

DevOps Project Report

G.Sravya - IMT2015014
M.Pranith Reddy - IMT2015025
M.Karthikeya Reddy - IMT2015026
Sowmya Dasari - IMT2015507

Abstract:

The aim of the project is to use the devops concepts and tools to automate each stage in software development lifecycle such as Build, Integration, Test, Deployment and Monitoring. We have to configure the various devops tools for each stage to automate the process in each stage. We describe how jenkins is used to connect all these tools along with the description of each tool.

Introduction:

Why DevOps?

DevOps helps create and improve products at faster pace. If the development team and operations team are separate it is difficult to say that the application is ready for operation. For a e-commerce website to run successfully in the competing world it needs to speed up its services. This is possible by automating the process of development, integration, testing, deployment and monitoring. So we need DevOps tools to automate our application.

About the application:

The application is a prototype of an e-commerce website, flipkart. There are three types of users in the application, buyer, seller and an admin. A seller can add some products to the website to sell them. A buyer can view all the items added by the sellers and have an option to buy them. Admin have the privilege to add the categories and subcategories. The Homepage Ui of the ecommerce website can seen in Figure 1.

The technology stack for development:

- Front-end technologies - HTML, CSS, Bootstrap and jquery
- Backend technologies - Java, Jersey, Hibernate
- Database - MySQL

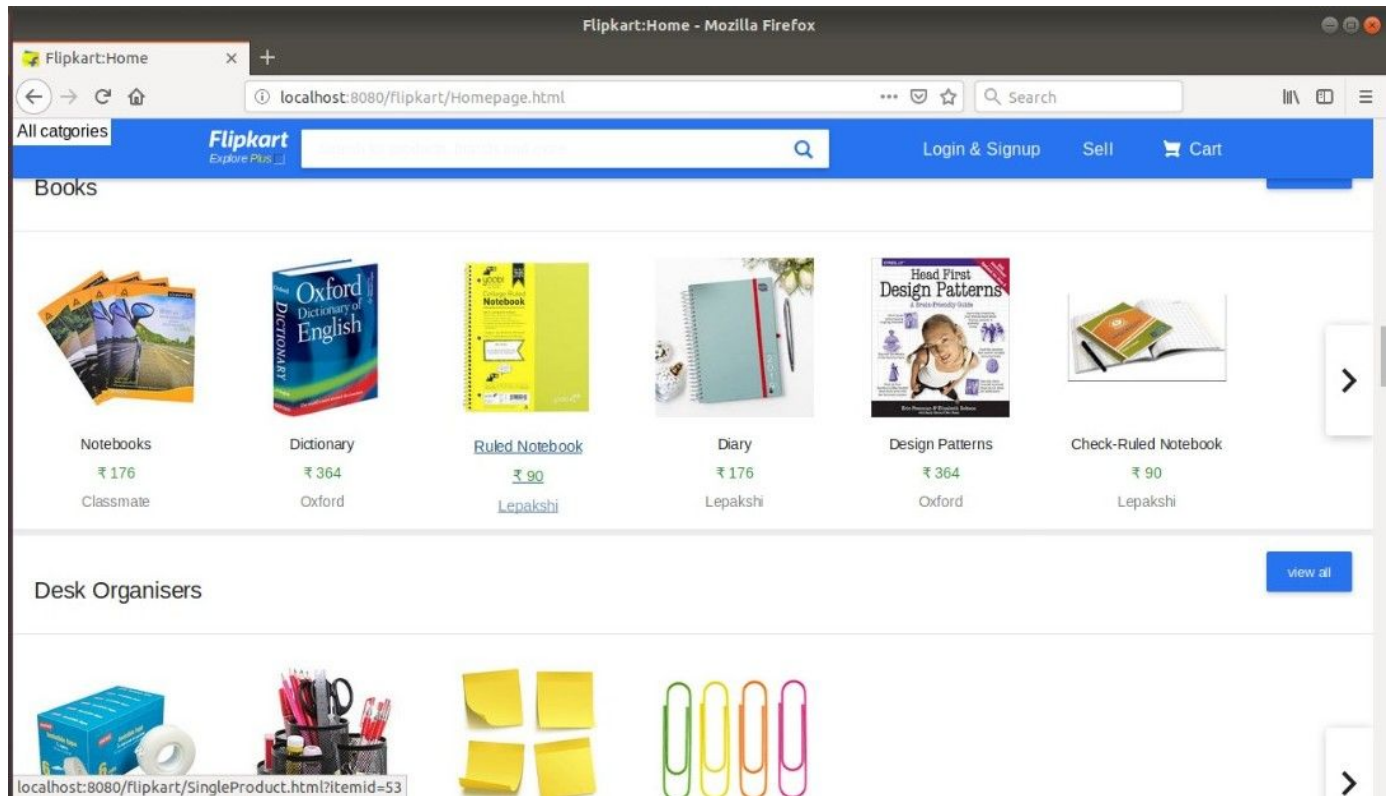


Figure 1 : Homepage of our flipkart prototype

Software Development Life Cycle:

Scope of the project:

The project uses various devops tools to automate every stage in software development life cycle of a web application such as Build, Test, Deploy and Monitor. Whenever any new commit is done to github, Build is triggered automatically followed by all other stages. The tools used in this project are Git, Maven, JUnit, Selenium, Jenkins, Rundeck and ELK stack. The application we have built is a ecommerce website.

Project Architecture:

The application architecture consists of a frontend, backend and a database as shown in the Figure 2. A request from client ui is sent to the backend using ajax calls, then respective api is called in the backend and required data is fetched from the database and the computed response is sent back to the frontend.

For example, when a user requests for login, he has to enter his email and password in the input and when he clicks on login, a request from front is sent to the backend to check if the

credentials of the user are valid. The userdetails from the database are fetched in the backend and checked if the the details given by user are valid.

