1. Introduction

Game Library is a Spring Boot application which keep track of your video game collection, One you sign up for an account, you can start adding games to your account. As one of my initial projects, the Game Library holds a special place in my development journey. It signifies the early stages of my exploration into software development. This course has offered a valuable chance to take my project to the next level transitioning from a personal project to a production ready application.

Game Library Demo
API Documentation

1.1. Deployment Process

The decision to deploy the Game Library application on virtual machines stems from my experience as a Java programmer in the billing system domain. In this industry, the use of cloud services is restricted due to specific billing constraints, leading me to op for virtual machines as a reliable deployment environment. Furthermore the selection of VMware Workstation due to respect for VMware because of its notable contributions to Spring Boot as framework.

2. Virtual Machine Setup

Being the deployment process by installing VMware Workstation on your host machine. This virtualization software will enable you to create and manage virtual machines.

VMware Workstation download link:
https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html

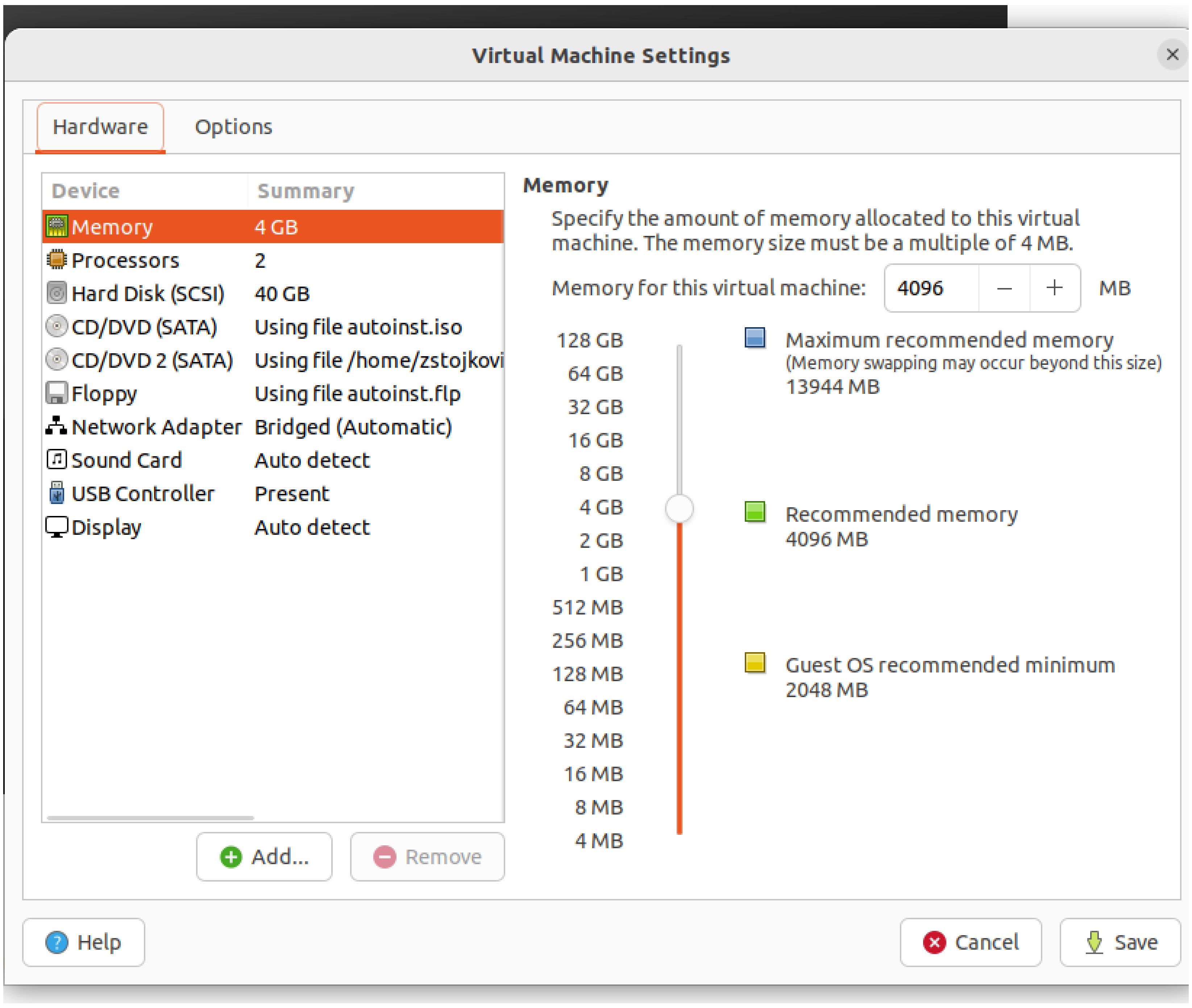
Create a new virtual machine and choose your preferred operating system. For this project, i utilized Ubuntu 20.04.6 LTS, which can be downloaded from this link: https://releases.ubuntu.com/focal/. I opted for Ubuntu due to my familiarity with the OS.

Please note that Hyper-V must be enabled in the BIOS for VMware Workstation to function properly.

Create a new virtual machine with Ubuntu 20.04.6 LTS as the operating system. During the installation. it's crucial to configure necessary settings, such as memory allocation, processor cores, and storage.

Hardware specification for my virtual machine:

- 4GB of RAM
- 2 Processor Cores
- 40GB Storage



Device	Summary
Memory	4 GB
Processors	2
Hard Disk (SCSI)	40 GB
CD/DVD (SATA)	Using file autoinst.iso
CD/DVD 2 (SATA)	Using file /home/zstojkovi
Floppy	Using file autoinst.flp
Network Adapter	Bridged (Automatic)
Sound Card	Auto detect
USB Controller	Present
Display	Auto detect

Memory

Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.

Memory for this virtual machine: 4096 MB

🟦 Maximum recommended memory
(Memory swapping may occur beyond this size)
13944 MB

🟩 Recommended memory
4096 MB

🟨 Guest OS recommended minimum
2048 MB

Disable LWS (Large Write Support), LWS is a fetaure that may cause compatibility issues with certain virtual machines so i decided to disable it during the installation to prevent potential conflicts.

During the Ubuntu installation process, you will be prompted to install additional software, Ensure that OpenSSH server is selected. This will install the SSH server on your virtual machine.
Using SSH (Secure Shell) access instead of directly connecting to a virtual machine advantages in terms of speed. Additionally, if you've already configured your terminal preferences - like using Zsh with various plugins for

features such as command history autocomplete, your work becomes significantly more efficient.

3. Deploy Spring Boot Application

Now we have functional VMware ready for deploying our spring boot application. It is also necessary to change network adapter which you can see in picture above to bridge so your VM have access to internet.

When you run this command you are making sure that your system has the latest information about available packages, and then upgrading the installed packages to their latest versions

```
sudo apt-get update && sudo apt-get upgrade
```

3.1. Set up firewall

First step is to open the ufw configuration file for editing and set ipv6 to No

```
sudo vi /etc/default/ufw
# set ipv6 to No
# save changes with wq!
```

In this part specific ports are allowed through the firewall using the `sudo ufw allow` command.

```
sudo ufw allow 22/tcp # SSH
sudo ufw allow 80/tcp # HTTP
sudo ufw allow 3306/tcp # MySQL
sudo ufw allow 8080/tcp # Apache Tomcat
```

This command enables the UFW firewall and applies all the previously configured rules. Once UFW is enabled, you can check the firewall status using the command `sudo ufw status`

```
sudo ufw enable
```

3.2. Install required packages for application deployment

Execute the following command to install the essential packges needed for deploying the application

```
sudo apt install apache2 openjdk-17-jre mysql-server mysql-client -y
```

apache2: The Apache HTTP Server.

openjdk-17-jre: The Java Runtime Environment (JRE) is necessary for running Java applications. In this case, it's OpenJDK version 17.

mysql-server: serve as the relational database management system (RDBMS).

mysql-client: provides command-line tools for interacting with the MySQL server.

The `-y` flag is used to automatically confirm and accept any prompts during the installation process.

3.3. Configuration of MySQL

Change the authentication method for the root user to 'mysql_native_password' and then run MySQL secure installation for additional configuration.

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'root';

sudo mysql_secure_installation
```

The first command modifies the authentication method for the root user to use mysql_native_password format with the password root.

MySQL secure installation command initiates a script that guides you through securing the installation. The prompts and response are as follows:

- Remove anonymous users? (Y/n): 'Y' , this will remove any anonymous MySQL user, enhancing security

- Disallow root login remotely? (Y/n): 'Y', this will prevent the root user from logging remotely, enhancing security again
- Remove test database and access to it? (Y/n): 'Y', this will delete test database. Since we don't need it
- Reload privilege tables now? (Y/n): 'Y' - This is going to reload priviledge tables and apply the changes made during the installation

Change the bind port in MySQL configuration file

```
cd /etc/mysql/mysql.conf.d
sudo vi mysqld.cnf
```

Inside the file locate the `bind-address` directive and set it to `0.0.0.0`. This allows MySQL to listen on every port instead of restricting it to localhost only.

3.4. Transfer application files to the server

In this step, we will transfer the necessary files, including the `application.properties` and previously build `JAR` file, to the server using the `scp` command.

