Web-based Application Services Orchestration using Docker and Kubernetes

Apridan Husaeni Muharam
Department Information System
Universitas Ahmad Dahlan
Yogyakarta of Indonesia

Imam Riadi
Department Information System
Universitas Ahmad Dahlan
Yogyakarta of Indonesia

ABSTRACT

Ahmad Dahlan University comprises a multitude of institutions and bureaus, each of which maintains a website that serves as a repository of profiles and services for students and staff within the university environment. These websites are managed by the Bureau of Information Systems (BSI), which is responsible for the administration of all information technology within Ahmad Dahlan University. BSI employs a virtualisation-based shared hosting architecture. The virtualisation-based shared hosting architecture, which is employed to host the bureau's website, is prone to failure, resulting in the server becoming inaccessible, thereby rendering the website inaccessible. In order to implement the proposed solution, architectural changes are required. These changes will involve the use of containerisation and orchestration techniques, utilising Docker and Kubernetes. The process of implementing these changes will begin by containerising each existing website, which will then be orchestrated by a Kubernetes cluster. This cluster will manage each container, perform scheduling and monitoring. The outcome of this research is the establishment of a Kubernetes cluster, which is employed to host web-based bureaus within the Ahmad Dahlan University environment. This solution offers enhanced levels of availability and reliability.

General Terms

Cloud Computing

Keywords

Kubernetes, Docker, Containers, Orchestration, Web Server, Web App.

1. INTRODUCTION

The website is currently one of the most effective tools used by organisations to interact with customers [1]. The reliability of the performance and availability of information technology, such as the web, is of great importance. If the information technology used cannot run properly, the pace of business process work will be disrupted [2].

The reliability of web applications is contingent upon the reliability and availability of the web server utilized. The architecture and implementation of the architecture can influence the reliability and availability of web servers [3]. Therefore, it is essential to design and build web server architectures and infrastructures in a manner that ensures optimal reliability and availability. This can be achieved by following best practices and utilizing appropriate design principles [4].

The responsibility for information technology at Universitas Ahmad Dahlan lies with the Information Systems Bureau. The Bureau of Information Systems plays a pivotal role in the development of applications, including web applications, network infrastructure, and servers, within the university environment. All institutional or bureau websites used at Ahmad Dahlan University (UAD) employ a virtualisation architecture and shared hosting. These institutional websites are utilised as institutional profiles and administrative services for students and staff.

The virtualisation architecture with shared hosting has inherent limitations in the separation of resources and other dependencies [5]. If the supporting dependencies experience problems, it will result in the web being inaccessible or even the server being down [6]. The data obtained during observation shows that on 12-27 October 2023, there were 15 server downs caused by one of the supporting dependencies having problems, which caused the entire web to be inaccessible.

Containerisation and orchestration technologies offer improved reliability and better availability of web servers by efficiently running and managing applications in containers [7]. These technologies have the advantages of isolating applications from other applications, portability, and consistent running in all environments, whether development or production [8]. Containerisation prevents interference between applications, while orchestration allows containers to be automated, such as deployment and on-demand scalability adjustments. Furthermore, orchestrator tools such as kubenetes have the ability to automatically fix problematic containers or pod [9]. There are a number of container technologies that can be employed, including docker, podman, and CRI-O. similarly, there are a variety of orchestrator technologies that can be utilised, such as kubernetes, apache mesos, and amazon ECS. This research employs kubernetes as the orchestrator and docker as the container isolator, due to its robust developer support, extensive community, and open-source nature, which is freely modifiable [10].

2. RELATED WORKS

In 2020, Saleh Dwiyatno, Edy Rakhmat, and Oki Gustiawan conducted research on docker container-based server virtualisation. This research utilises docker containers. The objective of this research is to replace the virtualisation architecture with docker containers in order to deploy a number of applications, and to test the effectiveness of Docker in utilising CPU and memory. The results of this research demonstrate that the use of a docker container-based architecture installed on ubuntu 18.04 LTS leads to enhanced stability of deployed web applications, with CPU and memory levels remaining below reasonable limits [11].

In 2020, Karim Manaouil and Adrien Lebre conducted research by testing kubernetes in an edge environment. The objective was to ascertain the behaviour of kubernetes in that environment and to identify the concepts and architecture used by kubernetes. The results of this research will be used to determine the impact of WAN on kubernetes. The findings of this study indicate that kubernetes demonstrates reliable results on the WAN architecture [12].

In 2021, Alowolodu Olufunso Dayo conducted research comparing the performance of virtual machines with Docker containers and kubernetes orchestrators. Calculators and fibonacci sequences were run on both architectures to ascertain the relative efficiency of the two architectures. The results demonstrated that the docker container architecture with kubernetes orchestrators was more efficient than the virtual machine architecture in running these calculations [13].

Lluís Baró Cayetano (2021) conducted research on kubernetes, docker compose, microk8s, and nested LXC containers with the objective of creating a kubernetes cluster built using three different container types. This research produces a kubernetes cluster that runs in three clusters simultaneously, namely in docker compose, microk8s, and nested LXC containers clusters. It also deploys a number of applications, including adminer, GitLab, jitsi, mattermost, nginx proxy manager, NextCloud, openLDAP, openVPN, redmine, and wxiki, in the aforementioned clusters [14].

In 2023, Faudji, Weda Adistianaya Dewa, and Nasrul Firdaus conducted research on the Siakad Stimata API. They implemented a docker container-based web server cluster architecture and kubernetes orchestrator with the goal of increasing server availability, optimising application resources, and reducing the failure rate of the server. The outcome of the research was the creation of a web server with a high level of availability, achieved by increasing the number of nodes in the cluster [15].

3. METHODOLOGY

The system comprises a kubernetes cluster, which will host WordPress-based bureau web services. Each web will be packaged using docker containers and combined with Kubernetes, which will manage these containers. A total of 16 bureau webs will be packaged into containers, including the Academic and Admissions Bureau (BAA), Student and Alumni Bureau (Bimawa), Bureau of Quality Assurance (KEP), Bureau of Human Resources (BSDM), Bureau of Information Systems (BSI), Islamic Centre (IC), Research Ethics Committee (KEP), Real Work Lectures (KKN), Office of Business Affairs and Investment (KUBI), Education Development Institute (LP2), Research and Community Service Institute (LPPM), Islamic Studies Development Institute (LPSI), Professional Certification Institute (LSP), Office of International Affairs (OIA), Educational Professional Development Centre (P3K), Bureau of Facilities and Infrastructure (Sarpras).

