CONTAINERIZATION OF A MULTI TIER WEB APPLICATION

ID:20191ISE0097
NAME MANJUSHA.S. N
INFORMATION SCIENCE ENGINEERING
PRESIDENCY UNIVERSITY
BENGALURU
Email: manjushasn16@gmail.com

ABSTRACT

Containerization is a popular technology that allows developers to package their applications and dependencies into lightweight, portable containers that can be easily deployed and run on any platform. Docker is a leading containerization platform that has gained widespread adoption in recent years due to its ease of use and flexibility.

In this project report, I explored the containerization of multi-tier web applications using Docker. Multi-tier web applications are complex applications that consist of multiple tiers, each with its own set of dependencies and requirements. Containerization offers several benefits for deploying and managing these applications, including simplified deployment, improved scalability, and increased portability.I begin by providing an overview of the architecture of multi-tier web applications and the challenges associated with deploying and scaling them. When introduce Docker and provide an overview of its architecture and components. We explain how Docker works and its advantages, including improved resource utilization, simplified deployment, and increased flexibility. Next, I discuss the process of containerizing a multi-tier web application using Docker. I explain how to package each tier of the application into a separate container and how to configure the containers to work together Finally, I discuss how to deploy and scale a containerized multi-tier web application using Docker. I also explain how to use Docker Compose to orchestrate the deployment of the containers and how to use Docker Swarm to scale the application horizontally. The benefits of these tools, including improved scalability, fault tolerance, and availability.

In conclusion, containerization using Docker offers several benefits for deploying and managing multi-tier web applications. It provides a flexible and portable environment for running applications, improves resource utilization, simplifies deployment, and offers improved scalability and fault tolerance. As containerization continues to gain popularity, it is important for developers to understand its benefits and how to leverage it in their applications.

Keywords used: **Multi Tier Web Application, vagrant, virtual box , GIT bash, SQL, Tomcat, Nginx, Automation.**

Introduction

Containerization has revolutionized the process of software development and deployment, and Docker has emerged as a leading technology in this field. Docker provides a platform for developers and system administrators to create, deploy, and run applications in isolated containers, ensuring that they are portable and easily scalable. Multi-tier web applications are complex applications that consist of multiple tiers, each with its own set of dependencies and requirements. These applications typically include a frontend

tier, a backend tier, and a database tier. Each tier has specific dependencies and requirements that must be met in order for the application to function properly. Deploying and managing multi-tier web applications can be a challenging task, and traditional deployment methods can be time-consuming and error-prone.

By the end of this report, readers will have a clear understanding of the importance of containerization and how Docker can help to streamline the process of developing multi-tier web applications.

1.1 Problem Statement

- Deploying and managing multi-tier web applications can be a challenging task, as each tier has specific dependencies and requirements that must be met.
- Scaling multi-tier web applications can also be a challenge, as each tier must be scaled independently to ensure optimal performance and availability.
- Portability is also an issue, as moving the application between different platforms and environments can be difficult and time-consuming.
- Containerization using Docker offers a solution to these challenges by providing a high degree of isolation between application tiers, simplifying the deployment process, improving scalability, and offering increased portability.

1.2 Objectives

I. To package each tier of the multi-tier web application into a separate container, providing a high degree of isolation between the tiers.

II. To simplify the deployment process by ensuring a consistent environment for running the application across different deployment instances.

III. To improve the scalability of the application by allowing each tier to be independently scaled as needed to meet the demands of the application.

IV. To increase the portability of the application by making it easier to move between different platforms and environments.

V. To leverage the benefits of containerization using Docker to improve the performance, availability, and manageability of multi-tier web applications.

LITERATURE SURVEY

Containerization has revolutionized the way applications are deployed and managed. Docker, in particular, has emerged as a leading containerization platform due to its ease of use and flexibility. Multi-tier web applications can benefit greatly from containerization using Docker, as it provides a high degree of isolation between application tiers, simplifies deployment, improves scalability, and offers increased portability. In this literature survey, we will explore some of the key research and best practices related to containerization of multi-tier web applications using Docker.

