Pages 1–8
2022

Chess Game with DevOps

CS 816 - Software Production Engineering

Team Members: Bharath Sai, Kousthubh Tadanki
Roll Numbers: IMT2018022, IMT2018048

May 12, 2022

Keywords: Javafx, Maven, GitHub, Docker, Ansible, JUnit Testing, ELK Stack Jenkins

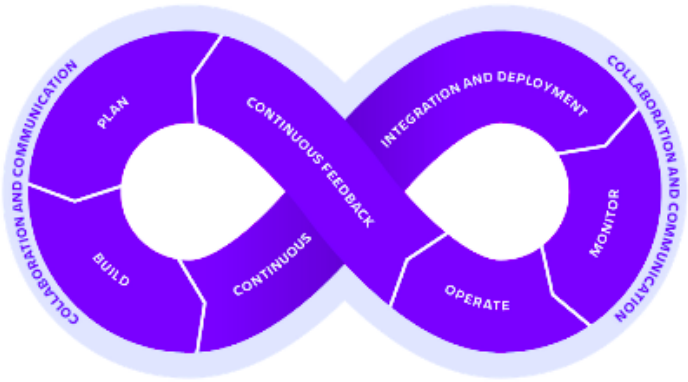
ABSTRACT

The objective of this project is to develop a chess game application using DevOps tools. The learning objectives include understanding and implementing Build, Continuous Integration, Continuous Delivery, and Continuous Monitoring using tools such as Maven, Jenkins, Ansible, ELK Stack.

GitHub / Public View

Docker Hub / Public View

CONTENTS



SDLC

1 INTRODUCTION TO DEVOPS

What is DevOps? DevOps is a set of practices, tools, and a cultural philosophy that automate and integrate the processes between software development and IT teams. It emphasizes team empowerment, cross-team communication and collaboration, and technology automation.

Why DevOps? Here are the top five reasons why the industry has been so quick to adopt DevOps principles:

– Shorter development cycles, faster innovation.

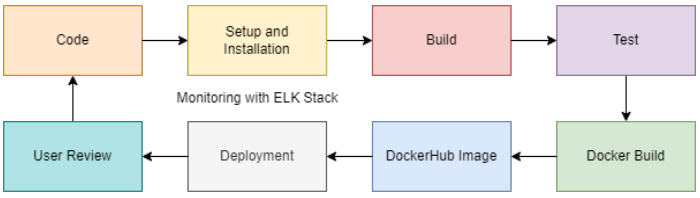
– Reduced deployment failures, rollbacks & time to recover

– improved communication & collaboration

– Increased efficiencies

– Reduced costs & IT headcount

In order to create the chess game application, the following pipeline was used:



Project Pipeline Schematic

2 CHESS GAME APPLICATION OVERVIEW

This is a fully functional Java-based chess game made using the JavaFX/Swing GUI libraries and architected on object-oriented programming concepts. It supports both human vs human(from same machine) and AI vs human with complete rules and extra features. It is built using the following technology stack.