3.1 Docker Container

Docker is a technology developed by Docker Inc. to package applications into isolated environments, referred to as containers. This approach allows for the separation of applications and other dependencies from the environment within the container [16]. Containers offer a high level of consistency, ensuring that the environment remains consistent across different contexts, including the development and production environments [17]. Prior to the advent of containers, the transfer of applications from the development environment to the production environment often resulted in inconsistency issues. While the developed application appeared to function correctly within the development environment, it would sometimes malfunction when deployed in the production environment due to the presence of different supporting dependencies [18].

Docker comprises several principal components, the most significant of which is the docker engine. This is the core of docker and is useful for the creation and management of containers. A container is a runtime instance of an image that runs on top of the Docker engine, which runs using the host's operation system. Dockerhub is a registry that can be used publicly or can be installed on a private server.

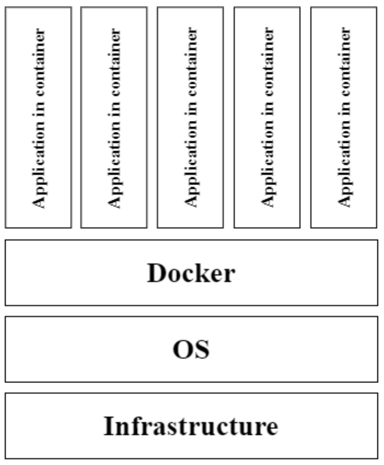


Figure 1: Docker architecture

Figure 1 provides an overview of the Docker container architecture. The Docker engine runs on top of the host operating system, which in turn supports the execution of applications in containers. This architectural approach is similar to the concept of virtualisation, but differs in that it is replaced by container engines such as docker.

3.2 Kubernetes

Kubernetes is a container orchestration tool that can be combined with docker containers to enhance the efficiency of container management [7]. Kubernetes automates the deployment, start-up, scheduling, and monitoring of containers, thereby reducing the manual effort required for container management [19]. Containers are run in pods by Kubernetes, preventing them from running independently [20].

Kubernetes is comprised of five primary objects: pods, application containers, services, volumes, and namespaces. Pods are the smallest components of kubernetes, within which application containers reside [21]. Services are entities that facilitate the exposure of pods within a cluster. Volumes are kubernetes objects that store data. Namespaces facilitate the sharing of resources within a cluster, even in the event of a pod failure or deletion. Containers, which are applications running within a pod, are designed to be resilient, ensuring that data is not lost in the event of a failure. Nodes, which are hosts or worker machines in the kubernetes environment, are responsible for executing containers and providing the necessary resources [9].

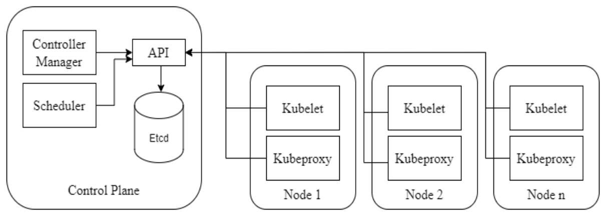


Figure 2: Kubernetes components

The kubernetes cluster comprises at least two nodes, each of which serves as a master node and worker node, as illustrated in Figure 2. The master node acts as a command centre for worker nodes, while the worker node is responsible for executing tasks assigned by the master node. In order to fulfil its duties, kubernetes is comprised of a number of key components. These include the kubernetes API, which serves as an interface for all operations within the cluster. The etcd system acts as a distributed data storage centre, while the scheduler is responsible for managing pod scheduling against nodes within the cluster. Finally, the controller manager ensures that the cluster runs in accordance with the configuration that has been set.

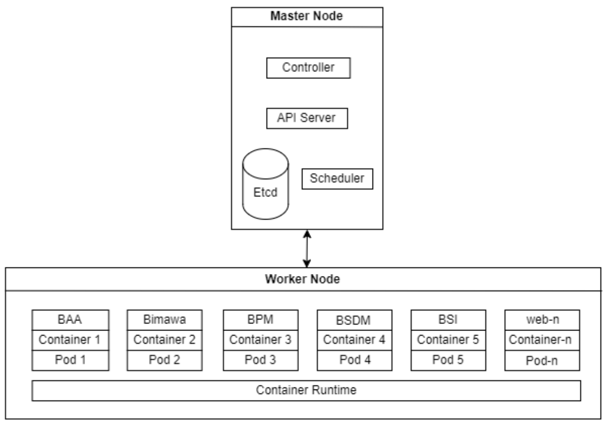


Figure 3: Cluster architecture

Figure 3 illustrates the architecture of kubernetes, particularly on the worker node. In this instance, the container manager, kubernetes, runs on the host operating system, while kubernetes also runs the pod, which is instructed to do so by the worker node based on the configuration that has been set before. The pod contains a container in which there is an application running. The cluster will be constructed using two nodes, comprising a master node and a worker node.

3.3 Data Collection Methods

The objective of data collection is to gain insight into the requirements of the system that will be deployed in the kubernetes cluster and to ascertain the current state of the operational system.

3.3.1 Observation

Observation is an activity that aims to observe the behaviour of the research object under study. In the context of this research, observations are made to determine the performance of the system currently in use and the results of these observations will be used as study materials in the research.

3.3.2 Literature study

The objective of a literature study is to examine a range of written sources on the subject of container technology and orchestration. The findings of such a study can then be used as a reference for subsequent experiments.

3.3.3 Experiment

The experimental method is a method used to test a hypothesis by controlling variables in a well-defined manner. This allows for the measurement of results and the generation of accurate data.

3.4 Implementation

The implementation process is conducted in a series of stages, as illustrated in Figure 4.

