Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 12 , December - 2025

Dolphin: Devopify Infrastructure As A Service (DIASS)

Dr. Ravikumar G. K
Department of CSE
BGSCET
Bengaluru, India

Aditya Srinivasan
Department of CSE
BGSCET
Bengaluru, India

Dr. Manjula
Department of CSE
BGSCET
Bengaluru, India

Manoja D
Department of CSE
BGSCET
Bengaluru, India

Vilas C.P
Department of CSE
BGSCET
Bengaluru, India

Abstract - **Dolphin is a people-friendly service, which is easy to deploy and scale server applications in the cloud. Previously developers had to cope with several instance of AWS or complex AWS Kubernetes configurations workload which necessitated a high level of expertise in cloud architecture and DevOps practices. Dolphin takes a different direction in which applications are defined based on source code and runtime environments and that the basic units of deployment are Docker containers. Once an application is whether specified in Dolphin, it handles deployment and automatically scales capacity based on user demand variations, eliminating low-level infrastructure concerns on the part of the developer. The paper presents the system architecture of Dolphin, its major components and the details of its implementation and discusses how the system enables the development and operation teams to work together, with the developers focusing on the development of application logic and features, and Dolphin being in charge of operational activities. The ultimate goal is to accelerate the application scaling and it minimizes the labor in charge of managing infrastructure. Dolphin is designed to streamline the deployment of a cloud-native application that is simple and resilient.**

I. INTRODUCTION

The method of software construction and usage has been changed by cloud computing. The current demands of people are that the apps have to be loaded fast, reliable and have to be always at their fingertips. Want to blow something up! This imposes a tone of pressure on the developers to scale out the system and accommodate additional users at the start. Nonetheless, it is not a simple task to develop an app that would be actually scaled. It is normally associated with new servers being added, load balancers being brought on, and complicated scripts to execute deployment. All that takes time, talent and labor that would be better applied towards enhancing the real product.

That's where Dolphin comes in. It is a developer-centric infrastructure that is meant to eliminate the effort of maintaining infrastructure. The developers do not have to worry about complex setup and server maintenance, but can merely state what their application should occur and they have the rest taken care of. Other aspects such as deploying and scaling of the containers, and all the running of the containers are all obviated by Dolphin. This will enable you to execute your app with ease across all locations without giving a second or third thought to any form of low level technicals that are usually prone to messing up with various cloud infrastructures.

Dolphin is simple enough to be used and comprehended by all. There is no longer a necessity of worrying about the backend configuration and server maintenance as developers. They are able to spend more of their time creating quality aspects, developing their applications. The remaining posts will delve into the various factors about establishing Dolphin, the way it goes about doing all its magic behind the scenes, its main parts, system architecture and interrelationship that it has about development and operation to worry free deployment to the cloud.

II. LITERATURE REVIEW

Cloud computing has become one of the chief support frameworks of modern distributed system design and it gives the development staff the scaling property, the shares dynamically of the resources, and the dynamically reacts to the frequent alterations in workload intensity. Among the first frameworks, assisting to understand the specified paradigm, Mahmood [1] presents the key features of cloud computing such as virtualization, an elasticity that is fast, and resource pooling and separates the models of deployment into the following: the public, the private, the hybrid and the community clouds. The concept has continued to be a guideline to system engineers and architects on the type of deployment strategies to be adopted whenever the workload requirements vary.

Continuing these principles, Guo et al. [2] also gave a glimpse at the impact of the shift to monolithic architecture and increased autonomy of service structures. Their proposed cloudware architecture is based on the application of the microservices, small and specialized components that are deployed to enhance the application agility, resilience and scalability of applications. Such a separation of applications into such small units enables a more finer control of the deployment process and scaling process and this factor has led to the increase in the use of containerization technologies. All told and said, the move of monolithic software towards microservice-based architectures will be a ground-breaking process when it comes to realizing the potential of cloud-native computing to the fullest.