– **Frontend Framework**: Javafx

– **Apart from frontend**: Java

– **Build Tool**: Maven

– **Testing**: JUnit

– **Source Control Management**: GitHub

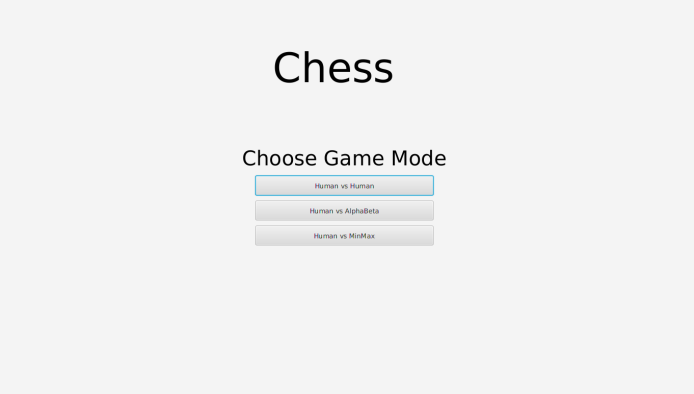
– **Containerization**: Docker

– **Continuous Integration**: Jenkins

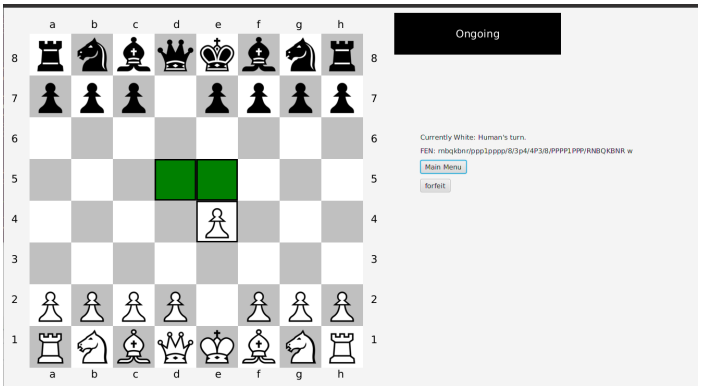
– **Continuous Deployment**: Ansible

– **Continuous Monitoring**: ELK Stack

The **src/main/java** directory contains the entire code for the Chess application. Since we have built every feature from scratch, we haven't used any APIs. So, there is no backend and database for this application as this being a non web application.



SnapShot of Chess Menu



Chess Board with all pieces

3 HOW TO RUN THE APPLICATION

To run the application you must have docker installed on your machine. Please follow the steps in this link according to your environment. Docker Installation.

Once done follow the below steps:

– **Pull image from dockerhub**: docker pull ellanti14/chess:latest

```
bharath@bharath-VirtualBox:~$ xauth list
bharath-VirtualBox/unix:  MIT-MAGIC-COOKIE-1  a8335c1383bd8d9b1b33ec4199f9d5ce
#ffff#626861726174682d5669727475616c426f78#:  MIT-MAGIC-COOKIE-1  a8335c1383bd8d
9b1b33ec4199f9d5ce
bharath@bharath-VirtualBox:~$ docker run -it --net=host -e DISPLAY -v /tmp/.X11-
unix ellanti14/chess bash
root@bharath-VirtualBox:/IdeaProjects/SPE-Chess# xauth add bharath-VirtualBox/un
ix:0  MIT-MAGIC-COOKIE-1  a8335c1383bd8d9b1b33ec4199f9d5ce
xauth:  file /root/.Xauthority does not exist
root@bharath-VirtualBox:/IdeaProjects/SPE-Chess# java --module-path /javafx-sdk-
11.0.2/lib --add-modules javafx.controls,javafx.fxml -jar ./target/SPE-Chess-1.0
-SNAPSHOT-jar-with-dependencies.jar

(java:10): dbind-WARNING **: 09:59:46.316: Couldn't register with accessibility
bus: An AppArmor policy prevents this sender from sending this message to this r
ecipient; type="method_call", sender="(null)" (inactive) interface="org.freedesk
top.DBus" member="Hello" error name="(unset)" requested_reply="0" destination="o
rg.freedesktop.DBus" (bus)
Gtk-Message: 09:59:46.376: Failed to load module "canberra-gtk-module"
Gtk-Message: 09:59:46.383: Failed to load module "canberra-gtk-module"
```

Snapshot of Commands to run the App

– Now run the following command on your local machine and store the output: xauth list

– **Running the docker image**: docker run -it –net=host -e DISPLAY -v /tmp/.X11-unix ellanti14/chess bash.

– **Adding Xauth** : xauth add <Paste the output of xauth list>

– **Running application inside Container**: java –module-path /javafx-sdk-11.0.2/lib –add-modules javafx.controls,javafx.fxml -jar ./target/SPE-Chess-1.0-SNAPSHOT-jar-with-dependencies.jar

4 DEVELOPMENT AND WORKFLOW

This section outlines each tool and how it was used to make the chess game application. This project is built using intelliJ IDE.