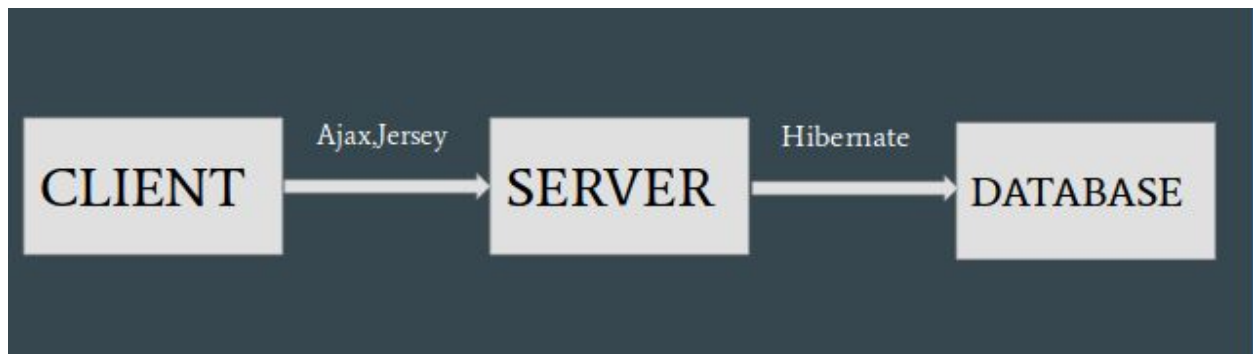


Figure 2 : Application Architecture

Workflow:

- There are three types of users in the application buyer, seller and admin. A seller creates an account for himself and adds items for the buyers to purchase.
- A buyer can see all the items available on the website and based on his purpose he can buy them. To buy an item he needs to register and login into his account.
- A buyer has an option to select a category and subcategory from the available categories and subcategories options. Then select an item of his choice from the list of the items available.
- We get an item description of the item selected. Then one can select from 'buy now' or 'add to cart' options.
- If he selects the 'buy now' option, he will be directed to the buying portal. In this page he has to add the address of delivery or select the existing address. If the buyer hasn't yet logged in he will get the option to login also in this page. Then select the quantity of the item and any offer, if it is under any offer.
- Then buyer has to select his account and enter the pin to pay money for the item. After the money is paid, the request is sent to the respective seller of the item. The items gets added in the 'My orders' of the buyer's profile page.
- If we select the 'add to cart' option in the item description page the item gets added to cart. If the buyer wants to buy multiple items he can add all of them to the cart.
- He can then go to the 'cart' option available on the bar top of the page. In the cart, he can see all the items he has added to the cart and click on 'buy now' and it directs to the buying portal and then buy the items same as described for buying the single item.

- He can also add items to the wishlist, from where he can then add it to the cart and then direct to the buying portal and purchase the item.
- Now after paying money for the item and seller receiving the request for the item, the seller changes the status of the item from 'payed' to 'shipped' once the item is shipped.
- Once item is received by the buyer he changes the status of the item in 'My orders' to 'received'.
- Admin has the option to add various categories, subcategories, colors and brands. He is the one who adds deals and then seller's can add their items to these deals as per their wish.

There are four stages in software development life cycle of the project. They are Build, Test, Deploy and Monitor.

SCM:

- The source code management tool used for the project is **Git**. We chose git as source code management tool because it allows multiple people to work on the same project collaboratively.
- Git SCM poll is used to connect the Build job in jenkins for every 1 minute to the github and if any new commit is seen, then the Build job is triggered automatically.
- The github link for the project code is :
https://github.com/gangishettysravva/devops_project.git

Build Tool:

- The tool used for Building the project is **APACHE MAVEN**.
- Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.
- Maven uses Convention over Configuration, which means developers are not required to create build process themselves.
- The command 'mvn install' is used to build the project. The command 'mvn install -DskipTests' is used to just build the maven project without executing any test cases.

- Maven installs all the project dependencies that are specified in pom.xml including plugins such as JSON, MySQL driver, Hibernate..etc which are needed to develop this project.

Test Tools:

- **JUnit** is used as a testing framework for this project.
- As the project is being developed in java environment, JUnit is a unit testing framework for Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks collectively known as xUnit, that originated with JUnit.
- The test cases are written in **test/java/FirstTestCase** class of src code of project, each test function can be used to test the functionality of the services that are written. If the data returned from the service to the test function is same as the given expected data then we can say that the test is successful. Example,
- testGetUserByEmailBuyer(), testgetSubCategoryById() etc as shown in Figure 3.

```
@Test
public void testgetSubCategoryById() {
    CategoryServices category = new CategoryServices();
    assertEquals("Clothing",category.getSubCategoryById(1).getName());
}
```

Figure 3 : JUnit Test Case Example

- These test cases need junit dependency in pom.xml as in Figure 4.

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.13-beta-1</version>
</dependency>
```

Figure 4 : JUnit Dependency in POM

- **Selenium** is used as web-based automation testing tool for this project.
- We used selenium's WebDriver as it provides a programming interface to create and execute test cases. Test scripts are written in order to identify web elements on web pages and then desired actions are performed on those elements.

- Since, WebDriver directly calls the methods of different browsers hence we have separate driver for each browser. For this project purpose, we used chrome WebDriver.
- We used WebDriverManager to automatically download the driver's binary files (.exe files) for Web Automation.
- The test cases are written test/java/AppTest class of src code of project. Each of these test cases are to test whether the functionality of certain buttons click functions, login, registration etc are as expected. Example, valid_UserCredential(), testSell() etc as shown in Figure 5.

```
@Test
public void valid UserCredential() throws InterruptedException {
    System.out.println("Starting test " + new Object(){}.getClass().getEnclosingMethod().getName());
    driver.get("http://localhost:8081/flipkart-prototype/Homepage.html");
    WebElement element = driver.findElement(By.id("idloginbutton"));

    ((JavascriptExecutor)driver).executeScript("arguments[0].click();", element);
    driver.findElement(By.id("email_or_mobile")).sendKeys("G.Sravya@iiitb.org");
    driver.findElement(By.id("password")).sendKeys("12345");
    WebElement element1 = driver.findElement(By.id("login_btn1"));

    ((JavascriptExecutor)driver).executeScript("arguments[0].click();", element1);
    System.out.println("Ending test " + new Object(){}.getClass().getEnclosingMethod().getName());
}
```

Figure 5 : Selenium Test Case Example

- These test cases need selenium, chromedriver, webdrivermanager dependencies in the pom.xml as in Figure 6.

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.141.59</version>
</dependency>

<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-chrome-driver</artifactId>
    <version>3.141.59</version>
</dependency>
<dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>webdrivermanager</artifactId>
    <version>3.4.0</version>
</dependency>
```