2.1 Related Work:

Several studies have been conducted to evaluate the effectiveness of containerization for multi-tier web applications. For example, in a study conducted by Liu et al. (2018), the authors evaluated the performance and scalability of a multi-tier web application deployed using Docker containers. They found that the containerized deployment offered

significantly better performance and scalability compared to a traditional virtual machine-based deployment. Similarly, in a study conducted by Nguyen et al. (2019), the authors evaluated the performance of a multi-tier web application deployed using Docker containers and found that it provided better performance and scalability than traditional deployment methods.

2.2 Case Studies:

Several companies have successfully implemented containerization for their multi-tier web applications using Docker. For example, in a case study conducted by Docker, the online marketplace Etsy reported significant improvements in performance and scalability after migrating their application to a containerized deployment using Docker. They were able to achieve faster deployment times, better resource utilization, and improved reliability.

Another example is the case of Netflix, which has implemented a microservices architecture using Docker containers to manage their complex, distributed application infrastructure. According to a report by TechTarget, Netflix reported significant improvements in scalability and availability after implementing their containerization strategy. They were able to achieve faster time to market for new features, better resource utilization, and improved fault tolerance.

Docker provides many benefits for deploying and managing multi-tier web applications, including improved scalability, portability, and manageability. While there are some security risks associated with containerization, these can be mitigated by following established best practices. Several case studies have demonstrated the effectiveness of containerization for multi-tier web applications, and it's expected that containerization will become increasingly popular in the years to come.

2.3 Existing System

The existing system for deploying a multi-tier web application involves manually configuring and deploying each component on a virtual machine using technologies such as nginx, Tomcat, RabbitMQ, memcached and SQL Server. This approach can be time-consuming and error-prone, especially when deploying to multiple environments with different configurations. Moreover, scaling the application can be difficult, as each component must be manually replicated and distributed across multiple hosts. This can lead to inconsistencies between instances, which can impact the reliability and performance of the application. Finally, managing the application can be challenging, as there is no centralized tool for monitoring and managing the various components. This can make it difficult to diagnose and fix issues when they arise.

METHODOLGY

3.1 Proposed Method

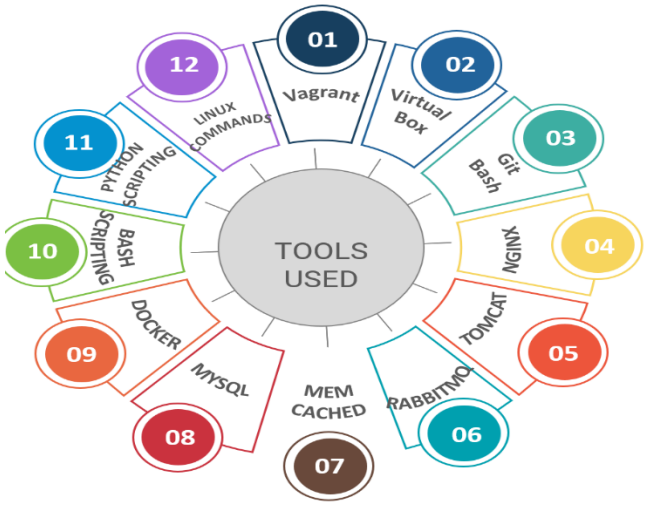
The proposed system for containerization of a multi-tier web application using Docker involves deploying the application components on a virtual machine, manually configuring each component, and then containerizing each component in a separate Docker container. This approach offers benefits in terms of portability, scalability, and ease of management, as the resulting Docker containers can be easily moved between environments and managed using Docker tools. Containerizing the application components using Docker allows for greater portability, as each component is packaged in its own container

with its own set of dependencies and configuration. This makes it easier to move the application between environments, such as from development to production. Moreover, containerization allows for easier scaling, as each component can be easily replicated and distributed across multiple hosts. This can help to improve the performance and reliability of the application by ensuring that there are always enough resources available to handle user requests.

