps

There has been an increased interest in DevOps, which necessitates identifying the challenges, practices, and tools needed to enhance the performance of CICD. A systematic review on the existing issues in DevOps from 2004 to 2016 was conducted by Shahin et al. (2017). Their study identified thirty techniques and tools that could assist in improving DevOps practice. The author utilized various search methods, search terms, data sources, inclusion and exclusion criteria, selection of relevant studies, and comprehensive data collection. Search methods were carried out through six libraries such as IEEE Xplore, ACM Digital Library, Springer, Wiley Online Library, Science Direct, and Scopus by using an automatic search method. A paper that is published in two different publishers and addresses the same issue should be rejected if it appears to address the same issue. After analyzing the paper author found the techniques that can increase the performance of CICD are (1) decreasing build and test time of Continuous Integration(CI), (2) enhancing accessibility and understanding in build and test scenarios of CI, (3) facilitation of semi-automated tests, (4) identifying errors and flaws in CI, (5) resolving security and compatibility issue while deploying pipeline, and (6) increasing dependency on the reliability of deployment methods. Researchers identified several critical factors to consider when performing continuous practices, such as testing, team awareness and operations, design principles, appropriate infrastructure, and customers.

Software testing procedures that are efficient and reliable are important because of system complexity and global competition. Before releasing the software, thousands of test cases are executed in manual testing. Tests must be thoroughly and efficiently carried out to ensure that the systems are properly tested. In Flemström et al. (2018) author, propose that ordering test cases before automating them can minimize test effort, assuming reuse sections of test cases have also been automated. Using a case study from a major Swedish vehicle manufacturer, the author analyzed multiple approaches to prioritization.

The research survey by Leite et al. (2019) discusses the benefits and challenges of DevOps. Several Finland software companies were interviewed for this survey. By implementing DevOps practice, the author observed that the software can be released more frequently and also tested more thoroughly. During analysis, there are challenges relating to communication structures, diverse development and working environments. There was a finding that DevOps is not appropriate for all organizations.

Development and IT Operations collaborate more effectively through DevOps practices. A tool called filling-the-gap (FG) is presented in Perez et al. (2015). The FG tool provides performance information to the developer, thereby enhancing and automating the software delivery process. In this tool, two things are accomplished: (1) providing a Quality-of-Service (QoS) model, and (2) providing at-runtime application behavior reports to developers. The tool consists of three components, namely: runtimes for FG, monitoring history databases, and design runtimes for FG. The FG tool records and analyses the data after application deployment to provide a QoS model. In FG design, estimation routines execute to fetch monitoring data from monitoring databases to update the application. Configuration and deployment are the first steps in the workflow for the FG tool. To launch configuration, deploy and configuration users interact with FG Design-time components. The time component analyzes the application behavior at runtime based on the configuration of FG Design and provides the application results along with the details regarding the application performance.

The concept of DevOps involves development (Dev) and operations (Ops) team collaborating more frequently. Such integration is driven by the need to continuously adapt enterprise applications (EAs) to provide variations in the business environment. Development and operations of software are known collectively as DevOps. In Brunnert et al. (2015) the author describes the operations, resources, and processes that facilitate consistency, an important quality attribute of software systems. The study outlines performance-based DevOps integration activities using measurement and model-based techniques. The author discusses ways to improve performance during the lifecycle of a software system and provides resources for doing so. It was discovered that the compatibility of the different methods and approaches is a vital aspect of all collaborative activities.

The software industry depends heavily on the development and operation of its software. DevOps is a fundamental component of their business. Artač et al. (2016) proposed a DevOps Quality-of-Service(QoS) enhancement method utilizing a DICER tool. The DICER tool consists of two modules (1) the modeling environment and (2) the deployment service. The deployment element can be dragged and dropped on the modeling environment dashboard and properly configured. ModaClouds4DICE has been implemented using MODACloudsML in the proposal. The MODACloudsML programming language outlines how multi-cloud applications will be deployed and provisioned. Once a user submits their initial model, the tool automatically transforms it to a TOSCA model. MODACloudML can be easily extended to support deployment models on top of TOSCA by extending the language. REST APIs are used to retrieve TOSCA blueprints from the front-end service. It simplifies the deployment procedure by allowing users to drag and drop files. With deployment service, a simple web browser can be used to run it, but the tool is designed to be integrated into large, complex DevOps workflows.

The author of cites inproceedings2 describes a security-centered DevOps approach. In the paper, the security controls were handled by a toolkit. The author used source code evaluation to secure the project. Integrating security into the DevOps process is the first step. In the production environment, however, additional features should be introduced, since more resources are needed to identify, detect, and respond to threats. The author identifies the main challenge to developing continuous integration and continuous development as the lack of automated, goal-oriented acceptance tests.

DevOps implementation poses some security issues. These issues need to be addressed quickly. PROMOTHEMEE-II was presented by Rafi et al. (2020) as a model to prioritize and develop a taxonomy to identify and classify DevOps security factors. PROMOTHEMEEE-II is based on the analysis of alternatives against each criterion in pairs. The phases in PROMOTHEMEE-II include normalizing the decision, comparing alternatives pairwise, calculating preference functions, calculating aggregate preference functions, calculating net outranking flow, and prioritization. From the results of the prioritization process, it was discovered that the absence of automation tools, security manual testing, and performance configuration were the primary security concerns in DevOps. - The study contributes to the understanding of ambiguity and vagueness associated with DevOps implementation to improve and streamline continuous software development.

Author Laukkarinen et al. (2017) discusses how DevOps affects medical device development. The research examines how to use IEC 62304 and IEC 82304 standards for clauses in DevOps. The IEC 62304 standard applies to medical devices that contain hardware from the manufacturer whereas the IEC 82304 standard applies to health software that executes on general-purpose equipment. Based on the experimental analysis, the author determined that IEC 62304 clauses related to software unit testing and software deployment were the primary obstacle to DevOps adoption. However, clauses regarding the repeatability of software development practices and deployments benefited DevOps. For DevOps, the post-market reporting clause of IEC 82304 posed an obstacle.

Many areas involve security issues, especially critical infrastructures like the national health system. To provide the best software and services to citizens, security processes, procedures, and resources are needed. It is not yet clear how security should be incorporated into the DevOps pipeline. Author Larrucea et al. (2019) the author discusses security management in the DevOps pipeline. Source code analysis is the first step in integrating protection into the DevOps process it must be performed during the integration phase. In research, real-life scenarios of the healthcare industry are used. An author points out steps to improving security in a sophisticated DevOps environment. The fact that malicious code contained inside files has been linked to some of the most severe security breaches is a major consideration. In this article, the author examines a DevOps approach of integration that includes security steps. During the DevOps pipeline, the security controls were managed using a toolkit. DevOps is becoming more secure through introducing protection into the process, but other features should be addressed later, such as in the production environment where there is a need for more resources to identify, detect, and stop threats.

Guerriero et al. (2015), presented SPACE4Cloud. SPACE4Cloud provides model-driven QoS and capacity allocation during cloud deployment, optimized at the design time. Java was used to implement the SPACE4Cloud environment. To perform the experimental analysis, EC2 instances are used with varying workloads and virtual machines (VMs). Upon deployment into MODACloud, SPACE4Cloud creates an optimization model for the application. By the end of the paper, the author argues that the proposed strategy achieves better performance at a time interval of 5 minutes.

