UiO : University of Oslo

An Automated Test Framework for Web-Pages using Git, Jenkins, Docker and OpenStack

by

Samiul Saki Chowdhury (589795)
HiOA: (S316611)

Supervisor
Kyrre Begnum

Service Management and Developer Operations (MS019A / INF4019NSA)

Network and System Administration (Department of Informatics)
Faculty of Mathematics and Natural Sciences
University of Oslo
Norway

Oslo
May 19, 2017

Abstract

In this report, an automated continuous integration suite is created in order to achieve a test framework for web-pages using Git, Jenkins, Docker and OpenStack. The main focus of this framework is the automation of the entire process. Using this framework the web-developers can observe if the newly modified HTML codes (which are pushed from Git-repository) are working successfully or not before it is released to the production. Upon successful test build, the web-server is automatically running in to production which ensures the complete automated continuous integration test suite for web-pages.

Contents

List of Figures

Chapter 1

Introduction

In this project, we are given the task to create an automated test framework for web-pages using Git, Jenkins, Docker and OpenStack for a a company called StarFleet Design. This is a web-design company which builds predominantly builds web-pages for other companies. All the web-pages are on a local GitLab server. The designers have one git repository per site. They also have one web-server per site which is running in their own OpenStack cloud. Their current workflow for modifying a web-page for a customer is described as follows:

1. They first check out the latest version from the Git server
2. Then they modify the HTML pages (if necessary)
3. When all the changes are made, they use Git to add a new tag for the new version (they don't use semantic versioning, but only v1, v2, v3 etc.)
4. Then they commit and push the changes to the Git server
5. The designers log in to the particular web-server and go to the folder serving the HTML files.
6. Finally they pull the latest tag.

1.1 Problem Statement

The above method works well for the developers. They can easily work from home and on the remote location because they have local versions of the site they are working on their laptops. The only problem is when it needs to be tested. There has been too many situations lately where there has been a problem with the HTML code so the webpage has not shown the correct information. Customers are getting irritated.

1.2 Project Objectives

Our task in this project is to create a framework where new code can be tested before it is released on the production servers. Our main focus is on automation. The idea is that after step 4 of the above mentioned workflow, a Docker container is created that will automatically configure itself as a web-server and clone the desired version of the code from the Git server and install it. An external script will be executed to check if the web-server is working properly. The script will also look for a particular text in the web-page. After the test is complete, there should be a message at the end informing the user / designer whether the test was a success or not. The web-designer will then proceed to step 5 only if the test was a success. The designers have heard of Jenkins as a test-scheduling system and as for their interest we need to use it in our automation design.

1.3 Report Outline

Rest of this report is organised as follows:

- In Chapter 2, the necessary features of the referred tools and enabling technologies which are adapted to this project are briefly discussed.
- In Chapter 3, the framework of the project design is described and system setup is elaborated in order to achieve the project goals.
- In Chapter 4, the implementation of technical design of the test suite is thoroughly followed up using diagrams to give the readers a extensive idea behind the design performance.
- In Chapter 5, overall results of the design implementation of this project is discussed along with potential usage of the framework.
- And finally, Chapter 6 outlines the conclusion of this project.

Chapter 2

Background

Many tools are developed and shaped in the need of user's/developers needs in order to reach the desired goals. In the modern day operational system and infrastructure management developers tend to choose the right device that can provide them to integrate not only the source code management but also the continuity of testing part of the large project. To avoid large project deployment and bulky resource consuming systems the best tools that fits well for the project are chosen. The production of the software and testing to make it error less is a cost-effective and long process. Continuity is also a big factor in this technical implementation part of the development. Several different type of mechanism or appliances are adapted by the vast majority of the software development and operational management teams. In this Chapter, the tools and frameworks are used to implement our design in a cloud based webserver are discussed.

2.1 Continuous Integration

Continuous integration (CI) has become big part of the software engineering practice which ensures the changes done in the software or other configuration settings are tested immediately. With the help of CI the developers can not only just test the newer version of the setup but also can automate the part where it is reported back to the users. The CI takes over when the large amount of codes are added to the infrastructure. The goal of it to rapidly send feedbacks so that if any defect or error occurs is introduced in the code base it can immediately identified and fixed as soon as possible. As a software tools it is mainly used for the testing and building automation of the entire project or specified part of it. In the software engineering practice testing and reporting error or defect of the code can be a infrastructure nightmare specially if the code base is large and developed by different teams. Since the concept of CI has created it has been evolved in many ways. When once it was just designed to make builds in daily basis, now it is developed to let each team members to commit and push their work in more frequent manner. The builds are conducting in this way have made a significant improvement in software developing field. Various benefits are introduced in CI such as constant feedback of the status of the software. The code defects are now mostly avoided in the building process as the CI can detect the imperfections in the build. Complexity of the codes are lessened and resolving the errors had became much easier. A best practice of CI includes committing the codes frequently, using a designated integration building machine, using continuous feedbacks, categorizing tests carried by the developers and may more.

The principle of CI can be applied to any iterative programming model. In the automated continuous integration, many tools are adopted in order to achieve the desired goals. The automation is done by Git which most adopted tool by developers and software engineers for tracking changes and version control of the computer files. Coordinating different files among multiple people is achieved using this tool. Additionally, one of the most well established tools that is being used to eliminate the need to deploy to remote servers and run integration tests in the same server with the build is Docker. In the part where the need to involve scale is also covered by using a interactive tool called Jenkins. In this Chapter, a brief introduction to tools are used for automated continuous integration are discussed [1].

2.1.1 Benefits of continuous integration

CI introduce the developers to reduced risks. It reduces the unknowability of long project dueness by finding the bugs in the code itself and the build system. Usually there is complete blind sport where developers find it hard to determine how long it can take to do the build or how far they are through the process. The long integration is eliminated with the help of CI users eliminate the blind spot as well. The team members can always know or keep track of what is the status of the project hence how far along they are in the project. Bugs in the programmes can jumble the development and can mess up the schedules or some times reputations. Also bugs in a system can be hard to track and fix once its in production. This makes harder to get the rest of the software working properly. CI cannot necessarily get rid of al the bugs but it does make it dramatically easier to find and remove while in development. Each small part of the code can be checked in each commit / push which is much quicker than looking through the entire project at the same time. Keeping track of the software in small part at a time is a simple and easy solution to avoid bulk error codes. The more bugs in the codes harder it is to find and remove.

Additionally, since developers can keep track of each own part of the code if something goes down it becomes quick and easier to track down who is in charge of that bit of the source code and send the error message to that specific member. In this way the member can easily recognise his/her own error and fix and build in quick manner. Frequent deployment plays a big role which is valuable because it allows users to get new features more rapidly, to give give more rapid feedback and generally become more collaborative in the development cycle. Both in production and process CI carries an important role in software development [1].

