Long Vu Hai

DEPLOYING WEB APP USING JENKINS AND AMAZON WEB SERVICES

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Information Technology
May 2020

Centria
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Centria University of Applied Sciences	**Date** May 2020	**Author** Vu Hai Long

Degree programme Information Technology	
Name of thesis DEPLOYING WEB APP USING JENKINS AND AMAZON WEB SERVICES	
Instructor Jari Isohanni	**Pages** 35
Supervisor Jari Isohanni	

This thesis is an object to provide a guidebook for developers who were interested in the combinations of software development and technology operation concepts to constructing a continuous integration process step by step. Indeed, the author endeavored to keep up with the latest technical things and simultaneously analyzed the features and relative strengths and weaknesses of the study case's technologies compared to its competitor. All of the services used such as CloudFront, Load Balancer, Route53, Certificate Manager and Jenkins are required to work around cloud storage and virtual machine.

Furthermore, the reason behind the author's choice of Jenkins and many Amazon Web Services are shown in the thesis. The development process for an automated application deployment system is an indication of the author's internship at Speys Oy. Certainly, the study case has been applied in practice at the company and has proven effective in shortening deployment time and avoiding crucial problems that may occur during the course. The essential integration pipeline is recognized to be able to meet the requirements of development and enhance work efficiency.

The result of this thesis is an entirely functional operation that has a significant position in the advancement project case and at the same time takes advantage of the combination of the two technologies to be able to handle numerous workloads every day.

Key words
Cloud solutions, continuous integration, Amazon Web Services, Jenkins

CONCEPT DEFINITIONS

AWS	Amazon Web Service
RDS	Relational Database Service
API	Application programming interface
UI	User interface
TCP/IP	Transmission Control Protocol / Internet Protocol
SSH	Secure shell
CI/CD	Continuous Integration / Continuous Delivery
SCM	Supply chain management
DSL	Digital subscriber line
SSL/TLS	Secure socket layer / Transport layer security
JSON	Javascript Object Notation
DDoS	Distributed denial-of-service
ECU	Engine control unit
EBS	Amazon Elastic Block Store

ABSTRACT
CONCEPT DEFINITIONS
CONTENTS

FIGURES

TABLES

1 INTRODUCTION

Over the past few years, software operation development has been an arguably ever-changing field and the building deploying methodologies have become completely different. Nonetheless, understanding what is best suited for application architecture is the foremost condition to develop any services – determining between using cloud storage or not and the further determination of continuous integration technology and service. With the plethora of tools available and the new ones rapidly popping up every day, most of the time developers find themselves confused about which way to go. Sometimes developers can find a few services that are suitable for their application at a time, but during long-term use and development, these tools gradually reveal some weaknesses and disadvantages that can not adapt to the application and may make the project progressively impossible. Therefore, it shows the importance of careful consideration appreciation of the feasibility and usefulness of any third-party services.

The main objective of this thesis is to be express an overview about designing a complete process with step by step instructions for comprehensively installation and arrangement of deploying progress on Amazon Web Services platform and Jenkins. At the time of completion, users can trigger an automated integration process that is preinstalled by Jenkins in a smooth way and deliver the analyzed action results to minimize time and possible risks. Manually deploying applications at AWS would be redundant to developers when the utility of the system brings them to eliminate lengthy deployment process and require unnecessary meticulousness. At this point, as well as the ultimate goal of the distribution system, only one click is all they need to complete the process.

Indeed, after finishing the creation for the target system, the author realized that this service chain still has aspects that need further research and development in the future even when it has met certain requirements. Although perfection is an impossible task in programming as the development and change in technologies take place day by day, achieving the highest optimization in the deployment structure is what this thesis is all about.

2 TECHNOLOGIES

This chapter is an overview of different software technologies that are used for the author's case study project and its influence on the development process. The author will provide a framework of several services from Amazon Web Services as well as Jenkins to make readers get an ideal jumping-off point of this thesis's topic.

2.1 Cloud Services

Cloud services are leveling the playing field in technology in astonishing ways and people are at the cusp of seeing those benefits accrue to organizations. While there is some disagreement on the exact nature of the "cloud", the fact is that it is difficult for any person to go anywhere without seeing advertisements for cloud services and cloud-based software offerings at all around. Therefore, ignoring the "cloud" is not really possible any longer. Since cloud services are so new, the outlook for cloud-based computing is rosy. (Hastings 2014.)

For a long time, in network diagrams, the complexity of the Internet was reduced down to a simple cloud icon. That is because the focus was on the servers, devices and switches IT professionals were responsible for and the Internet was beyond the horizon of what they needed to worry about from an architecture standpoint. Since software delivered over the Internet similarly does not require the customer to worry about how it is architected, people began to refer to this software as being "in the cloud". Today, the cloud comes in three flavors which forms the cloud stack: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). SaaS is what most people think of when they hear the name of a familiar cloud service that delivers software and applications via the cloud, which differs from other software in that it is far more scalable. PaaS is a cloud environment or platform for developers to build cloud applications and deliver a database, an operating system, programming language execution capabilities. Otherwise, IaaS is a platform offering raw computing power for service providers that includes the servers, file or object storage, load balancers, network firewall, and CDNs. (Kohgadai 2020.)

2.1.1 Overview

Cloud computing is the on-demand delivery of IT resources over the Internet with pay-as-you-go pricing. Instead of buying, owning, and maintaining physical data centers and servers, customers can access technology services, such as computing power, storage, and database, on an as-needed basis from a cloud provider like Amazon Web Services. (What is cloud computing 2020.) AWS was officially revealed to the world on March 13, 2006. On that day, AWS offered the Simple Storage Service, its first service soon shortened to S3. Later in 2006 came Elastic Compute Cloud (known affectionately as EC2). The overall pattern of AWS has been to add additional services steadily, and then quickly improve each service over time. AWS is now composed of more than 25 different services, many offered with different capabilities via different configurations or formats. This rich set of services can be mixed and matched to create an interesting and unique application, limited only by imagination or needs. (Golden 2013, 10-11.)

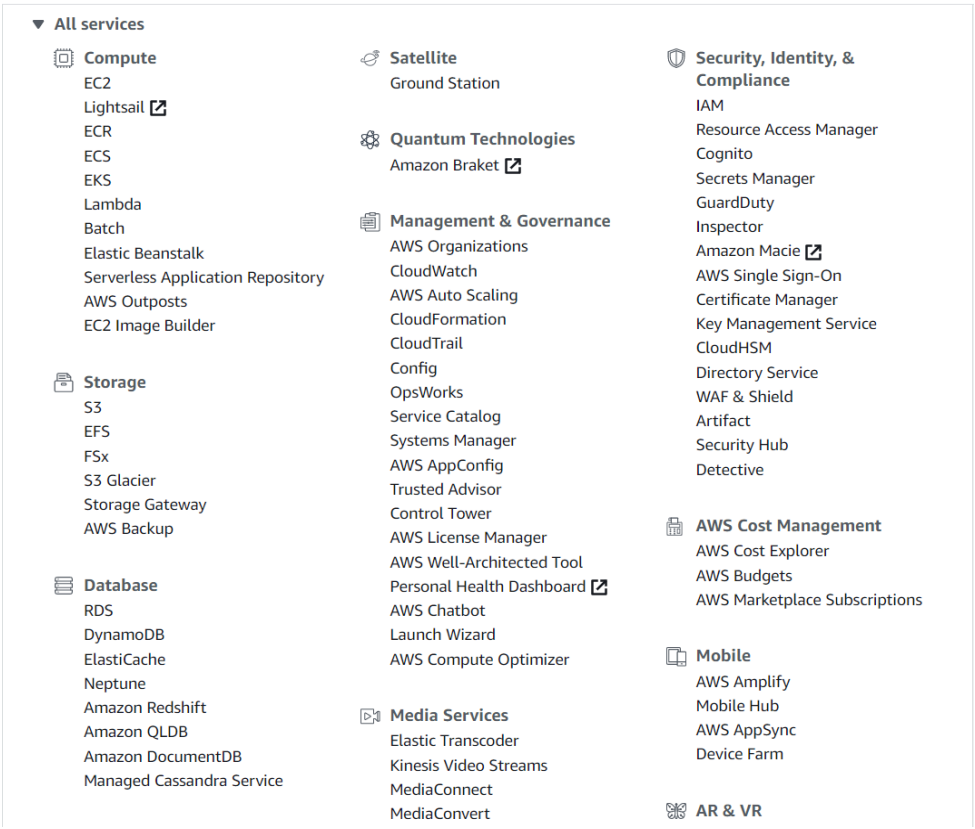

Figure 1. Services on AWS Management Console (adapted from AWS Services 2020)

Amazon Web Services, which consists of 175 products and services, is a leader in that cloud market-place. Now it can provide cloud storage, compute power, app deployment, user account management,

data warehousing, tools for managing and controlling the Internet of Things devices, and just about anything people can think of that a business needs. AWS really grew in popularity and capability over the last decade. One reason is that AWS is so reliable and secure. It is a gold standard and used by some of the most well-known brands in existence, such as Netflix, Uber, and Airbnb. (Brandon 2020.)

2.1.2 Setting up Cloud Services

The very first thing from establishing AWS server procedure is signing up and it is mandatory to able to use the services. Registering an account will require some usual data from customers like email, account name, company name (professional account type), phone number (personal account type), country, address, city and postal code for the first step. At this step, it is also requested to users read and agree to the terms of AWS Customer Agreement. The next step is filling payment information, name, billing address, VAT registration number so it is not only used for paying service charges but also to verify customer identity and for usage below AWS Free Tier Limits. There are three different available types of the free offer: Always free, 12 months free and Trials. Although depending on the product used, the customer is able to have more than 60 products to start building their cloud environment on the free tier. Once the second step completed, there will be a 1-dollar charge to credit card as a verification charge to make sure that the card is valid. Additionally, another verification that is coming up as a part of the sign-up process that involves receiving a phone call and entering a verification code on the phone keypad. Since the PIN code is authenticated, it will take two or three minutes for account set up by AWS and then there is a confirming email that is needed to click on a link in that e-mail to complete the registration process.

Figure 2. AWS Global Infrastructure Map (adapted from Global Infrastructure 2020)

AWS has a global infrastructure that will provide a more secure cloud environment for every company and business globally. Clearly observed, their AWS account is in the scope of the entire system, but their data can be located at a specific zone. Businesses can take advantage by storing data on limitlessness scalability infrastructures such as Regions, Availability Zones and Edge Locations according to different geographical areas. There are some of the most popular application names on the Internet like Instagram, Github, Pinterest, and Netflix as well as government activities like NASA and CIA. (Yvanovich 2016.)

2.1.3 Store Data in Cloud Services

In recent years, Amazon Web Services offers some distinct storage services: Amazon DynamoDB, Amazon RDS, Amazon SimpleDB or Amazon S3. (Amazon Web Services 2020.) In this thesis, the author has chosen Simple Storage Service (S3) bucket to be the data cloud storage as a result of its flexibility to become one of the most widely used services trusted by individuals and businesses.