2.2 Review on Continuous Deployment

Software development using Continuous Integration (CI) offers several benefits, including the development of high-quality software. However, many practices can optimize continuous integration. Zampetti et al. (2020), expose bad practices experienced by the developer when performing continuous integration. In total, 79 different CI bad smells were compiled and classified into 7 categories. Bad smells can be classified into the repository, infrastructure selection, build process organization, build maintainability, quality assurance, and delivery process. There was an insufficient build time for the commit phase, inappropriate build environment clean-up, use of monolithic builds, unnecessary tests in the build process, and a lack of notification mechanisms. A bad repository smell results from a poor repository organization and inappropriate use of version control systems (VCS). Infrastructure choice bad smell happens when CI pipelines are set up with irrelevant hardware and software components. Most bad smells occur because of the improper configuration of the CI pipeline, such as using monolithic builds in the wrong place. A build script malfunction leads to the build phase problem. Quality-of-Service(QoS) can be optimized by improving testing and statistical analysis before merging the branch. The configuration of the test coverage threshold, as well as the lack of clear separation between the test suits and the test activities, impacts QoS. Storage artifacts are accountable for bad smells associated with delivery, such as an improper deployment strategy. In a production environment, the bad smell can affect the software release.

Continuous Integration (CI) is a process for integrating code frequently. To avoid break builds and delays in getting fast feedback, it's essential to ensure that sufficient testing is performed on code. A cost-efficient continuous integration algorithm was proposed by Elbaum et al. (2014). When it comes to continuous integration, the author emphasizes two factors: continuously selected tests(RTS) and test case selection(TCP). When TCP is used, test cases are put in an appropriate order. When RTS is used, test cases that are important to run are selected for execution. A time window and RTS were created to track recent executed test cases. Based on Google's large data sets, an empirical study was conducted to determine how the RTS technique could significantly improve continuous testing's overall cost-effectiveness. With TCP techniques, developers can significantly reduce the feedback time in CI, allowing them to address problems more quickly. An experimental setup is based on the Google Shared Dataset of Test Suits Results (GSDTSR) which consists of 3.5 million test suits. In the algorithm, test cases are selected based on their prioritization, which is informed by previous test failure history. Test cases that fail will be executed first rather than being repeated after every successful one, which will save time.

Using DevOps, the software can be developed more quickly and efficiently. A long build cycle in Continuous Integration (CI) is one of the factors that slow down the DevOps process. The Continuous Integration phase could be optimized by making changes to the build test. A technique is presented by Marijan et al. (2018) that includes fault- and risk-based test case identification, as well as prioritization, to achieve a low runtime. Test cases that are most relevant to the code change and the risk functionality are significant in reducing runtime. Using a test optimizer, the author proposed selecting the appropriate tests from the test case database for updated code. As soon as the build phase is complete, a log file containing all test case results, i.e. pass or fail, will be created. Test cases will be retested if they fail. Code is deployed in a production environment after passing a successful test case. As soon as the application is deployed into the production environment, the monitoring process will begin, which will monitor it. It will update the operational log if there is an operational failure. If the developer provides an upgrade path to the optimizer, it will check the previously failed test cases from both operational logs and test logs during the patching phase. With the implementation of this log and the running of only critical and failed test cases, the build execution time has been slashed. On implementing the proposed technique, Cisco video conferencing system was used for experimental analysis and resulted in a 35% reduction in execution time.

Danglot et al. (2019) suggests that developers improve the overall test cases suite written by engineers to improve continuous integration(CI) stages of DevOps. DSpot is the name of their proposed system. DSpot is a tool that automates the Test Case Optimizer process for Java Unit Testing, based upon Tonella's and Xie's algorithms. Test cases written by developers can be improved by the proposed tool. To integrate the improved test cases into the main code branch of the test code, developers receive them as patch requests or pull requests. Ten real-world open-source software projects are used to develop 40 real-world unit test cases for the spot. The author found that by adding new assertions and triggering new behavior, 26 of 40 test cases could be improved. The author provided improved test cases developed by DSpot for 10 open-source software projects. 13/19 developers accepted the proposed improvement test case. DSpot is capable of automatically optimizing test cases for real-world large Java applications after tests were done with DSpot.

Lehtonen et al. (2015) discuss metrics that are suited for supporting continuous delivery and deployment using a comparative case study on a project of a mid-sized software company. Based on the tested data, the author investigates new metrics that describe the pipeline's properties. Information gleaned from metrics is used to enhance pipelines and related processes. When considering continuous delivery methodologies, the throughput of a pipeline was measured on how quickly functionality is delivered to an end-user before deployment. In the proposal, the author presented metrics on features per month (FPR), releases per month (RPR), and the fastest possible feature lead time.

An agile development approach has made software delivery more efficient by using Continuous Integration and Continuous Development (CICD). According to Arachchi and Perera (2018), introduces a new approach to CICD that consists of four optimization phases, including benchmarking, load testing, scaling, and provisioning. To monitor the abnormal behavior of the product, Nagios was used to monitor the Jenkins CI server, Git repository, Nexus repository with Ansible automation, and Jenkins CI server with Git repository was used for implementing pipelines. Tests were conducted by a simple software application built on XML. A benchmark phase includes first assessing the production environment's capabilities and limitations, and afterward, evaluating the product's benchmark level. The CI server initiates benchmarking and load-testing of APP4 using it as the benchmark server. Benchmarking and load testing are carried out to identify any performance issues with the new product. Based on the benchmark level, a scaling factor was used to determine new requirements for the software. In the next phase, provisioning is done, which is not mandatory. It was determined that each phase would take 20 minutes to complete after the performance analysis. The system became unstable when the load reached 100 percent. The response time of the system improves when scaled according to benchmarks. Accordingly, test benchmarks have demonstrated that the system continues to process the same amount of load when software is updated.

Dlugi et al. (2015), developed a CI plug-in for Jenkins that implements a model-based performance change detection process. The proposed model generates the resource profiles based on successful build tests. A resource profile demonstrates how a transaction moves between control and resources. Artifacts are stored as the resulting resource profiles. Every time a new release of the software is released, a resource profile is generated. Resource profiles were also used to compare the performance of the previously examined application versions. The resource profile compactor analyzes the control flow to find the reason for a performance regression found during the performance test. Based on the proposed model, a visual report and a listing of methods are generated for performance metrics.