2.2 Git

Before discussing about Git, it is also essential to understand the principal behind the version control system (VCS). It is a category of tool (software) which helps the team members (developers) to manage the changes of the source code over long period of time. Version control keep the developers to keep track of every modification have been made to a file or folder in case a mistake has been done and the builds / workspaces need to be rebuild to previous stable build. It is a special kind of database tracking system for source codes which helps minimizing disruptions to all the team members [2].

Software developers writing new source codes and changing them regularly. The folder structure or the file tree of the project can be monitored by applying VCS in it. This helps teams to solve the problems that is individually done or accidentally modified. This way the bugs of the code can also be avoided. Software testing becomes much more controllable and lot less error free. A stable version can be released until the new stable version is available to implement. The testing and development can be progressed simultaneously by using VCS. It is an essential part of the every-day of the modern software team's professional practice. The teams recognises the value of version control as it is one of the must have tool that is can be also used in non-software based projects as well.

Developing software without using version control is risky, like not having backups. It allows software teams to preserve efficiency and agility as the team scales to include more developers. VCS have seen great improvements over the past few decades and some are better than others. VCS are sometimes known as Source Code Management (SCM) tools or Revision Control System (RCS). One of the most popular VCS tools in use today is called Git.

2.2.1 Benefits of using Git

Git is one of best choice for most software development teams today. It is the most widely used version control system in the modern world today. It is an open source project (originally developed in 2005 by Linus Torvalds). In the wide range of Integrated development environments as well as operating system Git is well adapted by the developers. Open source to commercial projects, almost every software developing project uses Git for the version control. Git has a distributed architecture which is called as distributed version control system. Instead of storing only one main copy of the full version history, every team can store and commit their own version of the working copy and which contains the full history of every changes. Besides having distributing architecture this tool is also designed for performance, security and flexibility [3].

Performance

Comparing other alternatives for performance characteristics Git have a better and stronger approach. Among them committing new changes, merging, comparison previous versions and branching are all optimised. The knowledge about common attributes of real source code file trees and how are they usually modified over time are basic of algorithms implemented behind Git. It specifically focuses on content of the file itself instead of names of the file. The changes of source code (such as renaming, splitting and etc.) are part of the version control in Git. The Git repository files uses the combination of delta encoding which in simple word the differences of the content in each version. The compression and the categorical changes saves the directory contents with their metadata object. Having a distributed feature enables the significant performance benefit to this tool.

Security

Integrity of the managed source code has been the top priority for Git design principal. The content of the files, relationship between all the files and folders, commits, tags and metadata in Git repository are usually secured with secure hashing algorithm (SHA1). This protects the codes to be modified accidentally or maliciously and at the same time make the history of the changes more traceable. The companies which are relying on the software development can face a serious issue in information security with other version control tools. Git provides the authentic content history of the source code itself.

Flexibility

Flexibility is the key design of Git. In support of different kind of non-linear development workflows, projects and its comparabilities with many existing systems and protocols, this tool has the flexible approach. The tool has been designed to support branching and tagging as well as merging and reverting as part of the change history. This allow better control over large projects.

2.3 Docker

Docker is an open-source technology that is vastly embraced by proprietary software companies such as Microsoft and mostly UNIX systems. The most important reason behind using docker as has lower system requirements than of VM hypervisors, such as Hyper-V, KVM and Xen. Containers spawned in docker use shared operating systems, that means they are much more efficient than hypervisors when in comes the term for system resources. Containers rest on top of the Linux instances instead of virtualising hardware which means it is possible to leave behind all the bulky VM junks and just with a small capsule containers with a distinct application/s. Therefore with a perfectly tuned docker container system anyone can have as many as 4-6 times of the number of server applications instances than hypervisors. Although, docker is a new name in the developing field but spawning containers is quite old idea. This idea dates back at least year 2000. Other companies such as Oracle Solaris also has a similar concept called Zones but companies like Docker and Google concentrate more on open-source projects as OpenVZ and LXC (Linux Containers) to make it work better and more secured.

Docker is built on top of LXC and has its own file system, storage, CPU, RAM and so on. The key difference between containers and VMs is that the hypervisors abstracts an entire device while containers only abstract the operating system kernels. Hypervisors can use different operating systems and kernel such as Microsoft Azure can run on both Windows Server and SUSE Linux Enterprise Server but with the Docker, all the containers must use the same operating system and kernel. On the other hand, Docker containers can be used to get most server application instances running on the least amount of hardware. This approach can save a large amount of resources for the cloud providers or data centres. Using Docker the deployment of containers become easier and safer. Developers around the globe are using Docker to pack, ship and run any application as portable, self sufficient, lightweight containers in virtually anywhere. Docker containers are easy to deploy in the cloud. It has been designed to incorporate into most DevOps applications, including puppet, Chef, Ansible or just on its own to manage development environments. It makes easy to create and run ready containers with applications and it makes managing and deploying application much easier.

2.3.1 Benefits of using Docker

In the IT world running application instead of bulky virtual machines is a fast gaining momentum. The technology is one of the fastest growing in recent history due to its adaptation in industry level along with software vendors. Docker as a company and its software have grown immensely in technology field due to its usability.

Simple and fast configurations

One of the key benefit of Docker is the way it simplifies matter. VMs are allowing the users to run any platform with its own configurations on top of the infrastructure of the user where Docker has concentrated on the same benefit except reducing the overhead of a VM. An user can take their own configuration, put in into codes and deploy it in quick and easy manner. Docker containers can be used in wide variety of environments since the infrastructure requirements are no longer an issue for the environment of the application.

Increased productivity

Two major goals arises when when it comes to working in a developer environment, they are the bringing of the product as close to production as possible. It is done by running all the services on its own VM to show the application functionality However, any overhead needed to be avoid when compiling the application. The second goal is to make the development environment as fast as possible for interactive use. Receiving the feedback after tests is important in production level. Docker shows its functionality by not adding to the memory footprint and by allowing much more services at the same time.

Rapid deployment

The appearance of VMs took bringing up the hardware down to minutes. However, Docker manages to reduce this deployment time to mere seconds. The reason behind is that Docker containers creates every process but does not boot the OS. The cost of bringing up the system is next to null while creating and destroying the data in the Docker containers. The resources can be allocated in more aggressive manner with containers instances.

2.4 Jenkins

Jenkins, previously known as Hudson, is also an open source CI tool which is written in Java. Not until early 2011, Jenkins was part of the Oracle's project Hudson and the code has been renamed to Jenkins ever since. It is a cross-platform, continuous integration and continuous delivery application that increase developer's productivity. Developers or IT industry use Jenkins as a tool to build and test the software project continuously which made it easier to integrate change to the project. Additionally, it makes it easier for the users to obtain fresh build of the project. This allows continuous delivery of the software by providing ways to define build pipelines and integrating a large number of testing and deployment technologies It is easy to install, configure and rich in plug-in ecosystem. Jenkins integrates with virtually every SCM or build tool that exists. The tool allow the user to extend and modify most of its part.