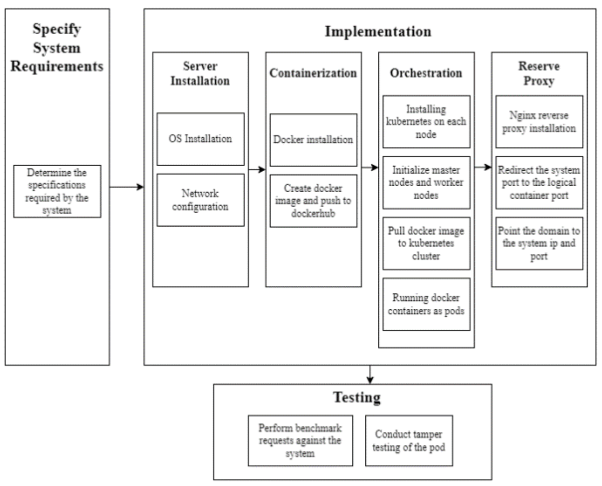

Specify System Requirements
Determine the specifications required by the system
Implementation
Server Installation
OS Installation
Network configuration
Containerization
Docker installation
Create docker image and push to dockerhub
Orchestration
Installing kubernetes on each node
Initialize master nodes and worker nodes
Pull docker image to kubernetes cluster
Running docker containers as pods
Reserve Proxy
Nginx reverse proxy installation
Redirect the system port to the logical container port
Point the domain to the system ip and port
Testing
Perform benchmark requests against the system
Conduct tamper testing of the pod

Figure 4: Stages of implementation

3.4.1 Specify System Requirements

The specification of the system to be used by the cluster is determined by considering the number of users who access the web in a given period, the size of each web file, the size of the database, and the minimum recommended specifications for the system to be used in the cluster [22].

3.4.2 Implementation

The implementation process is concerned with the installation of orchestration technology (kubernetes) and the docker container on a cluster that has been prepared and is capable of running well by orchestrating the existing containers.

Activity 1: Server Installation

The depelopment of a kubernetes cluster necessitates the presence of at least two hosts, which will serve as the master node and worker node, respectively [23]. While the operating system utilized on the two nodes may differ, it is recommended that they utilize the same operating system for the sake of ease of configuration. Once the operating system has been installed, the network configuration must be carried out in order for the two nodes to be able to communicate with each other. The two nodes can be configured to use the same LAN or even different ones, although it is preferable to place them on the same LAN for the sake of computing speed.

Activity 2: Containerisation

This stage involves the packaging of the application into a container. This process begins with the creation of a Dockerfile, which contains the configuration of the supporting dependencies of the application. Once the dockerfile has been executed, an image is formed that is ready to be pushed to the Dockerhub registry. This stage is completed when one wishes to create their own custom image, such as for a homemade application, for a commonly used application or tool, such as nginx or MySQL, which are already available on dockerhub [16].

Activity 3: Orchestration

Orchestration is the process of managing, automating, and operating containers. The orchestration tool employed in this research is Kubernetes. Each application that has been packaged into a container is pulled into the Kubernetes cluster, where it will run as a pod. The pod is exposed, and its service type is adjusted to align with the needs of the user. Among the service types in kubernetes are NodePort, clusterIP, and LoadBalancer. Each pod-exposed service within the Kubernetes cluster is assigned a virtual IP address and port, which can be accessed from both within and outside the cluster [24].

Activity 4: Reverse Proxy
A reverse proxy is a server that acts as an intermediary between the client and the backend server. When an incoming request is received, the reverse proxy will receive and forward it to the backend server [25]. In the context of this research, the backend server is the Kubernetes cluster, and the reverse proxy used is nginx. The configuration of Nginx involves entering the port and IP address used by each container in the pod. This enables Nginx to direct client requests to the configured port and IP address, thus facilitating access to the desired web content.

3.4.3 Testing
The objective of testing is to ascertain the extent to which the kubernetes cluster is capable of handling the load imposed by the client. This is achieved through the use of benchmarking tools such as apache beach, which are employed to simulate the simultaneous access of a defined number of users to the cluster. The results of the tests conducted were used to ascertain the number of successful requests and the time taken to respond to requests received [26].

4. RESULTS AND DISCUSSION
This research employs a containerised approach to web-based applications, utilising docker containers and orchestrated with kubernetes, The Kubernetes cluster, which has been constructed, comprises two nodes that have been customised to meet the specific requirements of. This represents a departure from the conventional shared hosting virtualisation architecture previously employed. The web application is based on word press, which has been adapted to align with the institution's profile and to benefit students and staff at Ahmad Dahlan University.

4.1 System Specification
The system specifications utilised by the two nodes were determined by observing the number of users, the size of web files, the size of the database, and calculating the minimum standard specifications required by kubernetes. The high number of users and the volume of traffic necessitate that the system be equipped with sufficient resources and be capable of processing the requests made by users to the system. Each of these parameters will be taken into consideration in order to determine the specifications that will be used by the kubenets cluster that will be created.

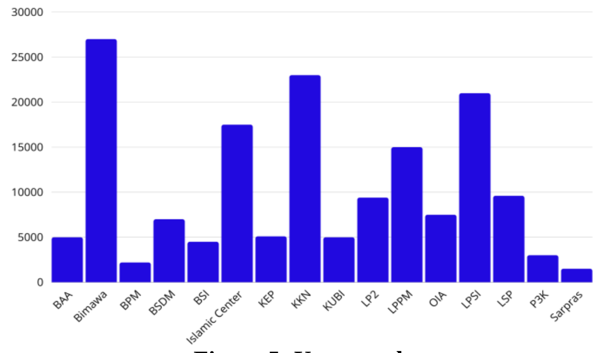


Figure 5: User graph

Figure 5 presents a graph of web user data from web bureaus at Ahmad Dahlan University, spanning the period from January to March 2024. The total number of users is in excess of 80,000. The Bimawa web ranks highest with a total of 27,000 users.

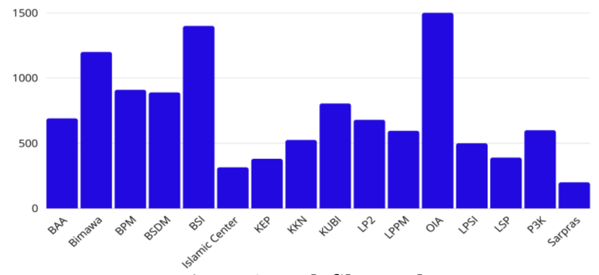


Figure 6: Web file graph

Figure 6 presents a graph illustrating the size of the web files to be containerised in megabytes. The total average file size of these web files is 734,125 megabytes. The OIA (Office of International) web occupies the largest position, while the web with the smallest file size is the Sarpras web (Sarana and Prasarana), with a total file size of 200 megabytes.