Figure 6 : Selenium Dependencies in POM

- On executing these unit test cases '.txt' and '.xml' files get created in surefire reports folder of target in jenkins workspace. We can observe this in Figure 7:

Workspace of Test on master



Figure 7 : surefire-reports Folder

Artifact:

- A war file is generated after the the project is Built successfully and Tested without any errors.
- Then a **docker** image is built which contains tomcat server and the war file generated from building the project will be copied to the same image. A dockerfile is written to perform this tasks, It contains the following commands.

```
FROM tomcat
COPY target/flipkart-prototype.war /usr/local/tomcat/webapps/flipkart-prototype.war
```

- The following command is used to built the docker image with the specifications in the dockerfile

```
docker build -t image_name -f Dockerfile .
```

- Similarly a docker image for the database is created using the following commands written in other dockerfile.

```
FROM mysql:5.7
COPY schema.sql /docker-entrypoint-initdb.d
```

- Where schema.sql is the sql schema file for the application database. Similarly the docker image for the database of the application is also built using the docker build command.

Deploy:

- **Rundeck** is used for Deploying the Flipkart-prototype Application.
- RunDeck is cross-platform open source software that helps you automate ad-hoc and routine procedures in set of nodes or cloud environments.
- We used rundeck to configure nodes such as systems and to deploy our applications in it.
- To deploy the application on the node we created a job '**Deploy the Services**', it pulls the docker_pull.sh and docker-compose files from github. Below Figure 8 shows the docker_pull.sh file.

```
1  #!/bin/sh
2  # This is a comment!
3  export PATH=/bin:/usr/bin:/usr/local/bin
4  docker pull gangishettysravva/devops_project:webimg
5  docker pull gangishettysravva/devops_project:dbimg
6  docker-compose up -d
```

Figure 8 : docker_pull.sh

- The docker_pull script file contains the docker commands to pull the images from the Docker hub and docker-compose file is used to deploy those pulled images in the node.

```
1  version: '3'
2
3  services:
4    db:
5      image: gangishettysravva/devops_project:dbimg
6      restart: always
7      ports:
8        - "3363:3306"
9      hostname: db
10     environment:
11       MYSQL_ROOT_PASSWORD: 1234
12       MYSQL_DATABASE: flipkartdb
13       MYSQL_USER: root
14
15     webapi:
16       depends_on:
17         - db
18       image: gangishettysravva/devops_project:webimg
19       hostname: webapi
20       ports:
21         - "8081:8080"
22       restart: always
```

Figure 9 : Docker-Compose file

Monitor:

- **ELK** stack is used for monitoring the data in the database of the application.
- ELK is acronym for Elasticsearch, Logstash and Kibana.
- A config file is given as an input to Logstash which fetches the data and send the data to elasticsearch where the data is indexed. Later, this data can be visualized in kibana.
- We used logstash to get data from the database for every one minute from various tables and from the data obtained we made some visualizations in kibana.
- The following is the command which in run in logstash/bin folder to give the conf file as input to logstash

logstash -f /path/to/the/conf-file.

- One of the .conf file written to fetch the data from a table of database is as below in Figure 10:

```
input
{
  jdbc {
    jdbc_driver_library => "/home/sravva/software/mysql-connector-java-5.1.47.jar"
    jdbc_driver_class => "com.mysql.jdbc.Driver"
    jdbc_connection_string => "jdbc:mysql://172.16.129.42:3363/flipkartdb"
    jdbc_user => "root"
    jdbc_password => "1234"
    schedule => "*/1 * * * *"
    statement => "SELECT * from Item"
    type => "item"
  }
}
output
{
  elasticsearch {
    hosts => ["http://localhost:9200"]
    index => "%{type}"
    document_id => "%{item_id}"
  }
  stdout {
    codec => dots
  }
}
```

Figure 10 : item.conf File

- The above .conf file is given as input to logstash which uses it to connect to the 'flipkartdb' database present on a docker running on a host through jdbc driver with the given user and password, executes the query given as statement in the conf file (in this

example select * from Item) and sends the data to elasticsearch runs on the port 9200 of localhost and the index is given as the table name.

- Similarly .conf for other tables are also written and all the files together are given as input to logstash by giving the path of folder containing all these files to logstash by following command:

```
logstash -f /path/to/the/folder
```

- Elasticsearch and kibana will should be running before the previous command is executed.
 - To run elasticsearch : Go the directory of elasticsearch and run the following command

```
bin/elasticsearch
```

- To run kibana: Go the directory of kibana and run the following command

```
bin/kibana
```

- Elasticsearch will be running on the port 9200 and kibana on port 5601.
- The following Figure 11 shows the data in kibana after a new index pattern with the given table name is added in kibana.

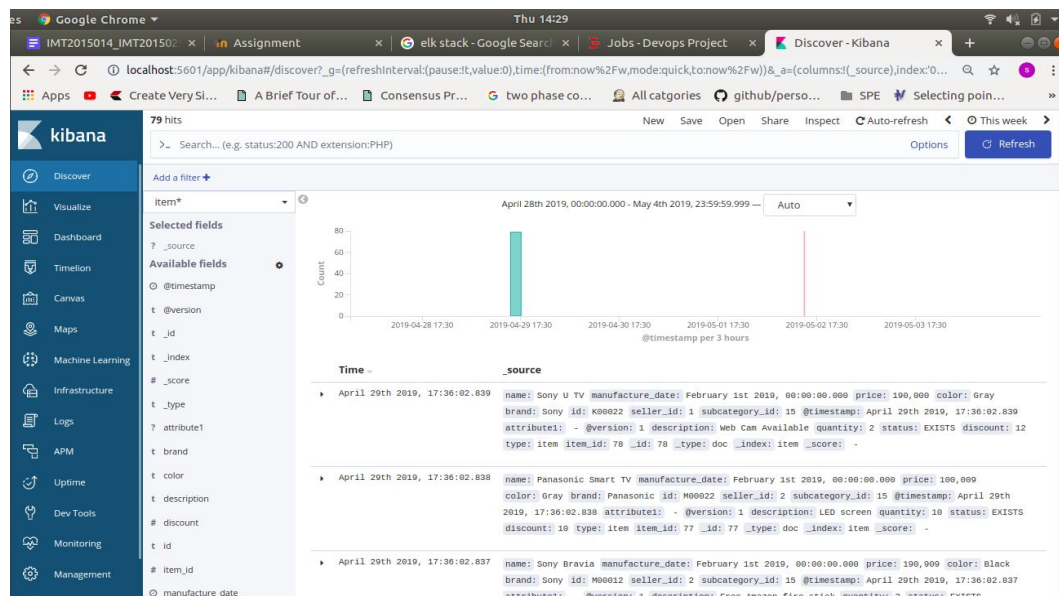


Figure 11 : Kibana Discover Tab showing the item data

- Then visualization such a pie chart, barchart etc can be created from the data obtained from elasticsearch using the visualisation tab.
- One of the following is an example of visualization we have created to illustrate the number of items in each subcategory is shown in the Figure 12.