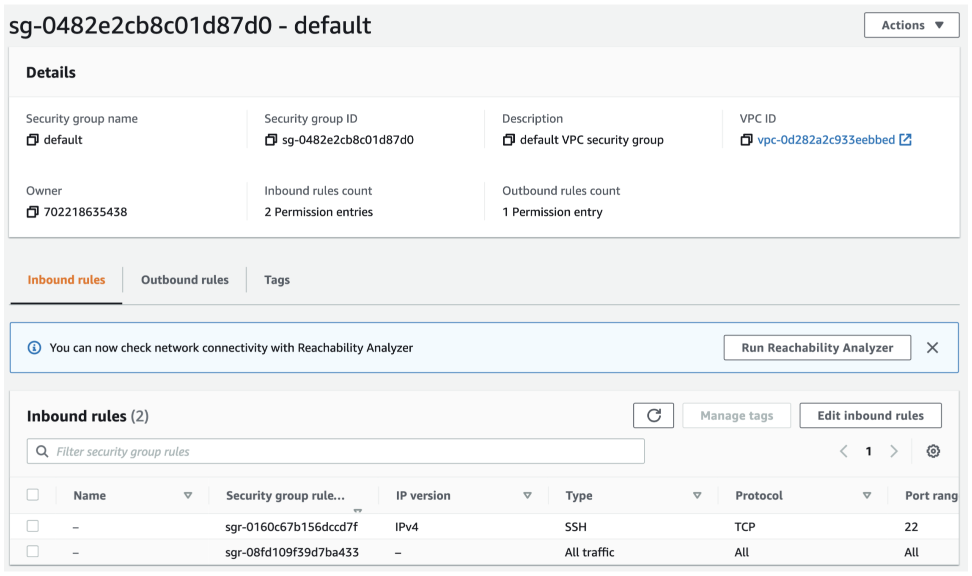
As DevOps is gaining traction and being adopted by many organizations, it is possible to integrate DevOps CICD with high-performance resources. Author of Sampedro et al. (2018) explains how CICD can ease challenges of software management for HPC (High-Performance Computing) resources. The authors explain how Summit is deployed in the HPC environment and why it uses tools such as Jenkins. Software deployment and continuous integration are performed with Jenkins, an open-source automation tool. A variety of configuration styles are supported. The pipeline has been used in the paper since it is easily integrated into a version control system. A containerized application called Singularity makes it possible to conduct reproducible research through mobility across compute resources. As a central location for managing singularity images, the singularity registry service was adapted from the cloud service Singularity Hub. Registries such as Singularity Registry provide local storage optimization and on-site authentication. Multiphase flow with interphase exchanges (MFiX-Exa) at exascale are presented for experimental analysis. National Energy Technology Limited(NETL) created MFiX, a computational fluid dynamics(CFD) model. As part of the CICD pipeline, Jenkins pools the code commits for MFiX-Exa. Jenkins begins CI jobs with a singularity registry. According to the results of the CICD, HPC workflow practices can further increase the likelihood of developing high-quality, reliable software.

2.3 Summary of related work

Developers and operators use DevOps to efficiently create and deploy applications and receive feedback rapidly. In Shahin et al. (2017), Leite et al. (2019), Laukkarinen et al. (2017) and Rafi et al. (2020) authors elaborate the existing tool, challenges and practices such as testing time for CI, deployment time while performing CD, security testing and lack of automation in DevOps. According to Leite et al. (2019), DevOps is not suitable for every organization to optimize CI performance. Further, Zampetti et al. (2020) uncovered problems while performing CI in DevOps. In Perez et al. (2015), Artač et al. (2016) and Dlugi et al. (2015) contribute to improving DevOps through automation and ensuring Quality-of-Service(QoS). By reducing execution time and cost, the Arachchi and Perera (2018), Elbaum et al. (2014) and Marijan et al. (2018) authors researchers improve continuous integration and continuous deployment. In my research, I will examine the existing challenges in continuous deployment, with a focus on improving the execution time in DevOps. Reduced CD execution time will also reduce deployment costs.

3 Methodology

This paper tries to optimize the performance of Continues Deployment lifecycle of software by reducing the time required to deploy software. The Amazon Web Service(AWS) platform is used to optimize performance. By injecting the custom code inside the AWS Elastic BeanStalk instance. Although AWS provides Elastic BeanStalk to fully automate the process of the CI/CD sometimes it requires time to fully deploy the software into production. The GitHub act as an application source provider which will trigger the custom code script to reduce the deployment time. AWS CodePipeline is used to provide connectivity between GitHub and Elastic BeanStalk. AWS CodePipeline is used to capture the time required by each phase during application deployment.



sg-0482e2cb8c01d87d0 - default

Details

Security group name	Security group ID	Description	VPC ID
default	sg-0482e2cb8c01d87d0	default VPC security group	vpc-0d282a2c933eebbed
Owner	**Inbound rules count**	**Outbound rules count**	
702218635438	2 Permission entries	1 Permission entry	

You can now check network connectivity with Reachability Analyzer

Inbound rules (2)

Name	Security group rule...	IP version	Type	Protocol	Port rang
–	sgr-0160c67b156dccd7f	IPv4	SSH	TCP	22
–	sgr-08fd109f39d7ba433	–	All traffic	All	All

Figure 2: Security group configuration

The initial step includes setting up the AWS Elastic BeanStalk and locating the instance created by Elastic BeanStalk and SSH into it. The custom code is written in python 3.7 and utilizes concurrent features to enable thread-level parallelism. With the help of thread-level parallelism, multiple files of applications deploy concurrently. The custom code is injected inside Elastic BeanStalk after SSH into the instance.

3.1 Material and Equipments

Using Amazon Web Service(AWS), this paper optimized the performance of Continuous Deployment(CD). As shown in figure **??**, by modifying the assigned security group inbound traffic rule i.e by allowing SSH rule. This will allow us to SSH into the Elastic BeanStalk instance to inject the custom code.

3.2 Sample Data

The sample application written in PHP is used to test the performance of AWS Elastic BeanStalk before and after injecting the custom code. For experimental purpose, various application size was used such as 1MB, 5MB, 10MB and 15MB.

4 Design Specification

This section describes the design specification of the proposed research work. Sub-section 3.1, elaborates the existing system problem. In sub-section 3.2, the proposed System Architecture flow is explained.