Amazon Simple Storage Service is an object storage service that offers industry-leading scalability, data availability, security and performance. This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases such as websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides easy-to-use management features so people can organize their data and configure finely-tuned

access controls to meet their specific business, organizational, and compliance requirements. Amazon S3 is designed for 99.999999999% of durability and stores data for millions of applications for companies all around the world. (Amazon Web Services 2020.)

According to AWS documentation, there are several overwhelming advantages of S3 that the author can clearly realize when compared to its competitor: safe, secure and highly scalable features. It is not only very simple to use via the web service interface but also allowing users to storage and retrieval data. AWS S3 has no concept of a folder, therefore, to deal with a number of level storage objects, it offers many storage classes including Amazon S3 Standard for storage data purpose and suitable to access frequently, Amazon S3 Standard – Infrequent Access (Standard – IA) for long-term data storages but infrequent use, Amazon Glacier is low-cost storage for data archiving. (Amazon Web Services 2020.)

In terms of implementation, buckets and objects are resources, and Amazon S3 provides APIs to manage them. For example, a customer can create a bucket and upload objects using the Amazon S3 API. They can also use the Amazon S3 console to perform these operations and the console uses the Amazon S3 APIs to send a request to Amazon S3. An Amazon S3 bucket name is globally unique, and the namespace is shared by all AWS accounts. This means that after a bucket is created, the name of that bucket cannot be used by another AWS account in any AWS Region until the bucket is deleted. Customers should depend on specific bucket naming conventions for availability or security verification purposes. To optimize latency, minimize costs or address regulatory requirements, choose any AWS Region that is geographically close to the customer. (Amazon Web Services 2020.)

2.1.4 Security in Cloud Services

There is a conspicuous question that can be an obstacle on the way when people are using the third-party proxies: “How can cloud provider fully protect the data in services that customer is using?”. Using Amazon Web Services, customer gains the control and confidence needed to securely run business with the most flexible and secure cloud computing environment available today. As an AWS customer, they will benefit from AWS data centers and a network architected to protect their information, identities, applications, and devices. With AWS, they can improve their ability to meet core security and compliance requirements, such as data locality, protection, and confidentiality with AWS comprehensive services and features. AWS allows people to automate manual security tasks so they can shift focus to scaling and innovating their business. All customers benefit from AWS being the only commercial cloud that

has had its service offerings and associated supply chain vetted and accepted as secure enough for top-secret workloads. (Amazon Web Services 2020.)

With AWS, automating security tasks enable customers to be more secure by reducing human configuration errors and give technical team to focus on other work critical to businesses. Select from a wide variety of deeply integrated solutions that can be combined to automate tasks in novel ways, making it easier for the security team to work closely with developer and operations teams to create and deploy code faster and more securely. Customers can build on the most secure global infrastructure, knowing they always own their data, including the ability to encrypt it, move it and manage retention. All data flowing across the AWS global network that interconnects data centers and regions is automatically encrypted at the physical layer before it leaves secured facilities. An additional encryption layer exists as well, for example, all Virtual Private Cloud cross-region peering traffic, and customer or service-to-service TLS connections. (Amazon Web Services 2020.)

Security of EC2 is called Security Group that acts as a firewall to take care of traffic to and from the instance and already installed on every instance by Amazon. A security group will have these following elements of network traffic access: traffic protocol, traffic source and traffic port to be able to allow traffic into the instance. At the starting point, there is a default Security Group that allows for all traffic at port 80 (HTTP) but it is not recommended to use it. Normally, when people create an EC2 instance, because there is an initial set of Security Group that no traffic is allowed to connect to the instance, they usually have one specific group ready for use. Hence, this is to benefit from the ease of control, maintenance and the ability to avoid confusion over long-term use. Security rules in Security Group will define what network traffic is allowed to connect with the instance by associate with it. It is critical to understand the importance of Security Group when designing an application to make it more secure. (Amazon Web Services 2020.)

As useful as EC2 undoubtedly is, many customers prefer a more secure offering. But with the best security practices regarding security groups, a potential vulnerability in applications is present when each EC2 instance has a public IP address. Fortunately, AWS addresses this problem with its Virtual Private Cloud (VPC) offering. In broad terms, VPC lets users segregate their instances and shield them from direct Internet access and operates by providing customers with a virtual network topology that is separate from the general AWS environment. Another way to say is that via the use of clever software, AWS provides a segregated computing environment. Instances are located within their own, private VPC, with no access to them other than via the VPC environment. Using VPC, people can create a separate set of

resources that carry private IP addresses within a range they select. People set rules for how traffic enters and leaves instances within the VPC. People can choose to make instances accessible to the public Internet via Elastic IP addresses. Moreover, people can create subnets (in effect, subdivisions of the overall VPC) and control access to and from the subnets and between subnets. (Golden 2013, 170-171.)

2.2 Continuous Integration

As a matter of fact, although not all projects can achieve great success, there are still methodologies and habits that can significantly increase the chances of a project's success and make the development process become a more comfortable experience. One of those mechanisms is to use Continuous Integration (CI). It was originally used as an extreme programming job, and its main purpose is to prevent integration issues and also to avoid Integration Hell occurs when a member on a technical team integrates their individual code.

Continuous Integration is a practice of constantly integrating the development made to the project and retesting it daily or more often. It requires the developer to integrate source code on the local machine with the server containing the source code that is shared with other developers. In agreement with FIGURE 3, this mechanism needs to happen as often and as soon as possible. This helps limit the differences in source code on the local machine of different members involved in the project. Additionally, if a project uses automated testing tools, this will make the source code of the programmer more reliable. Clearly observed, the technical team can all agree that having a problem right in the implementation day of the demo version is an unpleasant experience and CI can help them to minimize it. Automated build, test, and deployment can alleviate many of the common headache problems in projects. One of the CI statements is "Build Software at Every Change" and the goal of it is to avoid consequence like, "Oh, this part works normally on my computer".

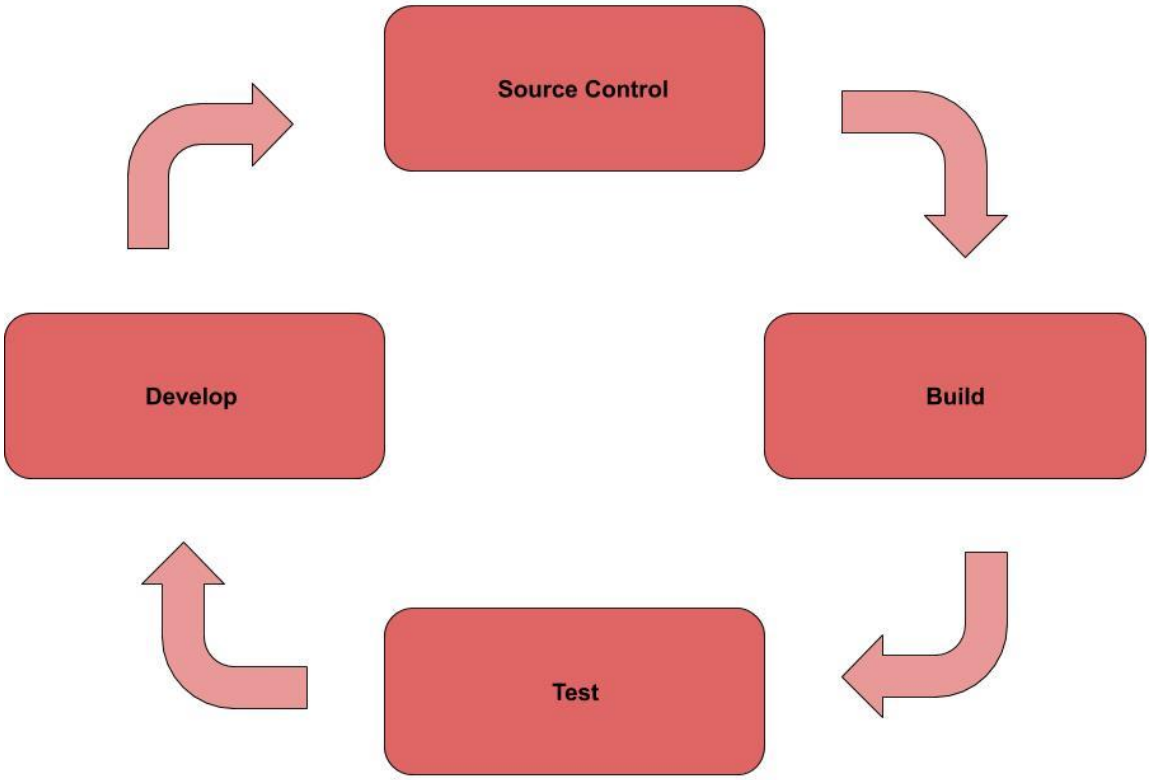

Figure 3. Continuous Integration steps

2.2.1 History

Because of conformance with the thesis subject's requirement, as a result, Jenkins has been selected as a CI tool to assist the author in completing the project. Hence, to help readers have more detail about this open-source engine, providing them with insight from history is a necessity.

Jenkins was originally developed as the Hudson project. Hudson's creation started in summer of 2004 at Sun Microsystems. It was first released in java.net in February 2005. Around 2007 Hudson became known as a better alternative to Cruise Control and other open-source build-servers. At the JavaOne conference in May 2008, the software won the Duke's Choice Award in the Developer Solutions category. During November 2010, an issue arose in the Hudson community with respect to the infrastructure used, which grew to encompass questions over the stewardship and control by Oracle. Negotiations between the principle project contributors and Oracle took place, and although there ware many areas of agreement a key sticking point was the trademarked name "Hudson", after Oracle claimed the right to the name and applied for a trademark in December 2010. As a result, on January 11, 2011, a call for votes was made to change the project name from "Hudson" to "Jenkins". The proposal was overwhelmingly approved by community vote on January 29, 2011, creating the Jenkins project. (Jenkins Wikipedia 2020.)

On February 1, 2011, Oracle said that they intended to continue development of Hudson and considered Jenkins a fork rather than a rename. Jenkins and Hudson therefore are continued as two independent projects, each claiming the other is the fork. As of June 2019, the Jenkins organization on GitHub had 667 project members and around 2,200 public repositories, compared to Hudson's 28 project members and 20 repositories with the last update in 2016. In 2011, creator Kohsuke Kawaguchi received an O'Reilly Open Source Award for his work on the Hudson/Jenkins project. On April 20, 2016 version 2 was released with the Pipeline plugin enabled by default. The plugin allows for writing build instructions using a domain specific language based on Apache Groovy. Jenkins replaced Hudson since February 8, 2017 in Eclipse. (Jenkins Wikipedia 2020.)

2.2.2 Automation concept

Jenkins, originally called Hudson, is an open-source Continuous Integration tool written in Java. Boasted a dominant market share, Jenkins is used by teams of all sizes, for projects in a wide variety of languages and technologies, including .NET, Ruby, Groovy, Grails, PHP and more, as well as Java. In essence, the questions what has made Jenkins such a success and why use Jenkins for the CI infrastructure are at the forefront. Firstly, Jenkins is easy to use. The user interface is simple, intuitive and visually appealing, and Jenkins as a while has a very low learning curve. However, Jenkins does not sacrifice power or extensibility, it is also extremely flexible and easy to adapt to self-purposes. Hundreds of open-source plugins are available, with more coming out every week. These plugins cover everything from version control systems, build tools, code quality metrics, build notifiers, integration with external systems, UI customization, games and much more. And installing them is quick and easy. (Smart 2011.)