4.1 Front-end Framework: Javafx

JavaFX is an open source, next generation client application platform for desktop, mobile and embedded systems built on Java. It is a Java library used to develop Desktop applications as well as Rich Internet Applications (RIA). We used Javafx for building the GUI of chess game i.e pieces, board, updation of game...etc.

Configuration

– To run the jar we need to download a javafx dependency from this site.

– Extract the downloaded zip folder.

4.2 Apart from frontend: Java

Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. We used Java to implement features and rules of chess game like movement of pieces, special moves, AI. We also used well known design patterns like strategy design pattern, template design pattern, factory design pattern...etc.

Configuration

– Install openjdk-11-jdk and then configure $ JAVA_HOME path.

4.3 Develop Source Code- Intellij

Following are the steps to create project.

– Open Intellij and select a new project.

– Select JavaFX from the menu.

– Select build environment as maven.

– Select testing environment as JUnit

– Select project sdk 11 and click next.

– Give a name and click Finish.

Intellij creates a file structure for our project. Go to src/main. Right click on java and select new/package. Our whole code will be residing here in the java folder. We have two folders chessgame and gui. chessgame folder contains all the backend code related to chessgame. GUI folder contains the front end code. In chessgame folder we have subpackages game, pieces, and player. Game package contains code related to playing game. Pieces package contains code related to different pieces. Player package contains code related to human and AI.