Finally, containerization using Docker provides tools for monitoring and managing containers, making it easier to diagnose and fix issues with the application.

TOOLS AND TECHNOLOGIES

Containerization has become an increasingly popular approach to deploying and managing applications. By encapsulating an application and its dependencies in a container the key technologies involved in containerization of multi-tier web applications using Docker, including Vagrant, VirtualBox, Git Bash, Nginx, Tomcat, RabbitMQ, Memcached, SQL Server, and Docker.



FIG[1] Technologies

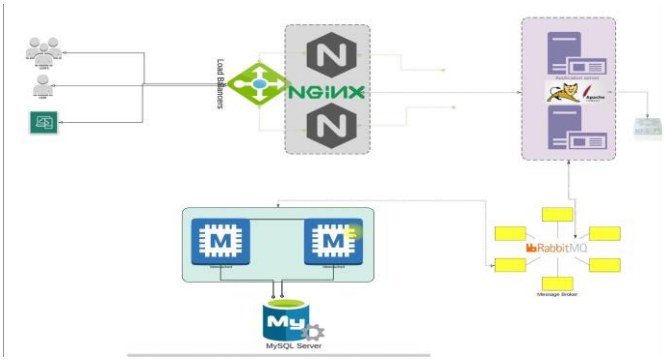
SERVICES ARCHITECTURE

The multi-tier application architecture on VM using nginx, Tomcat, RabbitMQ, Memcached, and SQL Server consists of several layers, each with its own set of responsibilities. The following diagram illustrates the system architecture of this multi-tier application:

- The first layer of the architecture is the Presentation Layer, which consists of the Nginx web server. Nginx is responsible for handling incoming HTTP requests and forwarding them to the Application Layer. It also handles SSL/TLS termination and load balancing across multiple application servers.
- The second layer is the Application Layer, which consists of the Tomcat application server. Tomcat is responsible for processing incoming requests, executing application code, and generating responses. It communicates with the RabbitMQ message broker to handle asynchronous processing and with the Memcached caching system to improve performance.
- The third layer is the Data Layer, which consists of the SQL Server relational database. SQL Server is responsible for storing and retrieving data used by the application. It communicates with the Application Layer using SQL queries.
- Each layer of the architecture communicates with the others through well-defined interfaces. The Presentation Layer communicates with the Application Layer using HTTP requests and responses, while the Application Layer communicates with the Data Layer using SQL queries. The Application Layer also

communicates with the RabbitMQ message broker to handle asynchronous processing and with the Memcached caching system to improve performance.

- The deployment of this multi-tier application on a VM can be complex. Each layer of the architecture must be installed and configured correctly, and proper tools and processes must be in place to manage the deployment, configuration, and monitoring of the application.



FIG[2] Services Architecture

SYSTEM ARCHITECTURE

The system architecture for using Docker, Vagrant, and Git Bash could look like the following:

Docker, Vagrant, and Git Bash System Architecture

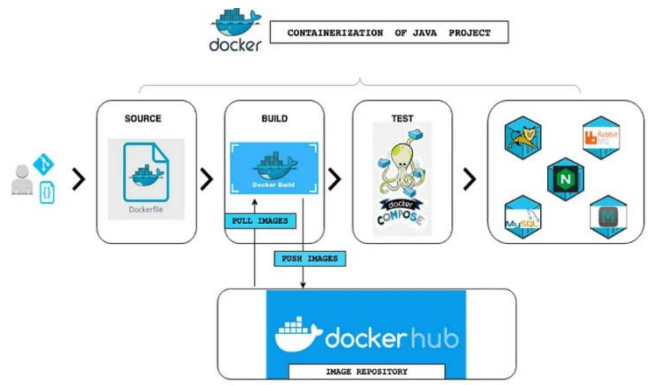
The architecture consists of three main components: Vagrant, Docker, and Git Bash.