Containerization has in many ways changed the deployment of applications. Containers unlike virtual machines do not package application components with exorbitant overhead hence providing lightweight and portable environments and reproducible environments. Wan et al. [3] created a system architecture on the launch of applications that are founded on microservices and Docker container architecture. They have demonstrated that the purpose behind the structures being container-based make configuration management easier and dependency conflicts addressed. They placed emphasis on portability of Docker images that implied that a team could create a seamless experience of deployment on development, staging and production with minimal variation. Here Taherizadeh and Stankovski [4] furthered their idea to propose a multi-level dynamic auto scaling of containerized systems. Their model dynamically commissioned the computing resources to the intensity of workload in a way that it possessed responsive provisioning capabilities which was effective in ensuring the efficiency of the system besides reducing the idle capacity justification. These publications rendered containerization one of the central ideas of a scalable cloud-native environment.

As the cloud environment evolved, researchers began to investigate how these models can be applied to scaling methods with respect to intelligent decision-making and predictive capabilities. To study the most efficient scaling intervals, Chouliaras and Sotiriadis [5] suggested a workload-based scaling behavior that used the statistics about the runtime, which includes the CPU utilization, latency, and throughput to conduct the study. They found out that reactive scaling can be able to handle the overload though can add delays in the assignment of the resources. That can be corrected by the predictive and workload-sensitive scaling that anticipates future changes in loads. Huaijun et al. [6] have proposed in this concept a container scaling strategy that is driven by the principles of reinforcement learning (RL). It is also a system which learns based on performance feedback and scales policies particularly with time. The performance to cost ratio was improved through the strategy as there was very little over-provisioning or under-provisioning. Shi et al. [7] then extended this theme to the geo-distributed configuration through which an auto scaling architecture was suggested, taking into consideration network latency, network bandwidth fluctuation, and geographical dispatchment fluctuation of demand distributed among many data centers. They determined that scaling in distributed clouds in the context that it does not have to be workload-intensive although there is some consideration of geographical factors.

At the same time, Gwydion developed by Santos et al. [8] is based on the idea of scalability of auto-scaling containerized workloads of large scale, or complex scale. Gwydion contains decision making modules, which can differentiate between temporary and any constant workload spikes and undertake more accurate scaling measures. It has been found to be significantly lower in orchestration delays in the experiment than in traditional methods and this means that it is more appropriate in real time scale in production processes. Altogether, these papers indicate a new way of cloud scaling research, which a more advanced approach is sought, and an extended development of the cloud scaling models as aspects of intelligence, context-sensitive, and self-optimizing models that operate effectively in dynamic environments is desirable.

Container orchestration is no longer young, and that is what has motivated the incorporation of DevOps practices and containerized frameworks as a necessity to continuous delivery and operational occurrences. Ugwueze and Chukwunweike [9] examined various strategies of continuous integration and continuous deployment (CI/CD). They have decided that the only way in which they can give liberty to the build, test and deployment procedures is through the automation to enable the process of software release to be fastened. They have found this adequate to that of Mustyala [10] who had suggested that even the usage of serverless models could be enhanced with the incorporation of DevOps that would enable swift application deployment without having to directly manage servers. In a systematic review, Azad et al. [11] discovered some critical facets of DevOps success, i.e. coordinate activity of team members, monitoring activities and automated feedback loops. They noted that automation of technical processes will not make DevOps flourish without backing by the organization. Similarly, as the hybridized DevOps paradigm created by Multiple Authors proved [12], the incorporation of agile principles in the ecology of DevOps will contribute to flexibility, in which the teams can respond faster to dynamic needs and make projects more reliable.

The study by Patil et al. [13] proposed a viable model of implementing the scientific workloads on open stack and showed how the demands of the research workloads with heavy computing and storage could be supported by IaaS systems. Their architecture; which embraced the use of modular building blocks of storage, compute and networking of OpenStack meant that they had adopted a scalable approach that would enable them to handle large scientific data sets. Their argument was very simple: being a mixture of DevOps processes and cloud orchestration, they were able to develop one system that is capable of supporting both enterprise applications and research systems.