4.4 Maven

Maven is a project management tool for software. For example, setting up a java project may need a large number of test suites, dependencies, and procedures. When done using Maven, however, a few lines in a pom.xml file may actually install the dependencies. A configuration file is included with this maven project (pom.xml). This is the fundamental file that Maven utilizes to gather all data. It's then used to 'construct' projects, which are subsequently tested using simple instructions. The build life cycle contains numerous steps, including verify, compile, and so on. The primary components that we will use are build, test, install, and clean.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>SPE-Chess</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>SPE-Chess</name>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <junit.version>5.8.1</junit.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-controls</artifactId>
            <version>11.0.2</version>
        </dependency>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-fxml</artifactId>
            <version>11.0.2</version>
        </dependency>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-web</artifactId>
            <version>11.0.2</version>
        </dependency>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-media</artifactId>
            <version>11.0.2</version>
        </dependency>
```

Part of **pom.xml**

Configuration

– Install maven in our local system and then we also add a maven plugin in IntelliJ.

– **mvn clean** command cleans the workspace, **mvn build** command builds the application, **mvn test** command runs the tests associated with the application.

– **mvn install** command can be used to perform the 2nd & 3rd task collectively.

Walkthrough of pom.xml

For our project we add lot of dependencies related to JavaFx, Junit and Log4j. All these dependencies are added by the Intellij when we configure the project.

4.4.1 Description Maven will go out and download the JAR files for those project dependencies. It reads the config file i.e. pom.xml. It will check the local repository that resides on your machine. If not found it will check the central repository(remote) for those dependencies. Save the file in the local repo. Use those dependencies to build and run the application. Target is the destination directory for compiled code. Automatically created by Maven.

4.4.2 JAR file Conversion and Running

– To convert the entire maven project to jar file: mvn clean install

– The above command should be run in the codebase directory(i.e SPE-Chess directory) where **pom.xml** file lies which does both build test. This will create a jar file with all the dependencies in the target folder.

– Make sure you have the javafx dependency that you have downloaded in section 4.1 in the project folder.

– **To run the jar file:** java –module-path /home/bharath/IdeaProjects/SPE-Chess/javafx-sdk-11.0.2/lib –add-modules javafx.controls,javafx.fxml -jar target/SPE-Chess-1.0-SNAPSHOT-jar-with-dependencies.jar

4.4.3 Log files- Log4j2

– Go to src/main/resources create log4j2.xml file.

– We have 3 significant tags i.e loggers, appenders, patternlayout.

– Logger is responsible for taking logging info. (stored in namespace hierarchy)

– Appender is responsible for publishing logging info to a variety of destinations, including a file, database, console outputs. . . etc.

– Layout is responsible for providing a format for the logging info.

4.5 Source Control Management: Git and GitHub

Source control (or version control) is the practice of tracking and managing changes to code. Source control management (SCM) systems provide a running history of code development and help to resolve conflicts when merging contributions from multiple sources. [4]

The chess game application uses Git and GitHub as source control management. Git is a distributed version control system, it is a tool to manage project source code history. Git is one of the most widely-used popular version control systems in use today.

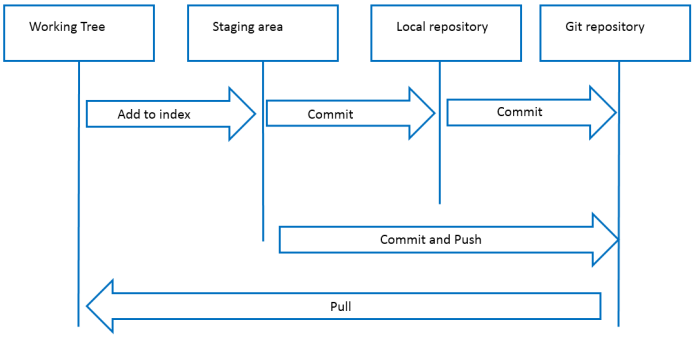
Github is a provider of internet hosting for software development and version control using Git. It offers distributed version control and source code management functionality of Git, plus its own feature.

Configuration

Install Git on your local machine and configure the global credentials in the local machine. Signup on github page and create a repository.Enter repo name and click create. In Intellij go to setting/Version Control/GitHub and click +. Log via Github or token.

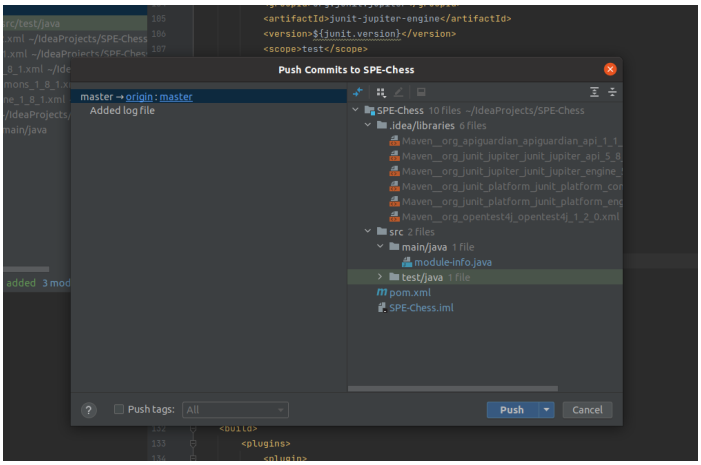
```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <File name="FileAppender" filename="Chess.log">
      <JSONLayout compact="true" eventEol="true">
        <KeyValuePair key="@timestamp" value="${date:yyyy-MM-dd'T'HH:mm:ss.SSSZ}" />
      </JSONLayout>
    </File>
  </Appenders>
  <Loggers>
    <Root level="debug">
      <AppenderRef ref="FileAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

Log4j2 File



Git and GitHub Workflow Schematic [5]

4.5.1 Commit, Push, Pull - Github We can commit and push our code from Intellij without running any commands. We need to first setup our repository in IntelliJ. Finally our repository looks something like in the below figure.