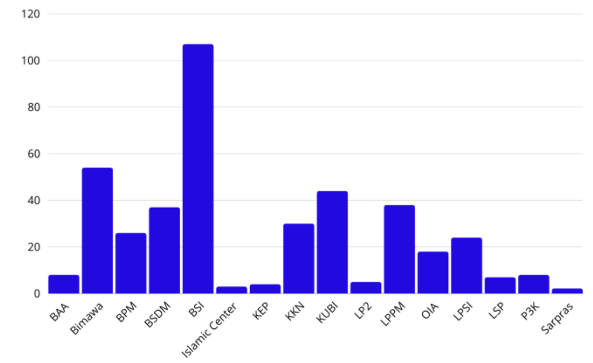


Figure 7: Database graph

Figure 7 presents a graph illustrating the size of the web database in megabytes, with an overall average of 25.98 megabytes. The BSI (Bureau of Information Systems) The web database occupies the largest position, with a size of 105 megabytes. In contrast, the Sarpras (Sarana and Prasarana) web database is the smallest, with a total file size of 2.2 megabytes.

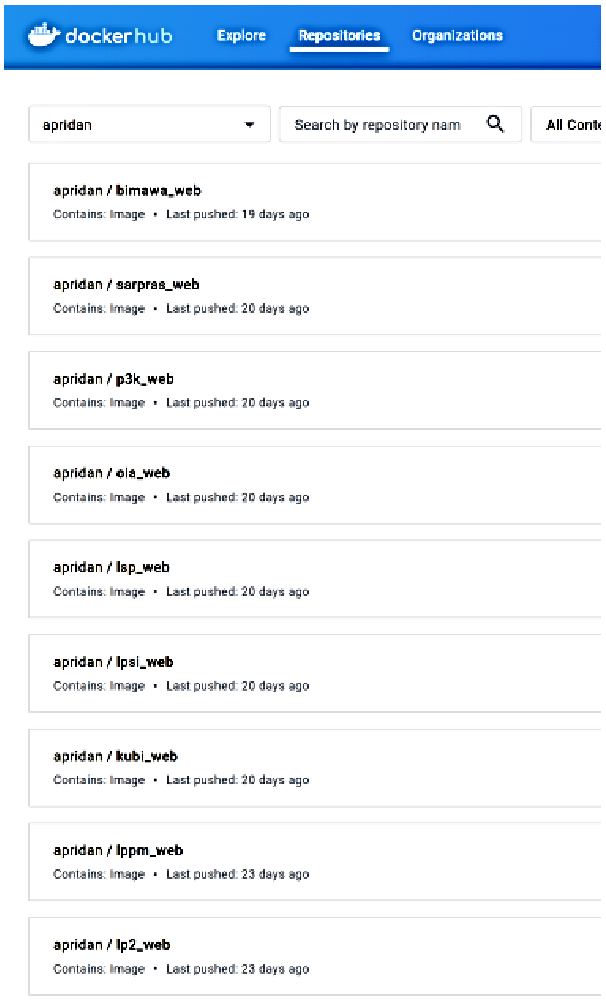
In consideration of the number of users, the size of web files and the size of databases associated with each existing web, the system specifications employed are as follows, as detailed in Table 1.

Table 1. Cluster Spesification

Host	CPU	RAM (GB)	Storage (GB)
Master Node	4	8	40
Worker Node 1	4	16	80

4.2 Containerisation
The entire existing web bureau is containerised using a Docker container. This is achieved by creating a Dockerfile that defines the dependencies that will be used by the existing web. As the application to be deployed is a web-based application, the Docker file must be defined to install a web server. Furthermore, a command is also needed to define the PHP

version if PHP is to be used. Once the image has been created, it is ready to be pushed to the Dockerhub registry container.



apridan / bimawa_web
Contains: Image • Last pushed: 19 days ago

apridan / sarpras_web
Contains: Image • Last pushed: 20 days ago

apridan / p3k_web
Contains: Image • Last pushed: 20 days ago

apridan / oia_web
Contains: Image • Last pushed: 20 days ago

apridan / lsp_web
Contains: Image • Last pushed: 20 days ago

apridan / lpsi_web
Contains: Image • Last pushed: 20 days ago

apridan / kubi_web
Contains: Image • Last pushed: 20 days ago

apridan / lppm_web
Contains: Image • Last pushed: 23 days ago

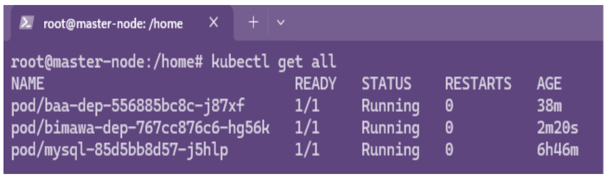
apridan / lp2_web
Contains: Image • Last pushed: 23 days ago

Figure 8: Dockerhub

Dockerhub is an official registry created by docker for the storage of images created by users. These images can be accessed from various environments, including the Kubernetes cluster used for the production environment. Figure 8 presents the creation of the entire web in the form of an image, which has been subsequently pushed to the Dockerhub. Each web is created as a new repository, with the objective of facilitating the updating of each web version and enabling its deployment to the Kubernetes cluster.

4.3 Orchestration

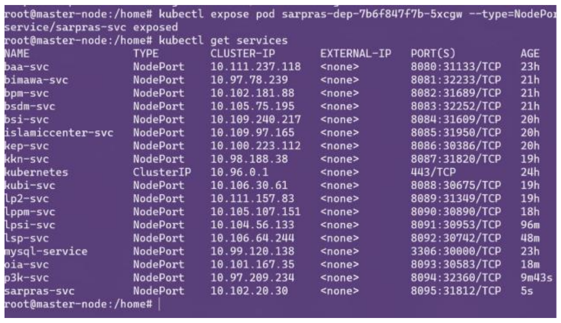
The image stored in the dockerhub registry is imported into the Kubernetes environment for orchestration. To transfer the image from the dockerhub to the kubernetes cluster environment, a YAML file is required that defines the rules that will subsequently serve as a reference for kubernetes. The objective is to sell containers as pods in the cluster. The contents of the YAML file are commands to pull images from dockerhub by defining tags and repository names that have been created, defining replicasets, and defining the internal port exposure of the container to be used. Each web has its own YAML file, with the same content but differing repository names and tags..



```
root@master-node:/home# kubectl get all
NAME                            READY   STATUS    RESTARTS   AGE
pod/baa-dep-556885bc8c-j87xf    1/1     Running   0          38m
pod/bimawa-dep-767cc876c6-hg56k 1/1     Running   0          2m20s
pod/mysql-85d5bb8d57-j5hlp      1/1     Running   0          6h46m
```

