Original Article

Managing the Stateful Applications for High Availability using Novel Kubernetes based Microservice Architecture over Cloud based Azure Virtualization Architecture

Sai Vimal Kumar V[1], Malathi K[2]

[1]Research Scholar, Department of Computer Science and Engineering, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, Tamilnadu. India, Pincode: 602105.
[2]Project Guide, Corresponding Author, Department of Computer Science and Engineering, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, Tamilnadu. India, Pincode: 602105.

Abstract

Aim: The aim of the study is to estimate the high availability of stateful applications using novel kubernetes based microservice architecture over cloud based azure virtualization architecture. **Materials and Methods:** Sample groups that are considered in the project can be classified into two each has 20 samples, one for Microservice architecture and other for Virtualization architecture, which are tested using 0.80 for G-power to determine the sample size and for t-test analysis. 20 stateful applications have been used in each group for estimating high availability. **Results:** The novel kubernetes based microservice architecture with efficient accuracy of 90.20%,which by far seems to be better than the cloud based azure virtualization architecture which gives around 88.20%. The significance is around 0.329 ($p>0.05$) and therefore there is a statistical insignificant difference among the study group. **Conclusion:** Novel Kubernetes based microservice architecture seems to be better for stateful applications over the cloud based azure virtualization architecture.

Keywords: Azure, Novel Kubernetes, Microservice Architecture, Container, Virtualization Architecture, Stateful application, High availability.

DOI:10.47750/pnr.2022.13.S04.184

INTRODUCTION

The microservices architectural style has received a lot of attention in the field, and the transition is well underway. Software applications can be created as a series of loosely coupled, independently deployable microservices using this architectural style (Buelta 2019). Microservices are small apps that execute specific business functions and connect with one another using APIs. The microservices architectural method can be used to overcome traditional monolithic architectures, in which the application is a large and complex code base. To take advantage of the advantages of the microservices architecture style, one should use technologies which are compatible with microservices' characteristics. Containerization has become a popular microservices deployment strategy, and Kubernetes is the main container platform that includes everything needed to deploy and run microservices (i.e., code, libraries, settings, etc.). Containers do not know about each other because they are segregated. As a result, a container deployment orchestration platform is required (De Santis et al. 2016). Kubernetes is the most widely used platform for containerized container application deployment, scaling, and administration. Since stateless microservices may be deployed as replaceable instances, they are simple to reproduce. The same cannot be said for stateful microservices. Because stateful microservice instances have different states, they cannot be swapped (Familiar 2015). The "state" aspect makes orchestration more complex than what the initial Kubernetes features and controllers were designed for. Deploying a replicated set of stateful microservices requires coordination of the different replicas to keep them synchronized, and the "state" aspect makes orchestration more complex than what the initial Kubernetes features and controllers were built (Vayghan et al. 2021).

There are around 23 IEEE papers and 17 google scholar papers have been published over the past 5 years. The most cited article is "Microservice Based Architecture:Towards High-Availability for Stateful Applications with Kubernetes". Azure Kubernetes Service is a Microsoft managed container orchestration service based on the open source Kubernetes framework. AKS can be used by an enterprise to manage key tasks like deploying, scaling, and managing Docker containers and container-based applications (Arundel and Domingus 2019). AKS was

released to the public in June 2018 and is primarily utilized by software developers and IT operations personnel. Kubernetes is the open source container orchestration tool, but it comes with a lot of cluster management overhead. AKS assists in the management of a large portion of the overhead, decreasing the complexity of deployment and management duties (Zhou and Choudhary 2021). AKS is for enterprises who wish to use the Azure architecture to build scalable apps utilizing Docker and Kubernetes. The Azure command-line interface (CLI), an Azure portal, or Azure PowerShell can all be used to set up an AKS cluster. Azure Resource Manager templates can also be used to build template-driven deployment choices (Tender and De Tender 2021). A control plane is automatically established and configured when a user creates an AKS cluster. The control plane is a managed Azure resource that the user does not have direct access to. The Azure platform configures secure communication between the nodes and the control plane based on the size and number of nodes specified by the user. There is at least one node in an AKS cluster (Redkar 2009). The node resources utilized to assist the node work as part of a cluster are the central processing unit and memory. Node pools are created when nodes with similar configurations are clustered together (Burns, Beda, and Hightower 2019). Two resource categories are also covered by AKS deployments. The Kubernetes service resource is in one group, while the node resource group is in the other (Abbadi 2014). To establish and administer additional Azure resources, It needs a service principal or managed identity (Familiar 2015).