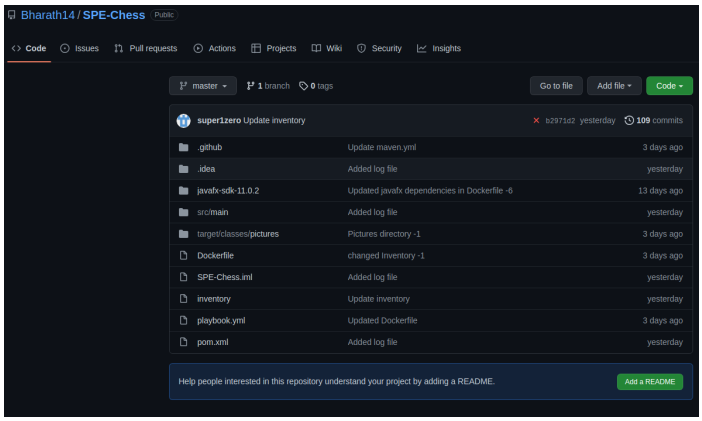


Intellij Github push.

4.6 Application Testing: JUnit Testing

JUnit is a unit testing framework for the Java programming language. Especially for Java and applications which require robust unit testing, Junit can be very helpful. This application is being run against unit tests due to the complexity of Functional Testing.

Configuration



Github Repository

We can easily include the junit dependency in the pom.xml file to install and manage the junit resources because we're using maven. After that,

```xml
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
</dependency>
```

Add the following dependency to **pom.xml** file.

we make a Junit test case for each function and execute it to check how it works. Each function is given its own test suite in Junit. For this we follow as below:

– right-click on the class

– create a Junit test case

– Write the test cases

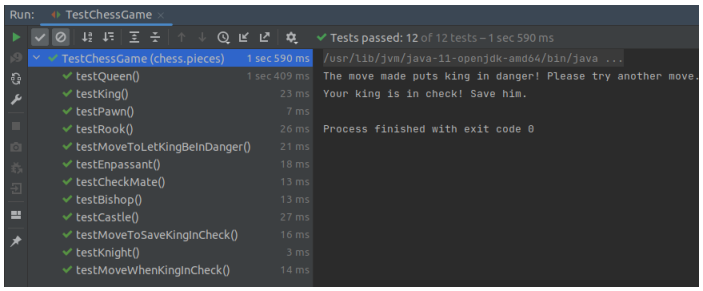
In each test case, we also specify the expected value of the test case and then wrap an assert statement around it.

– assertEquals(expectedValue, returnedValue)

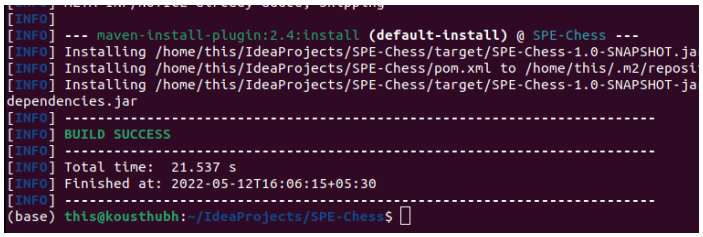
For testing, we have tested multiple piece movement scenarios, game status, whether king would be in danger or not, validation of moves...etc covering almost all functionalities except for AI algorithm.

4.7 Continuous Integration: Jenkins

Continuous Integration is a process of integrating a developer's code into a shared repository in a version control system frequently. CI continuously monitors the version-control system for any new changes. When there are changes CI starts building the code and tests the program.



JUnit testing. All tests passed

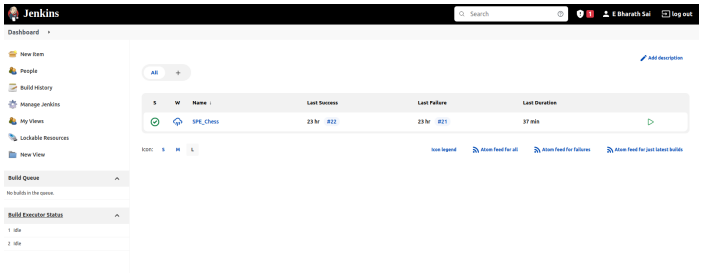


On running **mvn clean install** in terminal

If there are any errors then developers are notified about the error immediately. It will then allow you to deploy on a server automatically or as a one-click process.