Last, but certainly not least, much of Jenkins's popularity comes from the size and vibrancy of its community. The Jenkins community is a large, dynamic, reactive and welcoming bunch, with passionate champions, active mailing lists, IRC channels and a very vocal blog and Twitter account. The development pace is fast, with releases coming out weekly with the latest new features, bug fixes, and plugin updates. However, Jenkins also caters to users who are not comfortable with upgrading on a weekly basis. For those who prefer a less-hectic release pace, there is also long-term support (LTS) release line that lags behind the latest release in favor of more stability and a slower rate of change. New LTS releases come out every three months or so, with important bug fixes being backported. This concept is similar to the Ubuntu LTS releases. (Smart 2011.)

Jenkins is used to build and test software projects continuously making it easier for developers to integrate changes to the project and making it easier for users to obtain a fresh build. It also allows continuous delivery in software by integrating with a large number of testing and deployment technologies. With Jenkins, organizations can accelerate the software development process through automation. Jenkins integrates development life-cycle processes of all kinds, including build, document, test, package, stage, deploy, static, analysis and much more. (Saurabh 2019.) Jenkins offers a simple way to set up a continuous integration or continuous delivery environment for almost any combination of languages and sources code repositories using pipelines, as well as automating other routine development tasks. While Jenkins does not eliminate the need to create scripts for individual steps, it does give a faster and more robust way to integrate entire chain of build, test, and deployment tools than people can easily build themself. "Don't break the nightly build!" is a cardinal rule in software development shops that post a

freshly build daily product version every morning for testers. Before Jenkins, the best a developer could do to avoid breaking the nightly build was to build and test carefully and successfully on a local machine before committing the code. But that meant testing one's changes in isolation, without everyone else's daily commits. There was no firm guarantee that the nightly build would survive one's commit. (Heller 2017.)

2.2.3 Architecture of Continuous Integration

Jenkins uses a master/slave architecture to manage distributed builds. The main Jenkins server is the master. In a nutshell, the master's job is to handle scheduling build jobs, dispatching builds to the slaves for the actual execution, monitor the slaves (possibly taking them online and offline as required) and recording and presenting the build results. Even in a distributed architecture, a master instance of Jenkins can also execute build jobs directly. On the other hand, the job of the slaves is to do as they are told, which involves executing build jobs dispatched by the master. A project can be configured to always run on a particular slave machine, or a particular type of slave machine or simply let Jenkins pick the next available slave. A slave is a small Java executable that runs on a remote machine and listens for requests from the Jenkins master instance. Slaves can (and usually do) run on a variety of operating systems. The slave instance can be started in a number of different ways, depending on the operating system and network architecture. Once the slave instance is running, it communicates with the master instance over a TCP/IP connection (Smart 2011).

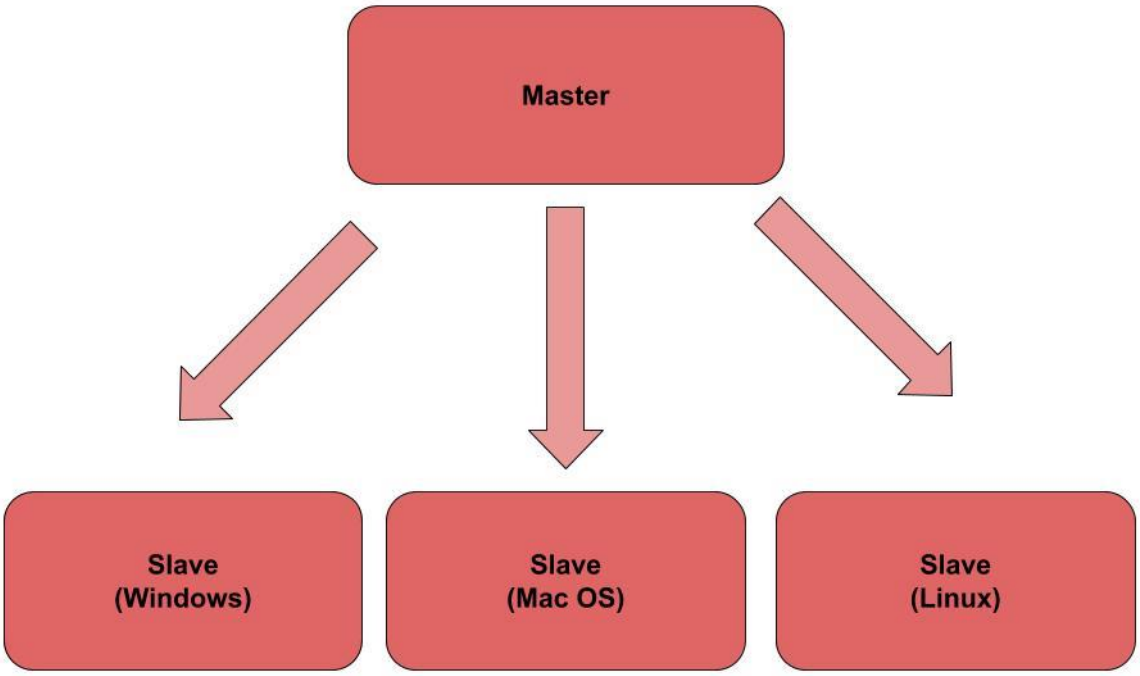

Figure 4. Master-Slave architecture (adapted from Long 2020)

All in all, to creating a slave node from Jenkins master machine, it is needed to provide for it a name and "Permanent Agent" type choice. Since Jenkins does not provide a higher level of integration with these agents, such as dynamic provisioning. This type can be selected if there is no other agent type applies such as adding a real computer or virtual machine from outside Jenkins. The name of the slave node should be unique identifies an agent within the Jenkins installation. This does not have to be the same as the agent hostname (where applicable) but it is often convenient to cause them the same. Spaces are accepted to apply to the name, but without spaces will make life easier to minimize issues in the future.

Description input is optional but the information on how many CPU cores, or how much RAM has installed, or physical locations are recommended to provide for the human-readable section. As can be seen, a number of executors bring a value of concurrent builds that can be performed on this node. Although a higher value can be configured and that would take a longer time to execute but could increment the overall throughput. The remote root directory is a spot where every Jenkins agent call its home. It is needed to provide a specific path that is able to use to build tasks and should not contain any critical file in this directory because when the build tasks are done everything will be delivered back to the master machine. Labels (or tags) are critical useful when grouping multiple agents into one logical group since the size of build architecture start growing up. Usage field accommodates two options: Use this

node as much as possible and Only build jobs with label expressions matching this node. The first option is default setting and whenever there is a build, Jenkins has full rights to use this agent that make the option to be the most frequently used to be served as the one people exactly want. Otherwise, the second choice formed the jobs only run whenever there is a label expression that matches the agent's name or labels. There is the case that customer wants to run on a specifically configured machine and does not want other jobs using it except the chosen one that contains matched label expression.

Jenkins provides three methodologies to launch slave machines: launch agent agents via SSH, launch agent by connecting it to the master and launch agent via execution of command on the master. Each option has its own way to connect slave machines and make the build tasks run. For example, launch agent agents via SSH will ask for the IP and credentials of the host machine in order to connect. There is also a button to extend the information field of the launch method that will require more specific details in case of missing required data. In addition, people can decide a running time to controls slave node when they launch and eliminate the agents by set options in Availability field: Keep this agent online as much as possible, bring this agent online according to a schedule ,bring this agent online when in demand and take offline when idle, that describes it all the way it works. People can also define more properties in the next section.

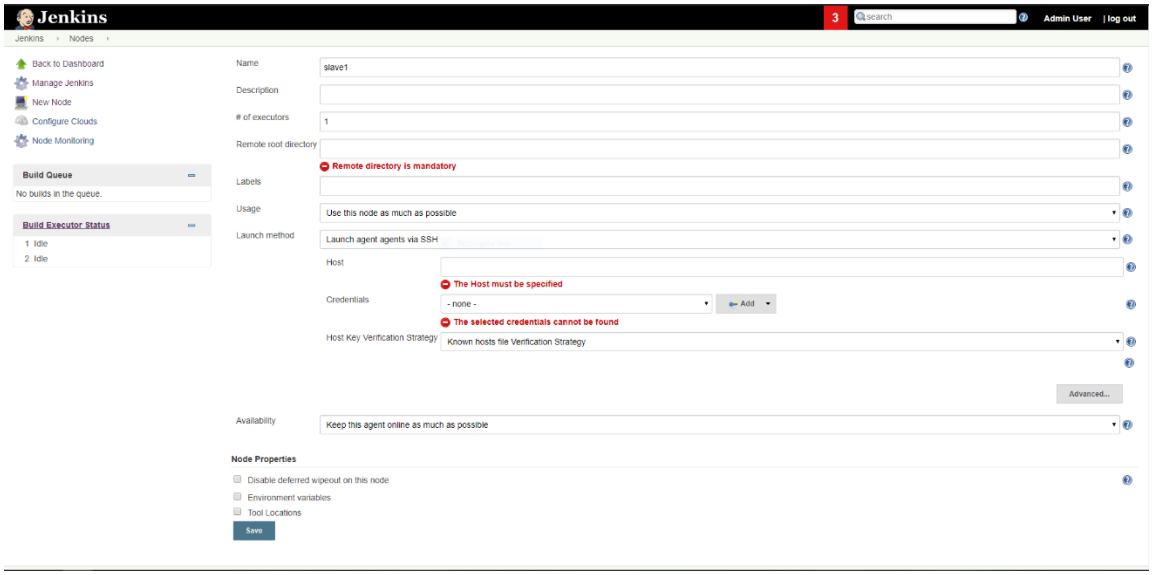

Figure 5. Jenkins slave node configuration

2.2.4 Continuous integration and continuous delivery pipeline

Continuous integration and continuous delivery pipeline (CI/CD) are one of the very important contents and by default, a DevOps must have a firm grasp and knowledge of it. In this section, the author will not delve into the CI/CD pipeline but will ascertain what the Jenkins tool supports and features to build a pipeline. It is the orderly set of tasks declared for product build, test and deploy. With CI/CD pipeline, whenever a change in software project source code is detected, the source code will be automatically built and tested by CI/CD servers like Jenkins. If it passes the quality control section and all tests passed, it will automatically be deployed.

A Jenkins Pipeline is a collection of jobs that brings the software from version control into the hands of the end-users by using automation tools. It is a feature used to incorporate continuous delivery in the software development workflow. Over the years, there have been multiple Jenkins pipeline releases including Jenkins Build flow, Jenkins Build Pipeline plugin and Jenkins Workflow. They represent multiple Jenkins jobs as one whole workflow in the form of a pipeline and these pipelines are a collection of Jenkins jobs which trigger each other in a specified sequence. The key feature of this pipeline is to define the entire deployment flow through code so all the standard jobs are manually written as one whole script and they can be stored in a version control system. It basically follows the ‘pipeline as code’ discipline. Instead of building several jobs for each phase, now people can code the entire workflow and put it in a Jenkins file. (Lateef 2019.)

