Research Article

Kube Devops Framework Algorithm

[1]Shaily Goyal, [2]Dr. Amandeep Gill
[1]Affiliation - Research Scholar, Department of Computer Science and Engineering, JECRC University, Jaipur
[2]Affiliation - Assistant Professor, Department of Electrical Engineering, JECRC University, Jaipur
Email id – [1]shailygoyal27@gmail.com, [2]aamangill.87@gmail.com

Article History: Received: 11 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 4 June 2021

Abstract: Every business in this age is migrating in the vicinity of the Internet expeditiously. However, to hold on to the customers, holding on to the efficiency of the web services is indispensable as a slow loading website can turn out to customer dissatisfaction. Sometimes customers can be lost due to some seconds or even milliseconds. This research paper targets designing an algorithm for the framework which focuses on the performance optimization of similar web services. This algorithm adds intelligence to the framework by running pre-tests on the docker image to analyse the local response time of the application if it falls under the standard requirements or code optimization is the necessity. The algorithm is responsible for performance tuning and the migration of the application. Besides, it also solves the speed lag issue that can be caused due to network congestion. The algorithm replaces the tedious manual effort of monitoring and optimizing the performance of the application post-deployment, making the proposed framework an asset to draw to customer satisfaction.

Keywords: Pod, Kubernetes, Response Time, Performance Tuning, Application Migration.

1. Introduction

Everything in the Internet world is putting a pedal to the medal. That pops out the word 'slow' from the web dictionary. With the rapid increase in the amount of information on the web, the urge for its rapid retrieval increases. All the users expect their web link to fire up in the blink of an eye. There exist several platforms like Word Press, Kubernetes, and servers like WAMP, XAMPP, LAMP that facilitate the deployment of a website application notwithstanding additional optimizations (Ahmed, Bezemer, Chen, Hassan, & Shang, 2016). Modifications are essential in the wake of achieving the sole objective of the paper discussed in the previous paper. In contemplation of achieving a better HTTP response time, as assured, an algorithm is the main ingredient in the development of the proposed framework. Optimizing the performance of the web application could be a tedious and time-consuming task, thus adding an algorithm in the framework is the viable solution to put up to the instant results (Alsmad & Alda, 2012). This algorithm concerns driving the customers towards our framework ensuring no slow loading, hence it brings forth the customer satisfaction which comes with the speed while accessing a webpage. Further, this paper discusses how the algorithm supports the objective and what all advantages it provides to the framework to serve its purpose. The algorithm aims to tune the performance of the framework. The research paper discusses in the forthcoming sections that most of the performance monitoring tasks are performed before the real-time deployment of the application, ensuring fewer efforts in optimization. The paper also explains how the algorithm handles the network congestion issues by load balancing and scaling.

2. Literature review

Response time is defined as the amount of time from the moment when a user sends some request regarding any activity to the moment when the system states the completion of that request. The response time is used as a performance measure for an application (Arora & Bali, 2015). As it is the total time from requesting to receiving the response, it is the most helpful tool in performance testing. For example, if the speed or performance of a webpage or website is to be tested, response time is calculated.

The framework that is based on the WAMP server is conceptualized as a client/server architecture consisting of three tiers (Azzam, Alkabi, & Alsmadi, (2012). The top tier is the presentation layer that displays the information; the middle tier consists of the processing layer which performs all the logical tasks. And the bottom tier is the data tier for storing the geographic data. This framework has adopted a thin client who helps the framework to ease up in updating the data and it requires negligible resources at the client-side. The server occupies the central position as all the processing tasks are executed at the server side itself whilst the client side is only responsible for communication through an interface and displaying of the results. The technologies like servlet, common gateway interface connect the webserver to the framework's server (Bora & Bezboruah, 2015).

