A Case Study on Containerizing Applications with Docker and Apache Tomcat

Hareesh Kumar Rapolu

hareeshkumar.rapolu@gmail.com

Abstract

This case study examines the Docker containerizing process for applications through Docker and Apache. It focuses on the integration of Apache Tomcat, a dominant Java-based web server and servlet container. The use of Docker proves to empower developers with the ability to build isolated environments for streamlined deployment and scaling of Tomcat-based applications. This study documents and explains the approach for containerizing Tomcat applications and tests the results and benefits to inform best practices of containerizing Tomcat applications. It reveals that containerization provides better scalability alongside key benefits such as improved scalability, environment consistency, and enhanced operational efficiency.

Keywords: Docker, Containerization, Apache Tomcat, Microservices, Deployment, DevOps, Kubernetes

I. INTRODUCTION

Docker and Apache are evolving as some of the best solutions within the container technology as they provide the needed capabilities for runtime environment. They are allowing a shift from the historical manual configuration of web servers by enabling the installation of correct dependencies and management of configuration files [1]. The Docker system creates independent application containers that maintain operational stability across development testing and production environments [1]. Docker containers provide complete application code alongside its entire set of dependencies, libraries configuration files and system files [2]. The containerization approach provided by Docker eliminates dependency conflicts while preventing configuration drift such that it stands as a preferred solution for application deployment on Apache Tomcat. Containerization with Docker helps developers create deployable application containers that run consistently across different environments [3]. Conversely, Apache Tomcat is a key web server instrument which offers Java-based open-source solutions for hosting Java servlet and JSP technology web applications. Organizations attain portability and scalability along with management efficiency across development and testing and production environments through Docker implementation on Tomcat-based applications. This paper provides a case study of how Docker and Apache Tomcat can streamline the deployment process for Java web applications.

II. RELATED WORKS

Docker and Apache Tomcat integration has had significant attention in academic and industry realms given the increased adoption of containerized environment in software development. A significant research attention focuses on the use of Docker in cloud environment. For example, Pahl [4] analyzed containerization technologies in the PaaS Cloud and underscored the Docker's benefits in the deployment of distributed applications across different cloud platforms. The study showed that containers are integral in dynamic scaling, strengths fault tolerance and offer resource efficiency in cloud-based infrastructure. Docker's adoption also explains the advent of microservices architectures where organizations decompose monolithic applications into smaller, standalone deployable services. A systematic review by Fowler and Lewis [5] shows that Docker's containerization benefits microservices since they break down applications into small independent deployable units. The current case study encapsulates its monolithic Tomcat application inside a Docker container while enabling the system for potential future microservices architecture deployment.Other studies have focused on critical challenges associated with deployment of Apache Tomcat in Docker environments. Combe et al. [6], for example, established that while Docker container allows for process-level isolation, additional configurations are required to achieve security of containers in production environment. Several areas of vulnerabilities exist in multitenant environment. The findings suggest the need for proactive security practices during the deployment of containerized applications. This consideration is made in the subsequent case study.

III. CONTAINERIZINGAPACHETOMCATAPPLICATIONS

Dockerization process of Java web application with Apache Tomcat and Docker is demonstrated in this case study. The steps involved include configuring the environment, Dockerfile creation for building docker images, container execution and management of scalable deployment [3]. These are elucidated in the subsequent sections.

a. Setting Up the Development Environment

The introductory step in containerization is Docker installation in the local computing system. Installation of Docker proves straightforward because many platforms supply installation packages and platform-specific package management solutions. Apache Tomcat and Docker must be locally installed to complete tests of the application prior to containerization. Apache Tomcat which is open source can serve as the server as well as a web server with secure connections (such as Tomcat supports Secure Socket Layer). It is also a Java application that run as web NMS server that is lightweight as it achieve resource utilization and has less memory [3]. The case study adopt a basic servlet-based application as a Web Application Archive (WAR) file. The WAR file will function within the Tomcat container.

b. Adding Dockerfile

A dockerfile is needed before creating a Docker image. A Docker file and its corresponding name are packaged as *Dockerfile* [3]. An important consideration is the naming convention and the saving such that the name starts with Capital D in addition to the appended file. The name conventions not likely to work include DockerFile, dockerFile, Docker or dockerfle [7]. The case Dockerfile serves as a set of instructions upon which the Docker image is build and encapsulates Apache Tomcat alongside the Java application. Its role is to define the base image, environment configuration and steps involved in the installation and configuration of application within the container.

```
Dockerfile

# Start with the official Tomcat base image
FROM tomcat:9.0

# Set environment variables for Tomcat
ENV JAVA_OPTS="-Djava.security.egd=file:/dev/./urandom"

# Copy the WAR file to the Tomcat webapps directory
COPY ./target/myapp.war /usr/local/tomcat/webapps/

# Expose port 8080 for external access to Tomcat
EXPOSE 8080

# Start the Tomcat server
CMD ["catalina.sh", "run"]
```

The Dockerfile extracts its official Tomcat image source from Docker Hub containing an already setup Apache Tomcat platform. When the container starts, Tomcat automatically deploys the Java web application (myapp.war) through its webapps directory. Port 8080 is the exposed for incoming HTTP traffic before the default launch commands Tomcat server to start [8].

c. Building the Cocker Image

The image is then built after creating the docker file. It is built through the docker build command containing the Dockerfile and generates an image with the application and Tomcat. The execution of the command generates an image named *myapp-tomcat* and is ready for deployment. It contains self-contained environment comprising the Tomcat and the Java application.

```
bash

docker build -t myapp-tomcat .
```

d. Running the Docker Container

The docker run command initializes launch of the container. The command initiates the container, binds the necessary ports, and runs Tomcat within the containerized environment. the -d flag instructs Docker to run the container in detached mode; -p 8080:8080 binds port 8080 on the host to port 8080 on the container. When Docker successfully generates the image it names it myapp-tomcat so it becomes ready for deployment.