2.4.1 Benefits of using Jenkins

The customization of Jenkins can be totally depend on user's preferences. The distribution of Jenkins can be built or load to multiple platforms. Some of the key benefits of Jenkins are discussed below [4]:

Governance and Community

Jenkins projects are managed by a open community whom are an independent board that includes long-time Hudson developers from different companies including Yahoo, CloudBees, Apture etc. The code to software in public interest (SPI) are donated to assure continued openness of the community.

Stability

Jenkins provide a more conservative and slower upgrade path to organizations A stable release with patches are announced in ca every three months.

Primary Platform for Plugins

Jenkins currently supports 392 plug-ins to be used by the users/developers. It is acting as the hub of the new applications development life-cycle with powerful and diverse functionality.

Cloud-Enabled

One of the key feature that Jenkins cover for its clients (users) is the lack of resources during peak periods. The tool provide a solution with DEV@cloud, providing developers with Jenkins build, test and package services without the need of configuring and maintaining their own build servers. Jenkins automates the non-human part of the whole software development process with common setup like CI. This empower the teams to implement the technical part of the continuous delivery. Based on server system its running in a servlet container such as Apache Tomcat. Jenkins supports SCM tools such as Git, Apache Maven, Subversion, Docker, RTC and many more. Jenkins builds can be triggered in various manner, e.g., by committing in a VCS, when the other builds queues have been completed, by scheduling via a CRON-like mechanism or simply by requesting a build from URL.

Security

Jenkins' security depends on two factors, access control and protection from external threats where access-control can be customized via two ways. They are user authentication and authorization as well as protection from CSPF external threats or such builds are also supported.

In the next Chapter, the technical design needed to meet this project goal and to achieve the automated continuous integration system to build a test framework for web-pages.

Chapter 3

Technical Design

The technical design part of the report focus on the design principle and design model to achieve test suite for web-pages using Docker, Jenkins, and GitLab. The framework can also be done using other alternative tools which can produce the similar result. As it is discussed earlier in Chapter 2, the benefits of above mentioned tools comes with a high recommendation and it is also necessary for this test to build on these tools since the developers/users of this framework like to use them. Following sections explain the design principle of the test suite.

3.1 Design Principle

To begin with the setup different steps have been taken in order to resolve the issue with automation in this project. The steps are briefly described in following:

- The repository has been created on the GitLab. It is a private repository in GitLab in which other team members (developers) can be added as group members. This repository can be cloned in any work machine in order to make changes and commit and/or push the changes. This will allow the HTML changes to be updated to the web-servers by taking the following steps.
- A server instance has been created in the Alto cloud and a floating IP is associated to it. The team will work on this server as well as the dedicated web-server, test-server and the Jenkins server will be running in this Alto cloud instance.
- Upon pushing/committing the changes on HTML files the GitLab repo will inform (trigger) a build job event request to Jenkins which is bonded by a webhook. A project webhook is used to trigger a URL if for example a new code is pushed or a new issue is created.
- Docker will be pre-installed and running in the server which will enable the team to spawn new containers for testing purposes.
- Jenkins is running on this server as one of the docker container as well.
- Jenkins will contact the server and send a build request for a temporary test-bench web-server as Docker container.
- The server will then build the temporary test-bench container, start it and download the latest copy of the HTML file/s from the repository to the specific folder.
- A test to check if the particular text/code is in the web-page.

- If the test is successful, Jenkins will request to remove the test-bench web-server and trigger a downstream job which only runs when the test result is successful or built is stable
- If the test was unsuccessful (specified text was not found on the web-page), Jenkins will send a notification to developer about fail test information and remove the temporary Docker container to make the job ready for the next test.
- Upon successful built and test, the downstream job will remove the previous copy of the HTML file from the default web-server and download and copy the latest stable version of the HTML code to the right folder.

3.2 Design Model

The model (Fig. 3.1) explains the principle behind implemented technical design of this project:

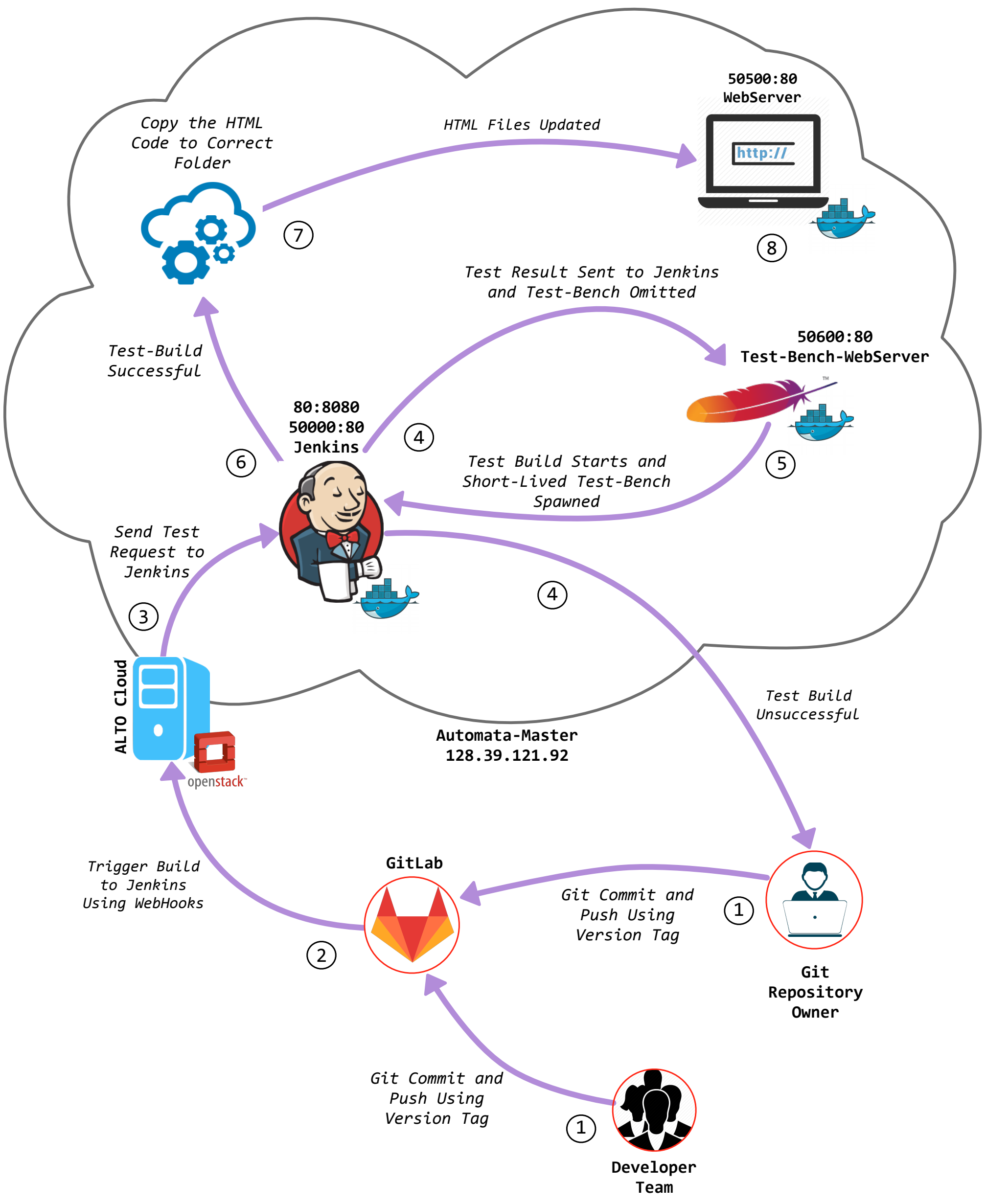


Figure 3.1: Illustration of the Technical Design Model of the Test Framework

The above diagram represents the design principal for an automated test framework for web-pages. The following section explains the software integration process that has been adapted to fit the project design principal in this project.