Jenkins is an open-source CI/CD server capable of orchestrating a chain of actions that helps to achieve the Continous Integration and Continous Deployment process in an automated way.

Jenkins has various plugins which help us to clone, build, and test the code. We can also containerize, and deploy code on servers using Jenkins. It provides the status of each stage and notifies about the failures if any.



Jenkins Dashboard

Configuration Of Jenkins

Install jenkins in your local machine and follow the steps to setup Jenkins

– Jenkins runs on the localhost at port 8080 by default.

– Open localhost:8080 in a browser.

– Print default password to login into jenkins

- cat /var/lib/jenkins/secrets/initialAdminPassword

– Choose install suggested plugins, configure user.

Internal Setup

– Create a new project by selecting a new item in the top left corner of jenkins dashboard.

– Give name for project select pipeline project.

– In build triggers, select Poll SCM. (checks github periodically triggers the pipeline if there are new commits)

– Give “ * * * * * ” so as to poll every 1 minute.

– Configure docker, ansible related settings in jenkins.

– Click on save then Build now.

– Finally check the console output.

The pipeline script does the following:

The registry in this case refers to your DockerHub’s repository. In the Docker-Jenkins integration stage, the registryCredential corresponds to the Id that we previously assigned to the Docker credentials. The code will be pulled from the given project repository in the first step of the ’Build’ stage. The next step will be to run maven, which will compile, execute, and test the code. If all goes well, it then builds. The next step is to build a Docker image. The image is then pushed to the DockerHub repository specified in the registry variable. Following a successful push, Ansible connects to the guest IP address indicated in the inventory file and runs the playbook.yml file, which instructs it to get the most recent image from the DockerHub repository. (specifics are discussed in section 4.9)

```
pipeline {
  environment {
  registry = "ellanti14/chess"
  registryCredential = "1"
  dockerimage = ""
  }

  agent any
  stages {
stage("Step 1 : Git clone") {
  steps {
  git "https://github.com/Bharath14/SPE-Chess.git"
  }
}
stage("Step 2 : Build Project") {
  steps {
  sh "mvn clean install"
  }
}
stage("Step 3 : Docker Build") {
  steps {
  script {
  dockerimage = docker.build registry + ":latest"
  }
  }
}
stage("Step 4 : Docker Push") {
  steps {
  script {
  docker.withRegistry('', registryCredential){
  dockerimage.push()
  }
  }
  }
}
stage("Step 5 : Ansible Deploy") {
  steps {
  script {
  ansiblePlaybook colorized: true, disableHostKeyChecking: true, installation: 'Ansible', inventory: 'inventory', playbook: 'playbook.yml'
}
```

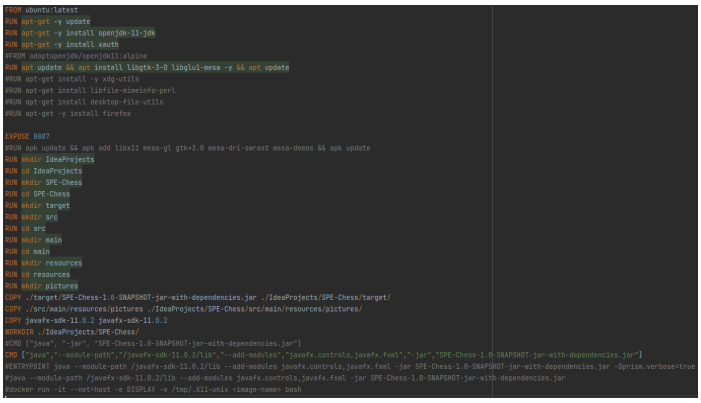
Pipeline Script

4.8 Containerization: Docker

Docker is a tool used to create, deploy, and run applications in an easier way using containers. A container contains software code and its necessary components like libraries, frameworks, and other dependencies. Containers are isolated from each other.

