Open Peer Review on Qeios

Deploying WordPress in Docker: A Scalable and Secure Solution

Ramkumar Lakshminarayanan[1], Bashair khalfan Al Wahaibi[2], Fatma said Al Kasbi[2], Sara said Al Araimi[2], Abdul-Malik sulaiman Al Barwani[2], Ahmed farag Al Gheilani[2]

1 Sur University College
2 University of Technology and Applied Sciences

Funding: No specific funding was received for this work.
Potential competing interests: No potential competing interests to declare.

Abstract

Deploying WordPress in Docker offers a flexible and efficient solution for managing web applications. This approach encapsulates WordPress and its dependencies within containers, facilitating easy deployment across various environments. Embracing Docker for WordPress deployment introduces a dynamic and efficient approach to elevate web application management and fortify security. WordPress in docker provides dedicated focus on load balancing. In our study, we implemented various load balancing algorithms and evaluated the performance and behavior in loading balancing the request.

Introduction

WordPress deployment in Docker provides a flexible and efficient solution for managing web applications. This method encapsulates WordPress and its dependencies within containers, allowing for simple deployment across multiple environments. This research aims to create a Docker-based solution for deploying WordPress with a focus on scalability, security, and efficiency. The primary objectives include designing a system that can seamlessly handle increased traffic, implementing strong security measures to protect sensitive data, and ensuring data persistence across container restarts. Additionally, efficient networking, clear documentation, and adherence to best practices are key priorities. The solution will be optimized for resource usage, and load testing will be conducted to validate its performance.

Docker

Docker is a platform for developing, shipping, and running applications in containers. Containers are lightweight, portable, and self-sufficient units that run applications and their dependencies that are isolated from the host system and other containers. Docker provides a set of tools and a platform to automate the deployment and scaling of containerized applications.

NGINX

NGINX is a webserver; it is also used as Proxy and load balance UDP traffic. NGINX supports Round Robin, Least Connections, IP Hash, Generic Hash, and Least Time algorithms for load balancing.

WordPress

WordPress is an open-source platform that allows users to develop and manage websites ranging from basic blogs to eCommerce storefronts, company websites, and artistic portfolios.

Literature Review

Containers for Docker Docker is a free and open-source project that automates the quicker deployment of Linux software within portable containers. A container can run directly on the VM if the cloud offers the appropriate native container runtime [1].

Containerization in conjunction with cloud computing offers a solution for meeting large data requirements, but it necessitates accurate and adequate load-balancing systems. With growing resource utilization, the strain on servers grows tremendously, making load balancing a must [2].

WordPress is a free, open-source CMS used to create a website, application, or blog using PHP and MySQL. It is hosted on the internet. More than 60 million websites use the most popular blogging tool. WordPress's main characteristics include a plugin architecture and a template system [3].

Docker's possibility of lightweight application and dependency packaging and deployment is attractive, and it is swiftly being accepted by the Linux community and making its way into production systems [4].

The author studied software containerization with Docker and demonstrated the reasons for the growing use of Docker as a software container compared to virtual machines as well as its integration with one of the most popular content management systems, WordPress. Docker has simplified the workflows for packaging software stacks by eliminating the need for dependencies and other configurations [5].

Both WordPress and Docker have their own unique strengths and can be incredibly useful in different scenarios. WordPress is perfect for creating and managing content-driven websites, while Docker is great for streamlining the deployment and management of applications [6].

The article's author provides an [7] overview of virtualization when it comes to software deployment, virtualization offers several advantages, including process isolation and resource control. Process isolation allows software developers to make strong assumptions about the state of the system, such as the operating system configuration and the exact software dependencies required for the system.

The author used Docker to deploy and test astronomy software in this research work. The author used Docker in development and production are its ease of bundling components, promotion of re-usability and maintainability, and faster continuous integration environments [8].

In this work, the performance evaluation of three environments (bare-metal, Docker containers, and virtual machines) is investigated to understand the differences between the characteristics of each environment. Also, addressed whether container-based virtualization can solve the problems of traditional virtualization. In addition, combined Docker with OpenStack to implement a container management platform. Finally, took Hadoop deployment as an example to verify whether Docker can solve the deployment problem and save time [9].

The author presents an overview of containerized environments such as Docker for big data applications with load balancing in this research. A novel container scheduling strategy for big data applications based on Docker Swarm and Microservice architecture is proposed by the authors [10].