```
# Transfer data to the server - JAR file and application
properties
scp -r /home/zstojkovic00/Personalno/Singidunum/3/Cloud\
Computing/Spring-Boot-Deploy/game-library/*
zstojkovic@192.168.1.8:/home/zstojkovic/gamelibrary
```

`Scp` stands for secure copy, and it is a command-line utility for securely copying files between hosts using the Secure Shell (SSH) protocol, `-r` stands for recursively copy entire directory. We need to provide source path and destination
in form of `username@server_address:destination_directory`

3.5. Starting Spring Boot Application

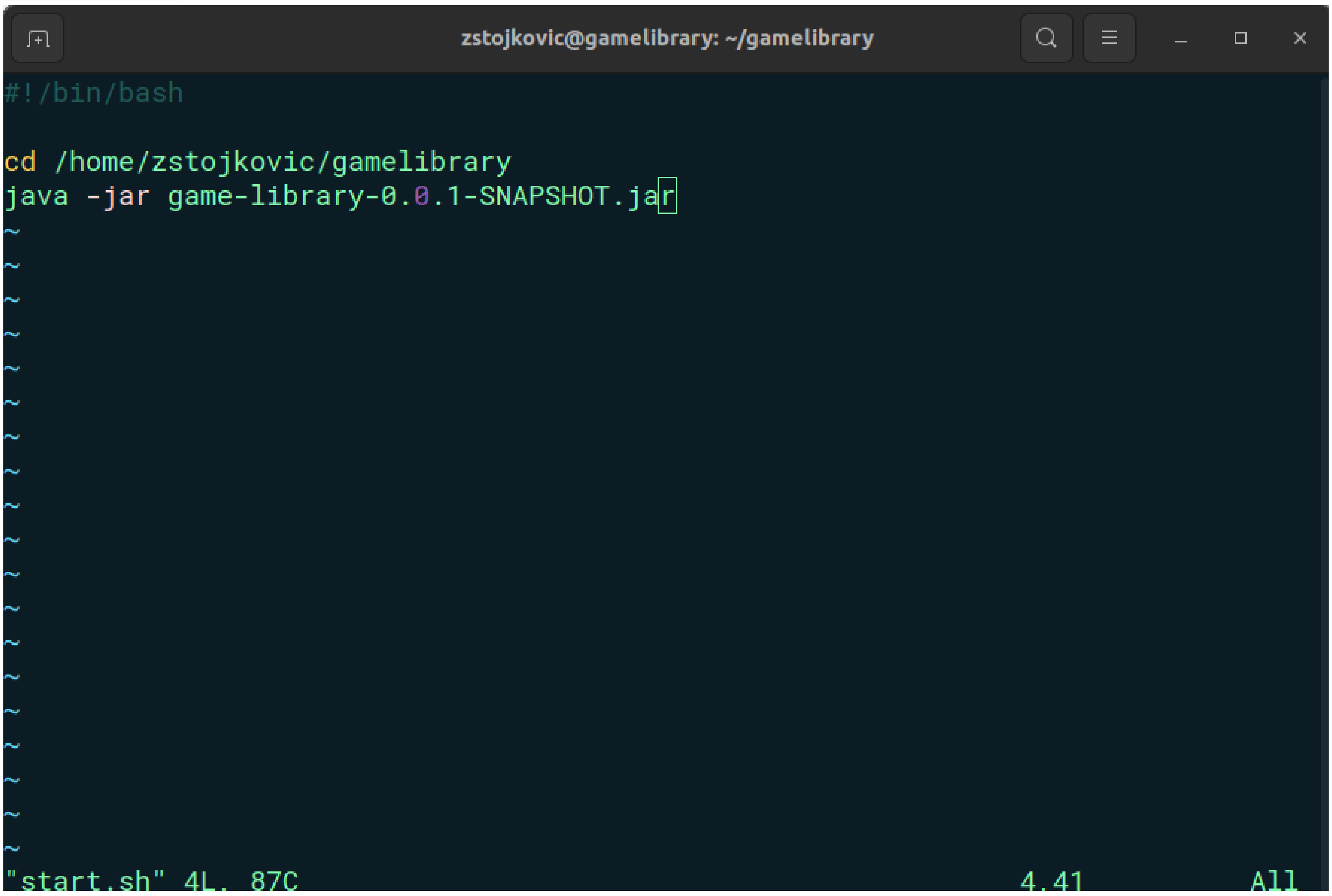
I used maven which is build automation and project management tool used primarily for Java projects. It is used to build the executable JAR file. Since this is a Spring Boot application which have spring-web dependency it comes with embedded Apache Tomcat application server. You can build java application by simply typing `mvn clean package` or `mvn clean install.` Once you get JAR file, launching your application becomes simple. Just type:

```
java -jar {JAR_NAME}
```

After I was sure that my application is working. I made `start.sh` script and corresponding service file to initiate the Java process as a Linux service. To create service file in this case named `gamelibrary.service`, we need to do following:

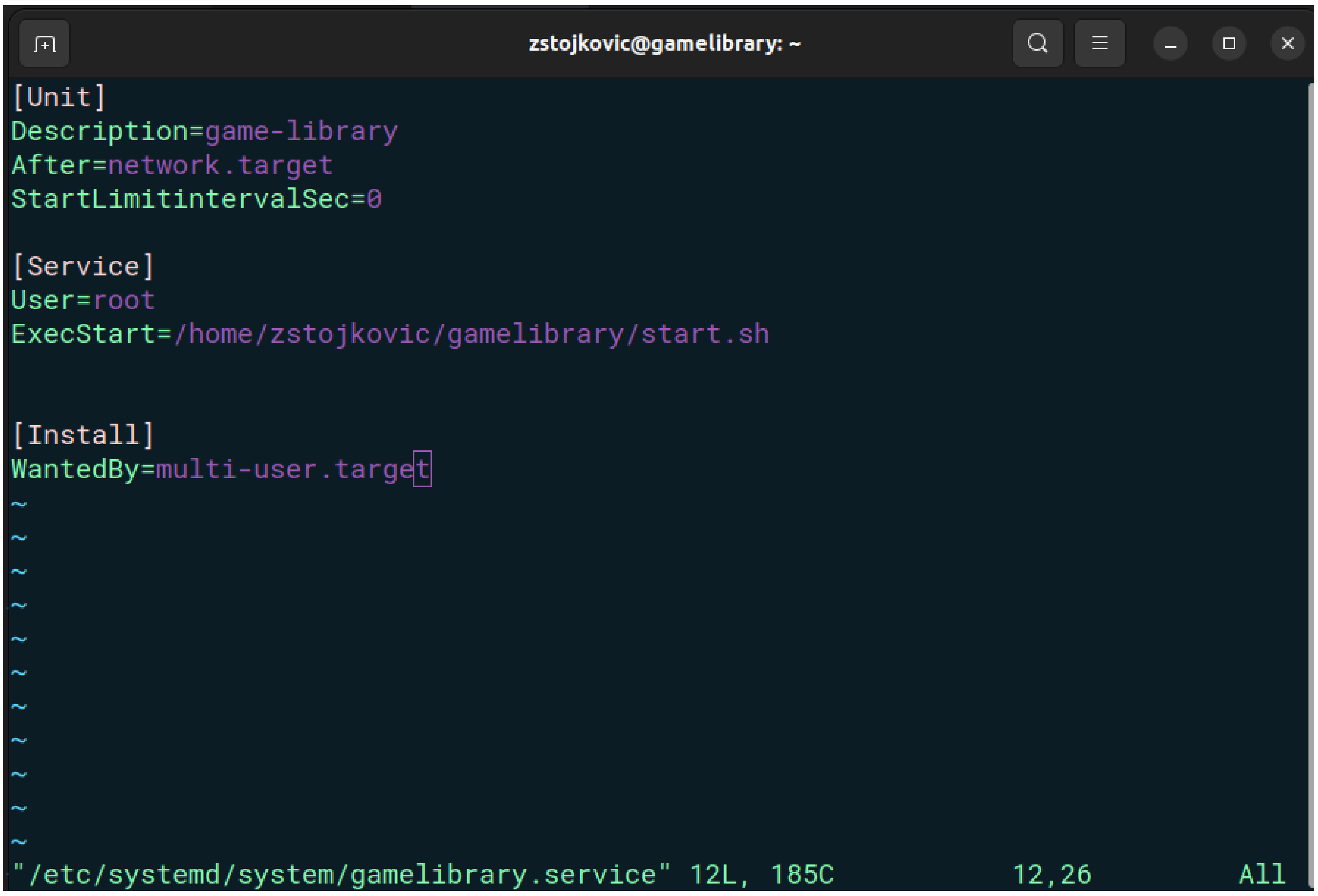
```
sudo cd /etc/systemd/system
sudo touch gamelibrary.service
sudo vi gamelibrary.service
# save changes with wq!
```

`start.sh`