3.3 Design Architecture

This section is integrated into several subprocesses in order to cover implementation in different part of the project. This section will give the readers thorough perception on the design setup and the adjustment done on the tools used to attain the project goals.

Server setup

In the Alto cloud a server is created as instance in order to setup the tools for the technical implementation. The instance name is set to **Automata-Master** as it fits the profile of the project task. The Server is created from a Ubuntu 16.04 64 bit image and given a RAM amount of ca 8 GB. It is a large disk with 80 GB of total disk space. The instance is given the connection to the **nsa_master_net**. To make it accessible over internet a floating IP (**128.39.121.92**) is associated to this server. The mandatory software and services will be installed and running in this server. It is not necessary for the all the team members to have access to this server as most of the developing procedure will be done using Jenkins Git. Additionally, the a custom TCP rule is added to the server with port range of **50500 - 50600**. This rule makes the ports available/open to the server to use. Later the use of this rule will be discussed thoroughly. That is all needed to setup the server for the desired project goals.

Docker setup

Docker is installed in the server for help the developers create and test builds/jobs. The instruction to setup Docker for Ubuntu Xenial 16.04 is taken from the official Docker website [5]. The setup requires to install linux-image-extra-* packages, which allow Docker to use the aufs storage drivers. Then the packages to allow apt to use a repository over HTTPS needed to be installed followed by adding the Docker's official GPG key. And finally setting up the stable repository from Docker for Ubuntu Xenial distribution and update the server before the Docker-engine is installed. Docker is now running in the server automata-master.

Jenkins setup

Jenkins is needed to be installed as Docker container in the server. Jenkins could have also be installed in the server on its own but it is avoided due to some routing/iptables setup conflicts. This will be discussed more in Chapter 4. A Docker container has been created from Jenkins image with port binding of 80:8080. This means the Jenkins container port 8080 (standard http port) is exposed to host's port 80. This is done by running the following command in the server:

```
sudo docker run -d -v /home/jenkins:/var/jenkins_home --name=jenkins -p 80:8080 -p
50000:50000 -u root jenkins
```

The similar port binding could be done using **–publish=80:8080**. With -v option host's **/home/jenkins** volume is attached to Jenkins container's home folder i.e., **/var/jenkins_home** . The **-u** option added the Jenkins to the user group **root**. Additional setup is required in order to Jenkins to be allowed to create/build/run Docker container outside the Jenkins container. This means the using Docker builder plugins to execute Docker commands inside Jenkins and create the containers in the host (server) itself. It is not possible unless the Docker Remote API is enabled. Docker remote API is used in various situation such as running the Docker host on a remote location and wanted to be connect to it from different machine (in this case the Jenkins container on port 80). In this scenario, remote API can be used to connect to the Docker using representational state transfer (REST) APIs as the Docker engine accepts the REST requests. To enable this, Jenkins need to find the host machine's TCP address. The Docker engine server need to be bond with the Unix socket as well TCP port **4243** with IP **0.0.0.0**. This means engine accepts connections from all IP addressees. The system file **docker.service** file in **/lib/systemd/system/** folder need to have slight modification on the line that contains ExecStart:

```
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:4243 -H unix:///var/run/
docker.sock
```

After saving the file Docker services need to be notified about modified configuration followed by restarting the Docker service:

```
sudo systemctl daemon-reload
sudo service docker restart
```

By running the **curl -X GET http://localhost:4243/images/json** command it can be confirmed that the configuration has been changed. The command will output all the images in the Docker host in JSON format. Now in Jenkins the initial setup needed to be done by finding the password from **/var/jenkins_home/secrets/initialAdminPassword** in the Jenkins container. Jenkins is accessible in the server's (host's) post 80. So by opening the **128.39.121.92:80** redirects to Jenkins webpage. After entering password and creating a new user (with all the security permissions for Jenkins) the maintenance page can be accessed. To continue with the TCP access to host for Jenkins plugins from **Manage Jenkins>Manage Plugins>Available** tab **docker-build-step** and **CloudShare Docker-Machine Plugin** needed to be installed.

Now in the **Manage Jenkins>Configure System>Docker Builder** tab the enter the address **tcp://172.17.0.1:4243** and test the connection which should result with connected to the address. From the same tab a Docker cloud needed to be added and give a name such Docker and same TCP URL. Upon testing the connection it should show version of the API. The Jenkins setup is done. Containers can be spawned in the host server and commands can be executed using this configuration.

Additionally, some other plugins are installed such as **embeddable-build-status**,**Gitlab Authentication plugin**,**GitLab Hook Plugin**,**GitLab Plugin**, **Slack Notification Plugin**(optional) and **TextFinder Plugin** is installed from available plugins to achieve the design requisite.

Git setup

The repository in GitLab (**INF4019NSA-Project1**) will be used to modify the HTML pages by the developers. To gain fully automated continuous integration the repo needed to trigger the jobs created on Jenkins. The repo need to setup in such that every time a developer/user commit and push the repo after making changes to the code (HTML) using Git it well send a notification to the Jenkins which in this case a build job request. The setup is a simple configuration where the Git repository will save the Jenkins project's 's URL with its automated generate secret token. To setup this, a project needed to be created by selecting **New Item>Choose any item name>Choose a project type**. In the next menu (project menu), **Build when a change is pushed to GitLab** with GitLab CI service URL need to be checked and choose the desired configuration to be achieved with the repo. These plugins for GitLab was downloaded when setting up the Jenkins server. A randomly generated token can be found under Secret Token in the advance settings by generating it. After saving this configurations this token along with the URL need to be copied in the GitLab repo webhooks settings. After entering the URL and token (collected from Jenkins project configuration) add webhook needed to be clicked after checking/unchecking trigger options. Optionally, SSL verification can also be enabled. Now any push to this particular repo by developers of the team after alternation of HTML codes will trigger the build command in the Jenkins.