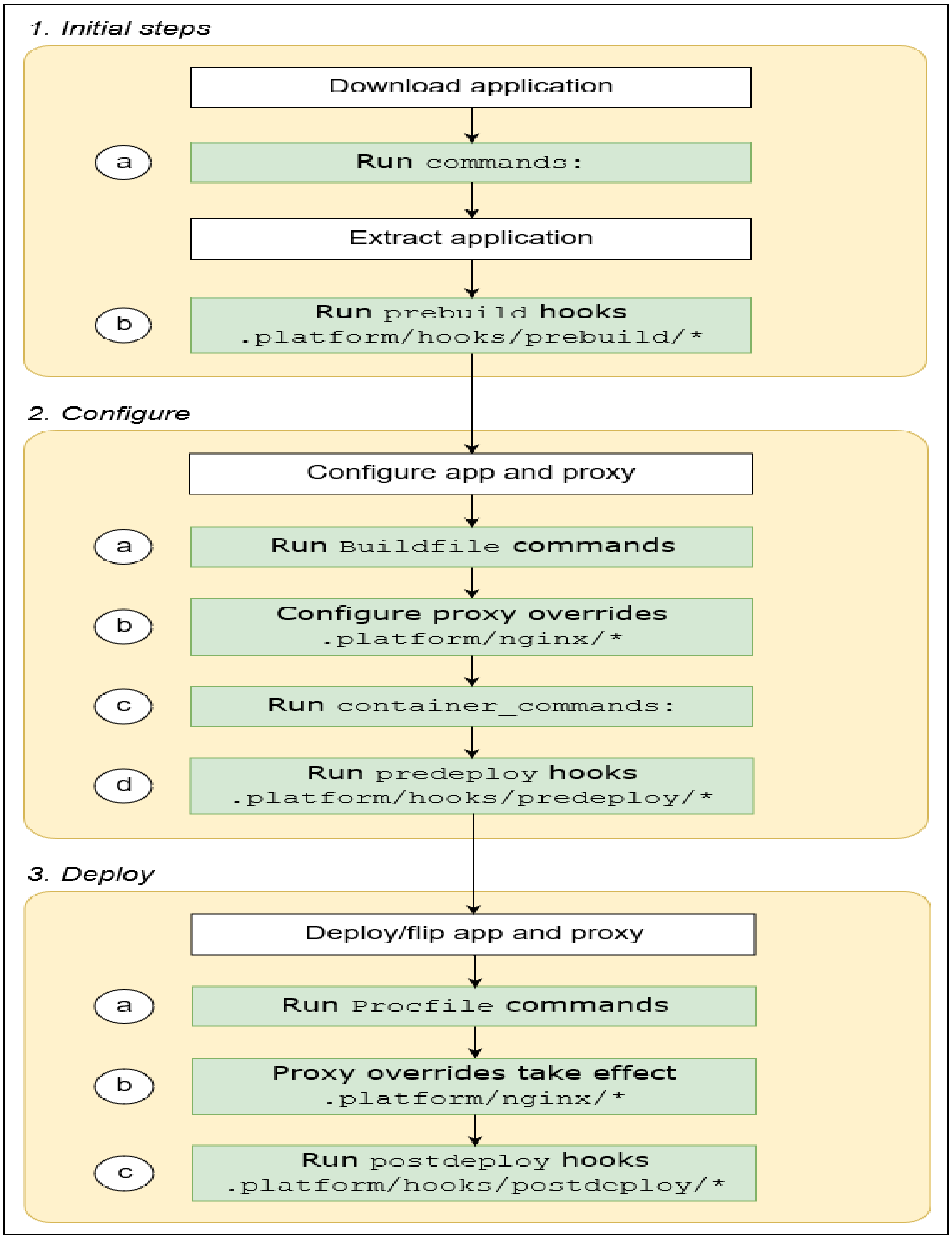


Figure 3: Existing System Architecture Amazon (n.d.)

As shown in figure 3, the internal architecture of AWS Elastic BeanStalk consists of Initial steps, Configure, Deploy. The initial phase begins with downloading the application zip package followed by running commands for extracting the application and prebuild hooks. The next phase is Configure phase, in this phase application configuration and proxy were set up. This phase includes running build file commands, configuring proxy overrides, container commands(if any), and finally running pre-deploy hooks. On successful completion of the above phase, the final phase begins i.e Deploy. In this phase profile, proxy overrides effects and post-deploy hooks take place. In the proposed research, we are going to focus on configuring the phase of Elastic BeanStalk by triggering our code during the pre-deploy phase.

4.1 Proposed System

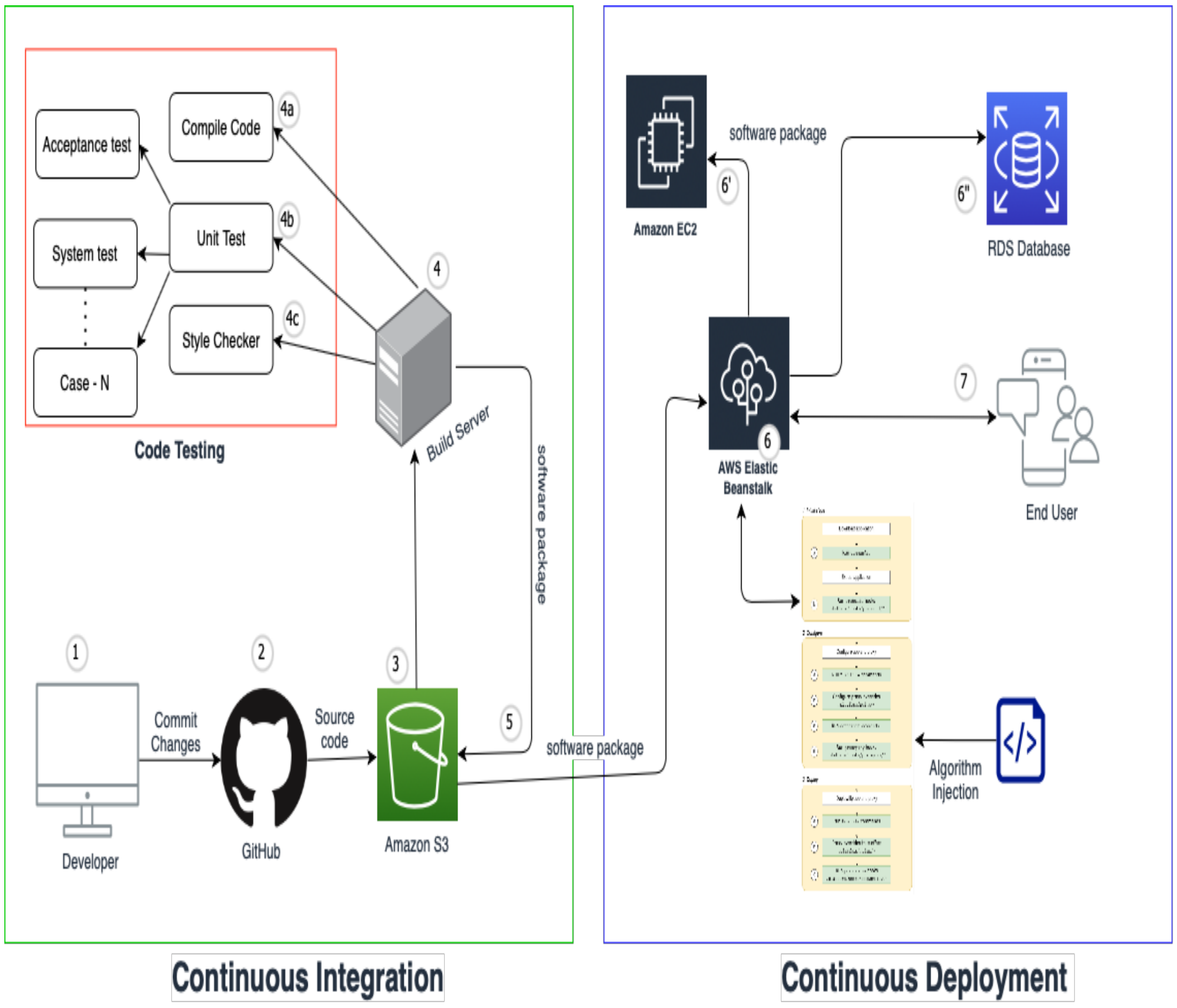
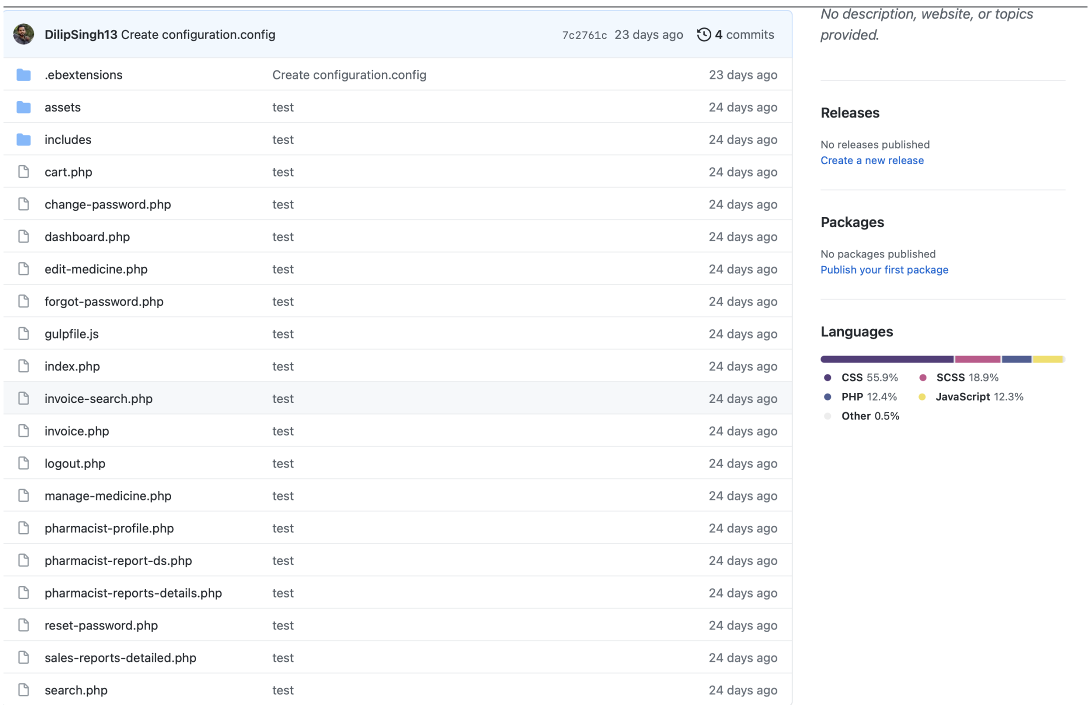