Our institution is passionate about high quality evidence based research and has excelled in various fields (Parakh et al. 2020; Pham et al. 2021; Perumal, Antony, and Muthuramalingam 2021; Sathiyamoorthi et al. 2021; Devarajan et al. 2021; Dhanraj and Rajeshkumar 2021; Uganya, Radhika, and Vijayaraj 2021; Tesfaye Jule et al. 2021; Nandhini, Ezhilarasan, and Rajeshkumar 2020; Kamath et al. 2020). In the existing research they didn't identify the availability and efficient platform for deploying applications. In this paper, we compare Kubernetes with Azure to see which platform offers the greatest deployment and administration services at the lowest cost and with the most ease of maintenance. To determine which platform was the most efficient, we used the same set of applications on each platform and compared the deployment time and correctness of the applications to determine which platform was the most efficient. Azure uses a virtualization design, while Kubernetes uses a microservice architecture. Microservice architecture allows a large application to be broken down into smaller, self-contained components, each with its own set of responsibilities. Virtualization architecture allows several consumers and organizations to share a single physical instance of a resource or application at the same time. This is accomplished by giving a physical storage a logical name and giving a pointer to that physical resource on demand. The main aim of our project is to identify the availability and platform efficiency for deploying simple calculator applications on kubernetes and microsoft azure platform and calculating the availability and deployment time.

Materials and Methods

The research work was performed in the Cloud Computing Laboratory, Department of Computer Science and Engineering, Saveetha School of Engineering, SIMATS (Saveetha Institute of Medical and Technical Sciences). The proposed work contains two groups. Group 1 is taken as Microservice Architecture Based Kubernetes and group 2 as cloud based azure virtualization architecture. The Microservice Architecture Based Kubernetes and cloud based azure virtualization architecture were executed and evaluated a different number of times with a sample size of 20(Buelta 2019) . The minimum power analysis for G-Power calculation is fixed at 0.8 and the maximum accepted error is fixed at 0.05 . Same set of 20 stateful applications are used to calculate the deployment time, management, scalability and availability of the applications for each architecture to get the accuracy of each architecture.

Testing setup for this proposed system used a Vmware workstation and Microsoft Azure. Vmware workstation is used to create a guest OS to deploy applications. Hardware configuration for this proposed system is Intel core i5 8th gen processor and requires 4GB random access memory and 256GB Solid state drive used. The configuration of the system is the Windows 10 operating system.

Procedure for Cloud Based Azure Virtualization Architecture

Step-1: Create a resource group

A logical group in which Azure resources are deployed and managed is called an Azure resource group. When creating a resource group, It will be asked to name it and give it a location. This is the address:

- The place where the resource group metadata is stored.
- Specify another region during resource creation, The resources will operate in Azure.

Step-2: Enable cluster monitoring

Check the connection to see if Microsoft.OperationsManagement and Microsoft.OperationalInsights are active. Register Microsoft.OperationalInsights and Microsoft.OperationalManagement if they aren't already.

Step-3: Create AKS cluster
Create an AKS cluster using the --enable-addons monitoring parameter to enable Azure Monitor for containers using the az aks create command.

Step-4: Connect to the cluster

- Using the az aks install-cli command, install kubectl locally.
- Using the az aks get-credentials command, configure kubectl to connect to the Kubernetes cluster.
- Using the kubectl get command, verify the connection to the cluster. A list of cluster nodes is returned by this command.