An additional setup to allow cloning of this private repository in any server is taken by attach a access token to the user. The personal access token can be created in GitLab profile settings of the user under the Access Token tab. By given name and choose a expiry date the token will be generated and can be copied to any Git clone repository as follows:

```
sudo git clone https://<user>:<private token>@git.example.com/myuser/myrepo.git
```

The similar setup could have been done using SSH keys by generating a key in the server and copying the public key to the GitLab account. The reason that personal access token is chosen because it is easy to distribute among all the team members to use it while cloning. For example, admin of the repository can give a cloning access token which can expires after certain time (which can be the project deadline). With the SSH key setup with GitLab it is necessary to share the private key with all the other team members who are developing the codes in a remote machine. But with the temporary token admin can give the specific group of team member a limited amount of time to work on the codes and the token expires itself after the time limit. Admin do not need to worry about the codes can be exposed to others when the project i done and the web-server is running. It is simpler to grant permission using private temporary token to the team than using SSH keys setup.

WebServer setup

A web-server is created from httpd image from docker hub which is running the Apache web-server. The latest version of the HTML codes will be running in this server. This is also a Docker container which is created from **httpd** (Apache) image with the port 80 of the container exposed to port 50500 of the server. It is available in the IP **128.39.121.92:50500** over internet. This would be the production web-server for the developers. The container is immutable and will not be altered in case the test-build is a failure.

The initial system setup have been done and the technical design is ready to be conducted in this infrastructure. The next Chapter will describe the technical design implementation of this project.

Chapter 4

Design Implementation

The previous Chapter explains the overall setup that has been done to automate the CI in the test suite for web-pages. Tools such as Jenkins, Git and Docker is integrated to work in CI mode. The implementation procedure is discussed in this section:

1 In the GitLab repository for this project a HTML file has been added named it **index.html**. Developer can change the code as needed and commit and push to the source repository with a specific version of tag with it.

2 Two freestyle projects/jobs have been created in Jenkins where the one project is to test if the modified HTML codes are running properly or if everything in the code is placed as demanded. This first project is named as **Test-Bench(Project1)**. The following configuration steps are taken with corresponding plugins to setup this project for test-bench.

Build Steps

- A Docker command to create a container from the image httpd (from docker hub) and named it test-bench-webserver. Under the port bindings field the 50600 port of server is exposed to container's port 80.
- Another command is to start the same container.
- The several commands to update the container, install Git, clone the copy of the latest push of the HTML code (index.html) with the personal access token using Git and finally copy it in the right folder (in the case of Apache its /usr/local/apache2/htdocs/) inside container is placed.
- An shell command is executed as follows: **curl -k http://128.39.121.92:50600** which gives the output of the HTML file link.

Post-Build Actions

- On the post-build section a text-finder plugin is executed to search for a specific word (in this case for Project 1 its only the simple "**Hello World**") in the console output of the build. If the particular text is found then it will show the test was a Success otherwise Failure.
- A build other project plugin to point to the another freestyle project which will only trigger only if the build is stable.
- Optionally, a slack notifications plugin is running at the end to send the build progress information to a specific channel. Team members can have a notification when the builds stats, stops and if it was successful or not.

- Finally, a docker command to remove the text-bench-webserver container.

3 The second project, named as **WebServer(Project-1)**, will be triggered if only the build of previous test-bench was stable (i.e., the container running successfully and specified text was found) have the following configurations:

  - Execute a docker command inside the default running production container named **web-server** to clone the latest stable tag from the repo, copy the HTML file to the right folder and replace the file.
  - Optionally, removing the downloaded files to avoid the duplication in the next build.
  - Slack notification to alert the team members of the developing team to show the status of build under the post-build action.

4.1 Design Analysis

Fig. 4.1 shows the flow diagram of the implementation of the design. These are the steps executed when a new version of the HTML code is changed:

Step: 1 The developer edit the code (index.html) and run a script to push the codes in the repo. A new tag version in included while committing.

Step: 2 GitLab receives this commit and triggers a job which redirects to the URL of the Test-Bench project in Jenkins. The trigger was done due to the webhook with the URL is specified in the repo.

Step: 3 Jenkins receives the trigger request.

Step: 4 Jenkins then starts the build schedule for the Test-Bench project.

Step: 5 Test-Bench project does the following steps:

- Creates a new Docker container in the server automata-master which is called test-bench-webserver. The container is available in 50600 port of the server.
- Starts the container.
- Runs the docker commands inside the container to install Git.
- Downloads the modified version of the index.html file using a personal temporary access token given by GitLab.
- The code is copied/replaced to the right folder.
- Executes a shell command to show the content of the index.html file in the console output of Jenkins build by having a curl command on the 50600 port of the server.

Step: 6 Runs a text-finder plugin to search for a specific text ("Hello World") in the content to test.

Step: 7 If the test was a success the container test-bench-webserver is removed to make the next test build available to execute. If not sends a test Failure notification to the user/developer and prevents triggering the build of second project after removing the test-bench-webserver container.

Step: 8 Upon success on the previous test the downstream project/job triggers to start.

Step: 9 WebServer build job downloads the latest version of the tag from repository in the production container called webserver. The container is available in the port 50600 of the server.

Step: 10 Then the index.html file is copied to right folder (/usr/local/apache2/htdocs/) in the container and removes the downloaded copy.

Step: 11 The latest changes can be viewed over internet at the production webserver, i.e., http://128.39.121.92:50500.