Figure 4: Proposed Research Architecture

Figure 4 depicts the proposed system architecture. The proposed system emphasis towards Continuous Deployment(CD) phase of application development. Whenever a development team commits a code change in the GitHub repository it will trigger the AWS Code pipeline which will sync the application source code from GitHub to the AWS S3 bucket. After that, the application package goes to the build server which will test the application against desired build case. If the application bundle successfully passes the build case then it will proceed further for deployment else it stops further deployment of the application. Build server responsible for code compilation, unit test, style checker and many more. After a successful build, the build server will create another repository inside AWS S3 which contains all the application files ready for deployment. Now the actual Continuous Deployment(CD) phase begins. In the CD phase, AWS Elastic BeanStalk will manage and deploy the application automatically. On the successful deployment of an application, Elastic BeanStalk provides the unique URL through which the deployed application can be accessed by end-users. In this process of CI and CD, the proposed system will improve the capability of AWS Elastic BeanStalk by injecting custom code inside the Elastic BeanStalk instance internal schema. The custom code was written in python 3.7 and the import concurrent feature library. Using a concurrent feature library enables thread-level parallelism in python. Concurrent features library provides asynchronous execution of a task with threads using ThreadPoolExecutor [1] further map function maps the source location(AWS S3 build the application) and destination location(Elastic BeanStalk instance). Irrespective of the application platform the developed custom code can be integrated with any application to decrease the application deployment duration.

5 Implementation

To implement the proposed research work, I have used Amazon Web Services(AWS) cloud. For performing CICD, AWS Elastic BeanStalk is used. The Elastic BeanStalk automatically creates an EC2 instance to deploy the application. In research, the main task is to finding and penetrate inside the Elastic BeanStalk EC2 instance. To penetrate inside Elastic BeanStalk instance we need to modify the existing security group as shown in figure 2. After modifying the security group we will SSH direct using the web interface. After penetrating the inside instance we will do some internal configurations such as:



DilipSingh13 Create configuration.config — 7c2761c 23 days ago — 4 commits

.ebextensions	Create configuration.config	23 days ago
assets	test	24 days ago
includes	test	24 days ago
cart.php	test	24 days ago
change-password.php	test	24 days ago
dashboard.php	test	24 days ago
edit-medicine.php	test	24 days ago
forgot-password.php	test	24 days ago
gulpfile.js	test	24 days ago
index.php	test	24 days ago
invoice-search.php	test	24 days ago
invoice.php	test	24 days ago
logout.php	test	24 days ago
manage-medicine.php	test	24 days ago
pharmacist-profile.php	test	24 days ago
pharmacist-report-ds.php	test	24 days ago
pharmacist-reports-details.php	test	24 days ago
reset-password.php	test	24 days ago
sales-reports-detailed.php	test	24 days ago
search.php	test	24 days ago

No description, website, or topics provided.

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

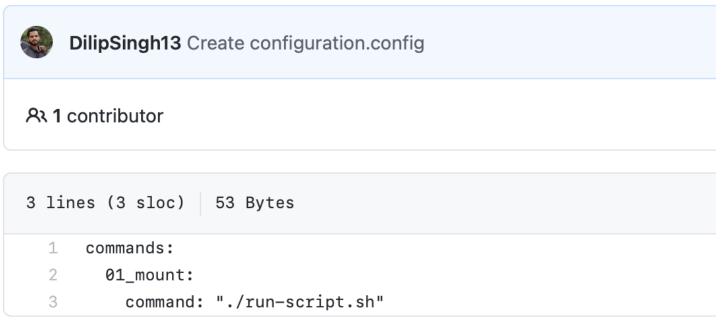
● CSS 55.9% ● SCSS 18.9%
● PHP 12.4% ● JavaScript 12.3%
● Other 0.5%

Figure 5: GitHub sample project structure

[1] `https://docs.python.org/3/library/concurrent.futures.html`

- Installing python 3.8, Boto3 and Git Clone the custom python code which will enable thread-level parallelism.
- We need to create a script(.sh) file at /root directory and make execute the custom code python file.
- As shown in figure 5, we need to create a repository called .extension in our application code and place a .config file. Config file will trigger the script file inside the EC2 instance to further run the python file.

As shown in figure 6, the configuration file which will trigger the python script for application deployment during deploy phase. The file has .config extension and contains command require to run a script(.sh) file which was penetrated inside Elastic BeanStalk instance.


DilipSingh13 Create configuration.config

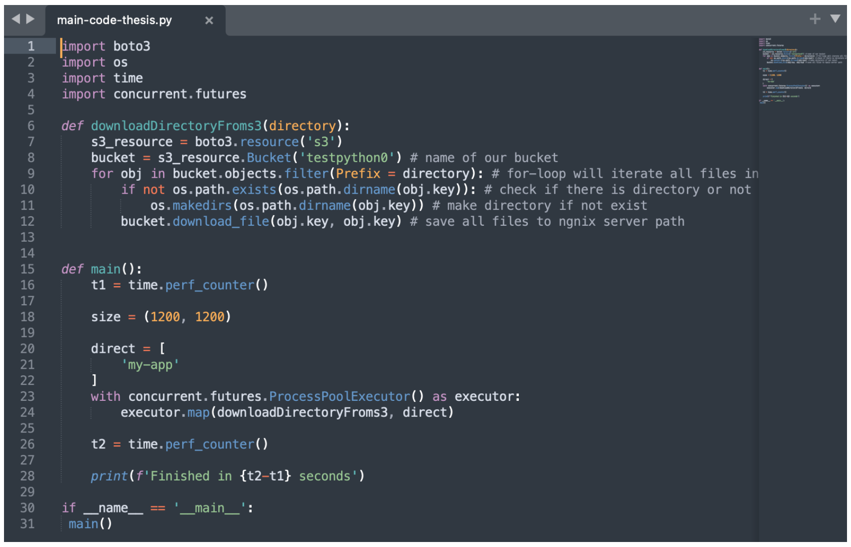
1 contributor

3 lines (3 sloc) | 53 Bytes

```
commands:
  01_mount:
    command: "./run-script.sh"
```

Figure 6: Configuration file to trigger python script

The figure 7 shows the custom code script written in python 3.8 which utilizes various libraries such as boto3, concurrent.features and os. The concurrent.features library enables thread-level parallelism with the help of ThreadPoolExecutor and further executes threads using the map() method.