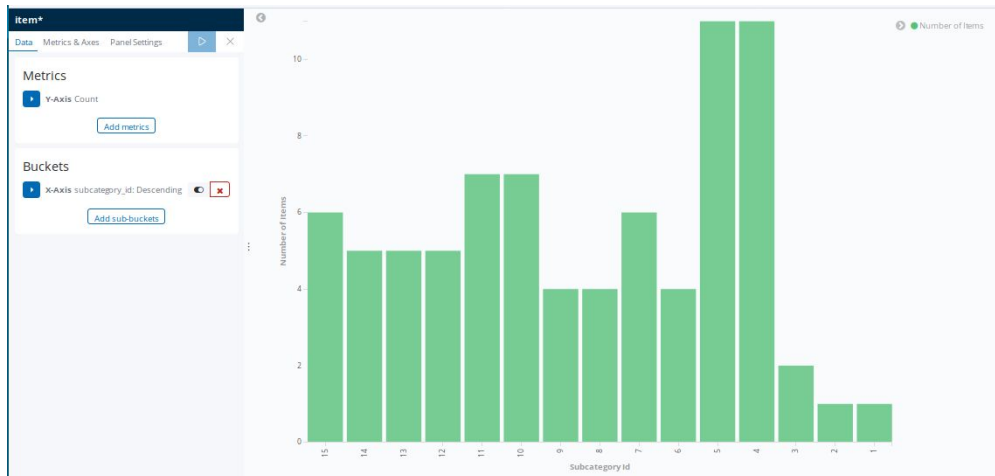


Figure 12 : Barchart Representing number of Items in each subcategory

- We also created some more visualizations using kibana which were pie charts representing number of items in each subcategory sold by each seller and other representing number of subcategories in each category. We then created a dashboard in kibana and added these visualization to it. dashboard created can be seen in Figure 13.

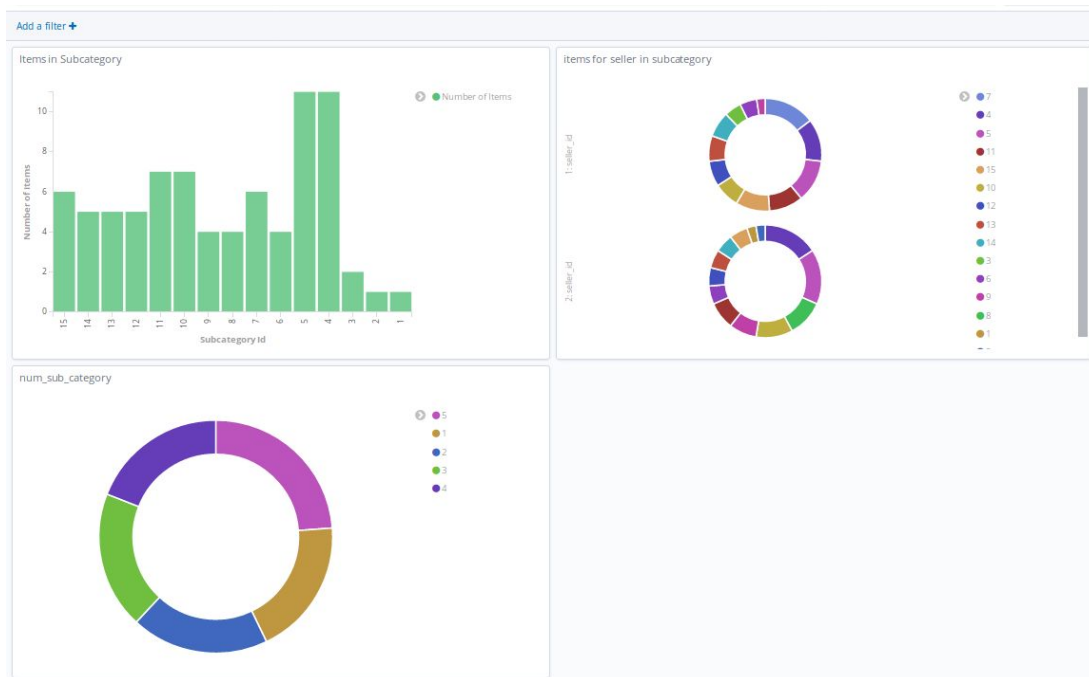


Figure 13 : Dashboard in Kibana

Continuous Integration Pipeline

Continuous Integration: Continuous Integration is a DevOps software development method where the developers merge their code changes into a central repository after which automated builds and tests are run.

Jenkins is used for Continuous Integration of the project.

- **Jenkins** is an open source automation server written in Java.
- Jenkins helps to automate the non-human part of the software development process, with continuous integration and facilitating technical aspects of continuous delivery.
- If Jenkins detects any changes in the source code, it triggers required builds and tests. The results of the builds and tests would be available on Dashboard (Notifications can also be sent).

We created 4 stages in jenkins such as Build, Test, Deploy, Monitor. For each stage a separate job is created in jenkins and respective actions are performed in each of them.

STAGE - BUILD:

- In Build stage the source code of the project from git need to be pulled to jenkins workspace, for this the git should be configured in jenkins.

To configure Git to Jenkins:

- Download Git plugin in Jenkins.
- In source code management tab of configure option of Build job created in jenkins, enter the github repository project url and the github credentials if the repo is private as shown in Figure 14.
- Git SCM poll is used to connect the Build job in jenkins for every 1 minute to the github and if any new commit is seen, then the Build job is triggered automatically.
- In the Build Triggers tab of configure, poll SCM is checked and the schedule is given as H/1 * * * * which checks if there is any new commit in the github for every 1 minute.