Figure 9: Pod

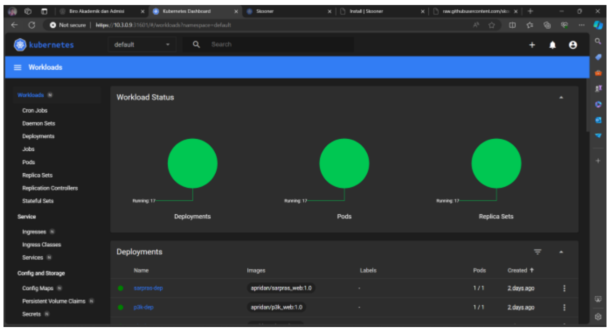
Upon execution of the YAML file, pods will be formed. The pod status can be ascertained via the kubctl get pods command. This command will display the pod name, pod status, number of restarts, and age for each pod in the cluster, as illustrated in Figure 9.



```
root@master-node:/home# kubectl expose pod sarpras-dep-7b6f847f7b-5xcgw --type=NodePor
service/sarpras-svc exposed
root@master-node:/home# kubectl get services
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
baa-svc             NodePort    10.111.237.118   <none>        8080:31133/TCP   23h
bimawa-svc          NodePort    10.97.78.239     <none>        8081:32233/TCP   21h
bpm-svc             NodePort    10.102.181.88    <none>        8082:31689/TCP   21h
bsdm-svc            NodePort    10.105.75.195    <none>        8083:32252/TCP   21h
bsi-svc             NodePort    10.109.240.217   <none>        8084:31609/TCP   20h
islamiccenter-svc   NodePort    10.109.97.165    <none>        8085:31950/TCP   20h
kep-svc             NodePort    10.100.223.112   <none>        8086:30386/TCP   20h
kkn-svc             NodePort    10.98.188.38     <none>        8087:31820/TCP   19h
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP          24h
kubi-svc            NodePort    10.106.30.61     <none>        8088:30675/TCP   19h
lp2-svc             NodePort    10.111.157.83    <none>        8089:31349/TCP   19h
lppm-svc            NodePort    10.105.107.151   <none>        8090:30890/TCP   18h
lpsi-svc            NodePort    10.104.56.133    <none>        8091:30953/TCP   96m
lsp-svc             NodePort    10.106.64.244    <none>        8092:30742/TCP   48m
mysql-service       NodePort    10.99.120.138    <none>        3306:30000/TCP   23h
oia-svc             NodePort    10.101.167.35    <none>        8093:30583/TCP   18m
p3k-svc             NodePort    10.97.209.234    <none>        8094:32360/TCP   9m43s
sarpras-svc         NodePort    10.102.20.30     <none>        8095:31812/TCP   5s
root@master-node:/home# |
```

Figure 10: Service

Pods that are functioning optimally are prepared for exposure. The exposure process determines which services will be utilized by the pod. Each web is exposed with the NodePort service, enabling it to be accessed from outside the cluster as illustrate in Figure 10. Each pod is assigned its own internal IP and port, or what is referred to as a cluster IP. If the pod is exposed using NodePort, an external port will be present outside the cluster. The port is a means of accessing the container from outside the cluster using the IP address of the host machine.



Name	Images	Labels	Pods	Created ↑
sarpras-dep	apridan/sarpras_web:1.0	-	1 / 1	2 days ago
p3k-dep	apridan/p3k_web:1.0	-	1 / 1	2 days ago

Figure 11: Kubernetes Dashboard

To facilitate cluster monitoring, a dashboard is used, the dashboard used is kubenetes dashboard, actually there are many options for dashboards including portainer, skooner, konstelle, and other. Inside the dashboard informs the number of deploymenets, pods, replicasets running, and other important information. The number of pods in this cluster is 17, with 16 web-bureau pods and one database pod as illustrate in Figure 11.

The database in question is the MySQL database, which is run as a pod in the cluster in a manner analogous to other pods. However, there is a distinction to be made between this and other pods, in that there is no need to create an image, as the MySQL image has already been created by its own development team and stored in the dockerhub. All that is required is the creation of a YAML file to pull the image into the cluster and expose it.

4.4 Reverse Proxy

Nginx is a web server that has a reverse proxy feature. A reverse proxy is configured to direct requests from clients to the cluster or web destinations in the cluster. Each web container within the cluster is placed in a pod, and the pod is assigned a unique external port that can be utilized by the Nginx reverse proxy to access the container's services and forward them to the client. In the configuration of the reverse proxy, the domain of each web container in the cluster is also embedded.

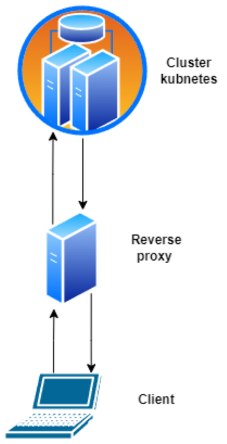


Figure 12: Proccess Reverse Proxy

Figure 12 illustrates the data flow when a client initiates a request. The client sends a request to the reverse proxy server, which will receive the request and forward it to the backend server. The latter is referred to in this research as the Kubernetes cluster. The cluster responds to the request and sends data according to the client's request, which is mediated by the reverse proxy server. Subsequently, the reverse proxy server transmits the response received from the cluster to the client. This mechanism enables domains to be directed to the reverse proxy server, and no directly to the cluster.

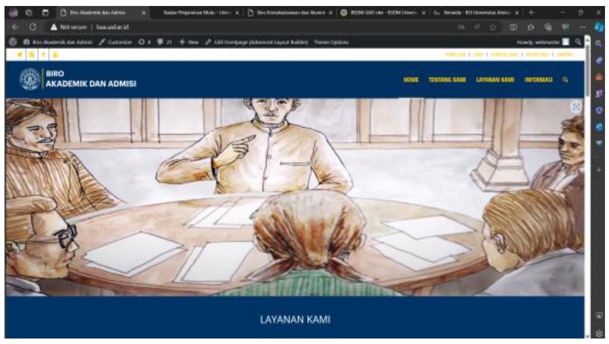


Figure 13: BAA website

Figure 13 represents the BAA (Academic and Admissions Bureau) web page, which is accessed by the client through the reverse proxy server.

