International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 09 | Sept - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Automated Web Server Deployment Using Docker Compose

Akshitha Katkeri
Department of Computer Science and Engineering
Assistant Professor
BNM Institute of Technology, Affliated by VTU
Bengaluru, India
akshithakatkeri@bnmit.in

Nehaa Sanchithi H
Department of Computer Science and Engineering
VII Semester, Student
BNM Institute of Technology, Affliated by VTU
Bengaluru, India
nehaasanchithi23@gmail.com

***Abstract*—The deployment and management of web servers is a critical task in modern computing environments, often requiring significant manual effort, which can lead to inconsistencies and errors across different systems. To address these challenges, this paper presents an automated approach for deploying the Nginx web server using Docker Compose. The solution leverages containerization to define services, networks, and port mappings within a single configuration file, ensuring reproducibility and consistency across environments. A shell script is developed to further automate the process, enabling quick setup, fast recovery, and seamless updates with minimal human intervention. This approach significantly reduces deployment time, enhances scalability, and simplifies integration with CI/CD pipelines. The proposed system demonstrates how Docker Compose can serve as a lightweight orchestration tool for small- to medium-scale deployments, offering a reliable and efficient alternative to complex orchestration platforms such as Kubernetes.**

***Index Terms*—Nginx, Docker Compose, Containerization, Deployment Automation, DevOps, Web Server Orchestration, CI/CD Integration, Scalability, Reliability.**

I. Introduction

In recent years, the adoption of containerization technologies has grown rapidly due to their ability to deliver scalable, portable, and efficient solutions for software deployment. Traditional methods of web server configuration, such as manual installation and setup of Nginx, are often time-consuming, error-prone, and difficult to reproduce across multiple environments. These challenges hinder the ability of organizations to maintain consistency, especially in development, testing, and production environments.

Docker, along with Docker Compose, provides a powerful solution for overcoming these challenges by allowing developers and system administrators to define and manage services, networks, and volumes in a simple configuration file. Docker Compose acts as a lightweight orchestration tool, enabling the deployment of multi-container applications with a single command, reducing manual effort while improving consistency and reliability.

Nginx, a widely used web server and reverse proxy, is commonly deployed for hosting websites, load balancing, and improving web performance. By automating its deployment using Docker Compose, organizations can streamline the setup process, reduce downtime, and ensure reproducible configurations. Additionally, this approach can be easily extended to integrate with Continuous Integration and Continuous Deployment (CI/CD) pipelines, further enhancing development workflows.

This paper focuses on automating the deployment of the Nginx web server using Docker Compose on the Fedora operating system. A deployment script is implemented to execute the entire process seamlessly, ensuring scalability, fast recovery, and simplified management. The proposed system highlights how Docker Compose can provide a reliable and efficient solution for web server deployment, serving as a practical alternative to complex orchestration platforms such as Kubernetes for small to medium-scale applications.

II. Related Work

A. Containerization and Orchestration

Containerization has emerged as a critical technology for simplifying software deployment and ensuring consistency across environments. Docker provides lightweight virtualization that enables applications to run in isolated environments, reducing dependency conflicts and improving portability [1]. To manage multi-container applications, Docker Compose was introduced as a lightweight orchestration tool, allowing developers to define services, networks, and volumes in a single configuration file [2]. Compared to Kubernetes, which is designed for large-scale deployments, studies highlight Docker Compose as more suitable for small- to medium-scale projects due to its ease of use and reduced complexity [3], [4].

B. Web Server Deployment and Automation

Nginx has become one of the most widely adopted web servers, primarily because of its ability to handle static content efficiently, act as a reverse proxy, and support load balancing. Several works emphasize its scalability and effectiveness in high-demand environments [4]. Traditional methods of deploying Nginx involve manual setup and configuration, which are error-prone and time-intensive. Recent works suggest using automation tools such as Docker Compose and Ansible to streamline deployment and reduce operational overhead [5], [6].