```
#!/bin/bash

cd /home/zstojkovic/gamelibrary
java -jar game-library-0.0.1-SNAPSHOT.jar
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"start.sh" 4L, 87C                                   4,41          All
```

`gamelibrary.service`



```
[Unit]
Description=game-library
After=network.target
StartLimitintervalSec=0

[Service]
User=root
ExecStart=/home/zstojkovic/gamelibrary/start.sh


[Install]
WantedBy=multi-user.target
~
~
~
~
~
~
~
~
~
~
~
"/etc/systemd/system/gamelibrary.service" 12L, 185C          12,26         All
```

After service file is in place, enable it with

```
sudo systemctl enable gamelibrary.service
```

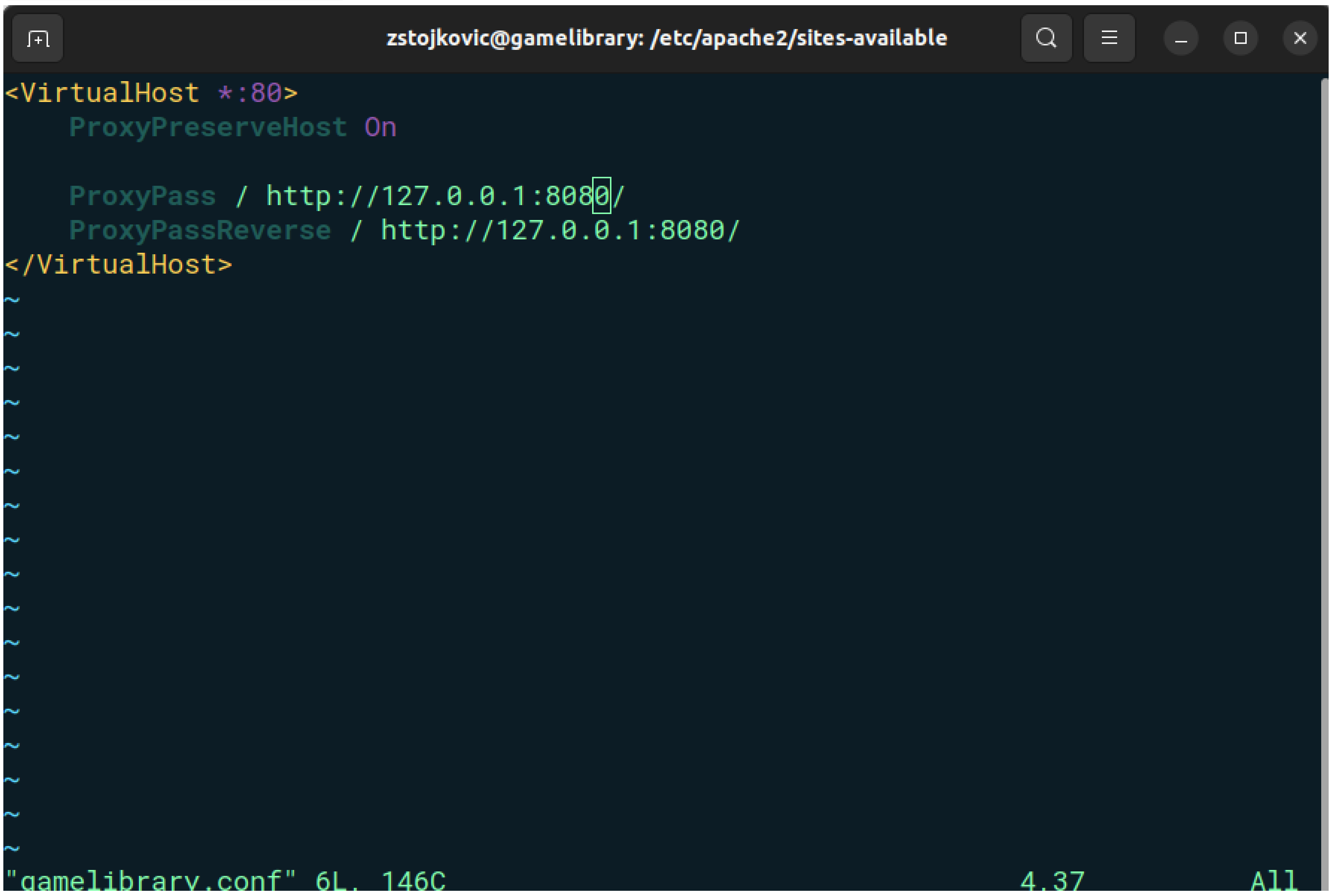
Using a service provides over java process provides several key advantages, including automation during system boot and consistent life cycle management. With services, there's no need to manually search for process IDs and subsequently termite them. Instead a simple command such as `sudo systemctl stop gamelibrary.service` is sufficient. Services also benefit of automatic restarts when necessary, enhancing administration of the application.

3.6. Configuring Apache as a reverse proxy

Create Apache VirtualHost Configuration

```
cd /etc/apache2/sites-available
sudo touch gamelibrary.conf
sudo vi gamelibrary.conf
```

`gamelibrary.conf`


zstojkovic@gamelibrary: /etc/apache2/sites-available

```
<VirtualHost *:80>
    ProxyPreserveHost On

    ProxyPass / http://127.0.0.1:8080/
    ProxyPassReverse / http://127.0.0.1:8080/
</VirtualHost>
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"gamelibrary.conf" 6L, 146C                                  4,37          All
```

Then disable the default configuration `000-default.conf` to ensure our custom `gamelibrary.conf` takes precedence

```
sudo a2dissite 000-default.conf
```

Enable the newly created VirtualHost configuration `gamelibrary.conf` to activate the Apache settings.

```
sudo a2ensite gamelibrary.conf
```

Enabling the crucial Apache modules is essential for ensuring a seamless restart.

```
sudo a2enmod proxy proxy_http proxy_balancer lbmethod_byrequests

sudo systemctl restart apache2
```

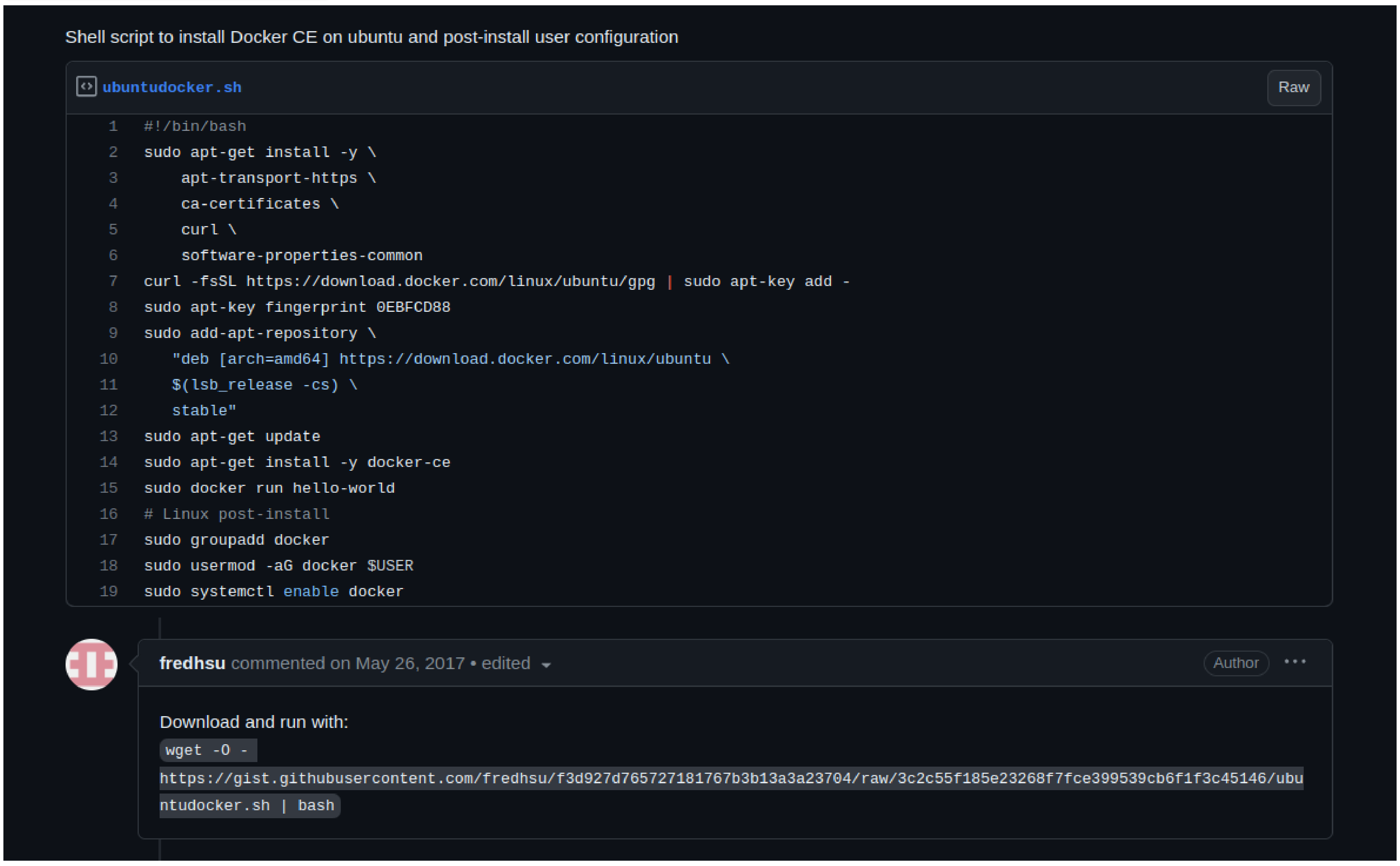
4. Docker

Install Docker on Ubuntu by following the official Docker documentation Visit: https://docs.docker.com/engine/install/ubuntu/

This script simplifies the installation of Docker on Ubuntu by fetching a script from a specific GitHub and running it with bash.