Every Docker container is built on top of a Dockerfile. A Dockerfile is a text file that specifies how to create a Docker image.It explains the container’s operating system, as well as the languages, environmental variables, file locations, network ports, and other components it will need, as well as what the container will do once it is launched. We must first create a Docker Image from the code with all dependencies, and then push it to Docker Hub. The Docker image is a portable file that specifies

Jenkins Builds



Docker File of Chess Image

which software components will be used by the container.

Configuration

– Install docker engine on your machine and configure it to run without sudo command.

– Create a repository in docker hub.

– Create a dockerfile in our project directory.(To let docker know how to build an image)

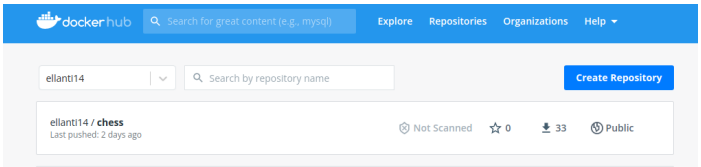
– After this, we clean, build, test the code using maven. A jar file with all dependencies is created.

– We use this jar file create docker image.

– As ours is an GUI application. Our Docker container needs to access the Display of our host system.

– For this we need to specify an authentication. Hence we are using Xauth to do this task.

Add docker hub credentials to our jenkins server. To do this: click manage Jenkins − > Manage Credentials− >Jenkins(store)− >Global Credentials− >Add credential. One important thing to note is that we need to give the same ID in the environment field for the registry credential.And give the "username/repo name" of the docker hub as registry. (in pipeline script)



Chess Image in Docker Hub

4.9 Continuous Delivery: Ansible

Ansible is an open-source automation tool for activities like configuration management, application deployment.. etc. It connects to network nodes (clients servers) and then sends each node a little programme called an Ansible module. It then executes these modules over SSH and then deletes them after they're finished. SSH keys are used as the authentication technique here. Ansible keeps track of which hosts are part of your infrastructure via an inventory file, and then accesses them to run commands and playbooks.

Installation and Setup

Install Ansible in your local machine and and install a ubuntu 18.04 desktop image in a virtual machine. Then try to connect our local host with the virtual machine using ssh key.
After installing VM other configurations, we need to create 2 files: inventory, playbook in project directory.

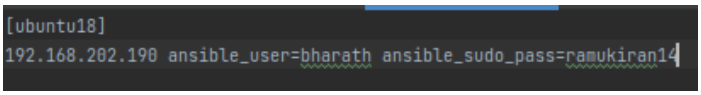
Inventory and Playbook

- Inventory contains information about remote user. First-line specifies some unique name for the machine. In the second line, we specify the IP address of the target machine and the user of the host machine.

- We create a playbook file called filename.yml where we specify the image that needs to be pulled from DockerHub. Here spemini is the repository of DockerHub that will be seen later in Jenkins Pipeline.

Ansible with Jenkins

Go to Manage Jenkins− >Global Tool Configuration. Click on add Ansible Use "which ansible" to get path to the executable directory.



Inventory file

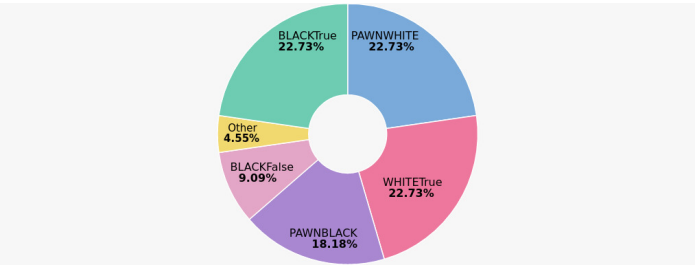
4.10 Continuous Monitoring: ELK Stack

Elasticsearch, Logstash, and Kibana make up the ELK stack, which is an acronym for a stack that consists of three popular projects: Elasticsearch, Logstash, and Kibana. The ELK stack, also known as Elasticsearch, allows you to aggregate logs from all of your systems and applications, analyse them, and visualise them for application and infrastructure monitoring, speedier troubleshooting, security analytics, and more.