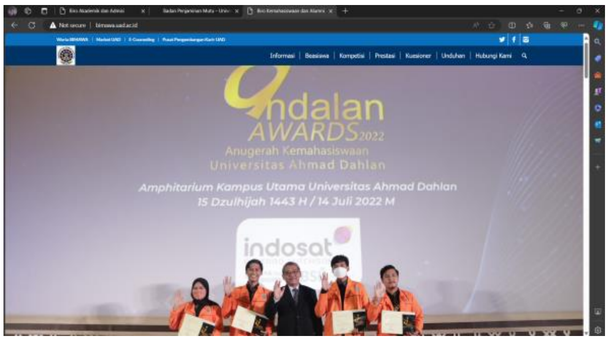

Informasi | Beasiswa | Kompetisi | Prestasi | Kuesioner | Unduhan | Hubungi Kami
Andalan AWARDS 2022
Anugerah Kemahasiswaan Universitas Ahmad Dahlan
Amphitarium Kampus Utama Universitas Ahmad Dahlan
15 Dzulhijah 1443 H / 14 Juli 2022 M

Figure 14: Website Bimawa

Figure 14 represents the Bimawa (Student and Alumni Bureau) web page, which is accessed by the client through the reverse proxy server.

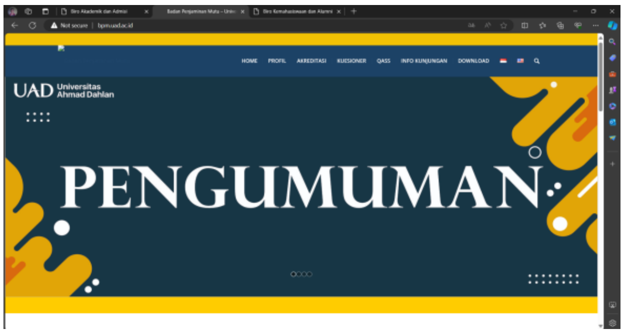


Figure 15: Website BPM

Figure 15 represents the BPM (Bureau of Quality Assurance) web page, which is accessed by the client through the reverse proxy server.

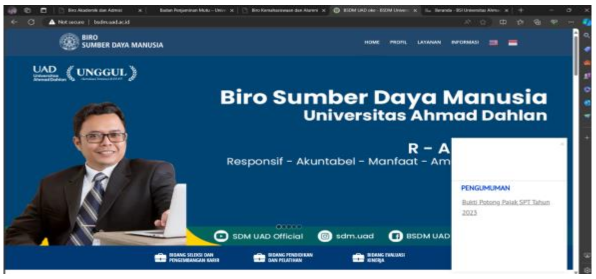


Figure 16: Website BSDM

Figure 16 represents the BSDM (Bureau of Human Resources) web page, which is accessed from the client through the reverse proxy server.

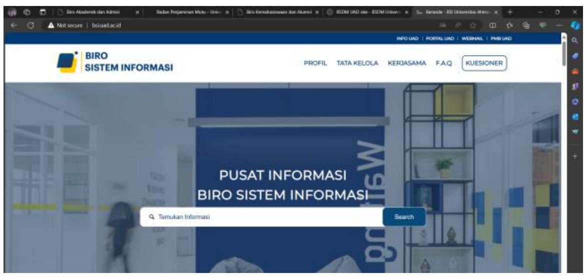


Figure 17: Website BSI

Figure 17 illustrates the BSI (Bureau of Information Systems) web page, accessed from the client via the reverse proxy server. The same method has been employed to access the websites of the Islamic Centre (IC), Research Ethics Committee (KEP), Real Work Lectures (KKN), Office of Business Affairs and Investment (KUBI), Education Development Institute (LP2), Research and The Community Service Institute (LPPM), Islamic Studies Development Institute (LPSI), Professional Certification Institute (LSP), Office of International Affairs (OIA), Educational Professional Development Centre (P3K), and Bureau of Facilities and Infrastructure (Sarpras) are also included. The results obtained demonstrate that the web can be accessed properly, indicating that all configurations have been successfully implemented and that the web is now ready to be accessed by users.

4.5 Testing

Two distinct types of tests are employed for the purposes of evaluating the performance of the kubernetes cluster. The first test assesses the cluster's ability to process a specified load of requests in a timely manner. This is accomplished by measuring the time required to process requests submitted by a client. The second test involves disrupting the pods within the cluster to ascertain the cluster's resilience in the face of such disturbances.

4.5.1 Load testing

Load testing is conducted by sending a number of requests simultaneously to the cluster. The tool used is Apache Bench, with a request delivery mechanism of 100, 500, and 1000 requests. The time required to respond to requests is calculated.

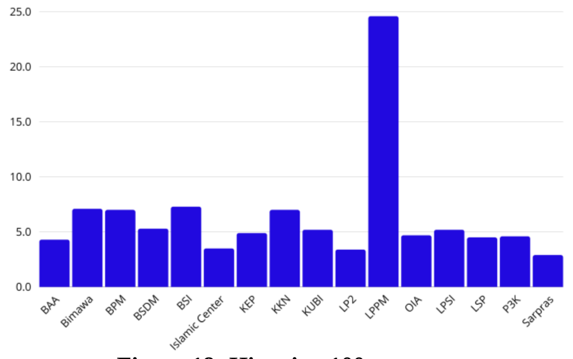


Figure 18: Hit using 100 requests

Figure 18 presents presents a graph of the outcomes of a test conducted on 100 requests to all websites in the cluster, with the results expressed in seconds. The results of the tests indicate that the Sarpras web (Bureau of Facilities and Infrastructure) is the fastest to respond to requests, with an average time of 2.9 seconds. In contrast, the LPPM web (Institute for Research and Community Service) is the slowest, with an average time of 34.6 seconds. The average time required by the entire web with a 100 request mechanism is 6.6 seconds.