```
wget -O -
https://gist.githubusercontent.com/fredhsu/f3d927d765727181767b3
b13a3a23704/raw/3c2c55f185e23268f7fce399539cb6f1f3c45146/ubuntud
ocker.sh | bash
```

`ubuntudocker.sh`


Shell script to install Docker CE on ubuntu and post-install user configuration

ubuntudocker.sh

```
#!/bin/bash
sudo apt-get install -y \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo apt-key fingerprint 0EBFCD88
sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
sudo apt-get update
sudo apt-get install -y docker-ce
sudo docker run hello-world
# Linux post-install
sudo groupadd docker
sudo usermod -aG docker $USER
sudo systemctl enable docker
```

fredhsu commented on May 26, 2017 • edited

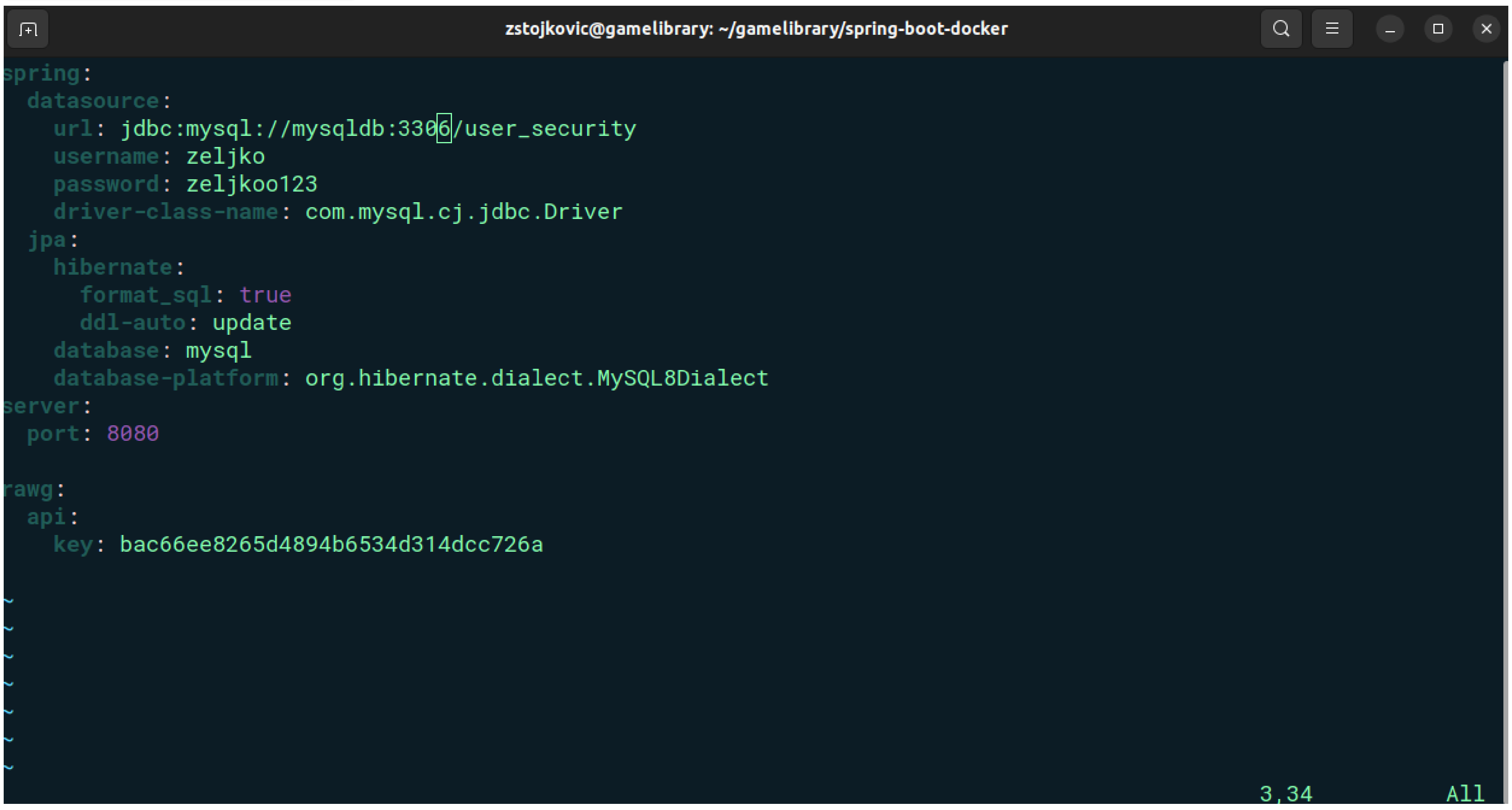
Download and run with:
`wget -O - https://gist.githubusercontent.com/fredhsu/f3d927d765727181767b3b13a3a23704/raw/3c2c55f185e23268f7fce399539cb6f1f3c45146/ubuntudocker.sh | bash`

Acknowledgments:
Special thanks to fredhsu for providing an amazing script.

Now we will transfer the necessary files, including the `application.properties` and previously built `JAR` file, as we did before without Docker. However, this time, we will also include two essential files - `Dockerfile` and `docker-compose.yml`, which are crucial for containerizing our Spring Boot application.

```
scp -r /home/zstojkovic00/Personalno/Singidunum/3/Cloud\
Computing/Spring-Boot-Deploy/game-library/*
zstojkovic@192.168.1.8:/home/zstojkovic/gamelibrary
```

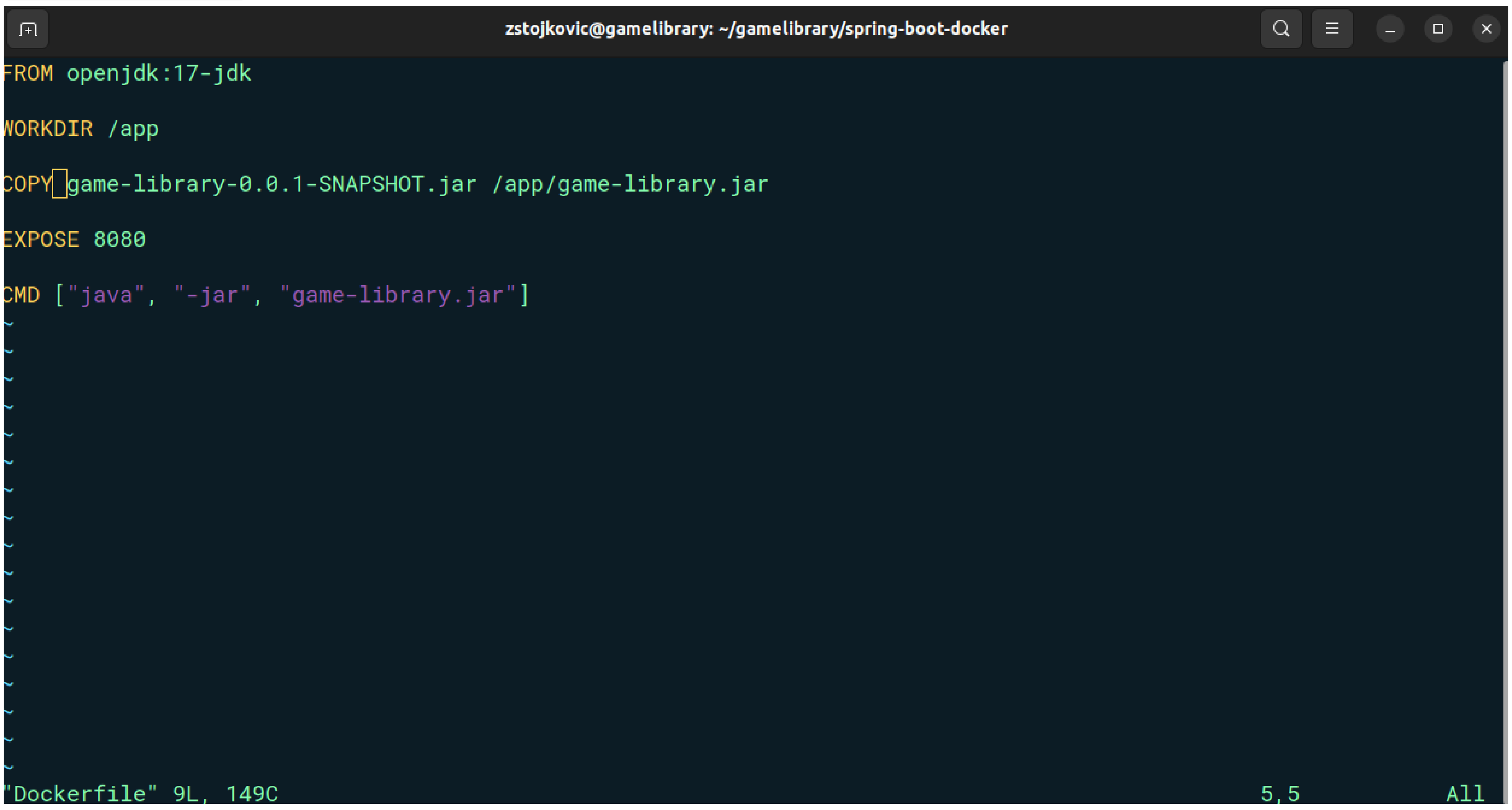
`Application.yml`