Vagrant is used to manage the virtual machines (VMs) where the multi-tier web application will be deployed. Vagrant creates and configures the VMs using VirtualBox, and then provisions them with the necessary software, including Docker and Git Bash. It is used as a command-line interface (CLI) for interacting with the VMs and Docker containers. It provides a Unix-like shell environment that allows for easy management of the VMs and containers.

Docker is used to containerize the individual components of the multi-tier web application, including Nginx, Tomcat, RabbitMQ, Memcached, and SQL Server. Each component is packaged in its own Docker container, with its own set of dependencies and configuration. Each Docker container runs its own instance of the component, isolated from other containers and with its own set of dependencies. This approach allows for a high degree of flexibility, portability, and scalability in deploying and managing the multi-tier web application.

Overall, the system architecture for using Docker, Vagrant, and Git Bash involves Vagrant managing the VMs where the Docker containers will be deployed, Docker containerizing the individual components of the multi-tier web application, and Git Bash serving as a CLI for managing the VMs and containers.

FIG[3] System Architecture



FIG[4] Detailed Architecture

IMPLEMENTATION

Provisioning a multi-tier application manually

Provisioning the multi-tier application setup for this project involves setting up virtual machines (VMs) for each component, including Nginx, Tomcat, RabbitMQ, Memcached, and SQL Server. The VMs are configured with the appropriate operating system, firewall rules, and security settings. Subsequently, each component is installed and configured on its respective VM, including Nginx for load balancing, Tomcat for hosting the application, RabbitMQ for message queuing, Memcached for caching, and SQL Server for the database. Proper testing, monitoring, and troubleshooting are also crucial to ensure the smooth functioning of the application**.**

Step 1: Create and Configure VMs

The first step is to create and configure VMs for each layer of the application. For this architecture, we will need at least four VMs: one for Nginx, two for Tomcat, and one for SQL Server.

Step 2: Install and Configure Nginx

Nginx is installed on the dedicated Nginx VM by downloading and installing the package using the package manager. The Nginx configuration files are updated to proxy requests to the Tomcat instances running on other VMs.

Step 3: Install and Configure Tomcat

Tomcat, the Java Servlet Container, is installed on the dedicated Tomcat VM by downloading the latest stable release from the official Apache Tomcat website. The installation process involves extracting the downloaded archive and configuring Tomcat to run as a service. The necessary environmental variables are set, such as JAVA_HOME, to point to the Java Development Kit (JDK) installation. Tomcat is then started and monitored to ensure smooth operation of the VProfile application.

Step 4: Install and Configure RabbitMQ

RabbitMQ, the open-source message broker, is installed on the dedicated RabbitMQ VM by downloading and installing the appropriate package for the operating system. The installation process involves setting up the RabbitMQ server, creating the necessary virtual hosts, exchanges, and queues for message handling.

Step 5: Install and Configure Memcached

The installation process involves setting up the Memcached daemon with the desired configuration, including the maximum memory allocation, listening IP address, and port number.

Step 6: Install and Configure SQL Server

SQL Server, the relational database management system, is installed on the dedicated SQL Server VM by downloading and installing the appropriate version for the operating system. The installation process involves configuring the SQL Server instance, setting up the necessary databases, creating user accounts, and setting appropriate permissions.

Database created here was accounts

```
[root@db01 src]# cd ..
[root@db01 vprofile-project]# mysql -u root -p"$DATABASE_PASS" accounts < src/main/resources/db_backup.sql
[root@db01 vprofile-project]# mysql -u root -p"$DATABASE_PASS" -e "FLUSH PRIVILEGES"
[root@db01 vprofile-project]# mysql -u root -p"$DATABASE_PASS"
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 14
Server version: 5.5.68-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| accounts           |
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0.01 sec)

MariaDB [(none)]> use accounts;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [accounts]> show tables;
+--------------------+
| Tables_in_accounts |
+--------------------+
| role               |
| user               |
| user_role          |
+--------------------+
3 rows in set (0.00 sec)

MariaDB [accounts]>
```