Nevertheless, there are still some significant obstacles. Most autoscaling solutions are pretty on paper, but difficult to implement in real life scenarios - they usually require elaborate configurations and expertise. Cloud providers and tools (such as AWS, Microsoft Azure, Kubernetes, or AWS Elastic Beanstalk) can provide immense power, but they also may need to be tuned by hand and team members must have substantial experience of infrastructure to get their policies and limits right. Even more so, the vast majority of the presented frameworks are centered on the automation of the back end and do not pay much attention to usability: they seldom provide convenient means of visualizing or controlling an applications architecture. The aspect of lack of alignment between the desires developers wish to create and the way infrastructure actually acts continues to demonstrate that development and operations do not interface with one another.

That is an area that Dolphin attempts to bridge. It is a newer generation deployment platform, having a visual, diagram-based interface, which allows developers to design their application structure without fussing with scripts or command line applications. Dolphin uses Docker containers as its fundamental units therefore deployments remain lightweight and portable. The workloads watched by Dolphin automated scaling engine then automatically manages resources eliminating much of the configuration and manual control that would otherwise be required. Dolphin simplifies DevOps activities and therefore eases the technical load on developers, as well as simplifying the process of making successful, fault-tolerant deployments with ease and without the need to have specific expertise of an infrastructure.

Concisely, the literature demonstrates that orchestration and scaling are becoming increasingly complicated and how complexity is a practical hindrance to lots of developers. Dolphin reacts to that by integrating automation, scalability, and usability in one, developers-centric platform. It is a smart, graphic substitute that will assist in gearing cloud deployment less complex and also assist in closing the historic divide between operations and software development.

III. PROBLEM STATEMENT

Deploying and scaling cloud applications is often much more difficult than it seems. Developers often struggle with setting up servers, writing configuration scripts, and managing multiple container tasks, all of which take a lot of time and require deep DevOps knowledge.

Even though tools like Docker make packaging and distributing applications easier, they don't completely remove the challenges of setting up, managing, and scaling many containers. Moving an app from a local setup to a reliable, production ready cloud service is still a difficult process, with many tricky and error prone steps.

Dolphin was created to fill this gap. The platform automates the deployment and scaling of containerized applications, freeing developers from the hard work of managing infrastructure. This automation ensures reliable, cloud ready deployments while letting development teams focus on writing code, building features, and speeding up delivery instead of dealing with maintenance and troubleshooting.

OBJECTIVES

- To make cloud deployment and scaling easier, we offer a simple platform that automatically manages application instances using Docker containers.
- This bridges the gap between development and operations. Developers can focus on writing code while Docker takes care of infrastructure setup and scaling
- We also offer better scalability and performance for server based applications by automatically managing resources and coordinating containers, without requiring deep DevOps knowledge.

IV. DESIGN TECHNIQUES

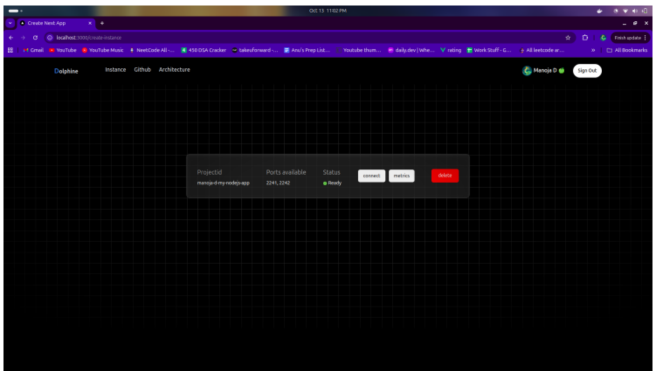
Dolphin is a scaling and flexible architecture which eases things and creates less cognitive burden to the developers. The site specifically specifies the difference between its frontend and the backend: the former deals with interaction with users, display, and monitoring of the system; the latter is in charge of the management of resources, optimization of the work with containers, and their isolation. This decoupling makes systems easier to manage on a global scale and each of the components can be scaled on its own i.e. any change or an increase in one area will not impact on the pace of the other.