```
zstojkovic@gamelibrary: ~/gamelibrary/spring-boot-docker
spring:
  datasource:
    url: jdbc:mysql://mysqldb:3306/user_security
    username: zeljko
    password: zeljkoo123
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      format_sql: true
      ddl-auto: update
    database: mysql
    database-platform: org.hibernate.dialect.MySQL8Dialect
server:
  port: 8080

rawg:
  api:
    key: bac66ee8265d4894b6534d314dcc726a

~
~
~
~
~
~
~
                                                                                              3,34          All
```

The `Dockerfile` is a script that contains instructions for building a Docker image. It specifies the base image, sets up the environment, and defines how the application should be configured inside the container.

`Dockerfile`



```
zstojkovic@gamelibrary: ~/gamelibrary/spring-boot-docker
FROM openjdk:17-jdk

WORKDIR /app

COPY game-library-0.0.1-SNAPSHOT.jar /app/game-library.jar

EXPOSE 8080

CMD ["java", "-jar", "game-library.jar"]
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"Dockerfile" 9L, 149C                                                                         5,5           All
```

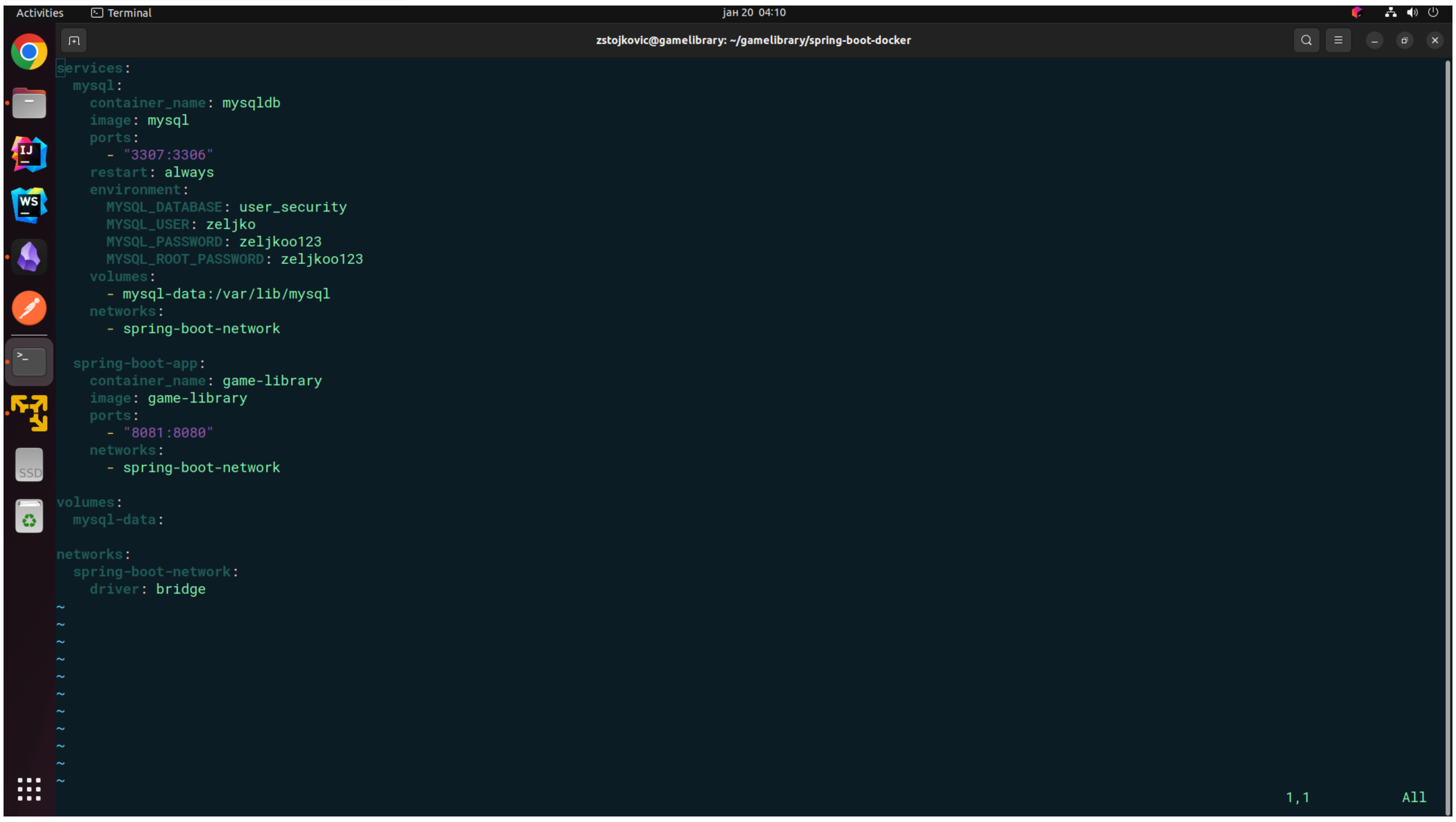
To create a Docker image, run the following command

```
sudo docker build -t game-library .
```

This command will build an image tagged `-t` as `game-library`

The `docker-compose.yml` file is used for defining and running multi-container Docker applications, it allows you to define services, networks and volumes.

`docker-compose.yml`



```
services:
  mysql:
    container_name: mysqldb
    image: mysql
    ports:
      - "3307:3306"
    restart: always
    environment:
      MYSQL_DATABASE: user_security
      MYSQL_USER: zeljko
      MYSQL_PASSWORD: zeljkoo123
      MYSQL_ROOT_PASSWORD: zeljkoo123
    volumes:
      - mysql-data:/var/lib/mysql
    networks:
      - spring-boot-network

  spring-boot-app:
    container_name: game-library
    image: game-library
    ports:
      - "8081:8080"
    networks:
      - spring-boot-network

volumes:
  mysql-data:

networks:
  spring-boot-network:
    driver: bridge
~
~
~
~
~
~
~
~
~
~
~
```

To run both the game-library container and the MySQL container simultaneously, use the following Docker command:

```
sudo docker compose up -d
```

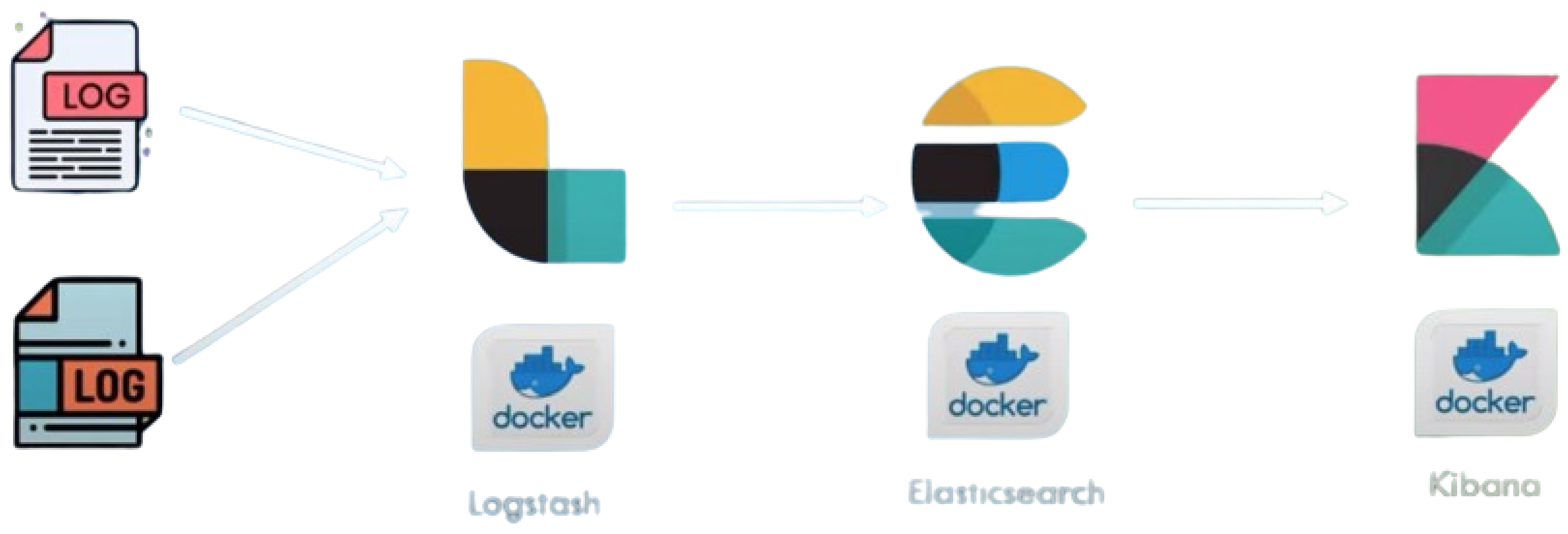
5. Hybrid (Docker + Regular Deployment)

In the context of a hybrid deployment, we use best of both worlds. This approach becomes particular useful when dealing with complexities such as integrating ELK (Elasticsearch, Logstash, Kibana) stack with a Spring Boot application. The decision to run Spring boot locally and the rest in Docker comes because when Spring Boot is deployed as Docker container it has to be in same network as other services for communication. And you need to configure application.yml and other files to use for example mysqldb instead of localhost.

5.1. Adding ELK to Spring Boot Application

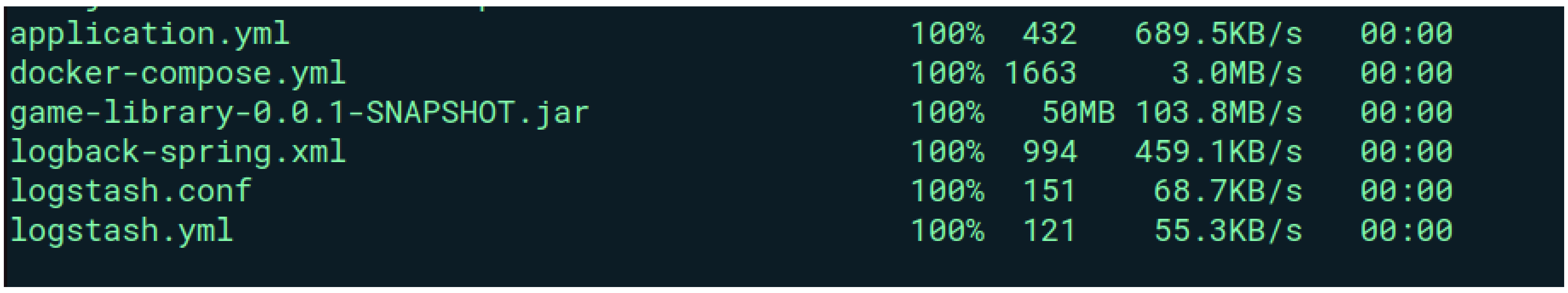
The ELK stack is combination of open-source tools for log management and data visualization

- **Elasticsearch:** A distributed search engine that stores and retrieves data quickly.
- **Logstash:** A data processing pipeline that ingests, processes, and sends logs to Elasticsearch.
- **Kibana:** A visualization tool that allows users to explore and visualize data stored in Elasticsearch.



Similar to the previous steps, we now need to obtain additional configuration files for our ELK (Elasticsearch, Logstash, Kibana) integration. These files, namely `logback-spring.xml`, `logstash.conf`, and `logstash.yml`, play a crucial role in instructing the Spring Boot application on how to send log files to Elasticsearch.

List of necessary files

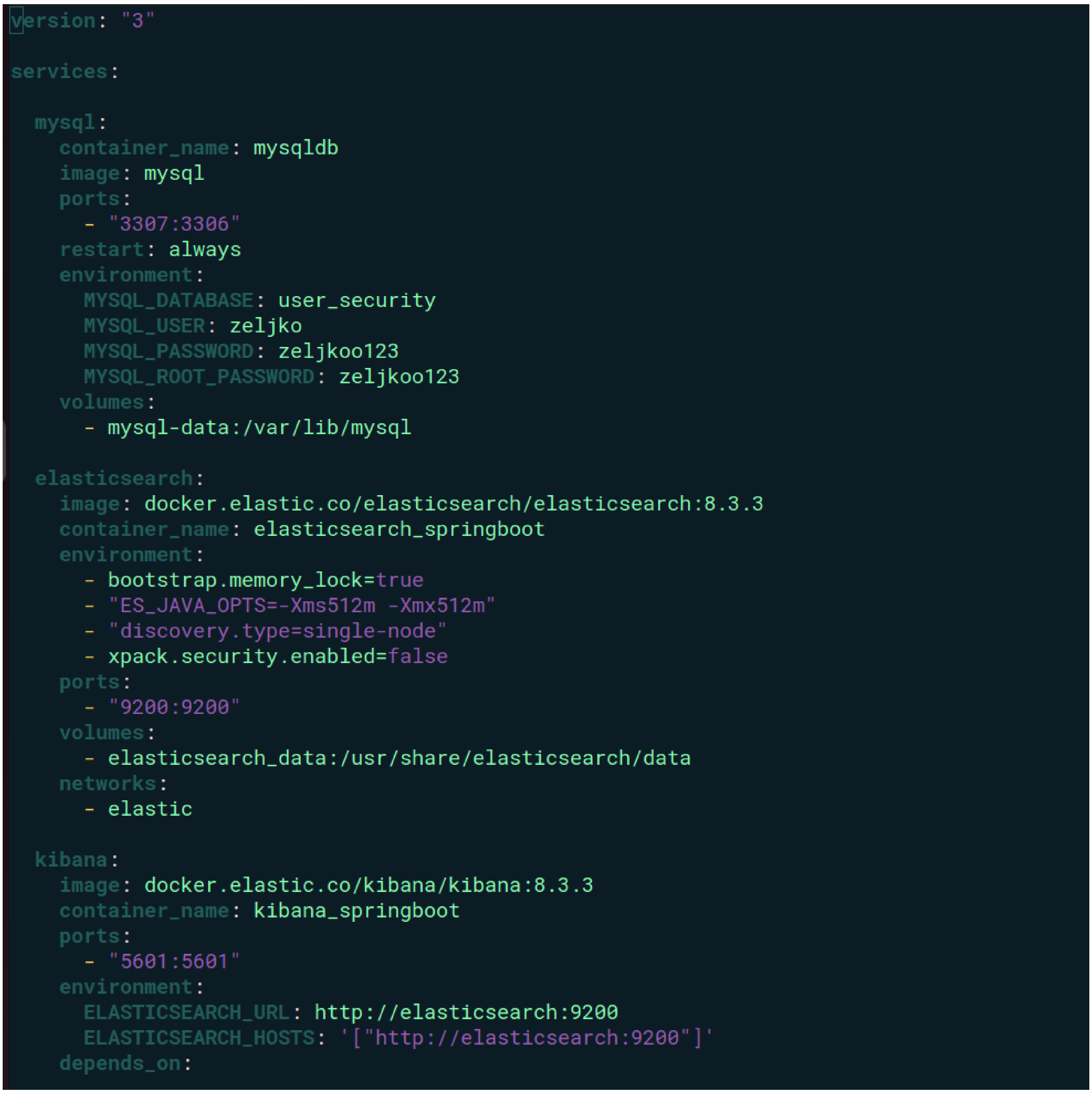


```
application.yml                                 100%  432   689.5KB/s   00:00
docker-compose.yml                              100% 1663     3.0MB/s   00:00
game-library-0.0.1-SNAPSHOT.jar                 100%   50MB 103.8MB/s   00:00
logback-spring.xml                              100%  994   459.1KB/s   00:00
logstash.conf                                   100%  151    68.7KB/s   00:00
logstash.yml                                    100%  121    55.3KB/s   00:00
```