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()
```

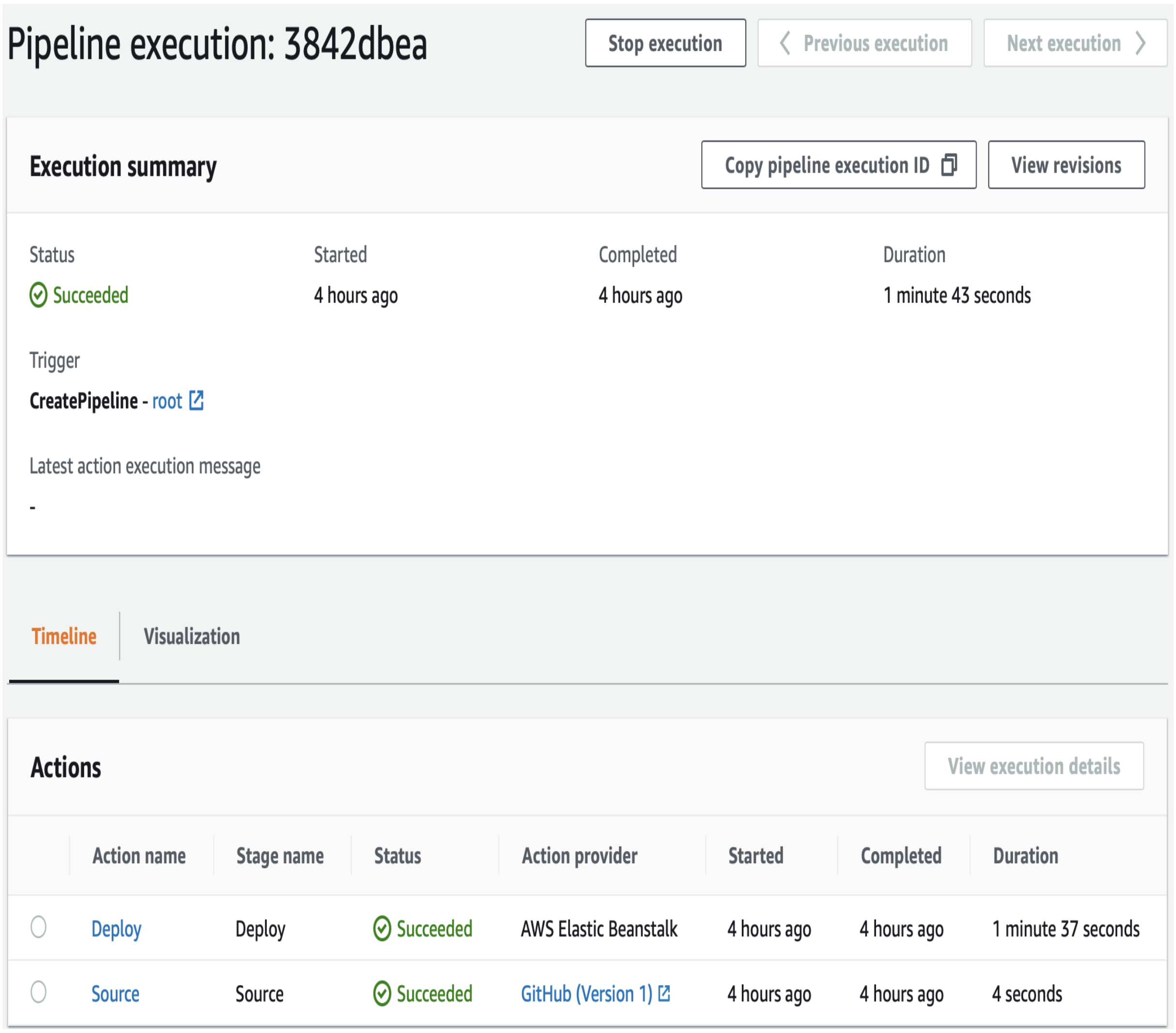
Figure 7: Code snippet of custom code written in python 3.8

6 Evaluation

The section compares four case studies for application deployment with custom code and without custom code. The evaluation was conducted on PHP applications of size 1MB, 2MB and 5MB respectively. The execution time graph is used to represent the difference between application deployment with thread-level parallelism and without thread-level parallelism.

6.1 Case Study 1 for 1MB file without custom code

As shown in figure 8, the deployment of a 1MB PHP application using AWS Elastic BeanStalk is 1 minute and 43 seconds. The sample application consists of four PHP files named index.php, connect.php, home.php and issuebooks.php. During deployment of application AWS Elastic BeanStalk copies application files one by one from AWS S3 to BeanStalk EC2 instance. which is time-consuming and also incurred additional cost to the user.



Pipeline execution: 3842dbea

Execution summary

Status	Started	Completed	Duration
Succeeded	4 hours ago	4 hours ago	1 minute 43 seconds

Trigger

CreatePipeline - root

Latest action execution message

-

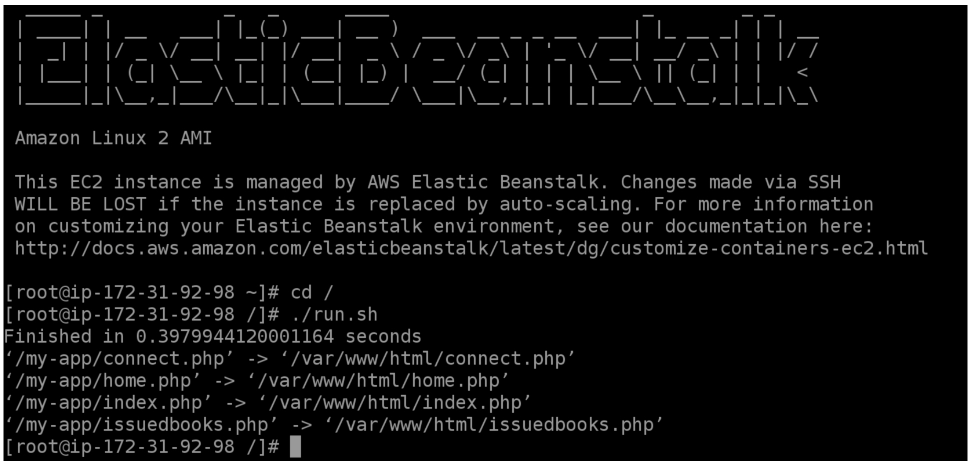
Actions

Action name	Stage name	Status	Action provider	Started	Completed	Duration
Deploy	Deploy	Succeeded	AWS Elastic Beanstalk	4 hours ago	4 hours ago	1 minute 37 seconds
Source	Source	Succeeded	GitHub (Version 1)	4 hours ago	4 hours ago	4 seconds

Figure 8: 1MB PHP application deployment without custom code

6.2 Case Study 2 for 1MB file with custom code

The figure 9 represents the application deployment using custom code which enables thread level parallelism. It can be seen that time required to deploy PHP application of 1MB was 0.397 seconds.

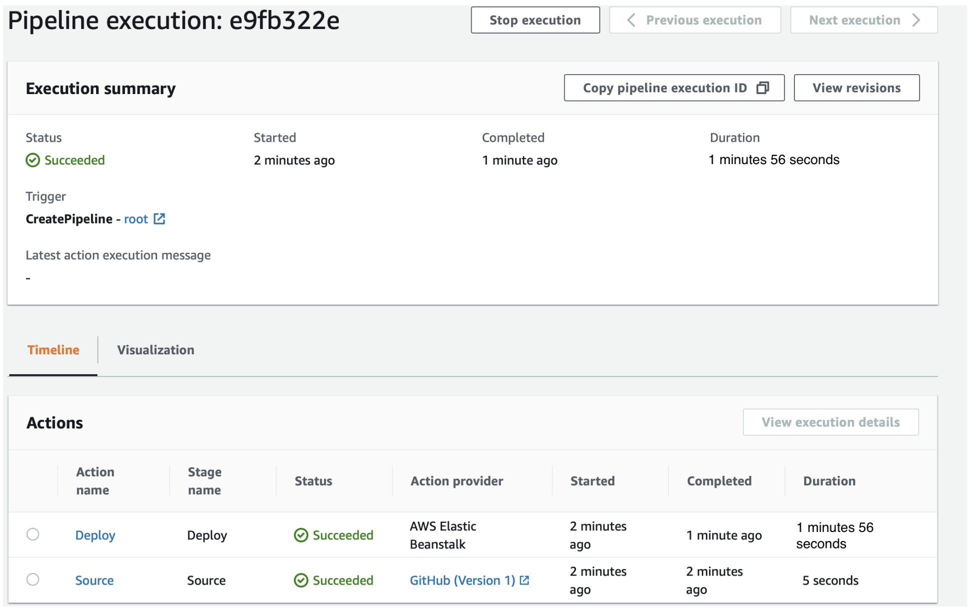