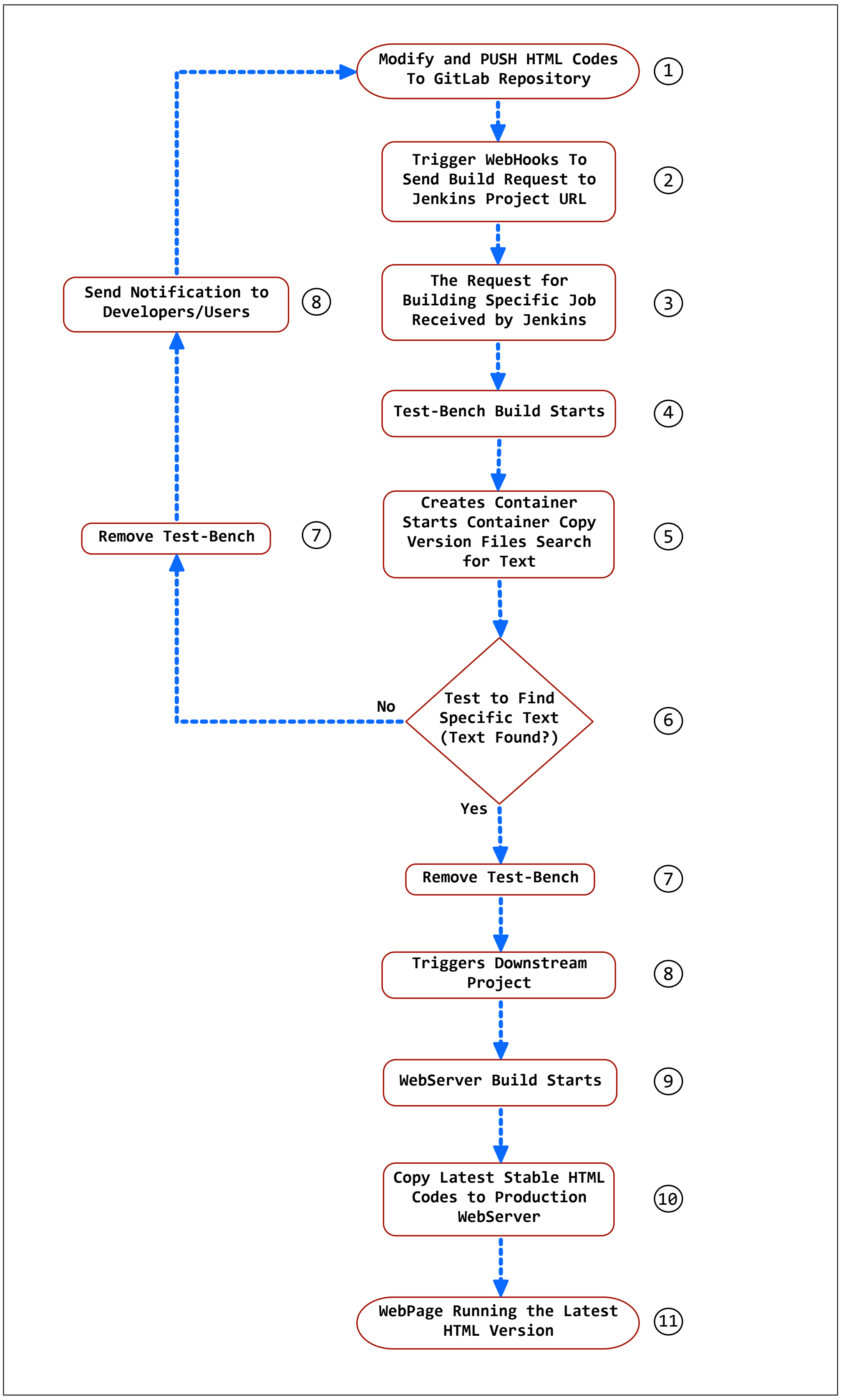

Modify and PUSH HTML Codes To GitLab Repository
1
Trigger WebHooks To Send Build Request to Jenkins Project URL
2
Send Notification to Developers/Users
8
The Request for Building Specific Job Received by Jenkins
3
Test-Bench Build Starts
4
Remove Test-Bench
7
Creates Container Starts Container Copy Version Files Search for Text
5
No
Test to Find Specific Text (Text Found?)
6
Yes
Remove Test-Bench
7
Triggers Downstream Project
8
WebServer Build Starts
9
Copy Latest Stable HTML Codes to Production WebServer
10
WebPage Running the Latest HTML Version
11

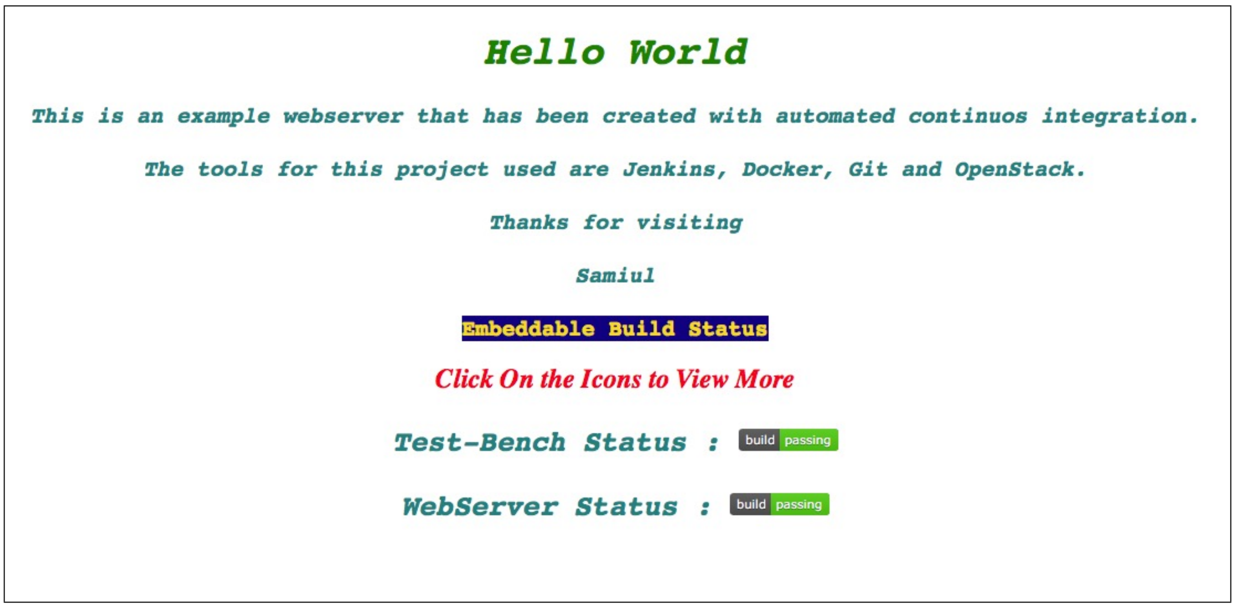
Figure 4.1: Illustration of the Flow Diagram of Design Implementation in Test Framework

Chapter 5

Discussion

In this Chapter, the overall success and future possible strategy can be taken are discussed. The HTML file (Appendix:A) that was changed was certainly a simple code to modify where we only modify the text "Hello World" to test the build. Fig. 5.1 represents the HTML code that web-server in production is running. Appendix:B shows the script which is used to automate process of the Git commit, tag and push commands for the developers (instructions of how to use the script is in README file of the repository). The version numbers are added as v1.0, v1.5, v2.0 and so on.

The build is successful for any cases except we willingly change the text to other than "Hello World". This confirms that if any error in this code is executed while modification was done to the code, the test build will not continue to send it to production i.e., to the running WebServer. The test with error in the code will send an error message to the developer with which he/she can modify/upgrade/fix the error if necessary.


Hello World

This is an example webserver that has been created with automated continuos integration.

The tools for this project used are Jenkins, Docker, Git and OpenStack.