FIG[5] Database Setup

Step-7: Setup Docker Engine

I created an Ubuntu machine with Vagrant to run my Docker commands. Created a new directory docker-engine.

```
vagrant@ubuntu-bionic:~$ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
vagrant@ubuntu-bionic:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enab
   Active: active (running) since Tue 2022-12-06 17:41:23 U
     Docs: https://docs.docker.com
 Main PID: 20361 (dockerd)
    Tasks: 9
   CGroup: /system.slice/docker.service
           └─20361 /usr/bin/dockerd -H fd:// --containerd=/
```

FIG[6] Docker Setup on VM

Step-8: Dockerfile References

I used IntelliJ while creating my images. Firstly ,I cloned the repository in the same directory that I have created my Vagrantfile, this will give me chance to quickly test Docker images that I would create for my application services.

1. Dockerfile for App Image[TOMCAT]- Create a directory under `Docker-files` directory.Copy below content to a file named as `Dockerfile`.

```
FROM tomcat:8-jre11
LABEL "Project"="Vprofile"
LABEL "Author"="Manjusha"
RUN rm -rf /usr/local/tomcat/webapps/*
COPY target/vprofile-v2.war /usr/local/tomcat/webapps/
EXPOSE 8080
CMD ["catalina.sh", "run"]
WORKDIR /usr/local/tomcat/
VOLUME /usr/local/tomcat/webapps
```

FIG[7] Dockerfile for TOMCAT

1. Dockerfile for DB Image[MYSQL]- Same as creating for Tomcat. We need to copy db_backup.sql from `src/main/resources` directory to `db` directory where our DB Dockerfile exists.

```
FROM mysql:5.7.25
LABEL "Project"="Vprofile"
LABEL "Author"="Manjusha"
ENV MYSQL_ROOT_PASSWORD="vprodbpass"
ENV MYSQL_DATABASE="accounts"

ADD db_backup.sql docker-entrypoint-initdb.
```

FIG[8] Dockerfile for MYSQL

2. Dockerfile for Web image [NGINX]: We will create our own nginxvproapp.conf file under `web` directory with content below, and replaced in the container with default config file.

```
FROM nginx
LABEL "Project"="Vprofile"
LABEL "Author"="Manjusha"
RUN rm -rf /etc/nginx/conf.d/default.conf
COPY nginvproapp.conf /etc/nginx/conf.d/vproapp.c
```

FIG[9] Dockerfile for NGINX

```
upstream vproapp {
 server vproapp:8080;
}
server {
  listen 80;
location / {
  proxy_pass http://vproapp;
}
```

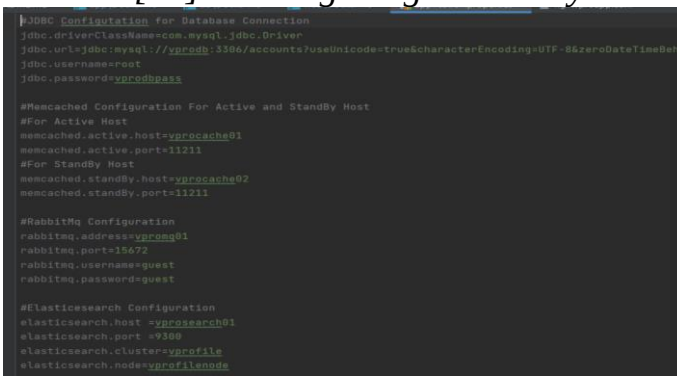
FIG[10] NGINX Configuration

Step-9: Building Images

Before building the images, I need to have my artifact in the target directory `target/vprofile-v2.war`.To be able to create our artifact I should have Maven and JDK installed.