A Jenkins file is a text file that stores the entire workflow as code, and it can be checked into an SCM on the local system. It is written using the Groovy DSL and can be created through a text/groovy editor or through the configuration page on the Jenkins instance based on two syntaxes, namely: Declarative pipeline syntax and Scripted Pipeline syntax. The first one is a relatively new feature that supports the pipeline as a code concept. It makes the pipeline code easier to read and write. Whereas, the scripted pipeline is a traditional way of writing code that is written on the Jenkins UI instance. Since Groovy script was not typically desirable to all the users, the declarative pipeline was introduced to offer a simpler and more optioned Groovy syntax. (Lateef 2019.

3 IMPLEMENTATION

In this chapter, there will be a full implementation process from the configuration step to deploy and testing to produce statistical data and thereby gain an overview of strengths and weaknesses, which can be improved and upgraded for the author's project. There are two main parts of the development and are attached with two main software tools: deploying project with Amazon Web Services and continuous integration with Jenkins. As can be seen, the whole development process of the project is not challenging, but the content of it as FIGURE 6 is the core for any software engineering team to have sight of scalability, cost savings and easier to maintain.

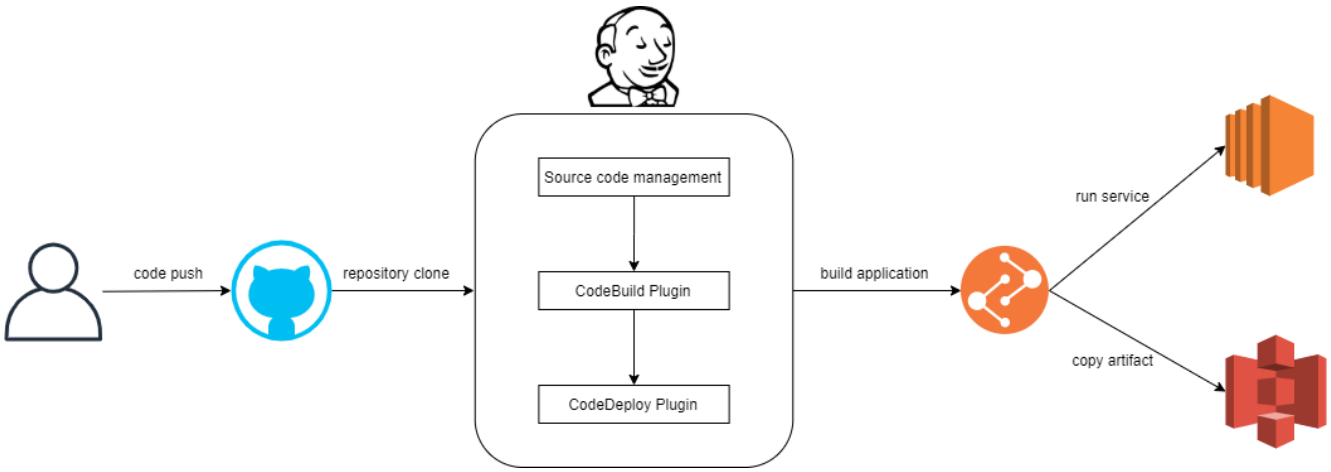

Figure 6. Use case pipeline architecture

3.1 Analysis and Configuration

It is important to sit down, be aware and set goals for specifying directions to be able to solve problems before embarking on any task. In the author's case, the goal is successful in delivery web application to online and everyone can access the website, also develop a continuous integration job to automate the test, build and deploy process and alleviate common problems. For AWS, there is a lot of work that needs to be done to get the web application online including setting up with S3, EC2, CloudFront, Certificate Manager, and Route53. With a new AWS account, after completing registration, Amazon has a free policy on some of its main services for 12 months such as 750 hours per month for Amazon EC2

and 5GB of standard storage with Amazon S3 with free tier. With Jenkins, it is needed to download from their website and choose the suitable release version for operation system.

3.1.1 Test case with React and .Net Core

The reason why this thesis decided to choose the frameworks to represent for implementation of the deploy and continuous integration process is because it is popular, attractive and its outstanding compared to other web technologies in the present time. Indeed, these two framework technologies are created by two big guys, Facebook and Microsoft, and quickly received the support and confidence to become as famous as it is now. Because of the thesis topic, both of React and .NET Core application will not truly be related to each other as their representation in the deploying web app process, therefore the author will not concentrate on the progress of building and explaining how the two operations like frontend and backend technologies collaborate by using API requests. Deploying operations is the first priority in this thesis as described from the start and the meaning of these two frameworks is just test cases to be able to show the steps in the development. Amazon Web Services and Jenkins have great diversity to allow to coordinate working with totally distinct frameworks, not just the two mentioned by the author, but React and .NET Core have easily surprised the author by the convenience and somewhat the favor of cloud technologies, making the deployment operations comfortable and advantageous for any developer.

There is a large number of various open-source React projects and libraries on the internet nowadays, since the author has found a random interesting source named “Emoji Search” that is providing simple searching components that can be easily implement to AWS.

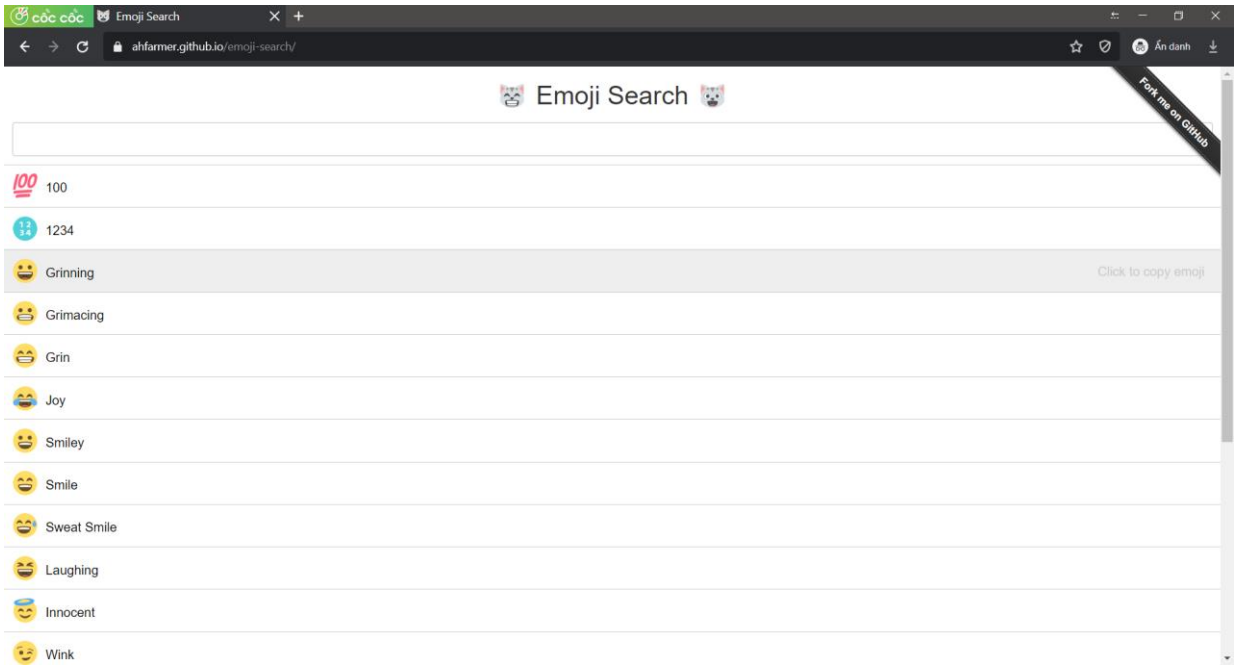

Figure 7. Live preview version of Emoji Search

On the other hand, a simple API application is created by the author based on the Microsoft tutorial documentation with some functionalities such as being able to send GET and receive POST requests. Hence the frontend and backend of the test case are not really linked, the author can use a third-party software called Postman to check that the .NET Core application actually works. As can be seen in the FIGURE 7 above, a POST request contains some JSON data that is sent to the root in the address bar and the API key is not required since it is made for testing intent. Therefore, it returned a status code 201 means “Created” and the data has been automatically appended with the id.

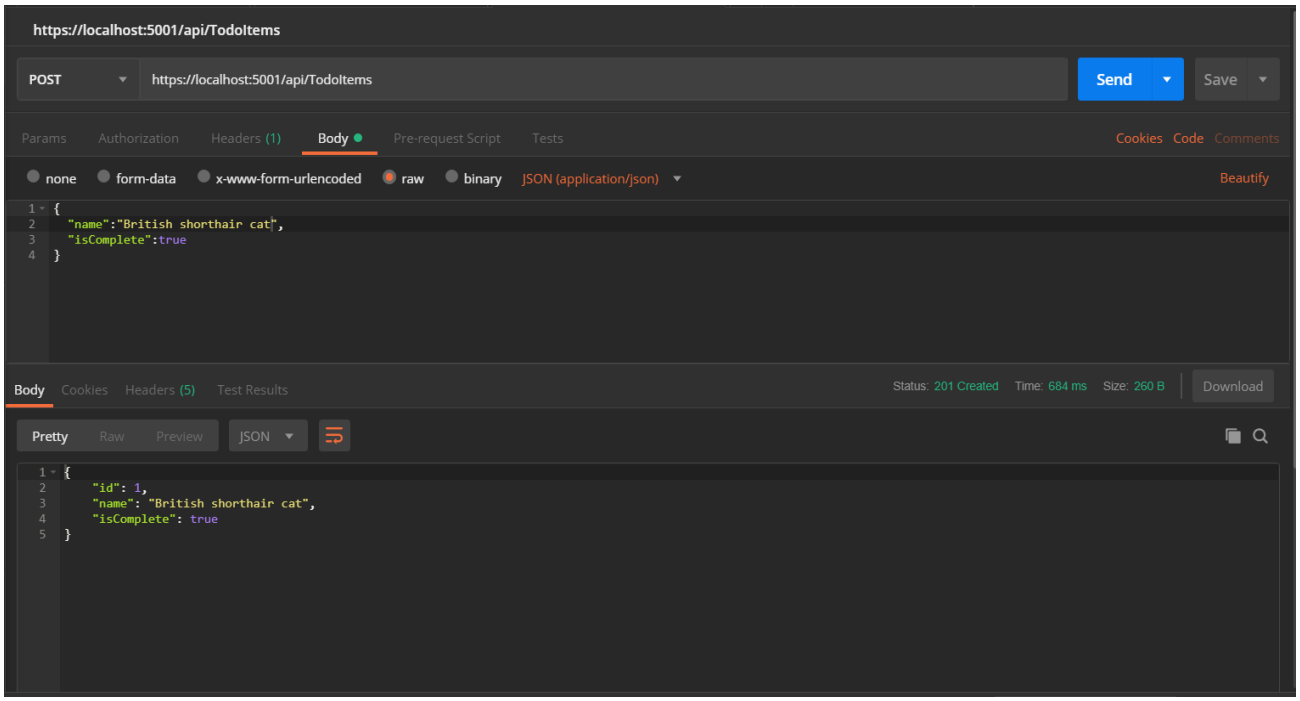

Figure 8. A POST request is sent by Postman to localhost

3.1.2 Initial Jenkins setup

After the installation from a valid version, Jenkins requires unlock password to ensure that it is securely set up by the administrator along with the password is automatically generated in a file named “initialAdminPassword” that it can be found in its own directory: “\root\Jenkins\secrets\inittialAdminPassword”. Lately, it is needed to fill in the local Jenkins server at “http://localhost:8080/” to be able to customize Jenkins step. Jenkins consists of a vast number of plugins that support many different needs so people have two options that they can choose what plugins they want to install, or the other choice is to install suggested plugins from Jenkins community finds most useful.