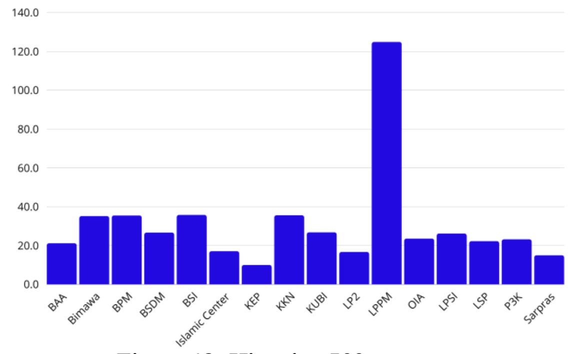


Figure 19: Hit using 500 requests

Figure 19 presents presents a graph of the results of testing 500 requests to all websites in the cluster in seconds. The results of the test demonstrate that the KEP (Research Ethics Committee) website is the fastest to respond to requests, with an average response time of 10 seconds. In contrast, the LPPM (Institute for Research and Community Service) website is the slowest, with an average response time of 124.9 seconds. The average time required by the entire web with a 500 request mechanism is 29.6 seconds.

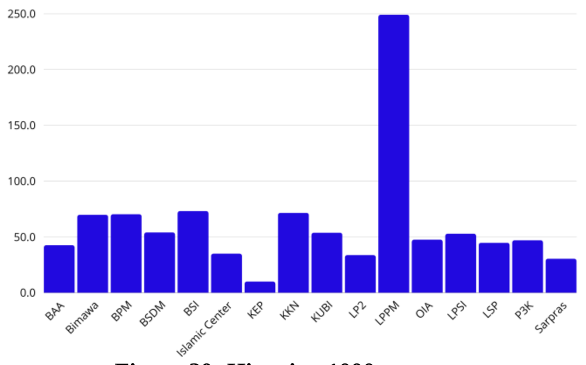
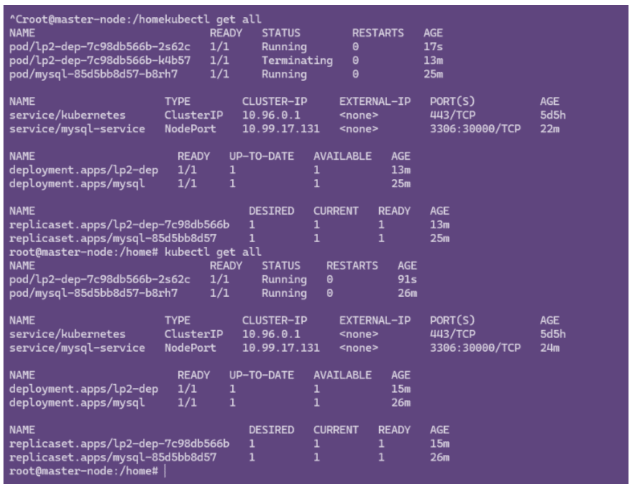


Figure 20: Hit using 1000 requests

Figure 20 presents a graph of the results of testing 1000 requests to all websites in the cluster in seconds. The results of the test indicate that the KEP (Research Ethics Committee) web is the fastest to respond to requests, with an average time of 10 seconds. In contrast, the LPPM web (Institute for Research and Community Service) is the slowest, with an average time of 249.3 seconds. The average time required by the entire web with a 1000 request mechanism is 58.2 seconds.

4.5.2 Pod resistance testing

The test is conducted by interrupting a running pod by deliberately removing it from the cluster, by attempting to delete it intentionally and observing the Kubernetes response to this action.



```
^Croot@master-node:/homekubectl get all
NAME                            READY   STATUS        RESTARTS   AGE
pod/lp2-dep-7c98db566b-2s62c    1/1     Running       0          17s
pod/lp2-dep-7c98db566b-k4b57    1/1     Terminating   0          13m
pod/mysql-85d5bb8d57-b8rh7      1/1     Running       0          25m

NAME                    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/kubernetes      ClusterIP   10.96.0.1      <none>        443/TCP          5d5h
service/mysql-service   NodePort    10.99.17.131   <none>        3306:30000/TCP   22m

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/lp2-dep    1/1     1            1           13m
deployment.apps/mysql      1/1     1            1           25m

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/lp2-dep-7c98db566b    1         1         1       13m
replicaset.apps/mysql-85d5bb8d57      1         1         1       25m
root@master-node:/home# kubectl get all
NAME                            READY   STATUS    RESTARTS   AGE
pod/lp2-dep-7c98db566b-2s62c    1/1     Running   0          91s
pod/mysql-85d5bb8d57-b8rh7      1/1     Running   0          26m

NAME                    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/kubernetes      ClusterIP   10.96.0.1      <none>        443/TCP          5d5h
service/mysql-service   NodePort    10.99.17.131   <none>        3306:30000/TCP   24m

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/lp2-dep    1/1     1            1           15m
deployment.apps/mysql      1/1     1            1           26m

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/lp2-dep-7c98db566b    1         1         1       15m
replicaset.apps/mysql-85d5bb8d57      1         1         1       26m
root@master-node:/home# |
```

Figure 21: Pod resistance

Figure 21 illustrates that the LP2 pod is in a terminating state, which is an abnormal status. This indicates that there is a disturbance to the pod. Kubernetes then restores the pod after a period of waiting. Once the pod is restored, it is running properly. The status of the pod has returned to running. Kubernetes responds that there is an irregularity in the cluster. It then attempts to replace the pod by restarting it according to the previous deployment in the YAML file. This demonstrates that Kubernetes is capable of managing containers automatically.

5. CONCLUSION

The implementation of a Docker containerisation architecture and Kubernetes orchestrator can be considered to increase the level of availability of web servers used by web-web bureaus in the Ahmad Dahlan University environment. Each web-web bureau is packaged using Docker containers separately from each other, which allows the web to run consistently in every environment, whether it is in a development environment or a production environment. Each container is managed by a container orchestrator, called Kubernetes. Kubernetes oversees the resources required by the container, coordinates scheduling, and facilitates container deployment, thereby optimizing efficiency. The test results demonstrate that the Kubernetes cluster is capable of passing load testing with the mechanism of sending requests of 100, 500, and 1000 requests alternately to the entire web well without any response failures. Additionally, pod reliability testing is conducted by disrupting pod performance. In such instances, Kubernetes is capable of responding to any abnormalities that may arise, including the replacement of a failed pod with a new one, as per the configured deployment.