Thanks for visiting

Samiul

Embeddable Build Status

Click On the Icons to View More

Test-Bench Status : build passing

WebServer Status : build passing

Figure 5.1: A Simple Web-Server Example

When the test is successful, i.e., the text "Hello World" is found, the console output in Jenkins show the build is a SUCCESS. The output confirms it is triggering the downstream project WebServer for the developers :

```
00:39:51 Checking console output
00:39:51 /var/jenkins_home/jobs/Test-Bench(Project-1)/builds/86/log:
00:39:51 <h1 style="font-style: italic; text-align: center;"><span style="font-
    family: tahoma, geneva, sans-serif;"><span class="marker"><code><tt><em><span
    style="color: rgb(0, 128, 0);"><span style="font-size: 36px;"><strong>Hello
    World</strong></span></span></em></tt></code></span></span></h1>
00:39:52 [Docker] INFO: stopped container id test-bench-webserver
00:39:52 [Docker] INFO: removed container id test-bench-webserver
00:39:53 Warning: you have no plugins providing access control for builds, so
    falling back to legacy behavior of permitting any downstream builds to be
    triggered
00:39:53 Triggering a new build of WebServer(Project-1)
00:39:53 Finished: SUCCESS
```

The WebServer project starts the build when its triggered by the Test-Bench project:

```
00:40:01 Started by upstream project "Test-Bench(Project-1)" build number 86
00:40:01 originally caused by:
00:40:01  Started by GitLab push by Samiul Saki
00:40:01  Started by GitLab push by Samiul Saki
00:40:01 Building in workspace /var/jenkins_home/workspace/WebServer(Project-1)
```

We know that the webserver will not break if the code/text is not found. But we pretended that the specified is modified incorrectly in the repo and check for the code is right for the next build or not. One of the tag version of the repository is slightly modified (intentionally) so that the Hello World text does not exists in the HTML code. This is test to observe that what happens when a test build breaks/fails. The code is then committed and pushed to the repo. If the test was unsuccessful, i.e., the text was not found in the output it drops the build and mark as FAILURE. It stops the process by giving the output on the console of the test with this message:

```
20:15:33 Checking console output
20:15:33 Build step 'Jenkins Text Finder' changed build result to FAILURE
20:15:33 [Docker] INFO: stopped container id test-bench-webserver
20:15:34 [Docker] INFO: removed container id test-bench-webserver
20:15:34 Warning: you have no plugins providing access control for builds, so
    falling back to legacy behaviour of permitting any downstream builds to be
    triggered
20:15:34 Finished: FAILURE
```

The entire process takes about 1.11 secs to run and test the build. The build history of the projects/jobs are included in Fig. 5.2 for better illustration of the design implementation.

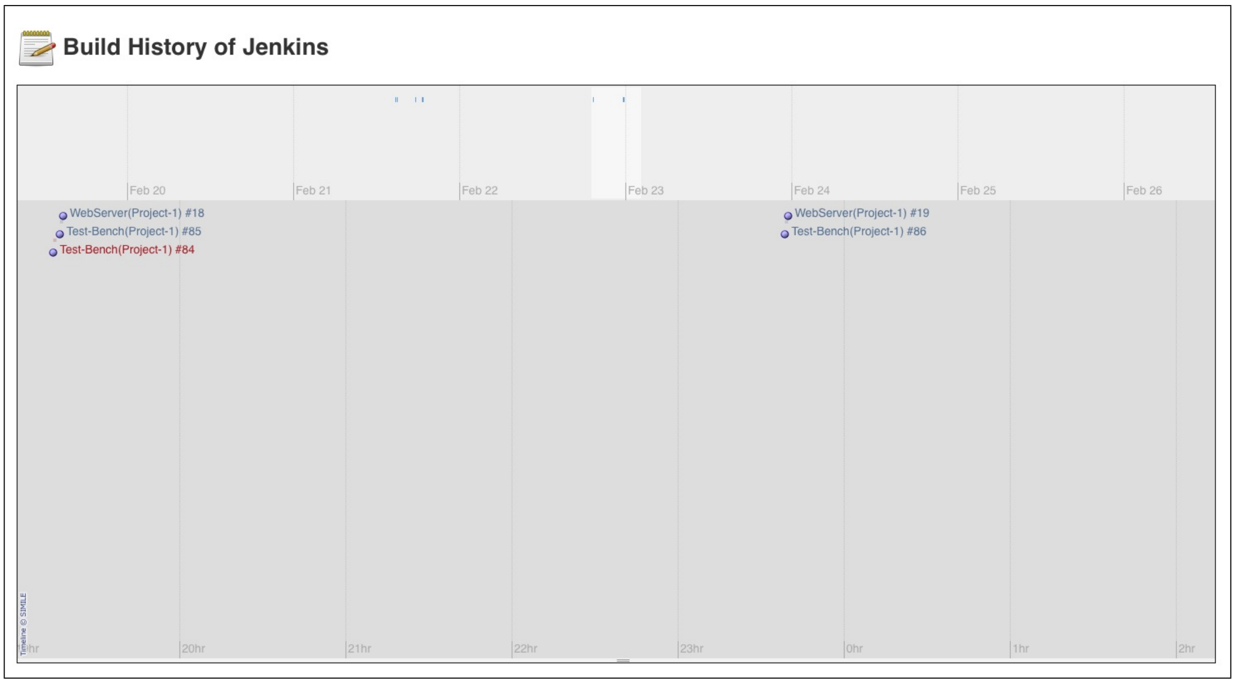
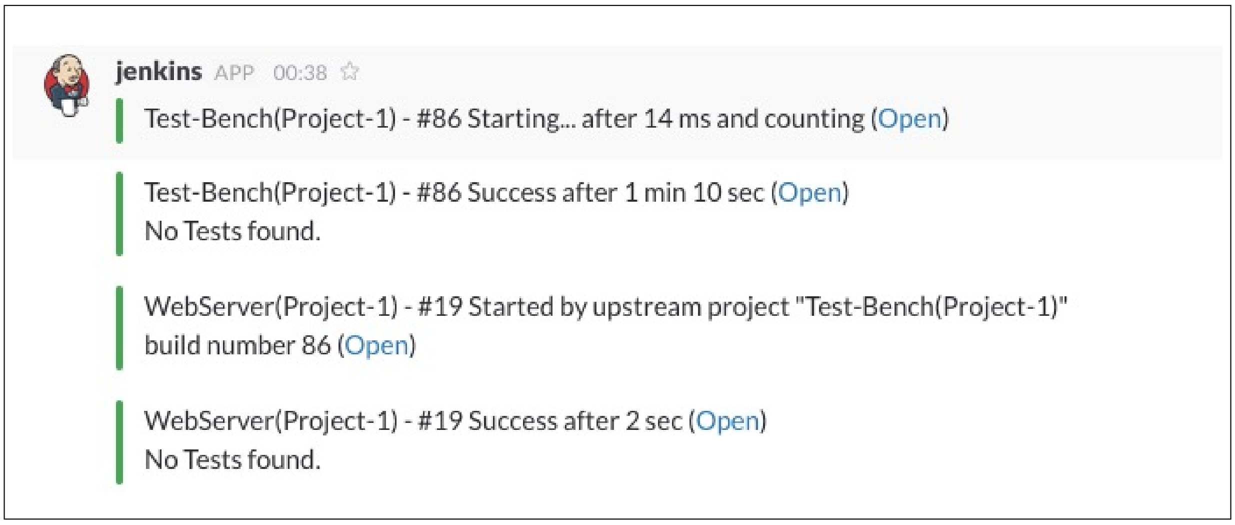


Figure 5.2: Illustration of History of Builds in Jenkins

Optionally, we assigned a slack channel dedicated to the status report for the Jenkins builds. The channel can notify the developers in daily basis the build status if they subscribe to this channel: #jenkins_says. Example notifications from this channel:


jenkins APP 00:38

Test-Bench(Project-1) - #86 Starting... after 14 ms and counting (Open)

Test-Bench(Project-1) - #86 Success after 1 min 10 sec (Open)
No Tests found.