In this research [11], the researchers examined the Kubernetes architecture with particular attention to load balancing, an algorithm for choosing leaders, and a technique for maintaining consistency based on leaders. After they studied two issues, the leader's inherent architecture placed severe burdens on it, and the Kubernetes leader election mechanism needed to distribute the leader equitably among nodes.

This research paper [12] stresses the significance of container-based virtualization and Docker in influencing the future of cloud technology. Container-based virtualization, also known as operating system virtualization, runs many instances of an operating system on a single kernel, with the virtualization layer running as an application within the operating system.

In [13], the authors compared the performance of resource control utilizing containers and virtual machines versus natively running workloads on hardware. It analyzes workloads in systems and real-world applications such as Redis and MySQL. The objective is to comprehend the overhead caused by virtual machines (KVM) and containers (Docker) compared to non-virtualized Linux. The article compares native, container, and virtual machine environments, identifies the critical performance effect of virtualization alternatives, and shows that containers are feasible even at full server scale with low impact.

Methodology

1. Install Docker.
2. Create the Network with Docker Network.
3. Create the mysql-container1.
4. Create the wordpress-container1 – mapping port 8080.
5. Create the mysql-container2.
6. Create the wordpress-container2 – mapping port 8081
7. Create the nginx load balancing with Round Robin Algorithm.
8. Test the performance with Apache bench Tool for Round Robin, Least Connection, IP Hash and Generic Hash

Algorithm.

Results and Discussion

With the support of Apache bench Tool, Round Robin, Least Connection, IP Hash and Generic Hash Algorithm and the results are analyzed.

Round Robin Algorithm

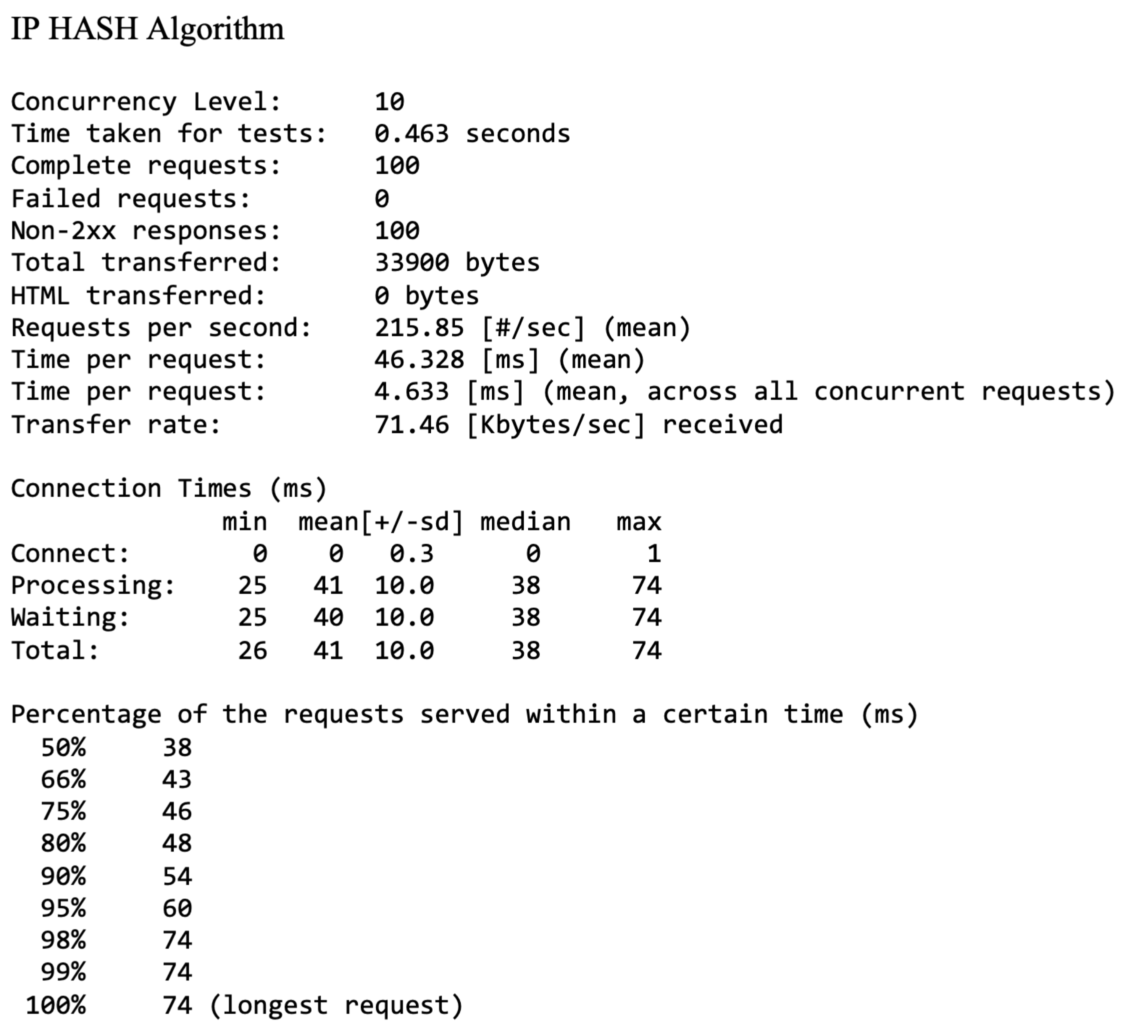
```
Concurrency Level:      10
Time taken for tests:   0.490 seconds
Complete requests:      100
Failed requests:        0
Non-2xx responses:      100
Total transferred:      33900 bytes
HTML transferred:       0 bytes
Requests per second:    204.21 [#/sec] (mean)
Time per request:       48.970 [ms] (mean)
Time per request:       4.897 [ms] (mean, across all concurrent requests)
Transfer rate:          67.60 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       1
Processing:    27   44  12.6     40      86
Waiting:       27   43  12.6     40      85
Total:         27   44  12.6     41      86

Percentage of the requests served within a certain time (ms)
  50%     41
  66%     46
  75%     50
  80%     54
  90%     64
  95%     70
  98%     78
  99%     86
 100%     86 (longest request)
```