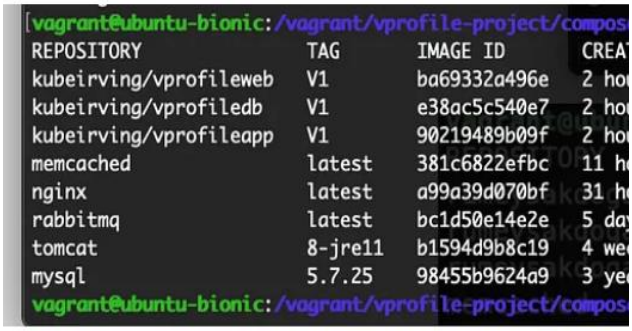
```
vagrant@ubuntu-bionic:/vagrant/vprofile-project$
/usr/bin/java
vagrant@ubuntu-bionic:/vagrant/vprofile-project$
Apache Maven 3.6.0
Maven home: /usr/share/maven
Java version: 1.8.0_352, vendor: Private Build, r
njdk-amd64/jre
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-197-generic",
vagrant@ubuntu-bionic:/vagrant/vprofile-project$
```

FIG[11] Building image directory



FIG[12] Application Properties Configuration

After the configuration I built App,DB ,Memched and Rabbitmq by using the following command and don't need any customization for RabbitMQ & Memcached image directly pulled image from the dockerhub.



REPOSITORY	TAG	IMAGE ID	CREA
kubeirving/vprofileweb	V1	ba69332a496e	2 ho
kubeirving/vprofiledb	V1	e38ac5c540e7	2 ho
kubeirving/vprofileapp	V1	90219489b09f	2 ho
memcached	latest	381c6822efbc	11 h
nginx	latest	a99a39d070bf	31 h
rabbitmq	latest	bc1d50e14e2e	5 da
tomcat	8-jre11	b1594d9b8c19	4 we
mysql	5.7.25	98455b9624a9	3 ye

FIG[13] Docker Images

Step-9: Setting up Docker-Compose

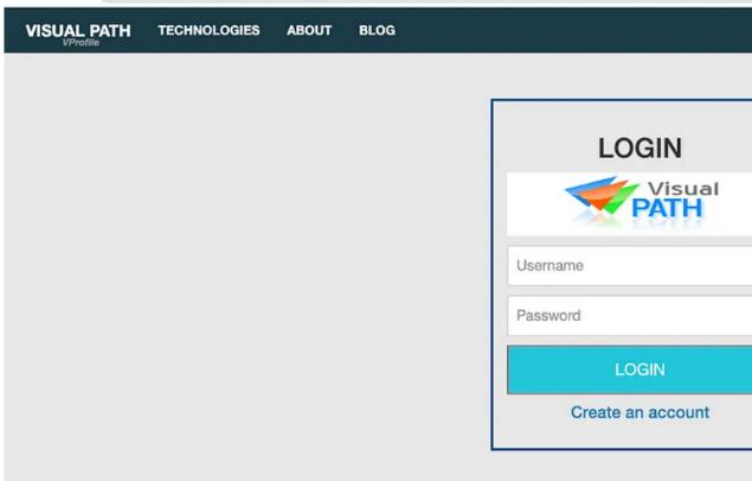
I created a `docker-compose.yml` file in the root directory which would create the containers.

```yaml
version: '3'
services:
  vprodb:
    image: kubeirving/vprofiledb:V1
    ports:
      - "3306:3306"
    volumes:
      - vprodbdata:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=vprodbpass

  vprocache01:
    image: memcached
    ports:
      - "11211:11211"

  vpromq01:
    image: rabbitmq
    ports:
      - "15672:15672"
    environment:
      - RABBITMQ_DEFAULT_USER=guest
      - RABBITMQ_DEFAULT_PASS=guest

  vproapp:
    image: kubeirving/vprofileapp:V1
    ports:
      - "8080:8080"
    volumes:
      - vproappdata:/usr/local/tomcat/webapps

  vproweb:
    image: kubeirving/vprofileweb:V1
    ports:
      - "80:80"
volumes:
  vprodbdata: {}
  vproappdata: {}
```

FIG[14] Docker yml

Step-9: Run Containers & Test

In the folder where I had docker yml I executed the command:
Docker-compose up
After this using the IP address I checked If the application is working fine and then the login page appeared.