```
 _____ _           _   _      ____                       _        _ _
| ____| | __ _ ___| |_(_) ___| __ )  ___  __ _ _ __  ___| |_ __ _| | | __
|  _| | |/ _` / __| __| |/ __|  _ \ / _ \/ _` | '_ \/ __| __/ _` | | |/ /
| |___| | (_| \__ \ |_| | (__| |_) |  __/ (_| | | | \__ \ || (_| | |   <
|_____|_|\__,_|___/\__|_|\___|____/ \___|\__,_|_| |_|___/\__\__,_|_|_|\_\

 Amazon Linux 2 AMI

 This EC2 instance is managed by AWS Elastic Beanstalk. Changes made via SSH
 WILL BE LOST if the instance is replaced by auto-scaling. For more information
 on customizing your Elastic Beanstalk environment, see our documentation here:
 http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customize-containers-ec2.html

[root@ip-172-31-92-98 ~]# cd /
[root@ip-172-31-92-98 /]# ./run.sh
Finished in 0.3979944120001164 seconds
'/my-app/connect.php' -> '/var/www/html/connect.php'
'/my-app/home.php' -> '/var/www/html/home.php'
'/my-app/index.php' -> '/var/www/html/index.php'
'/my-app/issuedbooks.php' -> '/var/www/html/issuedbooks.php'
[root@ip-172-31-92-98 /]#
```

Figure 9: 1MB PHP application deployment with custom code

6.3 Case Study 3 for 5MB file without custom code

The figure 10 represents the application deployment without usage of custom code. It can be seen that time required to deploy PHP application of 5MB was 1 minute and 56 seconds.



Pipeline execution: e9fb322e

Execution summary

Status	Started	Completed	Duration
✓ Succeeded	2 minutes ago	1 minute ago	1 minutes 56 seconds

Trigger
CreatePipeline - root

Latest action execution message
-

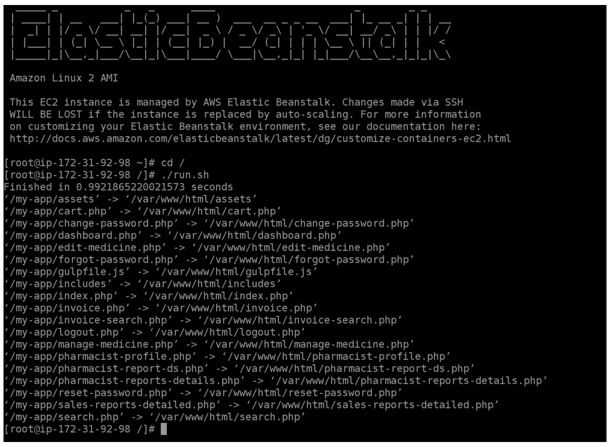
Actions

	Action name	Stage name	Status	Action provider	Started	Completed	Duration
○	Deploy	Deploy	✓ Succeeded	AWS Elastic Beanstalk	2 minutes ago	1 minute ago	1 minutes 56 seconds
○	Source	Source	✓ Succeeded	GitHub (Version 1)	2 minutes ago	2 minutes ago	5 seconds

Figure 10: 2MB PHP application deployment without custom code

6.4 Case Study 4 for 5MB file with custom code

The figure 11 represents the application deployment using custom code which enables thread-level parallelism. It can be seen that time required to deploy a PHP application of 5MB was 0.992 seconds.



```
 Amazon Linux 2 AMI

 This EC2 instance is managed by AWS Elastic Beanstalk. Changes made via SSH
 WILL BE LOST if the instance is replaced by auto-scaling. For more information
 on customizing your Elastic Beanstalk environment, see our documentation here:
 http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customize-containers-ec2.html