Step-5: Deploy the application
A Kubernetes manifest file specifies the desired state of a cluster, such as which container images should be used. Using the kubectl apply command, deploy the application and enter the name of the YAML manifest.

Step-6:Test the application
A Kubernetes service exposes the application front end to the internet when it operates. It may take a few minutes to finish this process. Open a web browser to the external IP address of the service to view the deployed application in action.
Algorithm for deploying application using Microservice architecture based kubernetes in azure is given below:

Input: Application A to deploy

```
A=state_application
R=ResourceGroup(*G)
ResourceGroup(*G)
      Create ResourceGroup(*G)=new ResourceGroup(*G)
Enable Cluster monitor()
      Register Microsoft.OperationsManagement()
      Register Microsoft.OperationalInsights()
K=AKS_Cluster(*AK)
     AKS_Cluster(*AK)=new AKS_Cluster(*AK)
Connect(R,K)
     Enable Kubectl()
          Register az_aks.kubectl()
     Configure Kubectl()
          P=port
          Configure(R,K,P)
          if(Configure(R,K,P)==true)
               Return Deploy()
          Else
               Error()
Deployment(A)
    Create Deployment D=kubectl apply(A)
G=Availability(A)
Availability()
    Check(Reaction Time)
              RT=Get(Reaction time)
    Check(Repair Time)
              RET=Get(Repair time)
    Check(Recovery Time)
              RCT=Get(Recovery time)
    Check(Outage Time)
             OT=Get(Outage time)
    compare(RT,RET,RCT,OT)
return(G)
```

Output:

Application Deployment
Deployment time
Availability

Procedure for Microservice Architecture Based Kubernetes

Step-1: KIND Set-up

- **Docker:** Docker must be installed in order for "KIND" to work. On Linux, Use the package manager that comes with the operating system, such as apt on Ubuntu to install docker.
- **Kubectl:** Once the cluster is up and running, It will need to use the kubectl command to interface with it. With the Linux distribution's packages manager, It can find kubectl install instructions, including methods to install it.
- **Kind:** Finally KIND can be installed using the package manager of the linux operating system.

Step-2: Creating the cluster

After installing all of these components, It is ready to start building the local Kubernetes cluster. Kind uses a Docker container to deploy a Kubernetes instance. It's preferable to stop any other containers that are operating on the system because they may conflict with the ports.

Step-3: Deploy an application

It can execute a process now that the cluster is up and running. All workloads in Kubernetes are described in a simple yaml format file called a "manifest." So, in order to run something on the cluster, First create a yaml file that describes what it wants to do.

Step-4: Expose the service

The process is ongoing. Kubernetes offers a scalable service layer for routing connections to the containers it manages. It must specify the ports that Pod will map onto the container when executing it. Then construct a 'Service' Kubernetes resource that will route requests to the processes operating in the Pods.

Algorithm for deploying application using Vmware Workstation based kubernetes is given below:

```
Input: Application A to deploy
A=application
C=Cluster(*C)
S=Service(*S)
E=expose()
P=port
Kindsetup(A)    //Setup kind Service for deployment
        Enable Docker()
        Kubectl(get nodes)
Create cluster(*C)       //Function to create cluster
        Create cluster(*C)=new cluster(*C)
Create service(*S)       //Funcion to create Service
       Create service(*S)=new service(*S)
configure(*CF)
       Configuration of(Cluster,Service,Port)
expose(*S)
       Expose service(*S)=Service deployment.Kubectl()
if( kindsetup(A) == true)
Create cluster(C)      //Create Cluster
Configure(P)          //Configure ports
Create Service(S)    //Create Service
Expose(Service (S) )       //Expose Service
G=Availability(A)
G=Availability(A)
Availability()
 Check(Reaction Time)
            RT=Get(Reaction time)
 Check(Repair Time)
            RET=Get(Repair time)
 Check(Recovery Time)
```

RCT=Get(Recovery time)
Check(Outage Time)
OT=Get(Outage time)
compare(RT,RET,RCT,OT)
return(G)
End if
Else
"Error while setting up Kind"
End Else
Output:
Application Deployment
Deployment time
Availability

Statistical Analysis

Statistical software used in the study is IBM SPSS version 26. The independent sample T-test calculation for analyzing equal variance, standard error, and levene's test are evaluated. Attributes like platform, Application number form the independent variables, Deployment Time, accuracy,availability are dependent variables. Independent sample T-test has been carried out for evaluating the accuracy.