It is almost complete the installation of Jenkins and there are only two final steps that are most important to archive this tool are creating an admin account and instance configuration. This step requires some simple information like username, password, full name, email address and arranging for Jenkins a root URL for absolute links to various Jenkins resources. It is all information that is needed for the installation process and after FIGURE 9 the next people are free to use it in the way they want.

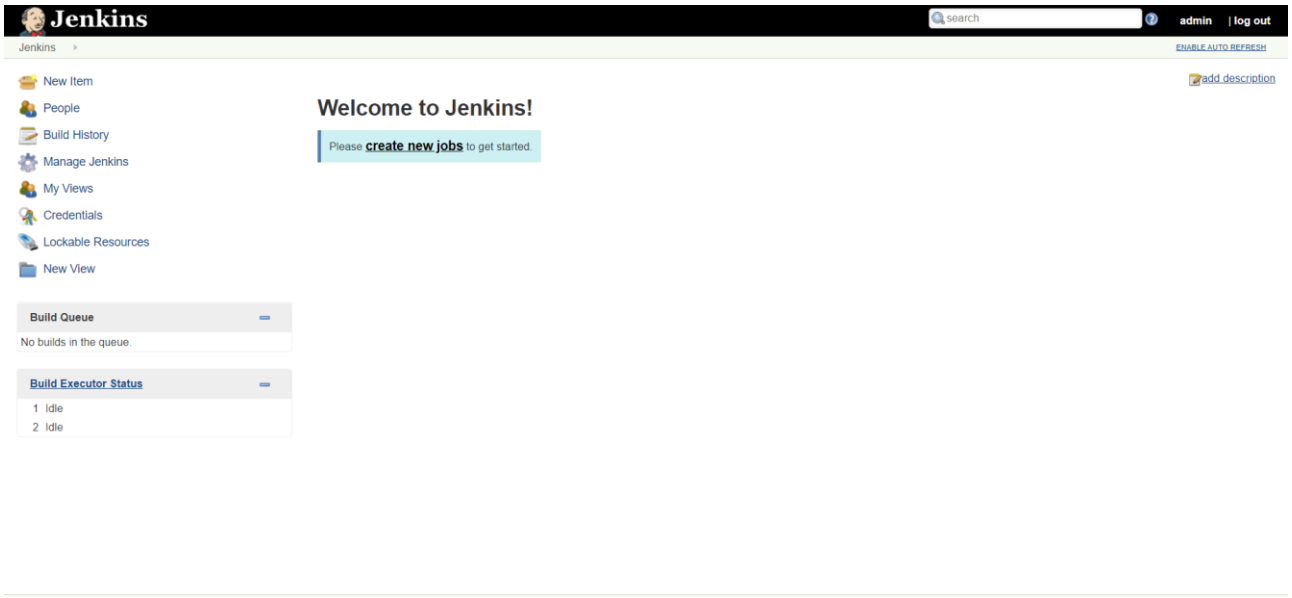

Figure 9. Jenkins homepage after installation

3.2 Development process

The development is divided into two fundamental parts as it is from the beginning of the thesis: Amazon Web service and Jenkins. The process's objective is deploying chain services that delivers applications and system to the customer in the possible fastest time.

3.2.1 Configuring Amazon Web Services

First of all, it is required to configure a number of services in order to develop a cloud environment for any application. Thanks to the easy and fast set up, S3 bucket is an excellent starting point for the configuration process. Especially complimentary capacity data for S3 bucket is 5GB which is a big plus for users to consider using the service. When configuring S3 service, this process does not have many points to be noted and most of the default settings can be preserved. However, there is still data that needs to be entered, such as the bucket's name and region, also its permission should be unselected "Block all public access". Although a warning message will pop up, allowing public access is needed for future environment plans.

After finishing the creating process, AWS just takes a few seconds in order to show the bucket in S3 console and it is the pleasant wait feeling when compared to the time it takes to launch other services. Nonetheless, it is not enough to make S3 work and there are still things to do with the S3 bucket policy

configuration. The policy uses JSON-based language to manage advanced permission to bucket's resources, so in the case of the thesis topic, it can be writing as below:

Bucket policy editor

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
                   "Sid": "AddPerm",
                   "Effect": "Allow",
                   "Principal": "*",
                   "Action": "s3:GetObject",
                   "Resource": "arn:aws:s3:::{bucketName}.{domainName}.com/*"
         }
                   ]
 }
```

FIGURE 10. Public JSON-based policy for S3 bucket

By default, there is no selected website hosting option in S3 bucket setting and it should be considered to set the bucket home page and error page directory, for example, index.html and error.html. Last but not least, the most important task of S3 bucket is to store data, then uploading data into the bucket is an indispensable task that can be automated in the future with Jenkins. At this step, Jenkins task was not available so it could be done manually drag and drop file to uploading section.

Contrarily, to improve the application's security and performance whereas adequately controlling charge, it is recommended that also arrange CloudFront Service to cooperate with S3 bucket to deliver and secure the content. Since CloudFront has a critical role in shipping data to customers by encrypting communications with a custom SSL certificate and also get DDoS protection with AWS shield standard by default, its architecture with S3 service can be described as FIGURE 11 shown below.

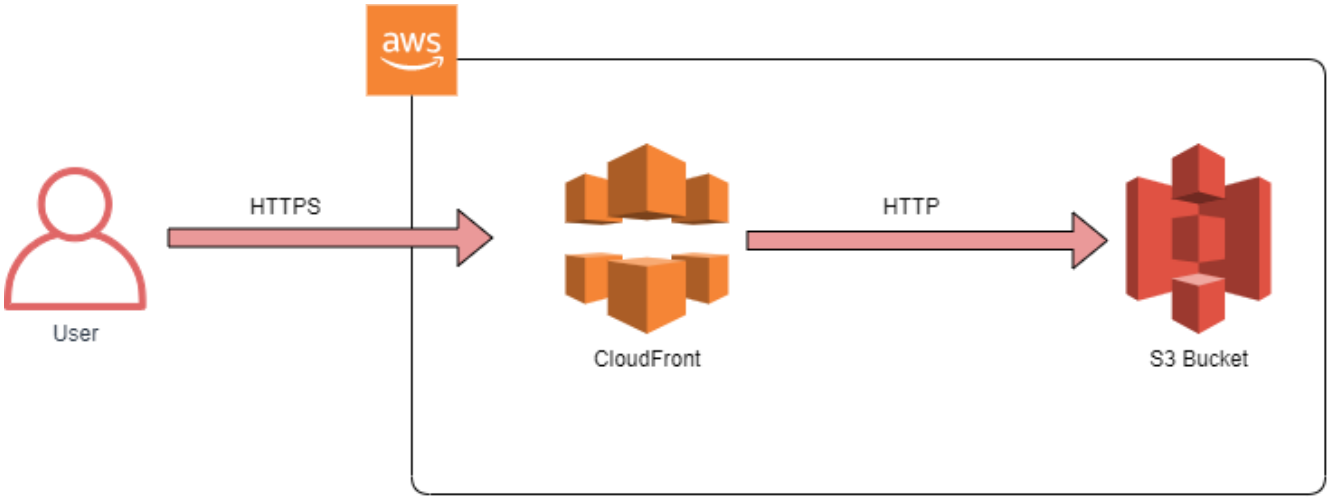

Figure 11. S3 Services diagram

CloudFront is needed to create in web distribution type that serves for static and dynamic contents. On the Create Distribution page, Origin Settings has one mandatory field that must be filled is Origin Domain Name by previous Amazon S3 bucket name and it will automatically input distribution ID for Origin ID field according to the domain. The rest of the setting fields such as Origin Path, Origin Customer Headers and Restrict Bucket Access are all optional and it is fully possible to leave it as default values. In addition, the next Default Cache Behavior Settings field is all right to leave as default at this step and the distribution behavior can be changed later when the customized SSL certificate for HTTPS connection is available. The last part is believed to be the most significant part of the whole process by the author is Distribution Settings that contains Alternative Domain Names (CNAMEs) input and SSL certificate options. People can choose to enter their personalized domain name that can be created with DNS Service as a CNAME record or using AWS origin that ends with ".cloudfront.net". Furthermore, it can be configurated to get the same Default Root Object as S3 for targeting to the right destination on the bucket.

In essence, construct a virtual machine on EC2 service is a long process and it requires an absolute perfect setup to avoid any mistakes that cause the machine to operate in a wrong way. For the thesis application, the author chose an Amazon Linux 2 virtual machine category as the machine image that is included considerable latest software packages through extras. For optimizing to fit the test case, t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1GiB memory, EBS only) type does not

only suit the best but also because of the free tier, there is 750 hours free per month for the first 12 months of a new account. Another momentous thing is provided for ec2 instances as a protection methodology is the security group that was mentioned above, and it is part of the setup steps for a new instance. Security group naming is not an easy subject that may be difficult and confusing in some cases even with the help of its description feature because of using for both EC2 instance and load balancer so it has to set up carefully. There is two connection that should be allowed in security group is SSH connection in port 22 for personal IP and custom Transmission Control Protocol (TCP) for next part load balancer connection in port 5000 same as the application running port. Before completing the launching instance process and if there is nothing goes wrong, it is needed to create a private key file for the first time as a security thing that consists of a password used to log into its correct instance. Hence, it is able to have a look for the setting of new instance that included critical information after a launching process as shown in FIGURE 11: Public DNS, IPv4 Public IP, Security Group Name, Private DNS and Private IPs.

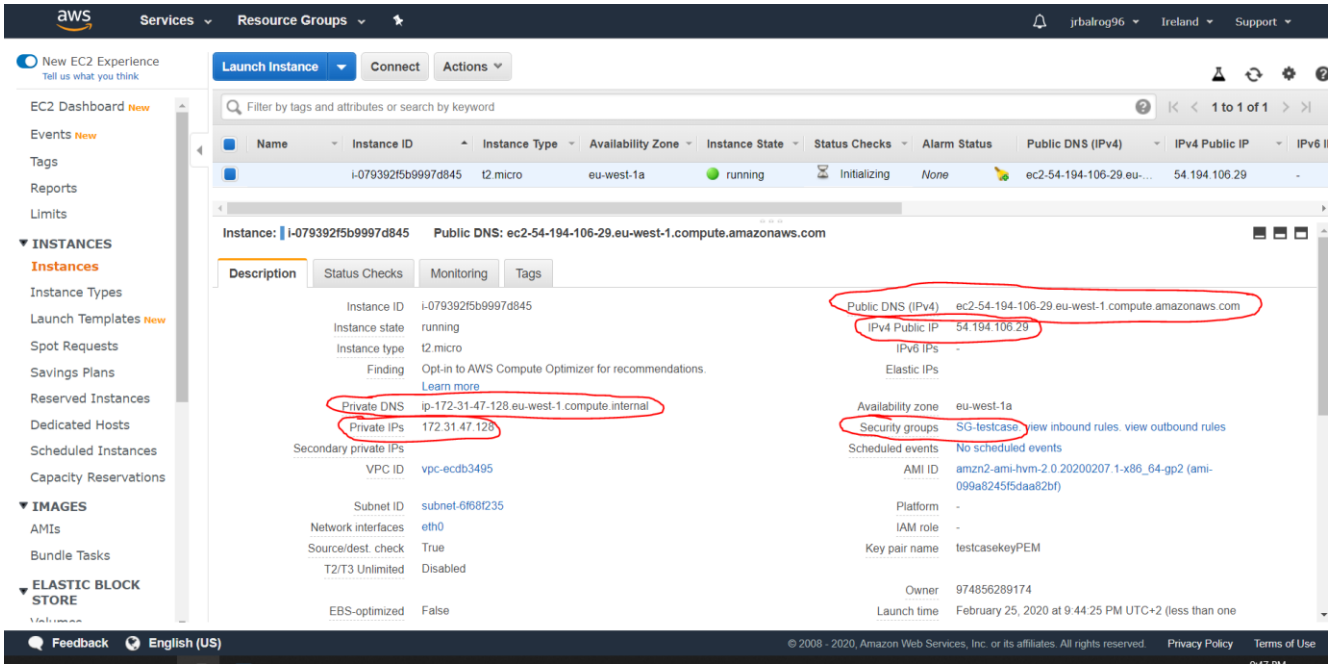

Figure 12. EC2 Instance overview

There are several ways to connect to the instance such as connecting by EC2 console or Command Prompt. Cleverly, Command Prompt seems to be the optimal and faster option in terms of speed and ease of control. The first thing to do when getting into the instance is applying all updates of AWS by running “sudo yum update” on Command Prompt. Then it is time for the register to Microsoft repository, install needed packages for the instance by run following commands:

Table 1. Required installation of .NET application

In terminal
sudo rpm -Uvh https://packages.microsoft.com/config/rhel/7/packages-microsoft-prod.rpm sudo yum update sudo yum install libunwind libicu sudo yum install dotnet-sdk-2.2.x86_64

Last but not least, FileZilla program will take responsibility for the transferring deployment file from local machine to EC2 instance with included key file from the settings. Afterall, running the application by command: “sudo ASPNETCORE_URLS=”https://*5000” /usr/bin/dotnet /home/ec2-user/publish/test.dll ENVIRONMENT=Development” (see FIGURE 13) and checking the application at URL “http://54.194.106.29/api/TodoItems” (see FIGURE 14).

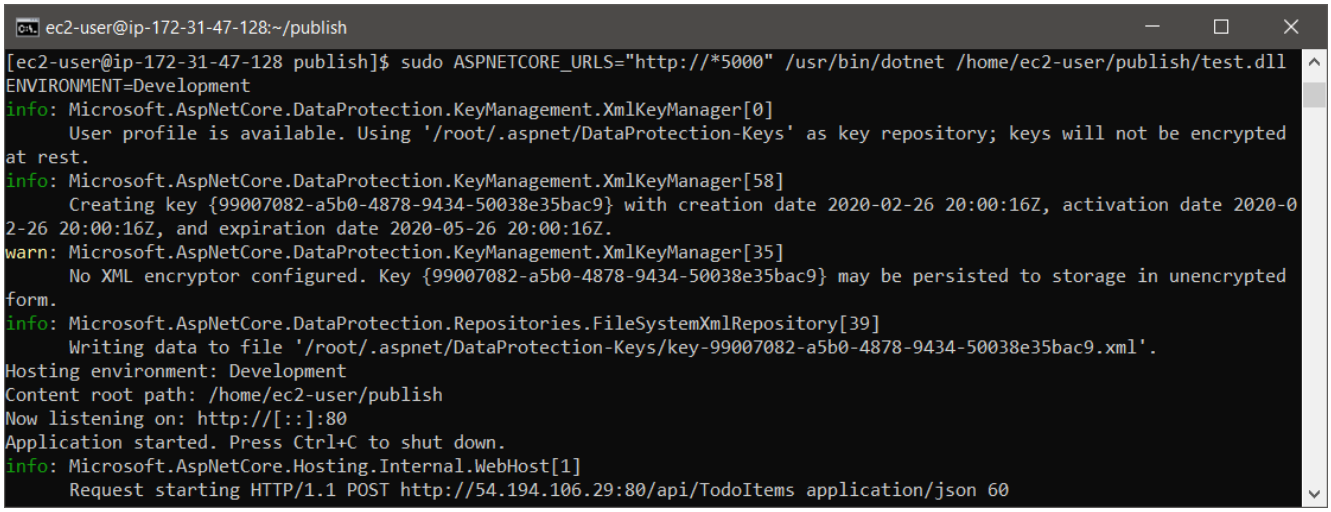

Figure 13. Running the application on Command Prompt

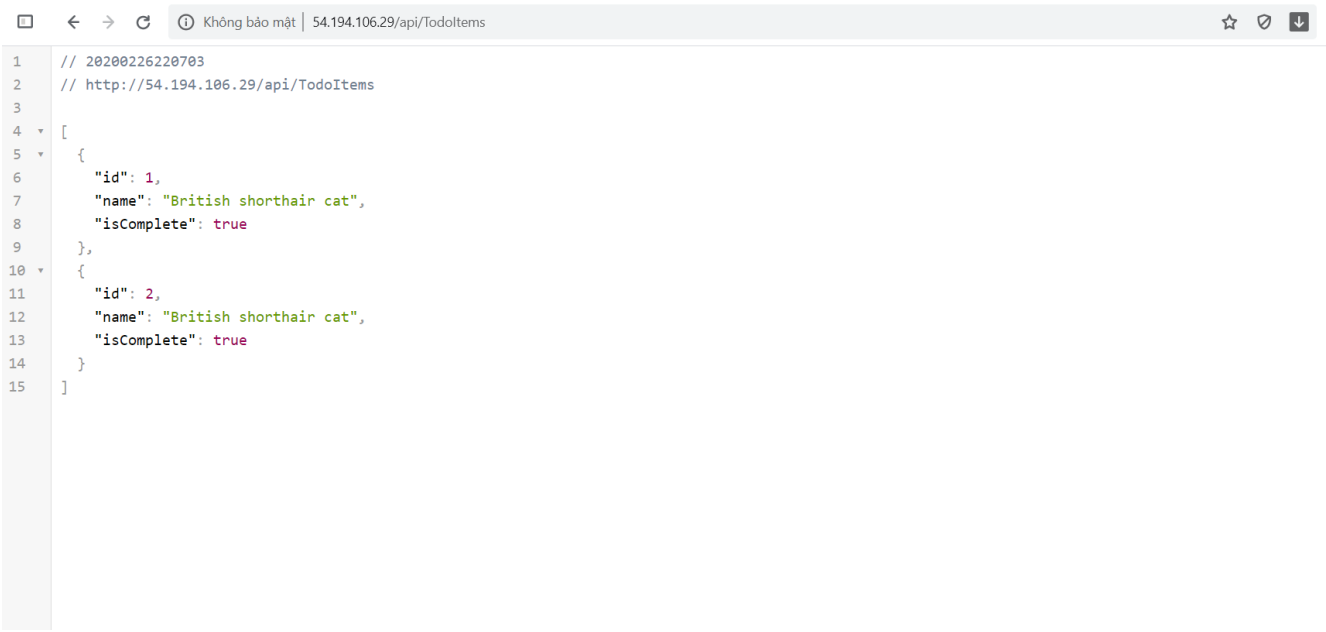

Figure 14. JSON data on Public DNS