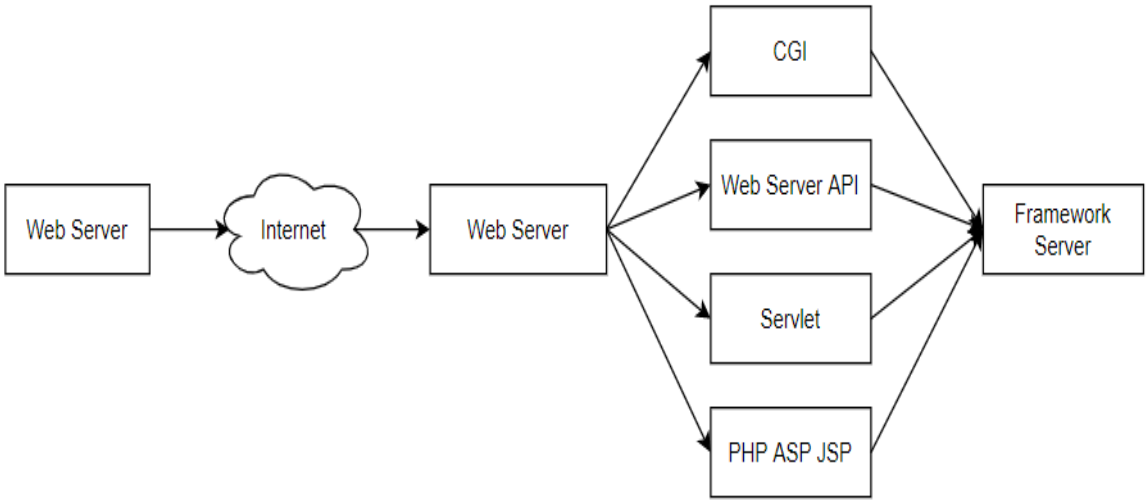

Figure 1. Flow Chart of the Framework based on WAMP

This framework has some ease in updating as the server system obtains centralized control over all the resources. It is easy to start and cheaper yet compromises local needs, advanced data formats, and most important response time (Esfandyari, 2015). As the client has a prerequisite of the completely processed data for the display, the framework requires persistent communication with the server even for simpler tasks like zoom. The proposed framework overcomes all these disadvantages in the best possible manner.

3. Methodology

The conventional method of deploying a web application on Kubernetes requires a docker file. It builds an image out of that docker file and deploys the pod containing the application; except this is a plain vanilla technical method that later requires a tedious manual effort for the finer performance of the application (Fageria & Kaushik, 2014). To inculcate a solution to this, the proposed framework has an intermediate layer that runs an algorithm that handles all the "traditionally required manual tasks". This algorithm is scripted mostly Python and partially Ansible. The algorithm looks up for all the essential performance monitoring preliminary to the actual deployment of our web application. The principal tasks performed by this algorithm are performance tuning and application migration. The algorithm supervises all the performance measures before it deploys the application on the platform. This helps in a pre-analysis of the real-time performance on the Web. When the user inputs a docker file for the creation and deployment of the web application before the file passes on to Kubernetes the algorithm intervenes. During its runtime, the algorithm builds an image from the docker file and tests for the local response time before deleting the current image from the system. This lays out the local response time of the image and provides the scope for the modification of the response time (Kelkar & Kandalgaonkar, 2015). Furthermore, it runs a pre-test for the response time analysis the same. If the local response time doesn't meet the expected response time, it means that the response time of the website in the standard form would be greater than the local (Kulkarni, Kotkar, Undale, Mankar, Mashal, & Komawar, 2015). Thus, the algorithm throws warnings stating the need for global optimization as a higher local response time doesn't work in favour of the objective. Post the warnings, its user's choice to deploy the application in the framework or not. Besides these, network congestion is another subject of concern which the algorithm has carefully handled (Kumar & Singh, 2015). For instance, if an HTTP page takes 10ms to respond and gradually the traffic increases leading the response time to around 80ms which is much greater than the threshold assumed (40ms). To deal with the delay in the response, the algorithm plans on scaling the number of pods such that the overall response for 10 users is not hindered even for 100. Ergo, the algorithm solves the congestion issues by scaling.