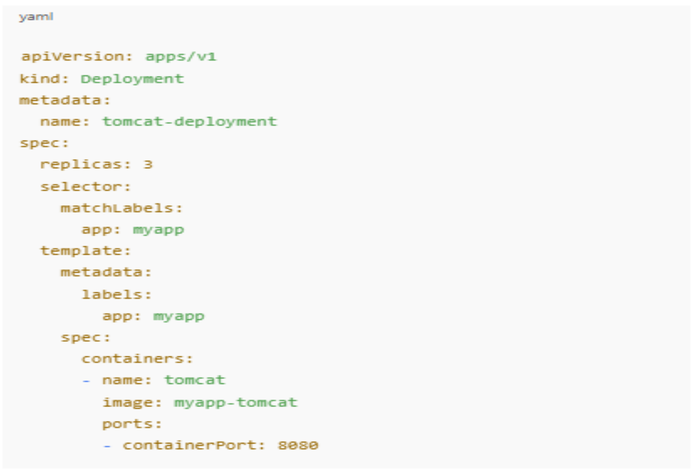
```
bash

docker run -d -p 8080:8080 myapp-tomcat
```

http://localhost:8080/myapp is created as the address to access the application.

e. Scaling and Orchestration with Kubernetes

Orchestration tools such as Kubernetes are needed to manage containers at large scales despite the simple deployment process through Docker [9]. Kubernetes can allow developers to deploy containers plus handle automatic scalability and management across different nodes. The case study utilized Kubernetes to establish multiple Tomcat container instances to manage large-scale traffic. The following Kubernetes deployment file was generated. The rationale is to manage the scaling of the Tomcat containers.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tomcat-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: tomcat
        image: myapp-tomcat
        ports:
        - containerPort: 8080
```

IV. RESULTS AND BENEFITS

The containerizing of an Apache Tomcat application through Docker resulted in substantial outcomes which affirmed enhancements of different aspects of software deployment. The main finding from using Docker for containerization is the uniformity between various
operating environments. Moving applications between development, testing, and production has traditionally grappled with varied configurations dependencies and platform setups [6]. Docker creates self-contained application packages which include the required dependencies to operate. The Tomcat application exhibited identical behavior across all targeted environments because the case study's approach solved the traditional "it works on my machine" limitations [10]. All developers, testers and operations staff in the same environment can minimize errors due to variations resulting from dissimilar testing environments [10]. The combination of Docker and Kubernetes proved crucial to orchestrate and manage Tomcat container scaling. Kubernetes deployment allowed for the automatic horizontal scaling such that application can handle fluctuations in requests without needing human interaction.

V. CONCLUSION

It is conceivable from this case that containerizing applications with Docker and Apache Tomcat create an organized and efficient deployment of Java-based web applications. Docker's capabilities reduce conflicts and environment inconsistencies, coupled with reliable Apache Tomcat platform for hosting Java web apps. The case study illustrates that blending Docker with orchestration tools such as Kubernetes can provide organizations with the ability to scale applications while managing dynamic workloads and achieving operational flexibility.

REFERENCES

[1] J. N. Acharya and A. C. Suthar, "Docker container orchestration management: A review," in *International Conference on Intelligent Vision and Computing*, Cham: Springer International Publishing, Oct. 2021, pp. 140-153.

[2] Y. A. Auliya, Y. Nurdinsyah, and D. A. Wulandari, "Performance comparison of docker and lxd with apachebench," in *Journal of Physics: Conference Series*, vol. 1211, no. 1, p. 012042, Apr. 2019.

[3]*Docker Documentation*, Docker, 2022. [Online]. Available: https://docs.docker.com/. [Accessed: Jan. 30, 2022].

[4] C. Pahl, "Containerization and the PaaS Cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24-31, 2015.

[5] M. Fowler and J. Lewis, "Microservices: A definition of this new architectural term," ThoughtWorks, 2014.

[6] T. Combe, A. Martin, and R. Di Pietro, "To docker or not to docker: A security perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54-62, 2016.

[7] Z. Lu, J. Xu, Y. Wu, T. Wang, and T. Huang, "An empirical case study on the temporary file smell in dockerfiles," *IEEE Access*, vol. 7, pp. 63650-63659, Mar. 2019.

[8] V. R. Apolinario, *Getting started with Windows Containers and Docker. Windows Containers for IT Pros: Transitioning Existing Applications to Containers for On-premises, Cloud, or Hybrid*, 2021, pp. 31-54.

[9] C. C. Chen, M. H. Hung, K. C. Lai, and Y. C. Lin, "Docker and Kubernetes," in *Industry 4.1: Intelligent Manufacturing with Zero Defects*, Aug. 2021, pp. 169-213.

[10] S. Surovich and M. Boorshtein, *Kubernetes and Docker-An Enterprise Guide: Effectively containerize applications, integrate enterprise systems, and scale applications in your enterprise*, Packt Publishing Ltd, Nov. 2020.