Elastic Load Balancing automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances. It can handle the varying load of application traffic in a single Availability Zone. Application Load Balancer is best suited for load balancing of HTTP and HTTPS traffic and provides advanced request routing targeted at the delivery of modern application architectures including microservices and containers. Operating at the individual request level (Layer 7), Application Load Balancer routes traffic to targets within Amazon Virtual Private Cloud (Amazon VPC) based on the content of the request. Elastic Load Balancing can also route traffic to healthy targets in different Availability Zones and the Amazon Elastic Load Balancing Service Level Agreement commitment is 99.99% availability for a load balancer. (Elastic Load Balancing 2020.)

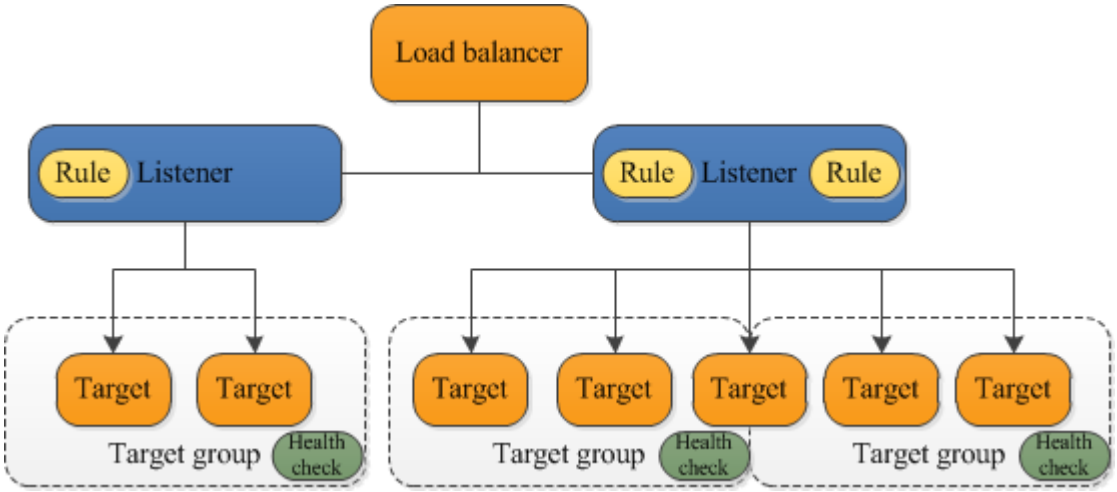

Figure 15. Load Balancer architect

There are three types of load balancer: Application Load Balancer, Network Load Balancer and Classic Load Balancer. Each balancer has its own uses and characteristics so that users can choose which one suits them best. With this author's situation, Application Load Balancer is the chosen one because of its suitability to use the element with HTTP and HTTPS requests in two Availability Zones are eu-west-1a and eu-west-1b of default VPC. A load balancer also requires another security group to control the traffic and it should allow from all source those HTTP requests in port 80 and HTTPS request in port 443 with a customized SSL certificate. Therefore, there will be two listeners of connections for HTTPS forwards to port 80 of EC2 instance and HTTP that must be redirected to port 443 of HTTPS. A target group is added to load balancer for routing requests to the EC2 instance in port 80 as the running port of .NET application.

3.2.2 Installing SSL certificate and sub domain

AWS Certificate Manager is a service that lets people easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with AWS services and internal connected resources (AWS Certificate Manager 2020). It is convenient to request an SSL certificate with just a few clicks in Certificate Manager service and easier for validating the certificate by functionality on this service that will automatically create a CNAME record in Route 53 service according to the information of the certificate or people can choose to create a record by information manually.

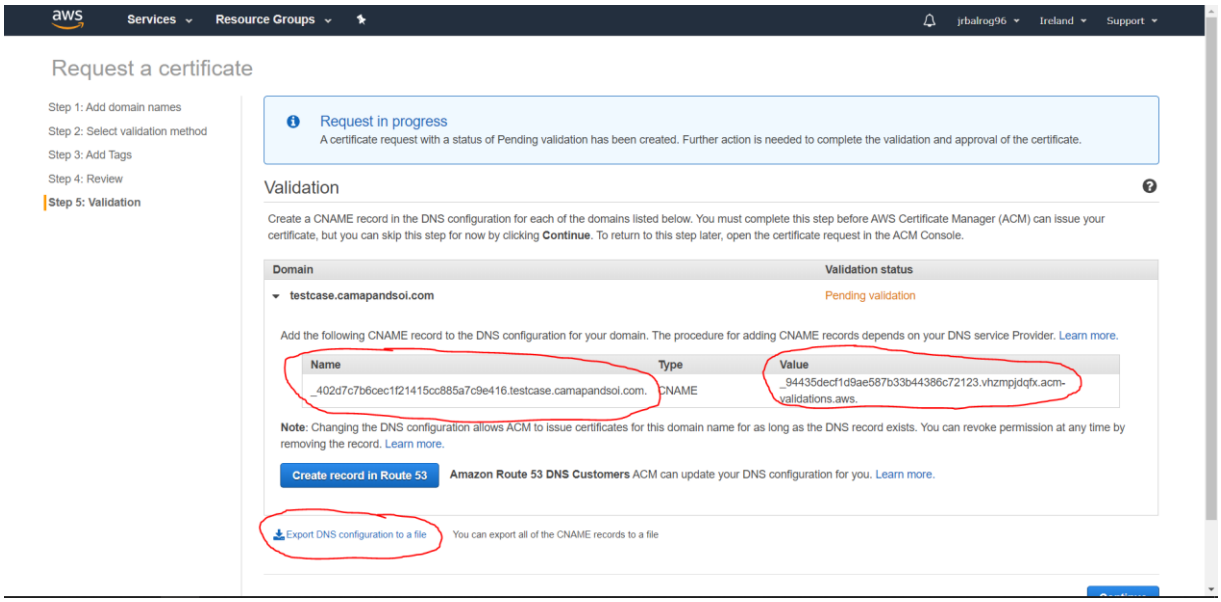

Figure 16. Validation step in requesting a certificate process

Amazon Route53 is a Domain Name System (DNS) service that is designed to give developers and businesses an extremely reliable and cost-effective way to route end-user to Internet applications by translating names like www.example.com into the numeric IP addresses like 192.0.2.1 that computers use to connect to each other (Amazon Route 53). It provides simple methodologies to registering hosted zones and manages domain names not only from AWS infrastructure but also outside of AWS. Hosted zone and domain names are required to share the same name for routing record's data to the right domain. The records for validatation certificate must be registered as CNAME type and the records for S3 bucket and EC2 instance must be certified as A-type. Values for record of sub-domains can be selected as Alias that Amazon will provide a list of Alias target included CloudFront and Application Load Balancer. It will take a few minutes to process and after SSL certificates are verified, it can be applied to CloudFront and Load Balancer for HTTPS connection.

3.2.3 Configuring Jenkins and plugins

A simple configuring Jenkins process includes several plugins and credentials for source code management as well as a build environment. Because of React application, NodeJS plugin indispensable to be able to run npm commands on the execute shell of Jenkins. The plugin can be easily found at Available tab on Plugin Manager dialog and there is no need for restart after installation. After that, the NodeJS section in Global Tool Configuration will available for adding NodeJS installer so then people can choose a suitable version from nodejs.org for providing Node and npm folder to PATH of Jenkins build tasks. Another plugin also needs to be installed is the SSH Agent plugin that allows users to provide SSH credentials to builds via its feature. What people need to do is just download it from Plugin Manager and then it will appear an SSH Agent option in the Build Environment section of the build task.

Jenkins server should be already integrated with Git and selecting Git option in Source Code Management scores many benefits of workspace management flexibility. Besides, it requires a global credential that contains the username and password of an account that is accessible to a specific repository. For connection to S3 buckets, the author used secret text options at build environment that consists of Username Variable and Password Variable with a root AWS credential or IAM credential with enough functionality to provide for the build task. Whereas in EC2 instance connection case, binding with its key file is an essential element for providing SSH credentials by SSH Agent plugin. The correct credential demands to be involved with username are ec2-user and private key data that can be entered directly by text. The proper key file information should start with "-----BEGIN RSA PRIVATE KEY-----" and end with "-----END RSA PRIVATE KEY-----".

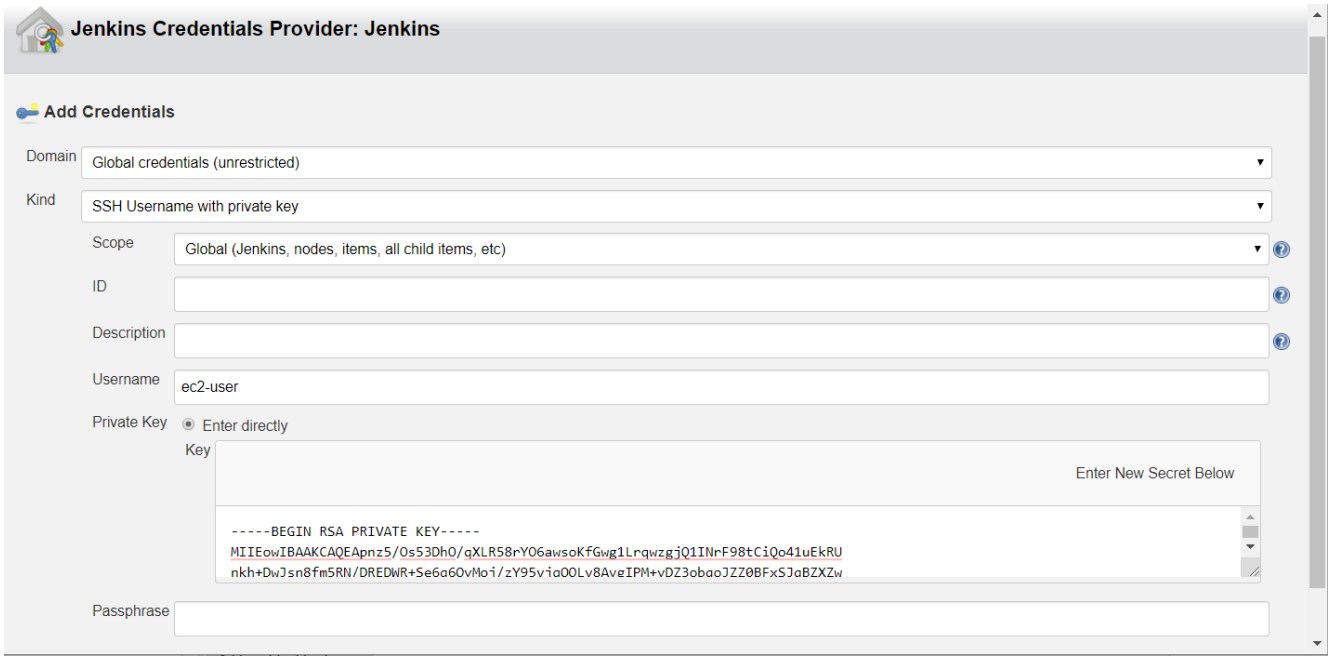

Figure 17. Adding credentials on Jenkins server

3.2.4 Jenkins build

The tasks for both of React and .NET applications are quite similar to each other with the same purpose is uploading files to the cloud environment. After completing configuration for source code management and build environment, there is still only one thing left to do is providing Jenkins scripts for executing in execute shell. Phenomenally, there are several options for executing scripts that depend on user demands and it is possible to have more than one execute step thus users can build an ordering execute script systems at their disposal. The scripts for React task are simple, include installing desired npm packages according to package.json file then build product folder for uploading it to S3 buckets. On the other hand, the scripts for .NET task is a little more complicated for controlling version and keeping backup files. Besides, to make transporting file to EC2 instance become a more pleasant and smooth experience, the author used to compressed and uncompressed the product version folder by tarball commands.