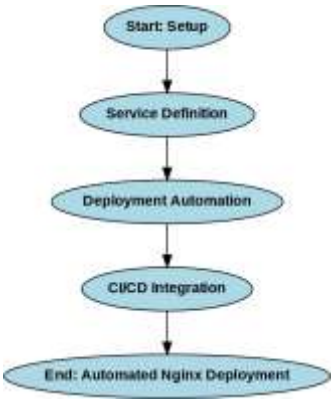

Fig. 1. Proposed Framework

C. Integration with DevOps and CI/CD Pipelines

The integration of containerization with Continuous Integration and Continuous Deployment (CI/CD) pipelines has gained significant attention, as it ensures faster release cycles, consistency, and reduced downtime [7]. While Kubernetes dominates in enterprise-scale CI/CD integration, lightweight approaches with Docker Compose offer a simpler alternative for small development teams and academic use cases. This allows for reproducible deployments, version-controlled configurations, and rapid recovery in case of system failures [2], [6], [7].

Building upon these studies, this paper focuses specifically on the automation of Nginx deployment using Docker Compose. Unlike prior works that either emphasize large-scale orchestration or manual deployments, the proposed approach demonstrates how containerization combined with automation scripts can provide reliability, scalability, and integration capabilities for small- to medium-scale environments.

III. Proposed Framework

The proposed framework introduces an automated approach for deploying the Nginx web server using Docker Compose, aiming to simplify the configuration, ensure consistency, and reduce manual errors. The architecture, as illustrated in Fig. 1, is composed of four major components: environment setup, service definition through Docker Compose, deployment automation using shell scripting, and integration with CI/CD pipelines.

A. Environment Setup

The first component of the framework focuses on preparing the execution environment. Docker Engine and Docker Compose are installed on the Fedora operating system, forming the foundation for container-based deployment. This ensures that all subsequent deployment steps are executed in a standardized and isolated environment. The setup also includes creating a custom Docker network to manage communication between containers, which is critical for scalability and multi-service applications.

B. Service Definition with Docker Compose

In the second stage, a docker-compose.yml file is created to define the Nginx service. This file specifies the base Nginx image from Docker Hub, port mappings (e.g., mapping host port 8080 to container port 80), and networking configurations. By encapsulating the entire deployment configuration within a single YAML file, the framework guarantees consistency and reproducibility across different environments. Version control can also be applied to track changes and maintain deployment history.

C. Deployment Automation via Shell Script

The third component involves a shell script (deploy.sh) that automates the entire deployment process. The script pulls the latest Nginx Docker image, runs the docker-compose up -d command to start the service in detached mode, and verifies that the containers are running as expected. This automation significantly reduces manual intervention, minimizes deployment errors, and allows for fast recovery in case of failures by simply re-running the script.

D. CI/CD Pipeline Integration

The final component focuses on the integration of the automated Nginx deployment with Continuous Integration and Continuous Deployment (CI/CD) pipelines. By embedding the docker-compose.yml file and deployment script into CI/CD workflows, updates to the Nginx configuration or web content can be deployed automatically after every code change. This not only accelerates release cycles but also ensures that deployments remain consistent across development, testing, and production environments.

The proposed framework thus provides a lightweight, scalable, and reproducible method for automating web server deployment. By combining Docker Compose with simple automation scripts, it delivers a practical alternative to complex orchestration platforms like Kubernetes, especially for small- to medium-scale applications.

IV. Implementation

The system was implemented on Fedora Linux with Docker Engine serving as the container runtime and Docker Compose as the orchestration tool. To streamline operations, a shell script (deploy.sh) was created to automate the entire deployment workflow, ensuring minimal human intervention. The implementation focused on achieving three goals: reproducibility, scalability, and simplicity, enabling the same setup to be replicated across different machines with consistent results.

A. Environment Setup

The first step involved setting up the environment by installing Docker Engine and Docker Compose. Docker Engine provides the base platform for containerization, while Docker Compose simplifies the orchestration of multi-container applications. The Fedora operating system was chosen for its robustness, compatibility with Red Hat tools, and support for

Fig. 2. Docker and Docker Compose installation on Fedora

Fig. 3. docker-compose.yml file used for defining the Nginx service

containerized workloads. After installation, the Docker daemon was enabled as a background service, ensuring containers could be launched immediately upon execution. Additionally, a custom Docker network was configured, which allows seamless communication between containers and paves the way for scaling beyond a single service if required.