[root@ip-172-31-92-98 ~]# cd /
[root@ip-172-31-92-98 /]# ./run.sh
Finished in 0.9921865220021573 seconds
‘/my-app/assets’ -> ‘/var/www/html/assets’
‘/my-app/cart.php’ -> ‘/var/www/html/cart.php’
‘/my-app/change-password.php’ -> ‘/var/www/html/change-password.php’
‘/my-app/dashboard.php’ -> ‘/var/www/html/dashboard.php’
‘/my-app/edit-medicine.php’ -> ‘/var/www/html/edit-medicine.php’
‘/my-app/forgot-password.php’ -> ‘/var/www/html/forgot-password.php’
‘/my-app/gulpfile.js’ -> ‘/var/www/html/gulpfile.js’
‘/my-app/includes’ -> ‘/var/www/html/includes’
‘/my-app/index.php’ -> ‘/var/www/html/index.php’
‘/my-app/invoice.php’ -> ‘/var/www/html/invoice.php’
‘/my-app/invoice-search.php’ -> ‘/var/www/html/invoice-search.php’
‘/my-app/logout.php’ -> ‘/var/www/html/logout.php’
‘/my-app/manage-medicine.php’ -> ‘/var/www/html/manage-medicine.php’
‘/my-app/pharmacist-profile.php’ -> ‘/var/www/html/pharmacist-profile.php’
‘/my-app/pharmacist-report-ds.php’ -> ‘/var/www/html/pharmacist-report-ds.php’
‘/my-app/pharmacist-reports-details.php’ -> ‘/var/www/html/pharmacist-reports-details.php’
‘/my-app/reset-password.php’ -> ‘/var/www/html/reset-password.php’
‘/my-app/sales-reports-detailed.php’ -> ‘/var/www/html/sales-reports-detailed.php’
‘/my-app/search.php’ -> ‘/var/www/html/search.php’
[root@ip-172-31-92-98 /]# 
```

Figure 11: 2MB PHP application deployment with custom code

6.5 Discussion

The graph 12 depicts the difference between deployment time for a PHP application of size 1MB, 2MB and 5MB. The time required for deploying an application without custom code was 1 minute and 43 seconds whereas with thread-level parallelism the time was 0.397 seconds. The execution time for deploying 5MB applications without custom code was 1 minute and 56 seconds and in 0.992 seconds with custom code. It can be observed that for small application size there was drastic change whereas when we keep increasing application size the time difference increases gradually.

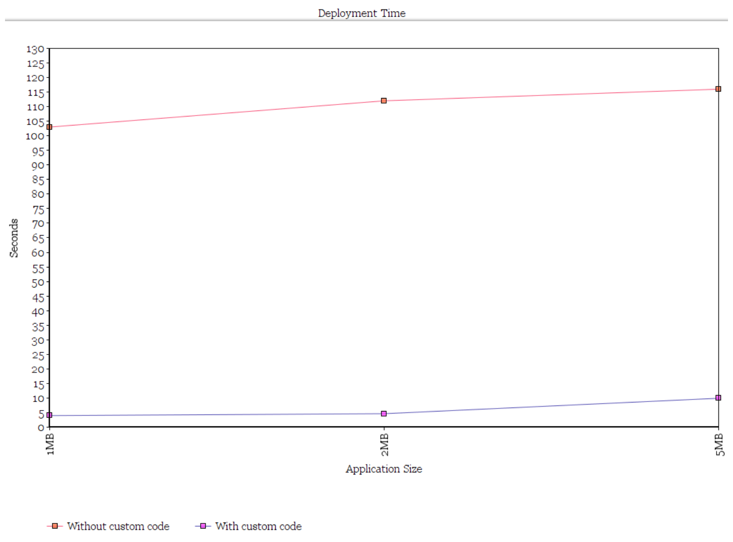


Figure 12: Application deployment time for various size of application

7 Conclusion and Future Work

This paper introduces task-level parallelism to decrease the time required during application deployment. This paper believes that the existing performance gap in application deployment in DevOps has gained a significant performance boost with custom code. The paper efficiently uses AWS S3 and AWS EC2 instances to improve the AWS Elastic BeanStalk performance. Comparing the result by the implementation of custom code and without code on various application sizes (1MB, 5MB, 10MB, 15MB) shows a drastic change in time duration. In research, it was found that due to thread-level parallelism the application files deployed parallelly from AWS S3 to Elastic BeanStalk instance which had resulted in saving time for the development team. To this aim, the research focuses on:

- Introduce platform-independent thread-level parallelism code.
- The usage of AWS CloudWatch for continuous monitoring of resource performance and tracking the time for application deployment.

Due to resource modification restrictions by AWS, this proposed researchable to evaluate the small analytic performance. Future work comprises reducing the time more efficiently for a large application. Also, various kinds of cloud storage can be used to optimize the performance and find the more suitable model for decreasing deployment time.

References

Amazon (n.d.). Extending Elastic Beanstalk Linux platforms. (accessed: 2021, April 2).
URL: *https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/platforms-linux-extend.html*

Arachchi, S. A. I. B. S. and Perera, I. (2018). Continuous integration and continuous delivery pipeline automation for agile software project management, *2018 Moratuwa Engineering Research Conference (MERCon)*, Moratuwa, Sri Lanka, pp. 156–161.

Artač, M., Borovšak, T., Di Nitto, E., Guerriero, M. and Tamburri, D. A. (2016). Model-driven continuous deployment for quality devops, *Proceedings of the 2nd International Workshop on Quality-Aware DevOps*, QUDOS 2016, Association for Computing Machinery, New York, United States, p. 40–41.
URL: *https://doi.org/10.1145/2945408.2945417*

Brunnert, A., van Hoorn, A., Willnecker, F., Danciu, A., Hasselbring, W., Heger, C., Herbst, N., Jamshidi, P., Jung, R., von Kistowski, J., Koziolek, A., Kroß, J., Spinner, S., Vögele, C., Walter, J. and Wert, A. (2015). Performance-oriented devops: A research agenda.

Danglot, B., Vera-Pérez, O. L., Baudry, B. and Monperrus, M. (2019). Automatic test improvement with dspot: a study with ten mature open-source projects, *Empirical Software Engineering* **24**(4): 2603–2635. CoreRank: A, JCR Impact Factor: 3.156.
URL: *https://doi.org/10.1007/s10664-019-09692-y*

Dlugi, M., Brunnert, A. and Krcmar, H. (2015). Model-based performance evaluations in continuous delivery pipelines, *Proceedings of the 1st International Workshop on Quality-Aware DevOps*, QUDOS 2015, Association for Computing Machinery, New York, Unites States, p. 25–26.
URL: *https://doi.org/10.1145/2804371.2804376*

Elbaum, S., Rothermel, G. and Penix, J. (2014). Techniques for improving regression testing in continuous integration development environments, *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, Association for Computing Machinery, New York, NY, USA, p. 235–245. CoreRank: A*.
URL: *https://doi.org/10.1145/2635868.2635910*

Flemström, D., Potena, P., Sundmark, D., Afzal, W. and Bohlin, M. (2018). Similarity-based prioritization of test case automation, *Software Quality Journal* **26**(4): 1421–1449. CoreRank: C, JCR Impact factor: 1.460.
URL: *https://doi.org/10.1007/s11219-017-9401-7*

Guerriero, M., Ciavotta, M., Gibilisco, G. P. and Ardagna, D. (2015). Space4cloud: A devops environment for multi-cloud applications, *Proceedings of the 1st International Workshop on Quality-Aware DevOps*, QUDOS 2015, Association for Computing Machinery, New York, Unites States, p. 29–30.
URL: *https://doi.org/10.1145/2804371.2804378*

Larrucea, X., Berreteaga, A., Santamaria, Izaskun", e. A., O'Connor, R. V. and Messnarz, R. (2019). Dealing with security in a real devops environment, *Systems, Software and Services Process Improvement*, Springer International Publishing, Edinburgh, United Kingdom, pp. 453–464.

Laukkarinen, T., Kuusinen, K. and Mikkonen, T. (2017). Devops in regulated software development: Case medical devices, *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*, Buenos Aires, Argentina, pp. 15–18.

Lehtonen, T., Suonsyrjä, S., Kilamo, T. and Mikkonen, T. (2015). Defining metrics for continuous delivery and deployment pipeline, *Proceedings of the 14th Symposium on Programming Languages and Software Tools*, Finland, pp. 16–30.

Leite, L., Rocha, C., Kon, F., Milojicic, D. and Meirelles, P. (2019). A survey of devops concepts and challenges, *ACM Comput. Surv.* **52**(6). CoreRank: A*, JCR Impact Factor: 7.990.
URL: *https://doi.org/10.1145/3359981*

Marijan, D., Liaaen, M. and Sen, S. (2018). Devops improvements for reduced cycle times with integrated test optimizations for continuous integration, *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Tokyo, Japan, pp. 22–27.

Perez, J. F., Wang, W. and Casale, G. (2015). Towards a devops approach for software quality engineering, *Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development*, WOSP '15, Association for Computing Machinery,

New York, NY, USA, p. 5–10.
URL: *https://doi.org/10.1145/2693561.2693564*

Rafi, S., Yu, W., Akbar, M. A., Alsanad, A. and Gumaei, A. (2020). Prioritization based taxonomy of devops security challenges using promethee, *IEEE Access* **8**: 105426–105446.

Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J. and Männistö, T. (2016). Devops adoption benefits and challenges in practice: A case study, *in* P. Abrahamsson, A. Jedlitschka, A. Nguyen Duc, M. Felderer, S. Amasaki and T. Mikkonen (eds), *Product-Focused Software Process Improvement*, Springer International Publishing, Trondheim, Norway, pp. 590–597.

Sampedro, Z., Holt, A. and Hauser, T. (2018). Continuous integration and delivery for hpc: Using singularity and jenkins, *Proceedings of the Practice and Experience on Advanced Research Computing*, PEARC '18, Association for Computing Machinery, New York, NY, USA.
URL: *https://doi.org/10.1145/3219104.3219147*

Shahin, M., Ali Babar, M. and Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices, *IEEE Access* **5**: 3909–3943.

Zampetti, F., Vassallo, C., Panichella, S., Canfora, G., Gall, H. and Di Penta, M. (2020). An empirical characterization of bad practices in continuous integration, *Empirical Software Engineering* **25**(2): 1095–1135. CoreRank: A, JCR Impact Factor: 3.156.
URL: *https://doi.org/10.1007/s10664-019-09785-8*