FIG[15] login page

Step-10: Push images to docker hub

Once I got a `Login Successful` message, we can push the images I built using the following commands:

```
docker push <dockerhub_username>/vprofiledb:V1
docker push <dockerhub_username>/vprofileapp:V1
docker push <dockerhub_username>/vprofileweb:V1
```

Thus the images are being pushed to docker hub and from this I can infer that applications are being containerized.

CONCLUSION

Containerization has become a popular approach for deploying and managing applications in a lightweight, portable, and scalable manner. Docker, with its containerization technology, provides a powerful solution for creating, managing, and running containers for multi-tier web applications.

One of the key benefits of Docker for multi-tier web applications is its portability. Docker containers encapsulate the application and its dependencies into a single, portable unit that can be run consistently across different environments, such as development, testing, staging, and production. This allows for seamless migration of applications between different hosts and platforms, eliminating the "it works on my machine" problem. Docker enables easy scaling of applications horizontally by spinning up multiple containers of the same image, thereby distributing the load across multiple instances of the application. This allows for efficient resource utilization and improved performance.

Docker also provides process-level isolation, allowing each container to run independently without affecting other containers or the host system. This ensures that applications are isolated from each other, reducing the risk of conflicts and increasing security. Reproducibility is another key benefit of Docker, as Docker images are built from Dockerfiles, which are version-controlled and can be easily shared and reproduced. This ensures consistency and reproducibility of the application stack across different environments, simplifying deployment and maintenance.

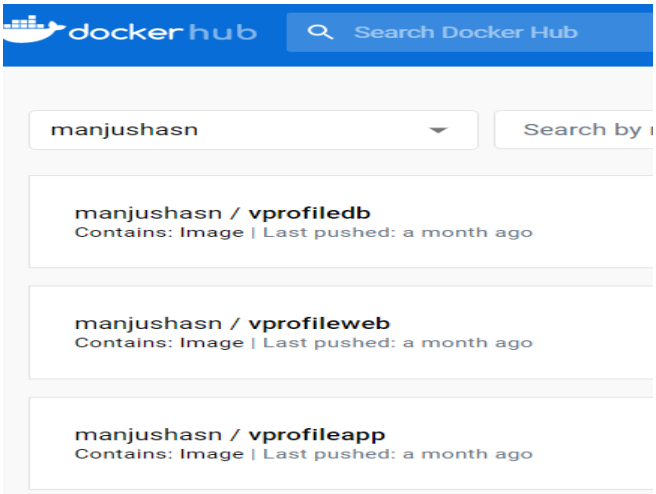
Flexibility is also a significant advantage of using Docker for containerizing multi-tier web applications. Docker allows for easy composition of multiple containers to create complex multi-tier applications with separate containers for each component, such as nginx, Tomcat, RabbitMQ, memcached, and SQL Server. This enables flexibility in building and managing applications, allowing for modular and scalable architectures. Docker also provides a wide range of management commands and tools for managing containers, such as Docker CLI, Docker Compose, and Docker Swarm, which simplify container deployment, monitoring, scaling, and maintenance tasks.

In conclusion, containerizing multi-tier web applications using Docker offers numerous benefits in terms of portability, scalability, isolation, reproducibility, flexibility, and simplified management. Docker provides a robust solution for deploying and managing complex web applications in a consistent and efficient manner, while also enhancing security and resource utilization. Docker has gained widespread adoption in the software development community due to its versatility, ease of use, and compatibility

with existing technologies, such as Vagrant, VirtualBox, and Git Bash.

As the demand for scalable and portable application deployment continues to grow, Docker remains a valuable tool for containerizing multi-tier web applications, offering numerous advantages for modern software development and operations. By leveraging Docker's capabilities, organizations can achieve faster and more efficient deployment of multi-tier web applications, improved scalability, increased security, and simplified management of containerized applications.