B. Service Definition with Docker Compose

The heart of the deployment lies in the docker-compose.yml file, which defines how the Nginx web server should run. This file specifies: The official Nginx image pulled from Docker Hub. Port mapping (mapping host port 8080 to container port 80) to expose the service externally. Network configuration, enabling container-to-container communication.

Using this YAML configuration ensures that deployments remain consistent and version-controlled. Any modifications to the service (such as port changes or additional services like databases) can be made centrally in this file, making updates easy to manage.

C. Deployment Automation Script

To minimize manual execution, a shell script named deploy.sh was written. The script executes three primary steps:

- Pull the latest Nginx image from Docker Hub, ensuring that the deployment is always up to date.
- Run docker-compose up -d, which starts the containers in detached mode.
- Verify running services to confirm that the Nginx container has been successfully launched.

The script can be re-run at any time, making it useful for fast recovery in case of server crashes or misconfigurations.

Fig. 4. deploy.sh script for automated Nginx deployment

This approach also reduces the learning curve for beginners, as a single command is sufficient to launch the entire system.

D. Running the Deployment

Once the environment, configuration, and automation script were prepared, the deployment process was executed by running ./deploy.sh. The script initialized the Docker Compose setup, launched the Nginx container, and mapped it to the specified host port. The deployed web server could then be accessed by navigating to http://localhost:8080 on the host machine's browser. This confirmed that the deployment was successful and that the server was operational.

E. Real-Life Usefulness

The proposed implementation addresses the common challenges of manual configuration, which is often error-prone and inconsistent across environments. By relying on Docker Compose and a simple script, the deployment becomes:

- Reproducible: the same YAML file and script can be used on multiple machines with identical results.
- Scalable: additional services (such as databases or application backends) can be easily added to the same Compose file.
- Resilient: in case of failure, the service can be redeployed within seconds, reducing downtime.
- CI/CD Friendly: the script and configuration files can be integrated into pipelines, allowing automatic redeployment after code changes.

This makes the implementation suitable not only for academic learning but also for real-world scenarios, especially for small- to medium-scale organizations seeking lightweight alternatives to Kubernetes.

V. Results

The proposed framework was tested on the Fedora operating system to evaluate its performance, ease of use, and reliability. The deployment process was carried out multiple times to measure consistency and recovery speed.

Upon execution of the deploy.sh script, the Nginx container was launched successfully using Docker Compose. The web server was made accessible on http://localhost:8080, confirming that the container was running correctly and serving requests. The Nginx default welcome page (shown in Fig. 5) served as proof of a successful automated deployment.

A. Reduction in Deployment Time

Traditional manual deployment of Nginx involves installing dependencies, configuring the server, and setting up networking, which can take several minutes and requires careful

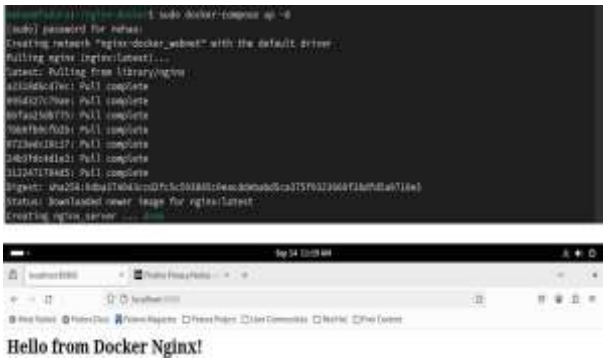

Fig. 5. Nginx default welcome page displayed in browser after automated deployment

error handling. In contrast, the automated approach using Docker Compose reduced the deployment process to less than a minute, with no manual configuration required. This demonstrates the efficiency of container-based deployment.

B. Consistency Across Environments

One of the key advantages of Docker Compose is environment consistency. The same docker-compose.yml file was tested on different machines, and each deployment produced identical results. This eliminates common problems such as "it works on my machine," which often arise in manual setups.

C. Scalability and Recovery

The system was designed with scalability in mind. Additional services such as databases or backend APIs can be integrated into the same Docker Compose file, enabling multi-service applications to be deployed easily. Moreover, in case of errors or system crashes, rerunning the automation script allowed for fast recovery, restoring the Nginx service in seconds without reinstallation.