The primary concept of Dolphin is with the help of container. Applications are in their individual Docker containers and this provides stability as well as predictability of the environment. This division allows the applications to expand or contract without impacting other applications, and such makes the system a trustworthy and secure system with demand fluctuation.

Dolphin has a web based terminal, which is a real time terminal to facilitate development. This allows developers to do commands directly in the browser and immediately get a response, not worrying about the underlying server and container communication. None of these technicals allows Dolphin to free developers to write code and create features rather than manage infrastructure.
The modularity and intuitiveness of the platform underscores the designer-oriented philosophy of the platform. The interface has reusable parts, intuitive displays on dashboards, and easy-to-access monitoring systems that render complicated technical measurements into the actions that can be taken. Basically, Dolphin hides the complexity inherent in deploying and scaling of the cloud, and allows developers full control of its operation with minimum friction and maximum efficiency.

V. IMPLEMENTATION

The environment around Dolphin is easy, user friendly and high performing, thus allowing the developer to focus on creating features instead of dealing with infrastructure. Next.js, Reactjs and TailwindCSS are used to do the frontend. Dolphin can be used by the developers using dashboard and a series of structured landing pages that simplifies the configuration process, the launching process and the monitoring process of application instances and it is simple to take everything under one roof. The design gives developers freedom to concentrate on the logic part of the app, but without much effort on how the app runs, it makes it more productive and causes a reduction in mental stress.



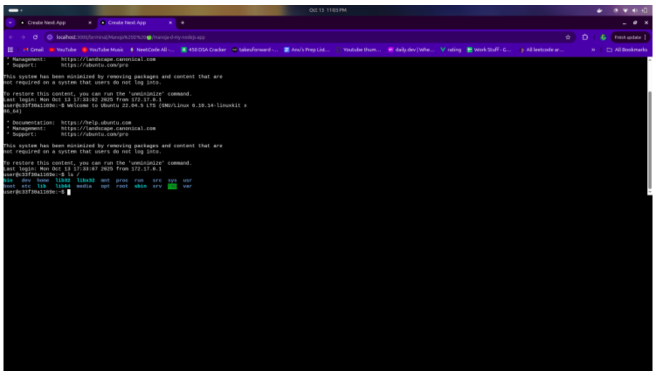
Projectid	Ports available	Status
manoja-d-my-nodejs-app	2241, 2242	Ready

Fig 1. Our Client App Instance Dashboard

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 12 , December - 2025

Dolphin also has a browser-based terminal that is developed using Xterm.js thus providing developers with an easy to use and interactive command-line interface. Web browsers are not capable of opening direct SSH connections with containers and this makes Dolphin use the Socket.IO library to open a secure WebSocket connection between the client browser and the server.

A command typed by a user in the browser terminal is forwarded to the server by a WebSocket connection. The command is executed on the server on the right Docker container and the result is relayed back to the browser in real-time. This provides immediate feedback and it acts just as using a regular terminal, although there is no need to log in to remote servers. Consequently, Dolphin simplifies the process of managing containers, and does so in a very rapid fashion, and it is through a very simple web-based interface.



```
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Oct 13 17:33:02 2025 from 172.17.0.1
user@c33f38a1169e:~$ Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.10.14-linuxkit x
86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Oct 13 17:33:07 2025 from 172.17.0.1
user@c33f38a1169e:~$ ls /
bin   dev  home  lib32  libx32  mnt  proc  run   src  sys  usr
boot  etc  lib   lib64  media   opt  root  sbin  srv  tmp  var
user@c33f38a1169e:~$ 
```

Fig 2. Our Client App Web-Based Terminal

At its core, Dolphin leverages Docker containers as the runtime foundation for all application instances. Containers are inherently lightweight and portable, offering a consistent execution environment that bridges the gap between development and production stages. When an additional instance is requested, Dolphin communicates with Docker to instantiate a new container configured with the specified computational resources.