Results

In this proposed system it was observed that novel Microservice architecture based kubernetes appears to be better than the cloud based Azure virtualization architecture based kubernetes. Microservice architecture based kubernetes enables the continuous delivery and deployment of large, complex applications. Table 1 represents the outcome of the deployment. Table 2 shows the statistical calculation such as mean, standard deviation and standard error mean for Microservice architecture based kubernetes deployment and cloud based Azure virtualization architecture based kubernetes deployment respectively. The mean, standard deviation and standard error mean for Microservice architecture based kubernetes deployment are 90.20, 1.924, 860 respectively. The mean, standard deviation and standard error mean for cloud based Azure virtualization architecture based kubernetes deployment are 88.20, 1.924, 860 respectively.

Table 1. Availability percentage and accuracy for Microservice architecture based kubernetes deployment technique and cloud based Azure virtualization architecture based kubernetes deployment technique.

Platform	Iteration no(n)	Deploy Time (in Sec)	Recovery Time (in Sec)	Failure Time (in Sec)	Availability(in %)	Accuracy (in %)
cloud based Azure virtualization architecture based kubernetes	1	38	1.738	3.462	99%	89%
	2	29	1.270	2.942	99%	91%
	3	43	2.050	3.676	99%	86%
	4	54	3.427	4.892	99%	88%
	5	36	1.543	3.216	99%	87%

Novel kubernetes Based Microservice architecture	1	41	1.546	3.231	99.99%	91%
	2	34	1.262	2.531	99.99%	93%
	3	49	1.942	3.333	99.99%	88%
	4	59	2.521	4.541	99.99%	90%
	5	45	1.325	2.903	99.99%	89%

Table 2. Group statistical analysis of Cloud Based Azure Virtualization Architecture Based Kubernetes With Mean Value Of 88.20 And Novel Kubernetes Based Microservice Architecture With Mean value of 90.20 and similarly the results of Standard Deviation and Standard Error Mean are given.

T-Test: Group Statistics

GROUP		**N**	**Mean**	**STD Deviation**	**STD Error mean**
ACCURACY	**Novel kubernetes Based Microservice architecture**	20	90.20	1.924	860
	Cloud based Azure virtualization architecture based kubernetes	20	88.20	1.924	860

It is inferred that the mean accuracy for T-test is far lesser than the comparison architecture. Moreover, the mean accuracy value of Microservice architecture based kubernetes is around 90.20 which seems to be superior to the cloud based Azure virtualization architecture based kubernetes. In Table 3, it was observed that the Levens test for equality of variance and its significance for Microservice architecture based kubernetes is 0.001 and 1.000 respectively and standard error difference and confidence interval are lower than cloud based Azure virtualization architecture based kubernetes.

Table 3. Independent Sample T-test Results with confidence interval of 95% and level of significance greater than 0.05 (Microservice architecture based kubernetes deployment seems to appear better for the high availability of stateful applications).

		Levene's Test for Equality of Variance	**Levene's Test for Equality of Variance**

									95% Confidence Interval of the Difference	
	Equal Variances	**F**	**Sig.**	**t**	**df**	**Sig.(2-tailed)**	**Mean Difference**	**Std. Error Differ -ence**	**Lower**	**Upper**
ACCURACY	**Assumed**	.001	.329	1.644	8	.029	5.600	2.000	-.805	4.805
	Not Assumed			1.644	8.000	.019	5.600	2.000	-.805	4.805

Fig. 1 represents the architecture for Deploying a Stateful application in kubernetes using different platforms. Cluster, Service are the important components in the architecture. Pods are used to configure ports and store status in the architecture..