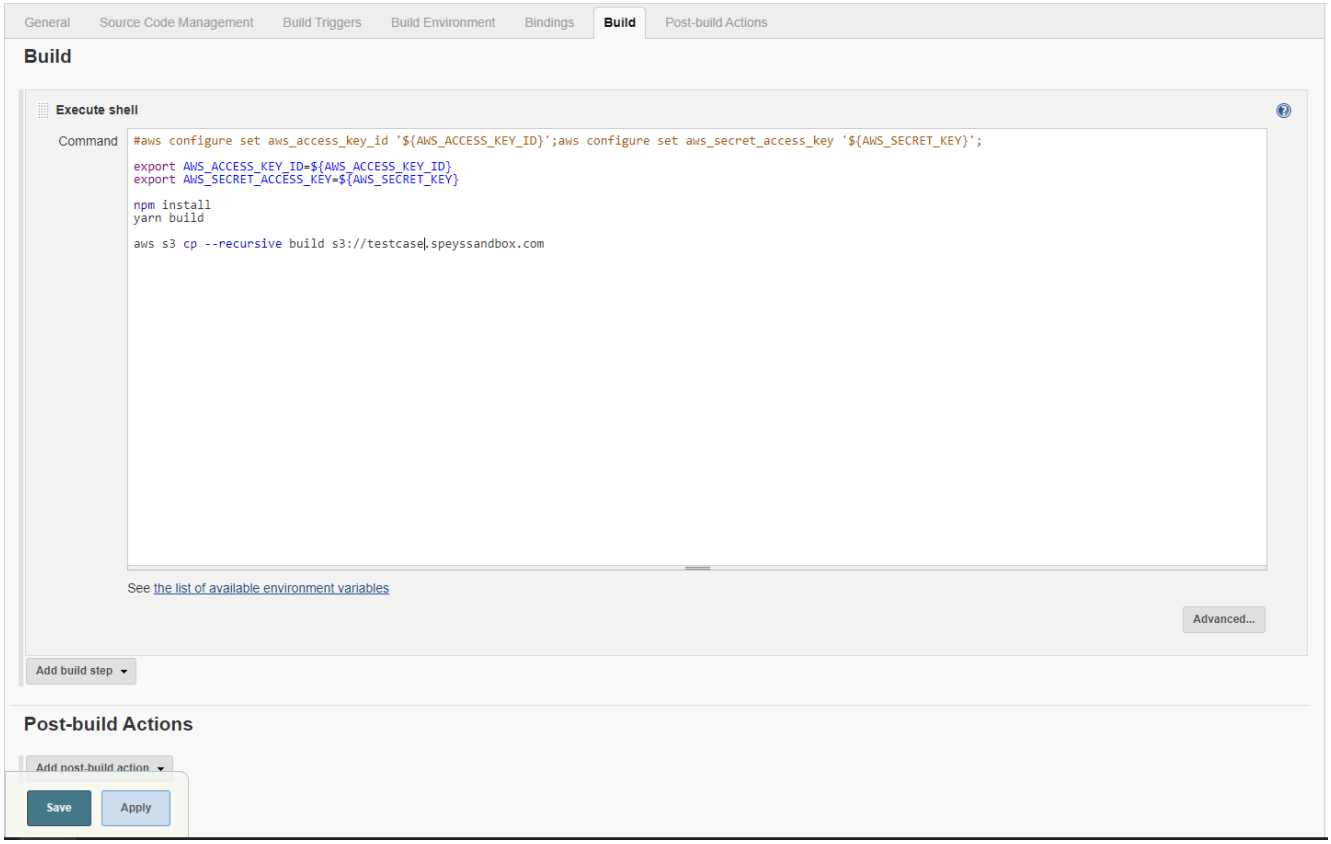

Figure 18. Build scripts for S3 buckets

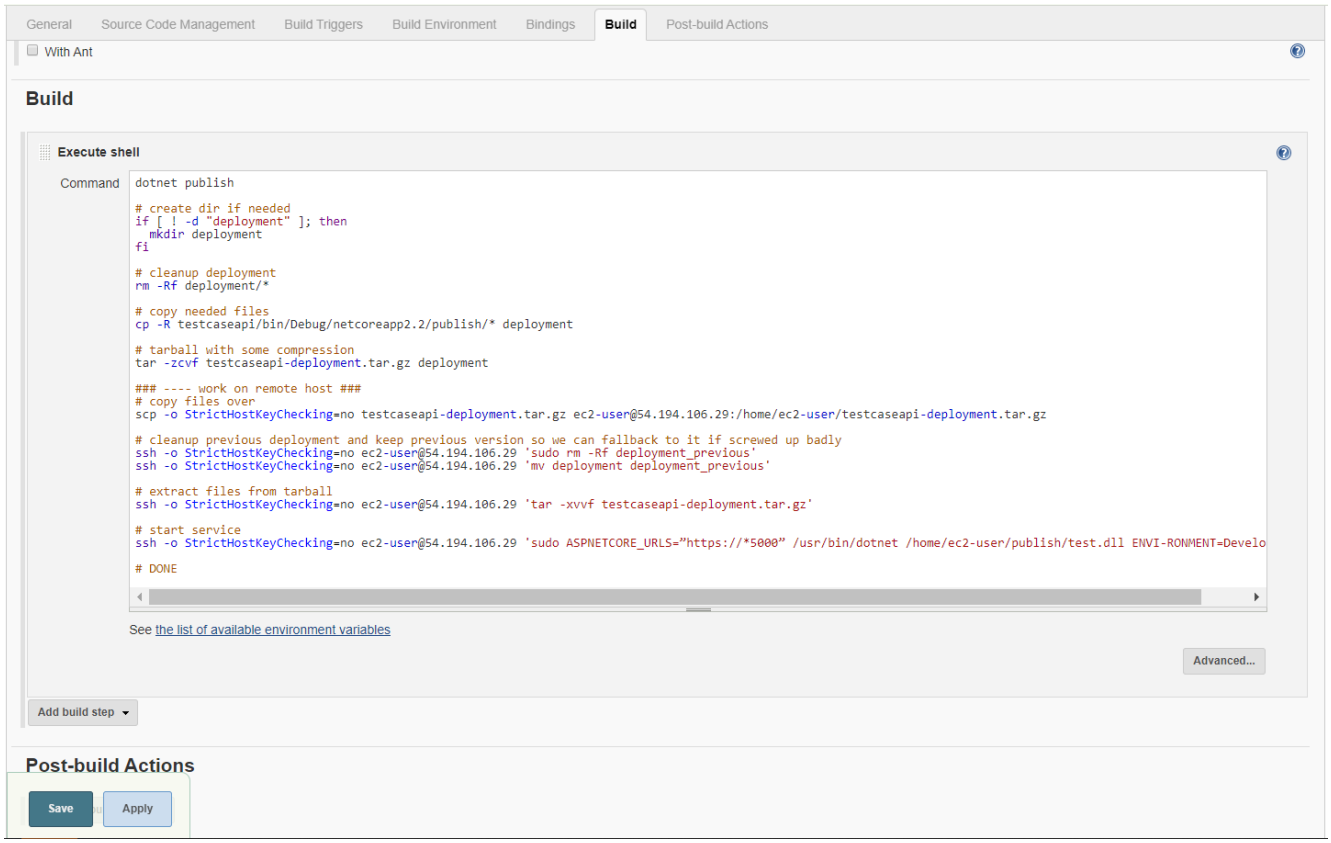

Figure 19. Build scripts for EC2 instance

3.3 Weakness and future improvements

Clearly observed, there is a number of lacking components that can be improved in the future. One weakness that can be found most clearly is the stability of the application on EC2 virtual machine and it leads to the missing information logging aspect for debugging the API operation. When there are somethings goes wrong and drive to the collapse of the system, the only actions that can be accepted to the developer is bringing back the backup image of the application without knowing anything about the cause of the crash. To keep the application stable, fortunately, there is a solution is creating background processes or known as Linux daemon that people can personalize their own service to handle running the application. In effect, a properly daemon service is included its descriptions for working environment, service scripts and user authorization. Specifically, whenever an incident occurred that caused shut down the system while the daemon Linux is in charge, it can bring the system back to work no matter how many times the system goes down.

In other aspects, the author believes that the project can still be further improved in terms of automation by automatically triggering Jenkins build on Github code push. It requires configuration on both Jenkins and Git sides by using Git Hook technique. There is a hidden folder in every Git repository named ".git" that is a hooks subfolder for executing scripts by Git to implements certain activities. From that functionality, users can use it to submit a request to an HTTP endpoint of Jenkins to polling for running a task.

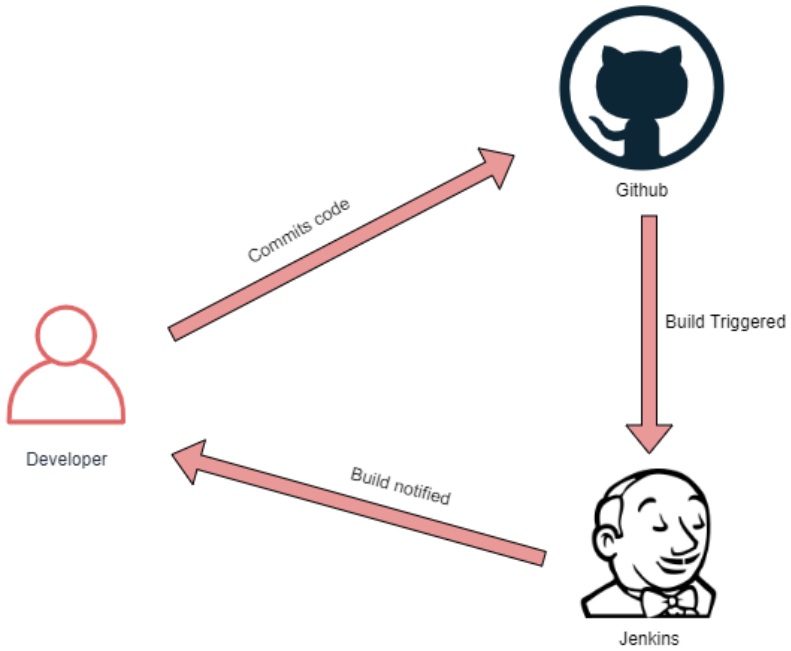

Figure 20. Build triggering diagram

4 CONCLUSION

Generally, this thesis aimed to demonstrate the concept of setting up an application by AWS and Jenkins. During the experience and implementation of the sampling process, since the author has built the system from scratch with the understanding he had when working on projects at his company, the development still took a week to be completed due to the complexity of AWS services interconnections. As stated at the beginning of the thesis, it is not possible to use a single service from AWS to be able to cater for all deployment needs and vice versa, requiring the user to have a clear and thorough understanding so that the establishment of the link between services can achieve high efficiency.

However, there are numerous affairs that have not been specified in the work, for example, is a collaboration with Github triggering build, AWS database services and Linux daemon service... All the examples cited are aimed at building a complete application deployment process to allow developers to save time and effort on stationing work. The subject can be expanded based on the level of automation that users expect. There is still room for further improvement in the future for the upcoming expansion feature of the system.

All in all, due to the scope of the thesis, it has fruitfully fulfilled research investigation and finalizes in illustrating the configuration course for application deployment by using Jenkins and AWS. The project is a model of a system that is operating at the moment and shows a certain efficiency by playing a critical role in the author’s company software development mechanism.

REFERENCES

AWS. 2020. What is cloud computing?. Available: https://aws.amazon.com/what-is-cloud-computing/. Accessed: 3 February 2020.

Yvanovich, R. 2016. Available: https://blog.trginternational.com/aws-powered-infrastructure-benefits-cloud-erp. Accessed: 30 March 2020.

Smart, F.G. 2011. Jenkins: The Definitive Guide. First edition. O'Reilly Media.

Saurabh. 2019. What is Jenkins? | Jenkins For Continuous Integration | Edureka. Available: https://www.edureka.co/blog/what-is-jenkins/. Accessed: 13 February 2020.

Lateef, Z. 2019. Jenkins Pipeline Tutorial: A Beginner's Guide to Continuous Delivery. Available: https://www.edureka.co/blog/jenkins-pipeline-tutorial-continuous-delivery. Accessed: 15 February 2020.

Kohgadai, A. 2020. What is Cloud Service. Available: https://www.skyhighnetworks.com/cloud-security-blog/what-is-a-cloud-service/. Accessed: 16 February 2020.

Jenkins Wikipedia. 2020. Available: https://en.wikipedia.org/wiki/Jenkins_(software). Accessed: 13 February 2020.

Heller, M. 2017. What is Jenkins? The CI server explained. Available: https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html. Accessed: 13 February 2020.

Golden, B. 2013. Amazon Web Services FOR DUMMIES. John Wiley & Sons.

Brandon, J. 2020. AWS: Your complete guide to Amazon Web Services & features. Available: https://www.techradar.com/news/aws. Accessed: 4 February 2020.

AWS. 2020. Working with Amazon S3 Buckets. Available: https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingBucket.html. Accessed: 11 February 2020.

AWS. 2020. Elastic Load Balancing. Available: https://aws.amazon.com/elasticloadbalancing/. Accessed: 7 March 2020.

AWS. 2020. AWS Certificate Manager. Available: https://aws.amazon.com/certificate-manager/. Accessed: 27 February 2020.

AWS Cloud Security. 2020. Available: https://aws.amazon.com/security/. Accessed: 11 February 2020.
Amazon S3. 2020. Available: https://aws.amazon.com/s3. Accessed: 8 February 2020.

Amazon S3 Storage Classes. Available: https://aws.amazon.com/s3/storage-classes/. Accessed: 30 March 2020.

Amazon Route 53. 2020. AWS-powered infrastructure benefits cloud ERP. Available: https://aws.amazon.com/route53/. Accessed: 7 March 2020.