Each container operates in isolation from others, ensuring both stability and security across the system. This isolation prevents resource contention and minimizes the risk of interference between concurrently running applications. Moreover, Dolphin dynamically manages container scaling automatically provisioning or terminating instances in response to real-time demand. This adaptive mechanism maintains efficient resource utilization while preserving predictable performance and uniform behavior across all deployments.

The broader software architecture is centered on a model of resource allocation, where clients will request resources from a central pool of servers to spin up their applications. There is a database used to track metadata like available ports, security policies, and resource usage; and to help track and administer individual instances. After a Docker container is running, developers are able to deploy and manage their application through the web terminal; thus, the developer/user is given a flexible, easy-to-use, and automated environment for deployment in a cloud service.

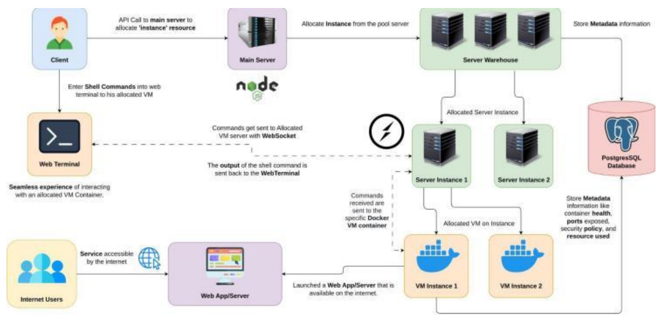

Client
API Call to main server to allocate 'instance' resource
Main Server
node
Allocate Instance from the pool server
Server Warehouse
Store Metadata information
Enter Shell Commands into web terminal to his allocated VM
Web Terminal
Seamless experience of interacting with an allocated VM Container.
Commands get sent to Allocated VM server with WebSocket
The output of the shell command is sent back to the WebTerminal
Allocated Server Instance
Server Instance 1
Server Instance 2
PostgresSQL Database
Commands received are sent to the specific Docker VM container
Allocated VM on Instance
Store Metadata information like container health, ports exposed, security policy, and resource used
Internet Users
Service accessible by the internet
Web App/Server
Launched a Web App/Server that is available on the internet.
VM Instance 1
VM Instance 2

Fig 3. Flow diagram of system architecture

VI. CONCLUSION

This article introduces Dolphin, a system that aims to simplify the deployment or scaling of containers in the cloud. Unlike current tools that require users to have an advanced aptitude for DevOps or time-intensive manual configurations of cloud resources, Dolphin utilizes Docker containers to automatically configure the architecture of the application and allocate cloud resources in a manner that corresponds to the workloads/usage patterns of the application. Dolphin abstracts infrastructure complexity and allows developers to focus on writing the functional code while Dolphin ensures their applications remain scalable, responsive, and efficient when the workloads/usage changes; and to able to take advantage of predictive scaling introduced by machine learning in the forthcoming future offering even broader functionality such as multi-cloud deployments, edge-to-cloud integration, security-aware auto-scaling, etc., enhancing Dolphin's flexibility and reliability.

This research also notes Dolphin may evolve into a greater cloud orchestration ecosystem. Dolphin focuses on simplifying the management of cloud-native applications with automation, intelligence, and simplicity in demanding and challenging distributed systems as a practical response developers and the organizations for which they work increasingly seek deployment strategies that are efficient, resilient, and scalable.