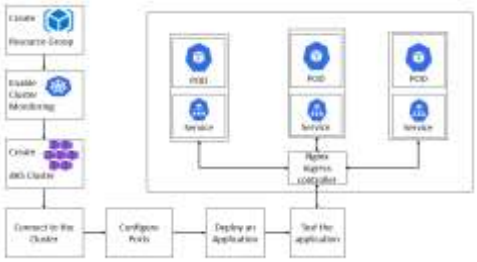

Fig. 1. Architecture for Deploying a Stateful application in kubernetes using different platforms. Cluster, Service are the important components in the architecture. Pods are used to configure ports and store status in the architecture.

Fig. 2 represents the deployed application in the kubernetes. Mean accuracy and mean loss graph is depicted in Fig. 3. Microservice architecture based kubernetes deployment seems to appear better for the high availability of stateful applications.

Fig. 2. Deployed application in kubernetes using Microservice architecture based kubernetes deployment and cloud based Azure virtualization architecture based kubernetes deployment respectively.

Fig. 3. Bar graph analysis of Novel kubernetes based Microservice architecture and cloud based Azure virtualization architecture based kubernetes. Graphical representation shows the mean Accuracy of 90.20% and 88.20% for the proposed platform (Kubernetes) and Azure respectively. X-axis : MZ vs VM, Y-axis : Mean Accuracy ± 1 SD.

Discussion

The proposed novel Microservice architecture based kubernetes deployment provides high availability for stateful applications compared to cloud based Azure virtualization architecture based kubernetes. The mean, standard deviation and standard error mean for cloud based Azure virtualization architecture based kubernetes deployment are 88.20, 1.924, 860 respectively. It is inferred that the mean accuracy for T-test is far lesser than the comparison architecture. Moreover, the mean accuracy value of Microservice architecture based kubernetes is around 90.20 which seems to be superior to the cloud based Azure virtualization architecture based kubernetes. A systematic review on high availability for stateful applications deployment techniques presents around 32 studies, the analysis of various papers shows that Microservice architecture based kubernetes deployment is the most used technique for containerized application deployment. Yearly deployment time and availability is compared to the previous years with different engines and architectures. Since most of the availability percentage of stateful applications are related to Microservice architecture based kubernetes deployment, the technique provides the best accuracy for deployment of stateful applications.In the industry, the microservices architectural style has gotten a lot of attention, and the transition to this design is well underway. The creation of software applications as a composition of loosely connected and independently deployable microservices is possible with this architectural style ("MICROSERVICES - Lightweight Service Descriptions for REST Architectural Style" 2010). Traditional monolithic architectures, where the application is a big and complex code base, can be overcome by the microservices architectural approach (Kamimura et al. 2018). To get the benefits of the microservices architecture style, one must employ technologies that are compatible with microservices' characteristics. Containerization has become a common deployment technique for microservices. AKS assists in the management of a large portion of the overhead, decreasing the complexity of deployment and management duties ("IMPLEMENTATION OF DEVOPS PARADIGM TO DEPLOYMENT AND PROVISIONING OF MICROSERVICES" 2021) (Talaat 2015). The Azure platform configures secure communication between the nodes and the control plane based on the size and number of nodes specified by the user. There is at least one node in an AKS cluster (Saha, Nanba, and Nishimura 2018). The node resources utilized to assist the node work as part of a cluster are the central processing unit and memory. Microservice architecture based kubernetes deployment seems to appear better for the high availability of stateful applications (Vayghan et al. 2019). The limitation of this research work is, it is limited to deploy containerized applications. Currently, it is not programmed to embed with other stateful applications. Further, this research work can be improved by deploying a model that deploys more applications in less time so that wait will be less and it can be easily manageable and scalable as in this research.

Conclusion

The Microservice architecture based kubernetes deployment technique provides the high availability for stateful application with better deployment time and maintenance compared to cloud based Azure virtualization architecture based kubernetes deployment technique.