Round Robin with Weight Algorithm

```
Concurrency Level:      10
Time taken for tests:   0.520 seconds
Complete requests:      100
Failed requests:        0
Non-2xx responses:      100
Total transferred:      33900 bytes
HTML transferred:       0 bytes
Requests per second:    192.40 [#/sec] (mean)
Time per request:       51.974 [ms] (mean)
Time per request:       5.197 [ms] (mean, across all concurrent requests)
Transfer rate:          63.70 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       1
Processing:    28   46  16.9     42     129
Waiting:       28   45  16.9     42     129
Total:         29   46  17.0     42     129

Percentage of the requests served within a certain time (ms)
  50%     42
  66%     47
  75%     51
  80%     52
  90%     60
  95%     87
  98%    112
  99%    129
 100%    129 (longest request)
```

Least Connection Algorithm

```
Concurrency Level:      10
Time taken for tests:   0.510 seconds
Complete requests:      100
Failed requests:        0
Non-2xx responses:      100
Total transferred:      33900 bytes
HTML transferred:       0 bytes
Requests per second:    195.98 [#/sec] (mean)
Time per request:       51.026 [ms] (mean)
Time per request:       5.103 [ms] (mean, across all concurrent requests)
Transfer rate:          64.88 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       1
Processing:    27   45  12.3     42      92
Waiting:       27   44  12.3     42      92
Total:         27   45  12.3     42      93

Percentage of the requests served within a certain time (ms)
  50%     42
  66%     48
  75%     52
  80%     54
  90%     61
  95%     66
  98%     78
  99%     93
 100%     93 (longest request)
```



IP HASH Algorithm

```
Concurrency Level:      10
Time taken for tests:   0.463 seconds
Complete requests:      100
Failed requests:        0
Non-2xx responses:      100
Total transferred:      33900 bytes
HTML transferred:       0 bytes
Requests per second:    215.85 [#/sec] (mean)
Time per request:       46.328 [ms] (mean)
Time per request:       4.633 [ms] (mean, across all concurrent requests)
Transfer rate:          71.46 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       1
Processing:    25   41  10.0     38      74
Waiting:       25   40  10.0     38      74
Total:         26   41  10.0     38      74

Percentage of the requests served within a certain time (ms)
  50%     38
  66%     43
  75%     46
  80%     48
  90%     54
  95%     60
  98%     74
  99%     74
 100%     74 (longest request)
```