```
scp -r /home/zstojkovic00/Personalno/Singidunum/3/Cloud\
Computing/Spring-Boot-Deploy/game-library-elk/*
zstojkovic@192.168.1.8:/home/zstojkovic/elk-gamelibrary
```

In this step, we've modified the `docker-compose.yml` file to integrate MySQL and ELK stack. We removed Spring Boot application from `docker-compose.yml` and now we define each service separately.

`docker-compose.yml`



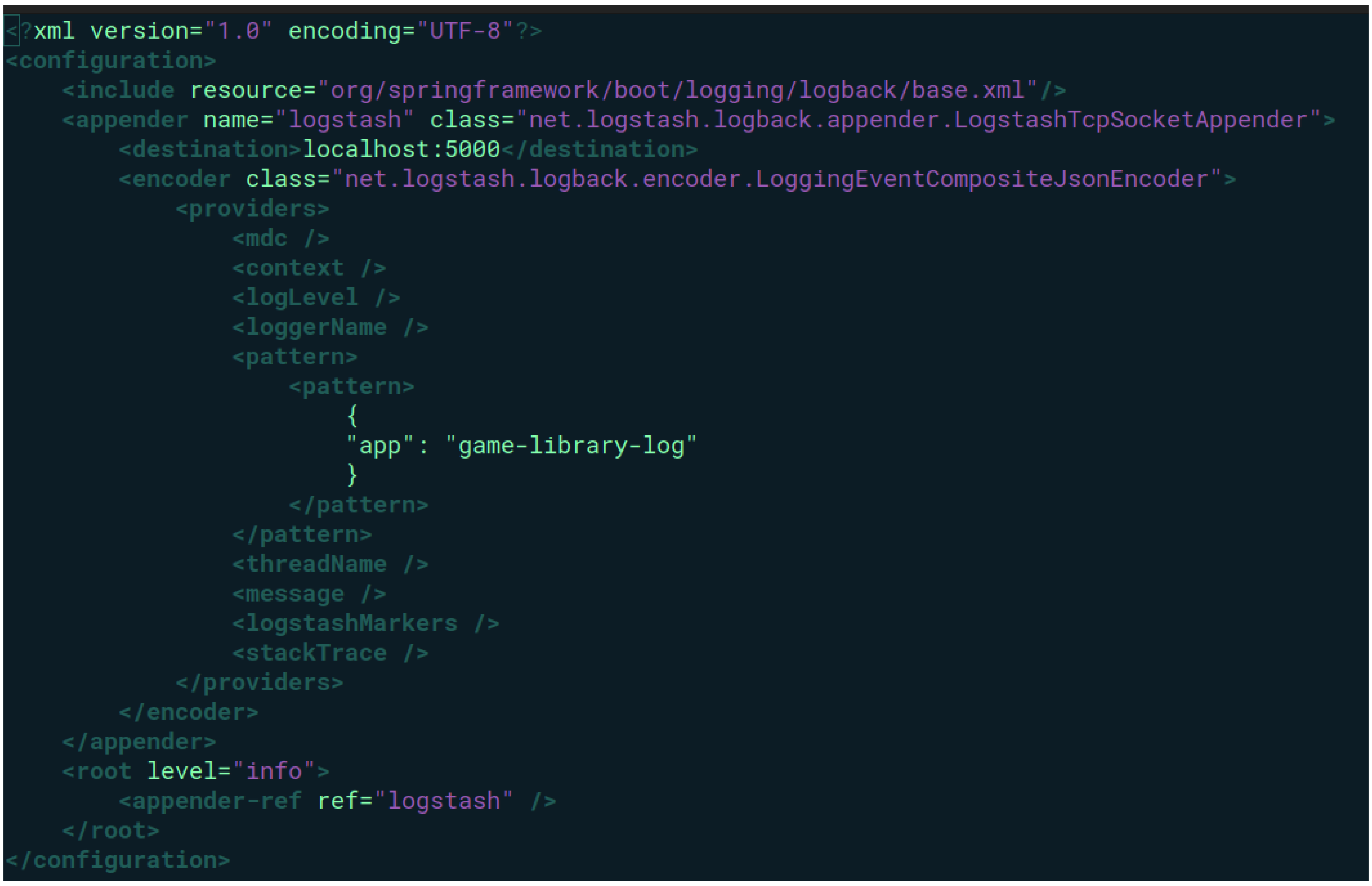
```yaml
version: "3"

services:

  mysql:
    container_name: mysqldb
    image: mysql
    ports:
      - "3307:3306"
    restart: always
    environment:
      MYSQL_DATABASE: user_security
      MYSQL_USER: zeljko
      MYSQL_PASSWORD: zeljkoo123
      MYSQL_ROOT_PASSWORD: zeljkoo123
    volumes:
      - mysql-data:/var/lib/mysql

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.3.3
    container_name: elasticsearch_springboot
    environment:
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
      - "discovery.type=single-node"
      - xpack.security.enabled=false
    ports:
      - "9200:9200"
    volumes:
      - elasticsearch_data:/usr/share/elasticsearch/data
    networks:
      - elastic

  kibana:
    image: docker.elastic.co/kibana/kibana:8.3.3
    container_name: kibana_springboot
    ports:
      - "5601:5601"
    environment:
      ELASTICSEARCH_URL: http://elasticsearch:9200
      ELASTICSEARCH_HOSTS: '["http://elasticsearch:9200"]'
    depends_on:
      - elasticsearch
    networks:
      - elastic

  logstash:
    image: docker.elastic.co/logstash/logstash:8.3.3
    container_name: logstash_springboot
    volumes:
      - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml:ro
      - ./logstash/pipeline:/usr/share/logstash/pipeline:ro
    ports:
      - "5044:5044"
      - "5000:5000/tcp"
      - "5000:5000/udp"
      - "9600:9600"
    environment:
      LS_JAVA_OPTS: "-Xmx256m -Xms256m"
    networks:
      - elastic
    depends_on:
      - elasticsearch

networks:
  elastic:
    driver: bridge

volumes:
  mysql-data:
  elasticsearch_data:
```

Now we are using `localhost` instead of the container name (`mysqldb`) in `application.yml` because the Spring Boot application is not dockerized.

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3307/user_security
    username: zeljko
    password: zeljkoo123
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      format_sql: true
      ddl-auto: update
    database: mysql
    database-platform: org.hibernate.dialect.MySQL8Dialect
server:
  port: 8080

rawg:
  api:
    base: https://api.rawg.io/api/games
    key: bac66ee8265d4894b6534d314dcc726a
```

`logback-spring.xml` is used to send log events in JSON format to a `Logstash` server at `localhost:5000`.

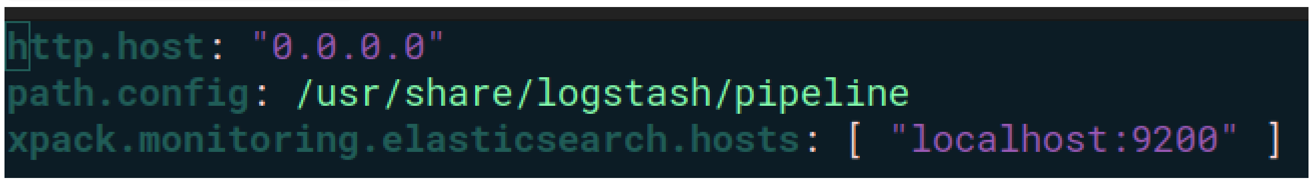
logback-spring.xml



```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <include resource="org/springframework/boot/logging/logback/base.xml"/>
    <appender name="logstash" class="net.logstash.logback.appender.LogstashTcpSocketAppender">
        <destination>localhost:5000</destination>
        <encoder class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
            <providers>
                <mdc />
                <context />
                <logLevel />
                <loggerName />
                <pattern>
                    <pattern>
                        {
                        "app": "game-library-log"
                        }
                    </pattern>
                </pattern>
                <threadName />
                <message />
                <logstashMarkers />
                <stackTrace />
            </providers>
        </encoder>
    </appender>
    <root level="info">
        <appender-ref ref="logstash" />
    </root>
</configuration>
```

`logstash.yml` file configure `Logstash` with specific settings such as http.host (bind to all available network interfaces, making it accessible from external hosts), specify files where logstash should look for its pipeline configuration files.