Source Code Management

☐ None
☒ Git

Repositories

Repository URL:

Credentials:

Branches to build

Branch Specifier (blank for 'any'):

Repository browser

Figure 14 : SCM in Jenkins Build Config

- After pulling the source code from git, we need to build the project and generate a war file. Here we use Maven to do project build in jenkins.

To configure MAVEN with Jenkins:

- Maven installation path is set in Global Tool Configuration option in Jenkins.
- In the Build tab of Build job config, invoke top level maven target is selected and the goal is set as 'install -DskipTests' as shown in Figure 15 to just build the project and generate the war file without executing the test cases.
- After the build is successful the war file generated is stored in the target directory.

Invoke top-level Maven targets

Maven Version:

Goals:

Figure 15 : Build Tab in Jenkins Build Config

- Now with the generated build files such as war, we created two docker images webimg, dbimg using docker build. webimg consists of application services(backend) and html files, dbimg consists of database related to that application. As these images need to

pushed in to DockerHub, we tagged them with the repository [gangishettysravva/devops_project](#) as shown in Figure 16.



Figure 16 : Commands to build docker images and tag them

- In all other stages such as Test, Deploy where the source code of project is required, instead of cloning the code from git, custom workspace is set for the job. In General tab of the config and advanced options, custom workspace option is checked and the workspace directory path is set to that of workspace directory Build as shown in Figure 17.



Figure 17 : Custom Workspace

- If the build stage is successful then the build for next stage(Test) begins.

STAGE - TEST:

- In the Testing stage, we deploy the application locally in the system(Testing environment) and run maven tests in **workspace of Jenkins** instead of running tests in the deployed **containers** as the containers are built with the artifacts of workspace.
- The Advantages of doing testing in Jenkins workspace instead of container are
 - Required plugins and packages are already installed while building the project.
 - It takes less time for testing in workspace than container.

- This means the size of the container can be decreased as some packages are not required.
- We need to deploy the containers even though we are doing testing in jenkins workspace because the test cases needs to interact with the database. Hence the database container needs to be deployed into the environment.
- But the problem that occurs in doing this way is that, application starts connecting to the container using hostname of container.

```
<property
name="hibernate.connection.url">jdbc:mysql://db:3306/flipkartdb
</property>
```

- Here 'db' is the hostname of the database container that will be running.
- The application won't be able to connect to database this way because there is no bridge between the local host machine and docker containers. But there will be a bridge network in containers, so that the containers can interact with each other but not the machine.
- To solve this problem, we used a dns-proxy-server tool which solves the above problem

DNS-Proxy-Server(DPS)

Solve your DNS hosts from your docker containers, then from your local configuration, then from internet.

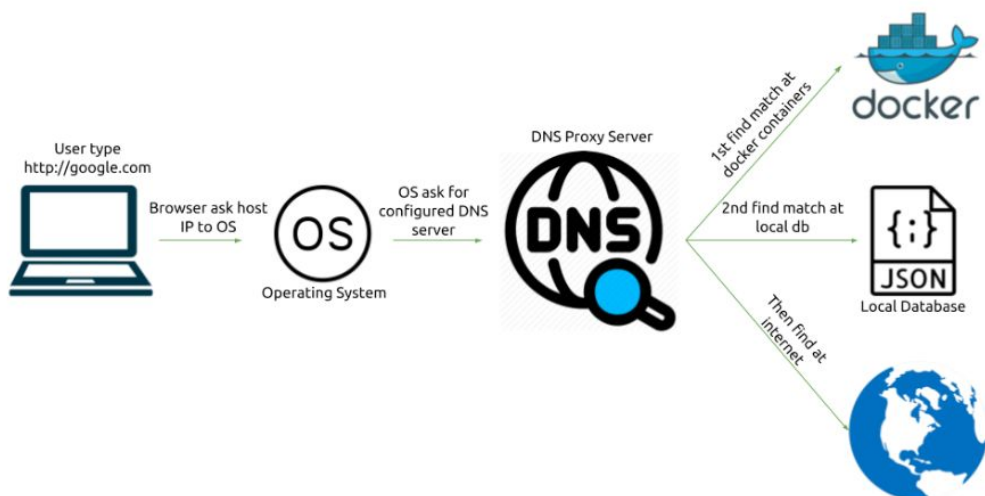


Figure 18 : DNS Proxy Server

- This [dns-proxy](#) can be run in a container using the following command

```
$ docker run --rm --hostname dns.mageddo \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /etc/resolv.conf:/etc/resolv.conf \
defreitas/dns-proxy-server
```

- This tool allows our application to connect to any container using hostname.
 - This way we can solve the problem of connecting to docker container from host machine but for that to happen dns-proxy container needs to be run before deploying the containers for testing.
- After running DNS-proxy and deploying containers the testing stage begins. Initially we download the Junit plugin from Jenkins plugin manager.
- In the configure of Test, in the add build step, we select execute shell to execute commands as given in Figure 19:



Figure 19 : Execute Shell For Test Job in Jenkins

- In the post-build actions, we select Publish JUnit test result report. When Maven runs unit tests in a project, it automatically generates the XML test reports in a directory called surefire-reports in the target directory. So in the Test report XMLs input we mention the path of these xml files as 'target/surefire-reports/*.xml' as shown in Figure 20.

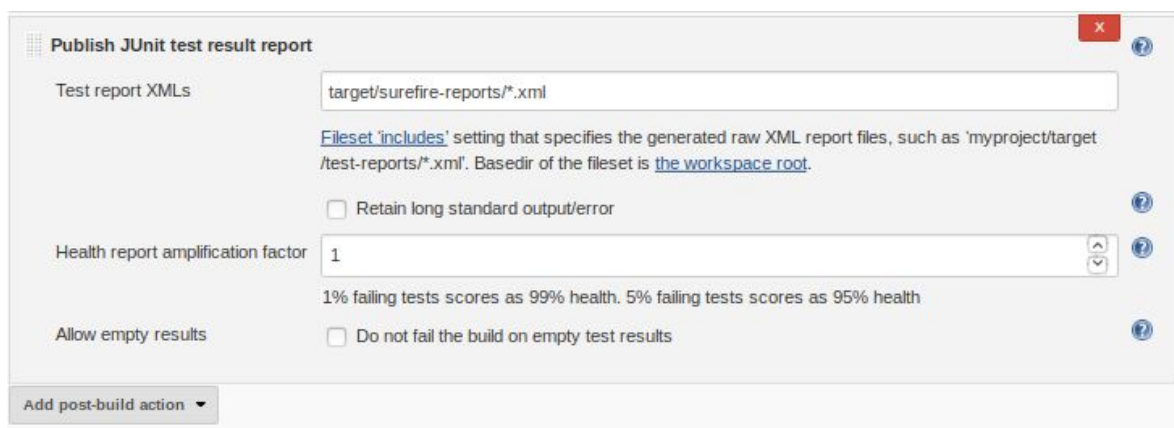


Figure 20 : Publish Test Reports

- After configuring the test, and building the test stage, we get the latest test results option and on clicking on it we get the Test Result in Figure 21:

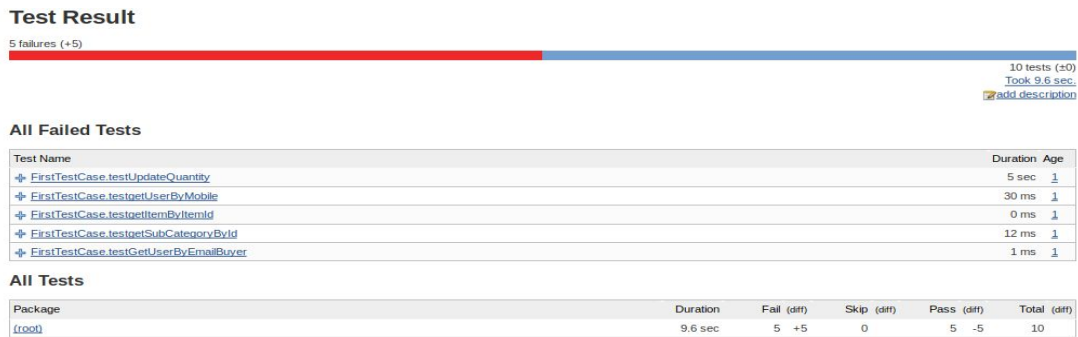


Figure 21 :Test Result

- We can also see the test results trend in jenkins, where we can see the count of test cases passed and failed in the various builds. We can observe this in Figure 22:



Figure 22 : Count vs Build Number

STAGE - DEPLOY:

- In the Deployment stage, the source has passed through all test cases and ready to be deployed into production. So, we need to deploy the created docker images into nodes. In Order to deploy, initially we push the images to [Docker hub](#) and pull them to the respective nodes. To push the docker images we use docker push command as given in Figure 23.



Figure 23 : Commands to push docker images to Docker Hub

- After pushing the images to Docker Hub, we need to integrate Jenkins with Rundeck so that the Rundeck job can deploy the application in nodes.

Configuring Rundeck:

To Deploy the application, we are using Rundeck which is an open-source automated deployment tool. Rundeck needs to be integrated with Jenkins, to do that we need to configure Rundeck first.

- To connect to a node and to run commands in that node, a project needs to be created first in Rundeck. Within the project any number of nodes can be created, here node can be any type of localhost machine or Docker container or VM or any other machine. To connect to a node IP address, hostname, authentication should be specified in an XML format as shown in Figure 24.

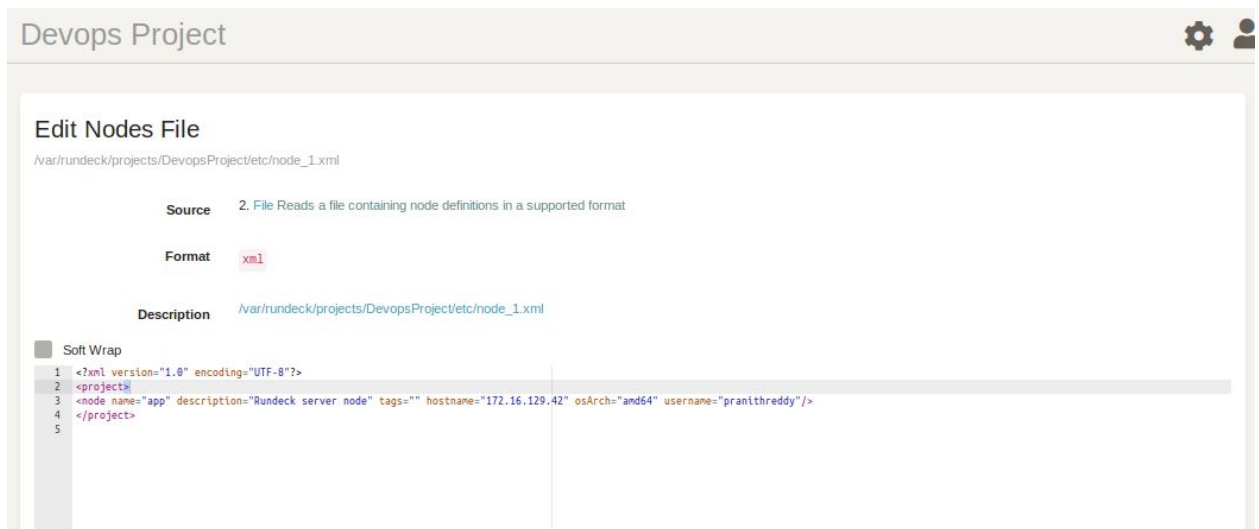


Figure 24 : resource.xml file of the client node

- SSH authentication access should be made from Rundeck server to client node in order to run any commands in it.

Authenticating access to the clients:

- Rundeck uses ssh keys to securely access the client nodes.
- We need to create a ssh key using the following command in Rundeck server.

```
cd /var/lib/rundeck
mkdir -p .ssh
cd .ssh
sudo ssh-keygen
```
- This will create a '**id_rsa.pub**' key and this key needs to be copied to the client node in a file named '**authorized_keys**'.
- This will ensure that the rundeck server is an authorized login. The following are the commands to create and copy the key to file.

```
mkdir -p .ssh
cat >> ~/.ssh/authorized_keys
<Paste the contents of id_rsa.pub file from the rundeck
server>
```

- After node creation, we need to create a job for that node in order to execute any commands in it. To automate the process of deployment we need to specify the commands in the job itself so that there will be no need to manually write the commands for each deployment stages as shown in Figure 25.