VII. FUTURE SCOPE

The Dolphin platform exhibits substantial promise in easing the process of deploying and scaling containerized applications, but there are various opportunities to improve the platform. One key direction is to add intelligent predictive scaling through reinforcement learning and time-series forecasting. This means Bayesian optimization to predict traffic patterns, to allow Dolphin to provision resources in advance of spikes and provide lower latency and costs. Also, increased multi-cloud and hybrid cloud orchestration support would allow applications to determine how to distribute work across heterogeneous cloud providers (or hybrid) dynamically, to achieve improved resilience, fault-tolerant, and cost-optimized geo-distributed deployments. In addition, automating CI/CD pipelines based on application architecture, characteristics, and dependencies would further reduce the amount of manual DevOps effort while achieving increased reliability and performance.

Alongside automation and scaling, Dolphin may offer resource-aware microservice scheduling for CPU, GPU and memory allocated to containerized components, especially for compute-intensive workloads in AI / ML inference and real-time analytics. Explainable AI methods would enable developers to see the 'why' and 'how' products made the scaling decision, and more effectively debug performance, cost tune, and maintain compliance with SLAs. Extending Dolphin to afford edge-to-cloud deployments would enable optimized bandwidth usage, lower latency, and real-time analytics for IoT and latency-sensitive applications. This may enable expanded use of Dolphin in distributed and hybrid solutions.

In summary, enhancing security-aware auto-scaling will allow Dolphin to automatically scale container instances based on identified anomalies or potential attack vectors - thereby scaling with automated threat mitigation. Collectively, these improvements would revolutionize Dolphin from a deployment tool for developers to an intelligent, fully autonomous, and secure orchestrator for the cloud. If predictive scaling were added, multi-cloud, and edge integration, automated DevOps, and unobtrusive AI-powered decisions and actions were implemented, Dolphin could be viewed as a full platform for a new generation of cloud-native applications management.

REFERENCES

[1] Wan, X., et al. (2018). "Application deployment using Microservice and Docker containers: Framework and optimization." *Journal of Network and Computer Applications*, 119, 97-120.

[2] Taherizadeh, S., & Stankovski, V. (2019). "Dynamic Multi-level Auto-scaling Rules for Containerized Applications." *The Computer Journal*, 62(2), 174-197.

[3] Chouliaras, S., & Sotiriadis, S. (2022). "Auto-scaling containerized cloud applications: A workload-driven approach." *Simulation Modelling Practice and Theory*, 121, 102651.

[4] Santos, J., et al. (2025). "Gwydion: Efficient auto-scaling for complex containerized applications." *Journal of Network and Computer Applications*, 206, 103443.

[5] Huaijun, W., et al. (2023). "Container Scaling Strategy Based on Reinforcement Learning." *Wireless Communications and Mobile Computing*, 2023, 7400235.

[6] Shi, T., et al. (2023). "Auto-Scaling Containerized Applications in Geo-distributed Clouds." *IEEE Transactions on Parallel and Distributed Systems*, 34(10), 2847-2860.

[7] Guo, D., et al. (2016). "Microservices Architecture Based Cloudware Deployment Platform." *Proceedings of the 2016 IEEE International Conference on Web Services*, 358-365

[8] Ugwueze, V. U., & Chukwunweike, J. N. (2025). "Continuous Integration and Deployment Strategies for Streamlined DevOps in Software Engineering and Application Delivery." *International Journal of Computer Applications Technology and Research*, 14(1), 1-24

[9] Azad, N., et al. (2023). "DevOps critical success factors — A systematic literature review." *Information and Software Technology*, 157, 107150

[10] Mustyala, A. (2023). "Serverless Computing and DevOps: Streamlining Application Deployment." *SSRN Electronic Journal*

[11] Mahmood, Z. (2011). "Cloud Computing: Characteristics and Deployment Approaches." *Proceedings of the 2011 11th IEEE International Conference on Computer and Information Technology*, 121-126

[12] Patil, M., et al. (2021). "OpenStack Cloud Deployment for Scientific Applications." *Proceedings of the 2021 IEEE International Conference on Electronics, Computing and Communication Technologies*

[13] Multiple authors (2022). "Implementation of DevOps based Hybrid Model for Project Delivery." *International Journal of Computer Science and Network Security*, 22(8), 251-263