6. REFERENCES

[1] Anista&Edy, "Analisis dan Pengembangan Sistem Informasi Manajemen Sragen," *Jurnal Sainstech Politeknik Indonusa Surakarta, 6, 1–8,* 2020.

[2] Listiani, I, "Analisis Pentingnya Sistem Informasi Manajemen dalam Teknologi Informasi dan Komunikasi Saat Ini," *Informasi, Teknologi Dan Komunikasi, 1, 1–15,* 2021.

[3] Lukman & Yudhiastari, Mayang, "Analisis Kinerja Web Server Apache Dan Litespeed Menggunakan Httperf Pada Virtual Private Server (VPS)," *Jurnal Teknologi Informasi, 16(2), 1-15,* 2021.

[4] Ramsari, N., & Ginanjar, A, "Implementasi Infrastruktur Server Berbasis Cloud Computing Untuk Web Service Berbasis Teknologi Google Cloud Platform*," Conference Senatik STT Adisutjipto Yogyakarta*, 2022, https://doi.org/10.28989/senatik.v7i0.472.

[5] Arsa, I. G. N. W., & Hendrawan, I. N. R, "Analisis Konsolidasi Server dengan Virtualisasi Menggunakan Proxmox VE*," Jurnal Eksplora Informatika, 13, 1-15,* 2020.

[6] Chandra, A. Y, "Analisis Performansi Antara Apache & Nginx Web Server dalam Menangani Client Request," *Jurnal Sistem dan Informatika (JSI), 48, 1-15,* 2020.

[7] Mukaj, Jon, "Containerization: Revolutionizing Software Development and Deployment Through Microservices Architecture Using Docker and Kubernetes," *Thesis, Epoka University,* 2023, doi:10.13140/RG.2.2.23804.51841.

[8] Felani, R., Al Azam, M. N., Adi, D. P., Widodo, A., & Gumelar, A. B, "Optimizing Virtual Resources Management using Docker on Cloud Applications," *Fakultas Ilmu Komputer, Universitas Narotama, Surabaya, Indonesia,* 2023.

[9] Senjab, K., Abbas, S., Ahmed, N., & Khan, A. u. R, "A survey of Kubernetes scheduling algorithms," *Journal of Cloud Computing: Advances, Systems and Applications,* 2023, doi: 10.1186/s13677-023-00471-1.

[10] Kuncoro, G., Cristanto, A., & Purbo, O. W, "Kubernetes untuk Pemula," *Penerbit Andi*, 2023.

[11] Dwiyatno, S., Rachmat, E., Sari, A. P., & Gustiawan, O, "Implementasi Virtualisasi Server Berbasis Docker Container," *Prosisko: Jurnal Pengembangan Riset dan Observasi Sistem Komputer 7(2), 165–175,* 2020, https://doi.org/10.30656/prosisko.v7i2.2520.

[12] Manaouil, K., & Lebre, A, "Kubernetes and the Edge?," *Research Report RR-9370, Inria Rennes - Bretagne Atlantique, pp. 19. hal-02972686v2*, 2020.

[13] Dayo, A. O, "A Multi-Containerized Application using Docker Containers and Kubernetes Clusters," *International Journal of Computer Applications, 183(44), 55–60*, 2021, https://doi.org/10.5120/ijca2021921843.

[14] Cayetano, L. B, "Creation of a Kubernetes Infrastructure," *January,* 2021, https://upcommons.upc.edu/handle/2117/344394.

[15] Faudji, Dewa, W. A., & Firdaus, N, "Implementasi Cluster Web Server Dinamis Berbasis Operating System-Level Virtualization Menggunakan Docker dan Kubernetes pada API Siakad Stimata," *Sistem Informasi, STMIK PPKIA Pradnya Paramita, Malang,* 2023.

[16] Kurniawan, M. I., & Dedi, R. T., "Virtualisasi Dengan Docker," STMIK Bina Sarana Global, 2020.

[17] Umar, I. A., Nurhadi, & Syafaah, L., Khaeruddin, "Analisis Efektifitas Implementasi Sistem Aplikasi Docker Terintegrasi OpenStack," *Program Studi Teknik Elektro, Fakultas Teknik, Universitas Muhammadiyah Malan,* 2021.

[18] Mursanto, P., Handayani, P. W., dkk, "Journal of Information Systems (Jurnal Sistem Informasi), 16(2)," *Faculty of Computer Science, Universitas Indonesia,* 2020.

[19] Putra, R. A., "Analisa Implementasi Arsitektur Microservices Berbasis Kontainer pada Komunitas Pengembang Perangkat Lunak Sumber Terbuka (OpenDaylight DevOps Community)," *Teknik Informatika, Fakultas Teknik, Universitas Muhammadiyah Jakarta*, 2020.

[20] Poulton, N., "The Kubernetes Book, 2023 Edition," JJNP Consulting Limited, 2022.

[21] Mondal, S. K., Pan, R., Kabir, H. M. D., dkk, "Kubernetes in IT administration and serverless computing: An empirical study and research challenges," The Journal of Supercomputing, 78(2), 2937–2987, 2022, https://doi.org/10.1007/s11227-021-03982-3

[22] Rahmi, E., Yumami, E., Hidayasari, N., "Remik: Riset dan E-Jurnal Manajemen Informatika Komputer," *Politeknik Negeri Bengkalis*, 2023.

[23] Huda, A. N., & Kusumawardani, S. S., "Kubernetes Cluster Management for Cloud Computing Platform: A Systematic Literature Review," *Department of Electrical and Information Engineering, Gadjah Mada University, Yogyakarta, Indonesia*, 2022.

[24] Poulton, N, “The Kubernetes Book, 2023 Edition,” *PublishDrive*, 2023.

[25] Saputra, P. S., Pratama, P. A., & Tjahyanti, L. P. A. S., “Perancangan dan Komparasi Web Server Nginx dengan Web Server Apache serta Pemanfaatan Reverse Proxy Server pada Nginx,” *Jurnal Komputer dan Teknologi Sains (Komteks), 2(1), 16-21*, 2023.

[26] Tejaya, W., Rahman, S., & Munir, A. (2023). Pengujian Website Invitees Menggunakan Metode Load Testing dengan Apache JMeter. Jurnal Kharisma Tech, 18(01), 99-112.