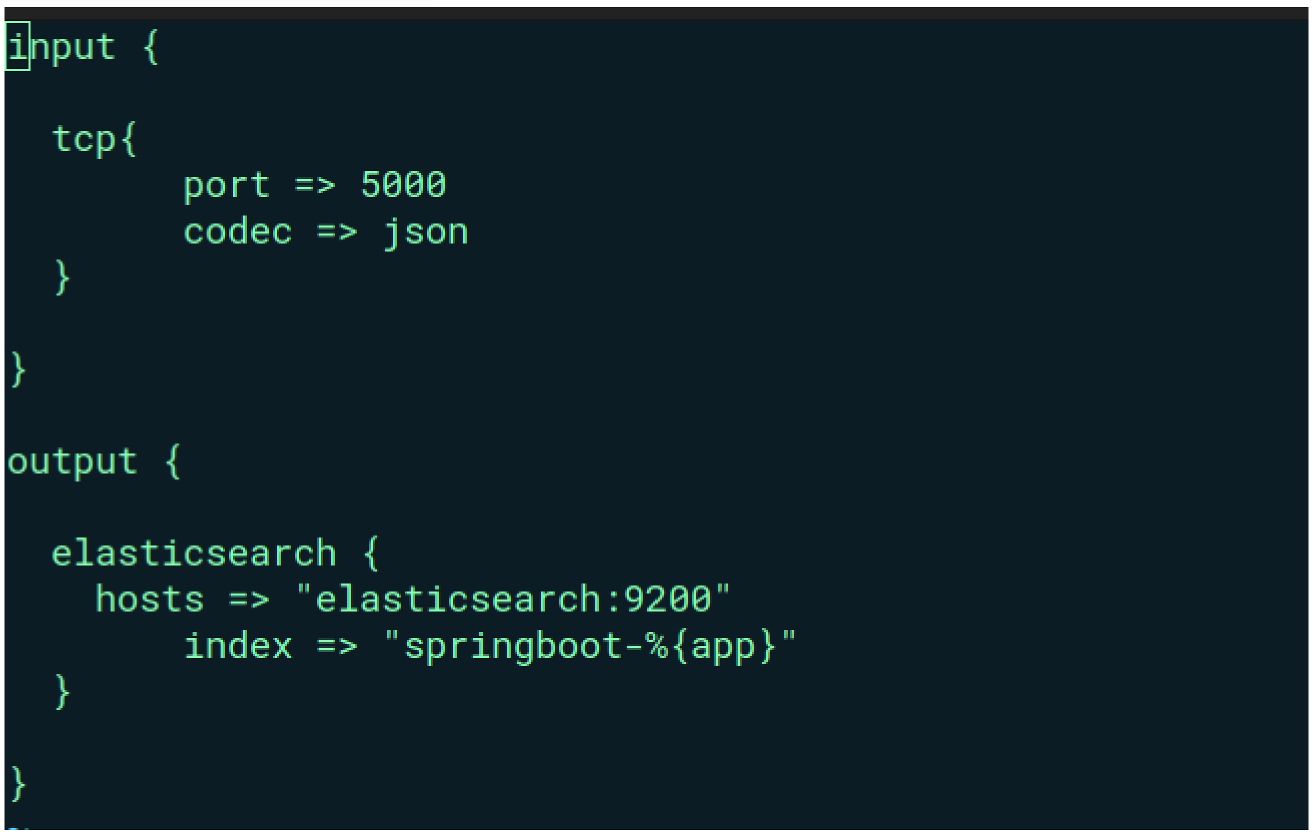
`logstash.yml`



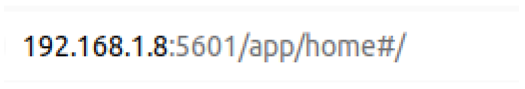
```yaml
http.host: "0.0.0.0"
path.config: /usr/share/logstash/pipeline
xpack.monitoring.elasticsearch.hosts: [ "localhost:9200" ]
```

`logstash.config` located in `Logstash` pipeline, defines the `logstash` pipeline configuration. Input section specifies that `logstash` will listen for incoming log events on `tcp` port 5000, and codec indicates that incoming messages are expected to be in `JSON` format.
Output specifies to send processed log events to `Elasticsearch`

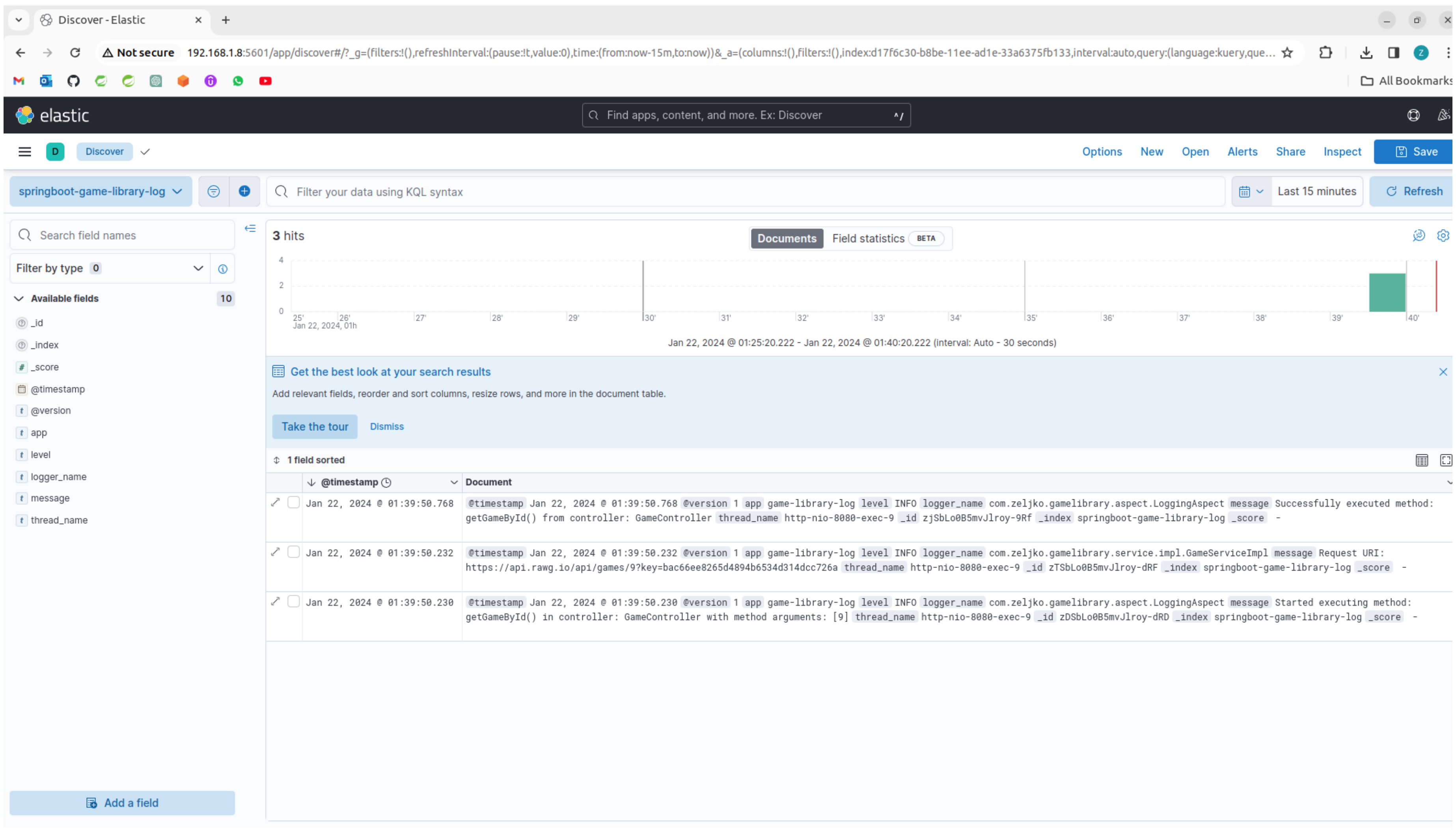
`logstash.config`



```
input {

  tcp{
        port => 5000
        codec => json
  }

}

output {

  elasticsearch {
    hosts => "elasticsearch:9200"
        index => "springboot-%{app}"
  }

}
```



192.168.1.8:5601/app/home#/



3 hits

Jan 22, 2024 @ 01:25:20.222 - Jan 22, 2024 @ 01:40:20.222 (interval: Auto - 30 seconds)

Get the best look at your search results

Add relevant fields, reorder and sort columns, resize rows, and more in the document table.

```
@timestamp                    Document
Jan 22, 2024 @ 01:39:50.768   @timestamp Jan 22, 2024 @ 01:39:50.768 @version 1 app game-library-log level INFO logger_name com.zeljko.gamelibrary.aspect.LoggingAspect message Successfully executed method: getGameById() from controller: GameController thread_name http-nio-8080-exec-9 _id zjSbLo0B5mvJlroy-9Rf _index springboot-game-library-log _score -
Jan 22, 2024 @ 01:39:50.232   @timestamp Jan 22, 2024 @ 01:39:50.232 @version 1 app game-library-log level INFO logger_name com.zeljko.gamelibrary.service.impl.GameServiceImpl message Request URI: https://api.rawg.io/api/games/9?key=bac66ee8265d4894b6534d314dcc726a thread_name http-nio-8080-exec-9 _id zTSbLo0B5mvJlroy-dRF _index springboot-game-library-log _score -
Jan 22, 2024 @ 01:39:50.230   @timestamp Jan 22, 2024 @ 01:39:50.230 @version 1 app game-library-log level INFO logger_name com.zeljko.gamelibrary.aspect.LoggingAspect message Started executing method: getGameById() in controller: GameController with method arguments: [9] thread_name http-nio-8080-exec-9 _id zDSbLo0B5mvJlroy-dRD _index springboot-game-library-log _score -
```

6. Conclusion

When deciding between regular deployment and Dockerized deployment, the choice depends on specific project requirements and scalability.

Regular Deployment is well-suited for smaller applications with straightforward configurations. It involves manual server environment setup and direct application management. On other hand, Dockerized deployment presents a more scalable solution. Docker containers encapsulate the application, its dependencies, and runtime environment. Additional advantages of Docker include cloud compatibility and seamless integration with orchestration tools like Kubernetes (K8s). Effective Docker usage demands expertise, as its adoption necessitates adjustments to the entire CI/CD pipeline. While Docker volumes provide persistent data storage, it may not be the ideal solution for production. Dockerized databases, however, prove beneficial for testing purposes, as exemplified by Test-containers.