4. Results

Source code is explained below:

Framewire.py: The following attached source code is responsible for the deployment of the proposed framework. The working of the code begins with the creation of a docker image for the application and further the application is deployed with an Nginx web server. The code also checks for the deployment and service status. To implement the above-stated working, the python libraries like time, sub-processes and sys are imported. These libraries include the methods and variables that are required for the implementation of the algorithm into a code. The following are the steps that are executed in the given code:

- A docker image is created to build the application.
- Above this image, the k8s framework is deployed.
- The two types of storage PV and NFS storage are created.
- PVC is created from a PV. This PVC acts as a cache for a web server running in a container.
- Now, the application is deployed using the Nginx server.
- After the deployment, the status of the deployment and the service is checked.

```
#!/usr/bin/python3

import time

import subprocess

import sys

## creating docker image for application

subprocess.getoutput('docker build -t shailyfr:v1 githubURL')

# deployment on k8s framework above created image will be used here

# first create PV from NFS storage

subprocess.getoutput('kubectl apply -f pv.yml -n kube-public')

# create PVC from PV

# Note: PVC will be acting as a cache for a web server running in the container

subprocess.getoutput('kubectl apply -f pvc.yml -n kube-public')

# deploying application with Nginx webserver

subprocess.getoutput('kubectl apply -f web.yml -n kube-public')

# checking deployment

subprocess.getoutput('kubectl get deployment -n kube-public')

# checking service

subprocess.getoutput ('kubectl get svc -n kube-public')
```

Graph.py: The following code is responsible for plotting the graphs of the time per request across all concurrent connection versus request per second for all the frameworks including LAMP, WAMP, XAMPP and the proposed K8s framework. This graphical representation helps in inferring the conclusion as it makes analysis and visualization easier. The python library that is used for displaying the output result in graphical format is matplotlib. This library has pre-defined methods that help us represent the required dataset in the form of various graphs like, line graphs, histograms, pie charts etc. Below are stated the steps to process the python code to achieve the desired graph.

- The code is fed the data as input.
- This data includes the number of requests made per second in the respective frameworks, and the time taken (in milliseconds) to process that particular request.
- import matplotlib.pyplot as plt

```
req_per_seclamp=[2000,2235,3000,3688,4010,4860]

req_per_secwamp=[2003,2265,3005,3698,4020,4870]
```

```
req_per_secxamp=[2008,2285,3010,3628,4030,4880]
req_per_secshailyfr=[2010,2295,3020,3688,4040,4890]
# order is LAMP , XAMP , localhost , WAMP , framework
# to process time taken by frameworks in MS (millisecond)
time_takenlamp=[0.447,0.467,1.453,1.986,2.162,2.323]
time_takenwamp=[0.547,0.560,1.653,2.286,2.962,3.123]
time_takenxamp=[0.540,0.550,1.753,2.186,2.732,3.029]
time_takenshailyfr=[0.448,0.450,1.353,1.686,2.072,2.129]
plt.title("HTML orented webapp")
plt.ylabel("time per request across all concurrent connection")
plt.xlabel("request per second")
plt.bar(req_per_seclamp,time_takenlamp,label="LAMP")
plt.bar(req_per_secwamp,time_takenwamp,label="WAMP",color="red")
plt.bar(req_per_secxamp,time_takenxamp,label="XAMP",color="green")
plt.bar(req_per_secshailyfr,time_takenshailyfr,label="Shailyfr",color="yellow")
plt.legend()
plt.show()
```

As discussed in the paper above, the algorithm is successful in tuning the performance of the application deployed. Since most of the monitoring task is performed before the real-time deployment, the maximum possibility of modification in the code is in the hands of the user before their application enters into the world Web for real. This, in turn, saves a lot of time and effort as modifying the codes and script of a running application is a lot more difficult job. The algorithm issues warnings in case of the need for optimization, hence it doesn't become compulsory for the user to amend the code, and the deployment of the application entirely is the user's choice. Therewithal lays a major challenge, as the number of users of the web service increases, the congestion in the network increases. The HTTP request might take longer than usual in such scenarios. But as mentioned earlier, the algorithm has inbuilt a feature of scaling that provides the same response time irrespective of the number of users. The algorithm serves its purpose thusly.