The screenshot displays the Rundeck job configuration interface. At the top, under 'Workflow', there are two radio buttons: 'Stop at the failed step.' (selected) and 'Run remaining steps before failing.'. Below these is a 'Strategy:' dropdown menu set to 'Node First'. A note states 'Execute all steps on a node before proceeding to the next node.' with an 'Explain' link. Below the workflow settings is a 'Global Log Filters' section with an '+ add' button. The main area shows a list of four steps, each with a terminal icon, a command, and control buttons (undo, redo, delete, info). The steps are: 1. 'curl -o docker-compose.yml https://raw.githubusercontent.com/gangishettysravya/devops_project/master/docker-compose.yml?token=AHLF7HTGMT52DTCE4FK6DM24...', 2. 'curl -o docker_pull.sh https://raw.githubusercontent.com/gangishettysravya/devops_project/master/docker_pull.sh?token=AHLF7HXP5EKGTYFMLZAGBC24Z2MC6', 3. 'chmod 755 docker_pull.sh', and 4. './docker_pull.sh'. At the bottom is an '+ Add a step' button.

Figure 25 : Commands that are executed in Rundeck Job

- After Creating a job in rundeck a Job Identifier will be created and displayed in job definition page. This identifier can be used to connect Jenkins with Rundeck. The identifier of the Job can be seen in the Figure 26 as UUID.

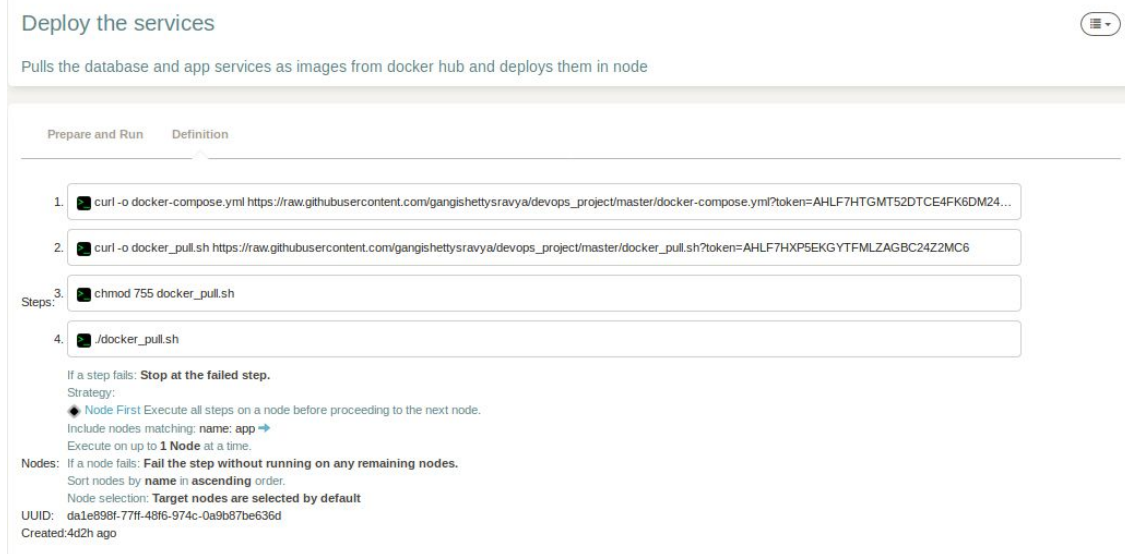


Figure 26 : Rundeck Job

Configuring Jenkins with Rundeck:

- In the Configuration of Jenkins, we need to connect to an instance of rundeck which means rundeck needs to be running at this point of time in the host. To connect to it URL, Username and Password needs to be specified in jenkins configuration as shown in Figure 27.

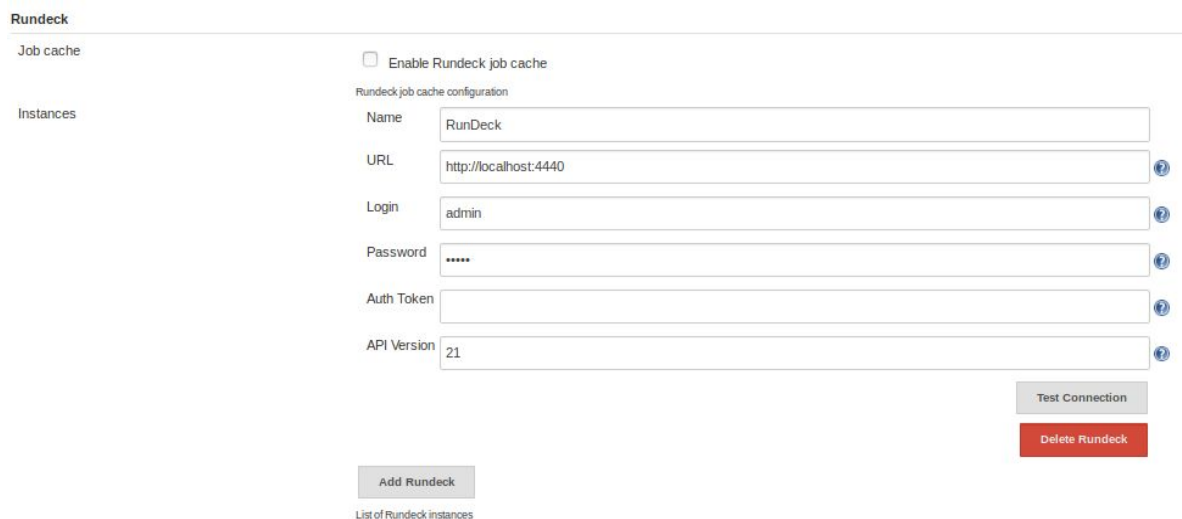
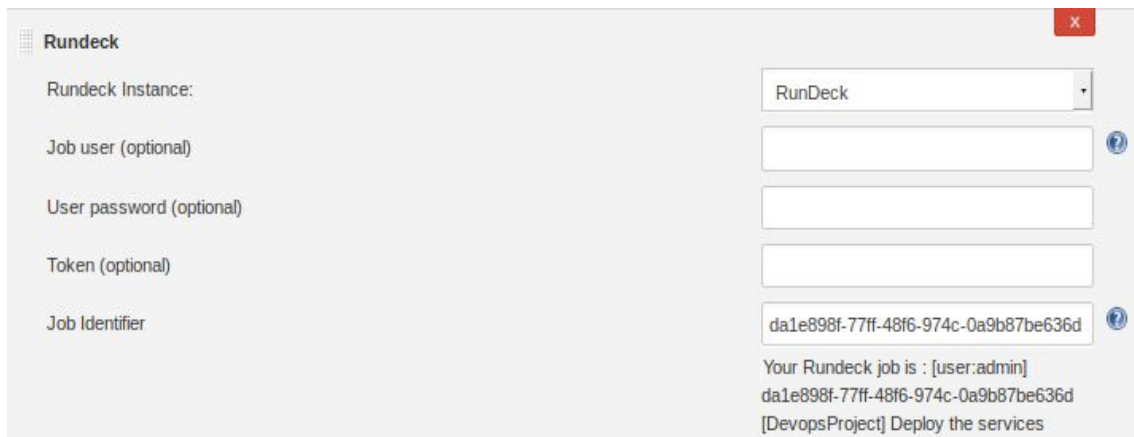