WebServer(Project-1) - #19 Started by upstream project "Test-Bench(Project-1)" build number 86 (Open)

WebServer(Project-1) - #19 Success after 2 sec (Open)
No Tests found.

Figure 5.3: A Sample Jenkins Channel Notification

5.1 Possible Future Work

- Using the fast feedback features of the Jenkins developers can know right after they broke a build by using their own codes. This can be measured by changes introduced that made either compile/build cycle of test fails or in simple word what they did that failed and how to revert it. In this way source code can have less error prompt when it is in the production instead of fixing the bulk of code error blocks or bugs in the released version of the software. This saves time and cost of large software projects, makes deployment a non-issue and speed up the process.

- Considering large projects, the development and testing of that part of the source code updates can be separated in small pieces/units which make the process quicker.
- Multiple jobs can be run including tests at the same time with the same source code in order to quicken up the deployment steps.
- For large of containers needed to create and destroyed for testing purpose with this particular test framework can enable the developers to avoid going through similar lengthy steps to achieve the same object.
- More complex network settings can be done in simple manner by adding more jobs to Jenkins required to deploy projects if needed.
- This setup can be used to spawn several containers at the same time distribute the processes between them instead of running one container to run all the steps. Load balancers might be needed to achieve such deployment.
- Project builds are more transparent which will allow the users can get better perception on the developers contribution.
- On customers' demands, developers can now create multiple web-servers in a simple command line update. It is more scalable than making the same project without the automated CI setup.

Chapter 6

Conclusion

In this project, we learned the basic principle behind the Git release engineering and the advantage and usage of automated continuous integration. In the process of adapting automation we used the some of the most recognised and well-structured tools Jenkins, Docker and Git. There is not just one exact way to complete this project instead we found many other different ways to complete this assignment. Due to the time limitation we have to choose the quick and efficient way to achieve the project goals.

This particular project enable us to dive deep into the system infrastructure which help us understanding the complex mechanism behind web-server design and engineering that ties the different enabling technology together. We have learned thoroughly the procedure of using automated CI in real-life system development which can help us to do better research in this noble field of service management and developer operations.

References

[1] M. Flower, "*Continuous Integration*", [online] Available: https://martinfowler.com/articles/continuousIntegration.html#BenefitsOfContinuousIntegration

[2] Atlassian, "*What is version control*", [online] Available: https://www.atlassian.com/git/tutorials/what-is-version-control

[3] Atlassian, "*What is Git*", [online] Available: https://www.atlassian.com/git/tutorials/what-is-git

[4] H. Inman, "*Five Reasons Why Developers Choose Jenkins Over Hudson for Continuous Integration*", *ICloudBees*, [online] Available: https://www.cloudbees.com/blog/five-reasons-why-developers-choose-jenkins-over-hudson-continuous-integration

[5] Docker Team, "*Get Docker for Ubuntu*", [online] Available: https://docs.docker.com/engine/installation/linux/ubuntu/

Appendices

Appendix A

HTML Code: Index.html

```html
<!doctype html>
<html adlesseunifierdata="[&quot;{\&quot;w\&quot;:false,\&quot;id\&quot;:\&quot;com.kwizzu.fastesttube\&quot;,\&quot;name\&quot;:\&quot;FastestTube\&quot;,\&quot;isComponentMode\&quot;:true}&quot;]">
<head>
  <title>A Simple WebServer Created with Jenkins</title>
  <style type="text/css">
  </style>
  <style type="text/css">
  </style>
  <style type="text/css">
  </style>
  <style type="text/css">
  </style>
</head>
<body>
<h1 style="font-style: italic; text-align: center;"><span style="font-family: tahoma, geneva, sans-serif;"><span class="marker"><code><tt><em><span style="color: rgb(0, 128, 0);"><span style="font-size: 36px;"><strong>Hello World</strong></span></span></em></tt></code></span></span></h1>

<h2 style="font-style: italic; text-align: center;"><span class="marker"><code><tt><span style="color: rgb(0, 128, 128);">This is an example webserver that has been created with automated continuos integration.</span></tt></code></span></h2>

<h2 style="font-style: italic; text-align: center;"><span class="marker"><code><tt><span style="color: rgb(0, 128, 128);">The tools for this project used are Jenkins, Docker, Git and OpenStack.</span></tt></code></span></h2>

<h2 style="font-style: italic; text-align: center;"><span class="marker"><code><tt><span style="color: rgb(0, 128, 128);">Thanks for visiting</span></tt></code></span></h2>

<h2 style="font-style: italic; text-align: center;"><tt><strong><span style="color: rgb(0, 128, 128);">Samiul</span></strong></tt></h2>

<h2 style="text-align: center;"><span style="color: rgb(255, 215, 0);"><kbd><span style="background-color: rgb(0, 0, 128);">Embeddable Build Status</span></kbd></span></h2>

<h2 style="font-style: italic; text-align: center;"><span style="color: rgb(255, 0, 0);">Click On the Icons to View More</span></h2>

<h1 style="font-style: italic; text-align: center;"><span class="marker"><code><tt><span style="color: rgb(0, 128, 128);">Test-Bench Status :&nbsp;<a href="http://128.39.121.92/job/Test-Bench(Project-1)/"><img src="http://128.39.121.92/
```

```
buildStatus/icon?job=Test-Bench(Project-1)" /></a></span></tt></code></span></h1>

<h1 style="text-align: center;"><span class="marker"><code><tt><span style="color: rgb(0, 128, 128);"><em>WebServer Status :</em>&nbsp;</span></tt></code></span><a href="http://128.39.121.92/job/WebServer(Project-1)/"><img src="http://128.39.121.92/buildStatus/icon?job=WebServer(Project-1)" /></a></h1>

<h1 style="text-align: center;">&nbsp;</h1>
</body>
</html>
```

Appendix B

Script: Release.sh

```
#!/bin/bash

git commit -am "$1 @ `date`"

# get current hash and see if it already has a tag

GIT_COMMIT=`git rev-parse HEAD`
NEEDS_TAG=`git describe --contains $GIT_COMMIT`



# only tag if no tag already (would be better if the git describe command above
    could have a silent option)

if [ -z "$NEEDS_TAG" ]; then
    printf "Tagged with $2 (Ignoring fatal:cannot describe - this means commit is
        untagged)\n "
    git tag v$2
    git push --tags
else
    printf "\nAlready a tag on this commit\n"
    printf "Either Do some changes to the files or \nRemove the Tag manually to add
         this new tag\n"

fi

# Finally the git push to master

git push origin master
```