Table 1. Comparison of the Proposed Framework with the Existing Frameworks

S.NO.	Parameters	Existing framework	Proposed framework
1.	Cost	High	Low
2.	Response time	High	Low
3.	Load balancing	External and Paid	Inbuilt
4.	High availability	External and Paid	Inbuilt
5.	Code storage	Not Provided	Implemented
6.	Jenkins technology	Not Provided	Implemented
7.	Portable	Very Less Chance	Extremely Portable
8.	Skills required	More	Fewer
9.	Migration	Typical	Super Easy

10.	Customer-friendly	Need More Tech Skills	Fewer Tech Skills

Table 1 below depicts all the features that make the proposed framework finer even at the lower cost levels proving the above-concluded results. It provides a clear vision of how the proposed framework outstands amongst the existing ones.

5. Conclusion

Overall, the algorithm helps better our framework than the existing frameworks by providing automated services to the framework. The inbuilt performance monitoring techniques of the framework are the outcome of the algorithm. It also provides the feature of application migration which increases the feasibility in customer service, as well as the auto scalability, which is the most fascinating characteristic. There are several parameters on basis of which the proposed framework has tested and verified to provide better results than all the existing ones. The proposed framework provides better performance than the existing ones. This conclusion can be drawn based on the parameters like cost, response time, load balancing, migration, availability, and other added technologies. The Kube Devops framework is comparatively more portable and is platform-independent.

Future Scope

The algorithm adds scalability to the proposed framework by maintaining the number of containers running. It regulates the response time of the web services running and as it fluctuates, it launches a new container or deletes the one in an already running state as per the requirement. The performance of the framework can be further modified by monitoring the slave nodes in the architecture rather than the containers. For instance, if the response time exceeds some pre-decided threshold, a new slave can be attached to the cluster and similarly if it decreases to a certain amount of time, a worker node can be detached from the cluster. Adding this feature along with the container might help avoid the creation of a large number of containers.

References

1. Ahmed T. M., Bezemer C., Chen T., Hassan A. E. & Shang W. (2016). Studying the Effectiveness of Application Performance Management (APM) Tools for Detecting Performance Regressions for Web Applications: An Experience Report, 1-12.
2. Alsmad I. & Alda S. (2012). Test Cases Reduction and Selection Optimization in Testing Web Services. Information Engineering and Electronic Business, 5, 1-8.
3. Arora I. & Bali V. (2015). A Brief Survey on Web Application Performance Testing Tools Literature Review. International Journal of Latest Trends in Engineering and Technology. 5(3), 367-375.
4. Azzam S., Alkabi M. N. & Alsmadi I. (2012). Web Services Testing Challenges and Approaches. Institute of Communication, Culture, Information and Technology (ICCIT), 291-296.
5. Bora A. & Bezboruah T. (2015). A Comparative Investigation on Implementation of RESTful versus SOAP-based Web Services. International Journal of Database Theory and Application, 8(3), 297-312.
6. Esfandyari A. (2015). A comparative study and classification on web service security testing approach. Advances in Computer Science: an International Journal, 4(4), 46-50.
7. Fageria P. & Kaushik M. (2014). Research of Load Testing and Result Based on Load runner. International Journal of Civil Engineering, 1, 1-4.
8. Kelkar D. & Kandalgaonkar K. (2015). Analysis and Comparison of Performance Testing Tools. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), 4(5), 1880-1883.
9. Kulkarni V., Kotkar P., Undale M., Mankar P., Mashal K. & Komawar K. (2015). Advanced Web Server Testing Tools. International Journal of Science and Research (IJSR), 4(10), 2064-2066.
10. Kumar R. & Singh A. J. (2015). A Comparative Study and Analysis of Web Service Testing Tools. International Journal of Computer Science and Mobile Computing, 4(1), 443-442.