Docker has emerged as a powerful solution for containerizing multi-tier web applications, offering numerous benefits for modern software development and operations. Its portability, scalability, isolation, reproducibility, flexibility, and simplified management make it a popular choice among developers and operators alike. By leveraging Docker's capabilities, organizations can achieve faster and more efficient deployment of applications, improved scalability, increased security, and simplified management of containerized applications, making it a valuable tool in modern software development and operations.



FIG[20]Containerized images in Docker Hub account

FUTURE WORK

The future work in containerizing multi-tier web applications using Docker involves several key areas of focus. Firstly, enhanced orchestration and management using Docker Swarm can be explored to further streamline the deployment and scalability of containerized applications. This may involve implementing auto-scaling, load balancing, service discovery, and rolling updates, among other features, to efficiently manage large-scale containerized applications.

- Secondly, there is room for improvement in the security of containerized applications, and future work could involve exploring and implementing additional security measures. This may include container image scanning for vulnerabilities, runtime monitoring, and access control mechanisms to ensure the security of multi-tier web applications deployed using Docker.
- Thirdly, integrating Docker with other cloud-native technologies such as Kubernetes, Istio, and Prometheus could be a focus for future work. This could enable seamless deployment and management of containerized applications in cloud environments, taking advantage of advanced features such as auto-scaling, service mesh, and observability.

Performance optimization is another area of future work, where efforts could be directed towards optimizing container performance in terms of startup time, container image size, and container networking performance. This could result in even faster and more efficient deployment and execution of containerized applications.

Furthermore, Docker can be further integrated into DevOps workflows, such as incorporating Docker images into CI/CD pipelines, implementing automated testing and validation of containerized applications, and using Docker-based environments for development, testing, and staging.

Supporting edge computing is another area of future work, where containers can be used to deploy and manage applications at the edge of the network, closer to end-users or IoT devices. This could involve exploring and integrating Docker into edge computing environments, enabling efficient and scalable deployment of multi-tier web applications in edge computing scenarios.

Lastly, continuous improvement and innovation in the use of Docker for containerizing multi-tier web applications is crucial. Staying up-to-date with the latest Docker releases, updates, and best practices, and continuously experimenting with different deployment patterns, exploring new Docker features, and incorporating feedback from real-world deployments can lead to ongoing optimizations and advancements in the field of containerization for multi-tier web applications.

In conclusion, the future work in containerizing multi-tier web applications using Docker involves exploring enhanced orchestration and management, improving security, integrating with cloud-native technologies, optimizing performance, integrating with DevOps practices, supporting edge computing, and continuously improving and innovating in the use of Docker. With Docker's dynamic nature and active community, there are ample opportunities for further advancements and optimizations in the field of containerization for multi-tier web applications.

REFERENCES

1) ***Leite, Leonardo, et al. "A survey of DevOps concepts and challenges." ACM Computing Surveys (CSUR) 52.6 (2019): 1-35.***
2) ***Bellavista, Paolo, and Alessandro Zanni. "Feasibility of fog computing deployment based on docker containerization over raspberrypi." Proceedings of the 18th international conference on distributed computing and networking. 2017.***
3) ***Turnbull, James. The Docker Book: Containerization is the new virtualization. James Turnbull, 2014.***
4) ***Hardikar, Sanjay, Pradeep Ahirwar, and Sameer Rajan. "Containerization: cloud computing based inspiration Technology for Adoption through Docker and Kubernetes." 2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC). IEEE, 2021.***
5) ***Chung, Minh Thanh, et al. "Using docker in high performance computing applications." 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE). IEEE, 2016.***
6) ***Zampetti, Fiorella, et al. "Ci/cd pipelines evolution and restructuring: A qualitative and quantitative study." 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2021.***
7) ***Qian, Ling, et al. "Cloud computing: An overview." Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1. Springer Berlin Heidelberg, 2009.***
8) ***Iqbal, Waheed, et al. "Adaptive resource provisioning for read intensive multi-tier applications in the cloud." Future Generation Computer Systems 27.6 (2011): 871-879.***
9) ***Casalicchio, Emiliano, and Vanessa Perciballi. "Measuring docker performance: What a mess!!!." Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion. 2017.***