Declarations

Conflict of Interests

No conflict of interest in this manuscript.

Author Contribution

Author SVKV was involved in data collection, data analysis, manuscript writing. Author MK was involved in conceptualization, guidance and critical review of manuscript.

Acknowledgments

The authors would like to express their gratitude towards Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences (Formerly known as Saveetha University) for providing the necessary infrastructure to carry out this work successfully.

Funding

We thank the following organizations for providing financial support that enabled us to complete the study.

1. Renaissance Technologies, Chennai.
2. Saveetha University.
3. Saveetha Institute of Medical and Technical Sciences.
4. Saveetha School of Engineering.

References

1. Abbadi, Imad M. 2014. *Cloud Management and Security*. John Wiley & Sons.
2. Arundel, John, and Justin Domingus. 2019. *Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud*. "O'Reilly Media, Inc."
3. Buelta, Jaime. 2019. *Hands-On Docker for Microservices with Python: Design, Deploy, and Operate a Complex System with Multiple Microservices Using Docker and Kubernetes*. Packt Publishing Ltd.
4. Burns, Brendan, Joe Beda, and Kelsey Hightower. 2019. *Kubernetes: Up and Running: Dive into the Future of Infrastructure*. "O'Reilly Media, Inc."
5. De Santis, Sandro, Luis Florez, Duy V. Nguyen, Eduardo Rosa, and I. B. M. Redbooks. 2016. *Evolve the Monolith to Microservices with Java and Node*. IBM Redbooks.
6. Devarajan, Yuvarajan, Beemkumar Nagappan, Gautam Choubey, Suresh Vellaiyan, and Kulmani Mehar. 2021. "Renewable Pathway and Twin Fueling Approach on Ignition Analysis of a Dual-Fuelled Compression Ignition Engine." *Energy & Fuels: An American Chemical Society Journal* 35 (12): 9930–36.
7. Dhanraj, Ganapathy, and Shanmugam Rajeshkumar. 2021. "Anticariogenic Effect of Selenium Nanoparticles Synthesized Using Brassica Oleracea." *Journal of Nanomaterials* 2021 (July). https://doi.org/10.1155/2021/8115585.
8. Familiar, Bob. 2015. "Azure, A Microservice Platform." *Microservices, IoT, and Azure*. https://doi.org/10.1007/978-1-4842-1275-2_4.
9. "IMPLEMENTATION OF DEVOPS PARADIGM TO DEPLOYMENT AND PROVISIONING OF MICROSERVICES." 2021. *Issues In Information Systems*. https://doi.org/10.48009/1_iis_2021_136-148.
10. Kamath, S. Manjunath, K. Sridhar, D. Jaison, V. Gopinath, B. K. Mohamed Ibrahim, Nilkantha Gupta, A. Sundaram, P. Sivaperumal, S. Padmapriya, and S. Shantanu Patil. 2020. "Fabrication of Tri-Layered Electrospun Polycaprolactone Mats with Improved Sustained Drug Release Profile." *Scientific Reports* 10 (1): 18179.
11. Kamimura, Manabu, Keisuke Yano, Tomomi Hatano, and Akihiko Matsuo. 2018. "Extracting Candidates of Microservices from Monolithic Application Code." *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. https://doi.org/10.1109/apsec.2018.00072.
12. "MICROSERVICES - Lightweight Service Descriptions for REST Architectural Style." 2010. *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence*. https://doi.org/10.5220/0002720105760579.
13. Nandhini, Joseph T., Devaraj Ezhilarasan, and Shanmugam Rajeshkumar. 2020. "An Ecofriendly Synthesized Gold Nanoparticles Induces Cytotoxicity via Apoptosis in HepG2 Cells." *Environmental Toxicology*, August. https://doi.org/10.1002/tox.23007.
14. Parakh, Mayank K., Shriraam Ulaganambi, Nisha Ashifa, Reshma Premkumar, and Amit L. Jain. 2020. "Oral Potentially Malignant Disorders: Clinical Diagnosis and Current Screening Aids: A Narrative Review." *European Journal of Cancer Prevention: The Official Journal of the European Cancer Prevention Organisation* 29 (1): 65–72.
15. Perumal, Karthikeyan, Joseph Antony, and Subagunasekar Muthuramalingam. 2021. "Heavy Metal Pollutants and Their Spatial Distribution in Surface Sediments from Thondi Coast, Palk Bay, South India." *Environmental Sciences Europe* 33 (1). https://doi.org/10.1186/s12302-021-00501-2.
16. Pham, Quoc Hoa, Supat Chupradit, Gunawan Widjaja, Muataz S. Alhassan, Rustem Magizov, Yasser Fakri Mustafa, Aravindhan Surendar, Amirzhan Kassenov, Zeinab Arzehgar, and Wanich Suksatan. 2021. "The Effects of Ni or Nb Additions on the Relaxation Behavior of Zr55Cu35Al10 Metallic Glass." *Materials Today Communications* 29 (December): 102909.
17. Redkar, Tejaswi. 2009. "AppFabric: Access Control Service." *Windows Azure Platform*. https://doi.org/10.1007/978-1-4302-2480-8_7.
18. Saha, Rony Kumer, Shinobu Nanba, and Kosuke Nishimura. 2018. "Clustering and Centralized Resource Scheduling of 3D In-Building Small Cells for Intra MAC Functional Split Control-/User-Plane Decoupled CRAN." *2018 IEEE International Conference on Communications (ICC)*. https://doi.org/10.1109/icc.2018.8422357.
19. Sathiyamoorthi, Ramalingam, Gomathinayakam Sankaranarayanan, Dinesh Babu Munuswamy, and Yuvarajan Devarajan. 2021. "Experimental Study of Spray Analysis for Palmarosa Biodiesel-diesel Blends in a Constant Volume Chamber." *Environmental Progress & Sustainable Energy* 40 (6). https://doi.org/10.1002/ep.13696.
20. Talaat, Sherif. 2015. "Azure Architecture Overview." *Pro PowerShell for Microsoft Azure*. https://doi.org/10.1007/978-1-4842-0665-2_1.
21. Tender, Peter De, and Peter De Tender. 2021. "Lab 7: Managing and Monitoring Azure Kubernetes Service (AKS)." *Migrating a Two-*