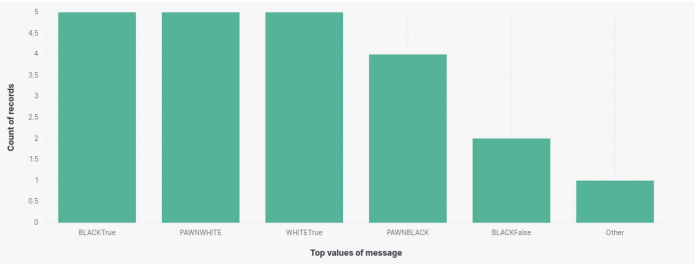
- Elasticsearch is a distributed search and analytics engine built on Apache Lucene.

```
---
- name: Pull docker image from dockerHub
  hosts: ubuntu18
  become: yes
  become_user: root
  remote_user : bharath
  tasks:
    - name: Pull chess image
      docker_image:
        name: ellanti14/chess
        source: pull
```

Playbook file



ELK Visualization:piechart



ELK Visualization:bargraph

- Logstash is an open-source data ingestion tool that allows you to collect data from a variety of sources, transform it, and send it to your desired destination.

- Kibana is a data visualization and exploration tool for reviewing logs and events.

Configuration

– We have installed Logstash, Elastic Search, Kibana and filebeats in different docker containers.

– We use docker-compose to manage these multiple containers.

– We have tried to use filebeats to get log files from our Application container. But it hsan't worked.

– We are manually uploading log files to the elastic search engine.

– Elastic search will be running on the localhost:5601 port.

We need to start the docker compose file containing the three containers elastic search, kibana, and logstash. The command is "docker-compose up". We need to run this command in the folder where the docker-compose file resides. Actually filebeat needs to collect the log files from various containers and send to logstash for filtering. Logstash after filtering then send elastic search for processing. This processed data from elastic search is then sent to Kibana for visualizations.

Due to some reasons this pipeline is not working on our system. So our procedure.

– Run the containers so that elastic search web app can be accessed at localhost:5601

– Collect the log files from application container manually and upload them to elastic search.

– Then we get the above visualizations.

4.11 Challenges

– As ours is a desktop GUI(non web application), when ran the docker container we weren't able to get a display for our GUI. Hence we weren't able to deploy this app in an ubuntu server. So we have deployed it in an Ubuntu desktop OS. Where the container shares the display of the host machine.

– Also to run the application in a container we need to have a base image that can run GUI. Hence while developing the docker Image we choose Ubuntu as our base image and install dependencies like java to run the jar file. Xauth provides authentication so that the container can access the display of the host machine. We have also installed gtk so that it can run GUI applications.

– If we run our image with a normal docker run command then we get "unable to open display" error message. So we have to mention in the docker run command itself that we need to use the display of the host machine.

– We tried to deploy our application in Azure but we couldn't succeed with our docker image. As ours is a non web application it couldn't access the display of azure server.

– So we have deployed our application in ubuntu 17 desktop OS.

– Another challenge was to access the pictures in jar file while running the application. At the time of creation of jar file the path to each file somehow gets changed among which is the pictures folder. So, we had to find a work around to be able to use them.

– We are sending pictures folder containing images of chess pieces to docker container as these are dependencies to the app but are not included in the jar file. Similarly we are also sending javafx dependencies to the docker container. By doing this we need not require any other dependencies to run the docker image.

– One major challenge was to somehow make this non web application into a web application so as to deploy it into web browsers. The work

around was to use some java plugins(gluon) or manually change some parts of the code using JPRO. Unfortunately, we weren't able to use any of these solutions.

4.12 Acknowledgements

We are grateful to Prof. Thangaraju B and the CS 816 course TAs for their assistance in completing this project.

4.13 References

– stackoverflow
– medium
– youtube
– Chess
– javafx
– Refered Github Repo-I
– Refered Github Repo-II