D. Integration Potential with CI/CD Pipelines

The deployment approach also aligns well with DevOps practices. By storing the configuration files (docker-compose.yml and deploy.sh) in a version control system such as Git, deployments can be triggered automatically via CI/CD pipelines. This ensures that updates to configuration or code can be reflected immediately in production environments.

Overall, the results validate the effectiveness of the proposed framework in achieving faster, more reliable, and scalable Nginx deployments. While Kubernetes offers more advanced orchestration features, this lightweight approach using Docker Compose demonstrates significant value for small- to medium-scale environments, academic projects, and rapid prototyping scenarios.

VI. Conclusion and Future Work

This study presented an automated deployment framework for Nginx using Docker Compose, focusing on efficiency, reliability, and ease of use. The experimental results demonstrated that the proposed approach significantly reduces deployment time, ensures consistency across different environments, and enables rapid recovery in case of failures. By automating repetitive setup tasks, the framework minimizes human error and facilitates scalable deployments suitable for small- to medium-scale applications, academic projects, and prototyping.

The integration potential with CI/CD pipelines further enhances the utility of the framework, enabling continuous delivery and streamlined updates. Overall, the proposed method offers a lightweight, reproducible, and efficient alternative to traditional manual deployment methods, while also serving as a foundation for more complex containerized applications.

Through this implementation, several key learnings emerged. The research highlighted the importance of containerization in simplifying deployment workflows and ensuring environment consistency. Hands-on experimentation with Docker Compose provided practical insights into dependency management, networking configurations, and automated recovery strategies. Additionally, the study reinforced the value of systematic testing, version control, and documentation in achieving reliable and reproducible deployments. Overall, the project enhanced understanding of modern DevOps practices, bridging the gap between theoretical concepts and practical application.

A. Future Work

Several avenues exist to extend and enhance this framework:

1) Monitoring and Logging Integration: Incorporate tools such as Prometheus, Grafana, or the ELK stack to provide real-time metrics and log analysis for deployed containers.
2) Multi-Service Applications: Expand the docker-compose setup to include databases, backend APIs, and other microservices for full-stack deployment scenarios.
3) Kubernetes Migration: Evaluate the framework's migration to Kubernetes for advanced orchestration, load balancing, and scaling capabilities in larger production environments.
4) Automated Testing and Validation: Integrate automated testing scripts to verify deployment correctness and service availability post-deployment.

By implementing these extensions, the framework can evolve into a more robust, production-ready deployment solution, maintaining its core benefits of speed, reliability, and reproducibility, while also serving as a stepping stone toward more complex cloud-native and microservices architectures.

References

[1] Y. Liu and V. I. Borisov, "Containerization and automation of web application deployment: Analysis and practical implementation," Journal of Applied Informatics, vol. 17, no. 1, pp. 45–53, 2022.
[2] B. Piedade, J. P. Dias, and F. F. Correia, "Visual notations in container orchestrations: An empirical study with Docker Compose," arXiv preprint arXiv:2207.09167, 2022.
[3] Y. Mao, Y. Fu, S. Gu, S. Vhaduri, L. Cheng, and Q. Liu, "Resource management schemes for cloud-native platforms with computing containers of Docker and Kubernetes," arXiv preprint arXiv:2010.10350, 2020.
[4] Z. Wang, M. Goudarzi, J. Aryal, and R. Buyya, "Container orchestration in edge and fog computing environments for real-time IoT applications," arXiv preprint arXiv:2203.05161, 2022.

[5] A. S. Seisa, S. G. Satpute, and G. Nikolakopoulos, "Comparison between Docker and Kubernetes based edge architectures for enabling remote model predictive control for aerial robots," arXiv preprint arXiv:2212.05966, 2022.
[6] E. Ksontini, M. Mastouri, R. Khalsi, and W. Kessentini, "Refactoring for Dockerfile Quality: A Dive into Developer Practices and Automation Potential," arXiv preprint arXiv:2501.14131, Jan. 2025. arXiv
[7] M. Ccallo and A. Quispe-Quispe, "Adoption and Adaptation of CI/CD Practices in Very Small Software Development Entities: A Systematic Literature Review," arXiv preprint arXiv:2410.00623, Oct. 2024.