Tier Application to Azure. https://doi.org/10.1007/978-1-4842-6437-9_9.
22. Tesfaye Jule, Leta, Krishnaraj Ramaswamy, Nagaraj Nagaprasad, Vigneshwaran Shanmugam, and Venkataraman Vignesh. 2021. "Design and Analysis of Serial Drilled Hole in Composite Material." *Materials Today: Proceedings* 45 (January): 5759–63.
23. Uganya, G., Radhika, and N. Vijayaraj. 2021. "A Survey on Internet of Things: Applications, Recent Issues, Attacks, and Security Mechanisms." *Journal of Circuits Systems and Computers* 30 (05): 2130006.
24. Vayghan, Leila Abdollahi, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. 2019. "Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes." *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. https://doi.org/10.1109/qrs.2019.00034.
25. ———. 2021. "A Kubernetes Controller for Managing the Availability of Elastic Microservice Based Stateful Applications." *Journal of Systems and Software*. https://doi.org/10.1016/j.jss.2021.110924.
26. Zhou, Zach Zhizhong, and Vidyanand Choudhary. 2021. "Impact of Competition from Open Source Software on Proprietary Software." *Production and Operations Management*. https://doi.org/10.1111/poms.13575.

Table 1. Availability percentage and accuracy for Microservice architecture based kubernetes deployment technique and cloud based Azure virtualization architecture based kubernetes deployment technique.