Generic IP HASH

```
Concurrency Level:      10
Time taken for tests:   0.465 seconds
Complete requests:      100
Failed requests:        0
Non-2xx responses:      100
Total transferred:      33900 bytes
HTML transferred:       0 bytes
Requests per second:    214.91 [#/sec] (mean)
Time per request:       46.532 [ms] (mean)
Time per request:       4.653 [ms] (mean, across all concurrent requests)
Transfer rate:          71.15 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       1
Processing:    27   41   9.3     40      72
Waiting:       27   41   9.3     40      72
Total:         27   41   9.3     40      72

Percentage of the requests served within a certain time (ms)
  50%     40
  66%     45
  75%     47
  80%     51
  90%     56
  95%     59
  98%     61
  99%     72
 100%     72 (longest request)
----------------------
```

Conclusion

IP HASH achieved the highest request rate (215.85 requests/second), while Round Robin – 204.21, least connection- 195.98, Generic hash – 214.91. All algorithms achieved a completion rate of 100% with no failed requests. Processing times varied slightly between algorithms, likely due to inherent fluctuations in server load and network conditions.

Other References

- Felter, Wes, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. "An updated performance comparison of virtual machines and linux containers." In 2015 IEEE international symposium on performance analysis of system.

References

1. ^*Bernstein, David. "Containers and cloud: From lxc to docker to kubernetes." IEEE cloud computing 1, no. 3 (2014): 81-84.*
2. ^*Singh, Neelam, Yasir Hamid, Sapna Juneja, Gautam Srivastava, Gaurav Dhiman, Thippa Reddy Gadekallu, and Mohd Asif Shah. "Load balancing and service discovery using Docker Swarm for microservice based big data applications." Journal of Cloud Computing 12, no. 1 (2023): 1-9.*
3. ^*"WordPress", https://wordpress.org/documentation//. October 10, 2023.*
4. ^*Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." Linux j 239, no. 2 (2014): 2.*
5. ^*"Docker", http://hub.docker.com/. October 23, 2023*
6. ^*Di Tommaso, Paolo, Emilio Palumbo, Maria Chatzou, Pablo Prieto, Michael L. Heuer, and Cedric Notredame. "The impact of Docker containers on the performance of genomic pipelines." PeerJ 3 (2015): e1273.*
7. ^*Santos, Eddie Antonio, Carson McLean, Christopher Solinas, and Abram Hindle. "How does Docker affect energy consumption? Evaluating workloads in and out of Docker containers." Journal of Systems and Software 146 (2018): 14-25.*
8. ^*Caturano, Francesco, Nicola d'Ambrosio, Gaetano Perrone, Luigi Previdente, and Simon Pietro Romano. "ExploitWP2Docker: a Platform for Automating the Generation of Vulnerable WordPress Environments for Cyber Ranges." In 2022 International Conference on Electrical, Computer and Energy Technologies (ICECET), pp. 1-7. IEEE, 2022.*
9. ^*Morris, Dave, S. Voutsinas, Nigel C. Hambly, and Robert G. Mann. "Use of Docker for deployment and testing of astronomy software." Astronomy and computing 20 (2017): 105-119.*
10. ^*Shih, Wen-Chung, Chao-Tung Yang, Rajiv Ranjan, and Chun-I. Chiang. "Implementation and evaluation of a container management platform on Docker: Hadoop deployment as an example." Cluster Computing 24, no. 4 (2021): 3421-3430.*
11. ^*Singh, Neelam, Yasir Hamid, Sapna Juneja, Gautam Srivastava, Gaurav Dhiman, Thippa Reddy Gadekallu, and Mohd Asif Shah. "Load balancing and service discovery using Docker Swarm for microservice based big data applications." Journal of Cloud Computing 12, no. 1 (2023): 1-9.*
12. ^*Nguyen, Nguyen, and Taehong Kim. "Toward highly scalable load balancing in kubernetes clusters." IEEE Communications Magazine 58, no. 7 (2020): 78-83.*
13. ^*Singh, Sachchidanand, and Nirmala Singh. "Containers & Docker: Emerging roles & future of Cloud technology." In 2016 2nd international conference on applied and theoretical computing and communication technology (iCATccT), pp. 804-807. IEEE, 2016.*