Figure 27 : Rundeck Instance configuration in Jenkins

- Then in the post-build actions of Deployment stage we need to specify rundeck and it's Job Identifier to trigger the rundeck job as shown in Figure 28.



RunDeck

RunDeck Instance: RunDeck

Job user (optional):

User password (optional):

Token (optional):

Job Identifier: da1e898f-77ff-48f6-974c-0a9b87be636d

Your RunDeck job is : [user:admin]
da1e898f-77ff-48f6-974c-0a9b87be636d
[DevopsProject] Deploy the services

Figure 28 : Rundeck Job Trigger from Jenkins

STAGE - MONITOR :

- In Monitoring stage, ELK stack is used where the conf file folder path is given to logstash as an input, and logstash sends the data from the database server to elasticsearch.
- The following command is written in jenkins Monitor job execute shell, to execute the logstash and start fetching the data from database server once the deploy is successful.



Build

Execute shell

Command: `./logstash -f /home/sravva/Logs &`

[See the list of available environment variables](#)

Advanced...

Add build step

Figure 29 : Logstash command in Jenkins execute shell

- After this command is executed, the logstash starts fetching the data from server and sends it to elasticsearch, which then can be visualized using kibana.

Build Pipeline :

The pipeline view in Jenkins after all the stages are completed successfully is shown in Figure 30.

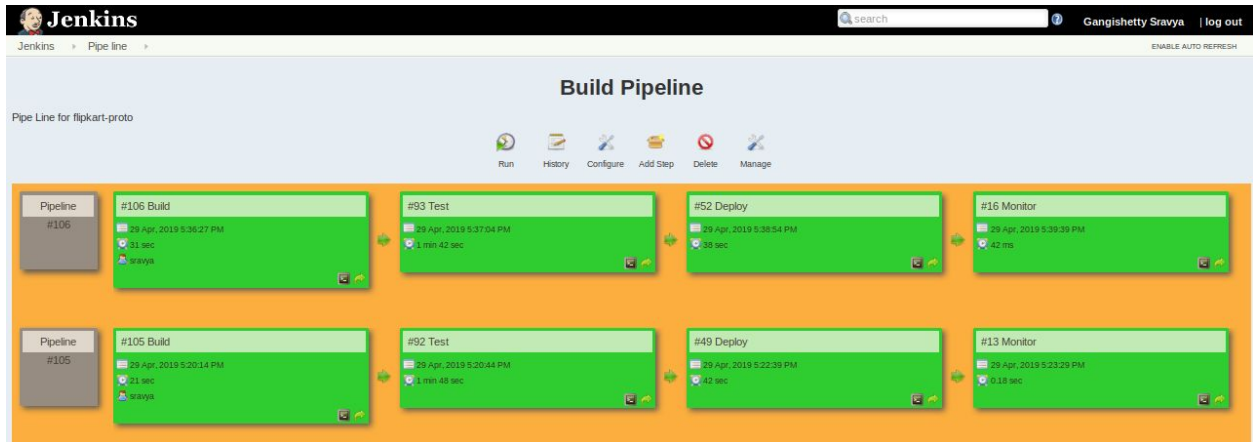


Figure 30 : Pipeline View in Jenkins

Results and Future work:

The average time taken by each stage is given below:

- Build : 30 sec
- Test : 1 min 40 sec
- Deploy : 1 min 50 sec

The size of the docker images are given below:

- webimg - 230 mb (contains services)
- dbimg - 124 mb (contains database)

The application is divided into two containers one is for services and the other for database. Hence if there is any traffic then the number of service nodes can be increased, similarly to increase performance more containers containing databases can be deployed. This way the application becomes more scalable. Presently our system configuration performs continuous monitoring on the database, we can extend the system to perform continuous monitoring for application server health and application server logs.

Conclusion:

In the project we have successfully built a ecommerce website used a tool in every stage of software development lifecycle to automate that stage and this makes addition of a new feature

easier and faster to deploy. We have built a pipeline in Jenkins to connect all the tools used. Using DevOps tools and concepts help in continuous delivery of new features at high speed and reliability.

References:

1. <https://dzone.com/articles/top-5-reasons-why-devops-is-important>
2. <https://shadow-soft.com/why-devops-important/>
3. <https://github.com/mageddo/dns-proxy-server>
4. <https://maven.apache.org/what-is-maven.html>
5. <https://docs.rundeck.com/docs/manual/introduction.html>
6. <https://www.rundeck.com/what-is-rundeck>
7. <https://github.com/CS816SPE/Rundeck-Lab>
8. <https://github.com/Alakazam03/ELK-Tutorial>
9. https://www.seleniumhq.org/docs/01_introducing_selenium.jsp
10. <https://github.com/CS816SPE/maven-setup-instructions>
11. <https://www.tutorialspoint.com/junit/>
12. <https://www.5balloons.info/write-selenium-test-cases-with-junit-maven-and-eclipse/>
13. <https://www.toolsqa.com/selenium-webdriver/first-test-case/>
14. https://www.bogotobogo.com/DevOps/Jenkins/Jenkins_GitHub_Java_Application_Project_Build_Configuration_Maven.php