Platform	Iteration no(n)	Deploy Time (in Sec)	Recovery Time (in Sec)	Failure Time (in Sec)	Availability(in %)	Accuracy (in %)
cloud based Azure virtualization architecture based kubernetes	1	38	1.738	3.462	99%	89%
	2	29	1.270	2.942	99%	91%
	3	43	2.050	3.676	99%	86%
	4	54	3.427	4.892	99%	88%
	5	36	1.543	3.216	99%	87%
Novel kubernetes Based Microservice architecture	1	41	1.546	3.231	99.99%	91%
	2	34	1.262	2.531	99.99%	93%
	3	49	1.942	3.333	99.99%	88%
	4	59	2.521	4.541	99.99%	90%
	5	45	1.325	2.903	99.99%	89%

Table 2. Group statistical analysis of Cloud Based Azure Virtualization Architecture Based Kubernetes With Mean Value Of 88.20 And Novel Kubernetes Based Microservice Architecture With Mean value of 90.20 and similarly the results of Standard Deviation and Standard Error Mean are given.

T-Test: Group Statistics

GROUP		N	Mean	STD Deviation	STD Error mean
ACCURACY	Novel kubernetes Based Microservice architecture	20	90.20	1.924	860
	Cloud based Azure virtualization architecture based kubernetes	20	88.20	1.924	860

Table 3. Independent Sample T-test Results with confidence interval of 95% and level of significance greater than 0.05 (Microservice architecture based kubernetes deployment seems to appear better for the high availability of stateful applications).

		Levene's Test for Equality of Variance		Levene's Test for Equality of Variance						
	Equal Variances	F	Sig.	t	df	Sig.(2-tailed)	Mean Difference	Std. Error Differ -ence	95% Confidence Interval of the Difference	
									Lower	Upper
ACCURACY	Assumed	001	329	1.644	8	029	5.600	2.000	.805	4.805
	Not Assumed			1.644	8.000	019	5.600	2.000	.805	4.805

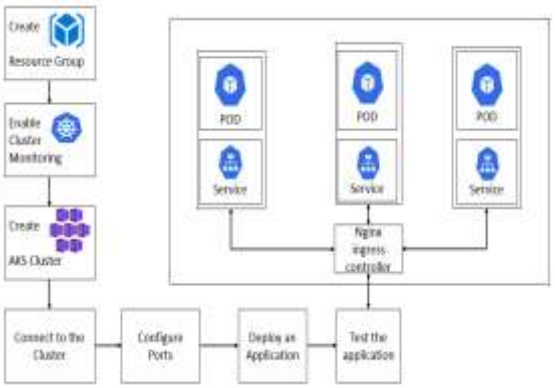

Fig. 1. Architecture for Deploying a Stateful application in kubernetes using different platforms. Cluster, Service are the important components in the architecture. Pods are used to configure ports and store status in the architecture.

Fig. 2. Deployed application in kubernetes using Microservice architecture based kubernetes deployment and cloud based Azure virtualization architecture based kubernetes deployment respectively.

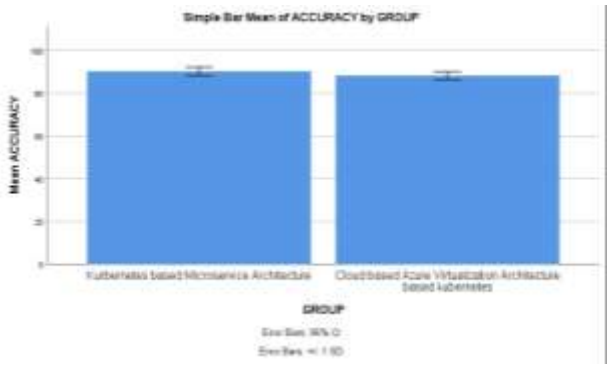

Fig. 3. Bar graph analysis of Novel kubernetes based Microservice architecture and cloud based Azure virtualization architecture based kubernetes. Graphical representation shows the mean Accuracy of 90.20% and 88.20% for the proposed platform (Kubernetes) and Azure respectively. X-axis : MZ vs VM, Y-axis : Mean Accuracy ± 1 SD.