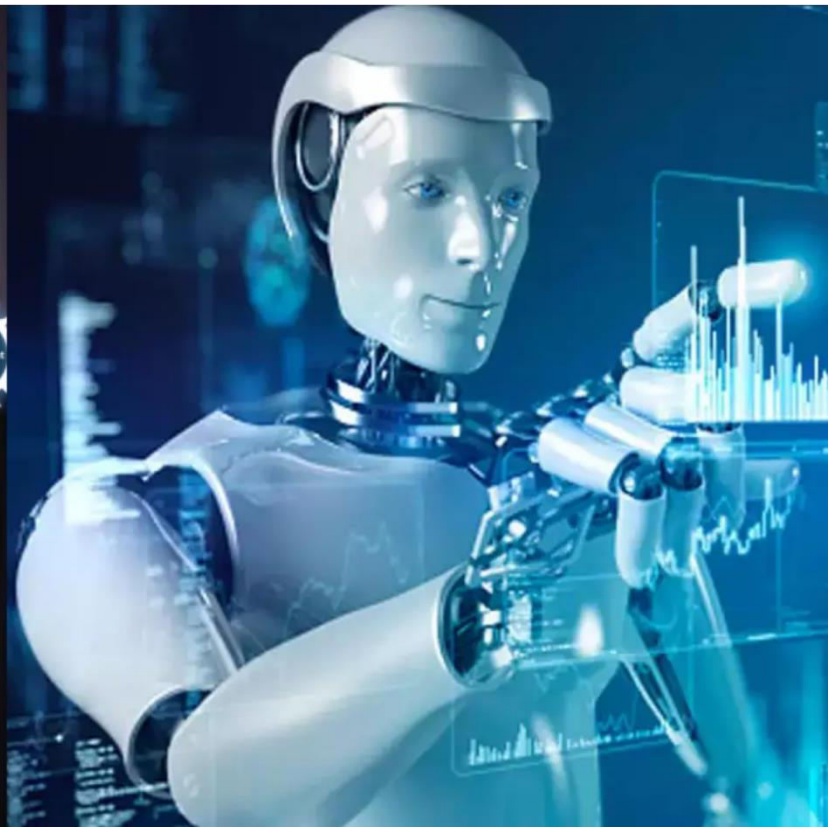
IJIRSET
International Journal of Innovative Research in Science | Engineering | Technology

ISSN(O): 2319-8753
ISSN(P): 2347-6710

International Journal of Innovative Research in Science Engineering and Technology (IJIRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.699

Volume 15, Issue 4, April 2026

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

Integrating Predictive Analytics with Functional Graph Semantics for Cloud- Native Resource Optimization

Mrs Yogapriya.D [1], Dr.Balamurugan M[2], Divakar .A[3], Karthikeyan P[4], Dhilip R[5], Harish S[6]

Assistant Professor, Department of Information Technology, The Kavery Engineering College (Autonomous), Mecheri, Salem, Tamil Nadu, India[1]

Head of the Department, Computer Science Engineering, The Kavery Engineering College (Autonomous), Mecheri, Salem, Tamil Nadu, India[1]

UG Students, Department of Information Technology, The Kavery Engineering College (Autonomous), Mecheri, Salem, Tamil Nadu, India[3-6]

ABSTRACT: A common way to perform proper management of cloud-native systems is to utilize container orchestration systems for the purpose of deploying scalable, distributed applications. Making efficient use of resources; however, still remains a challenge to implement. The majority of container orchestration systems utilize periodic monitoring systems and develop scaling policies that react only after a workload fluctuation is perceived by the application. Reactive scaling results in delayed responses to changing traffic patterns, which can result in resource deficiencies and ultimately cause performance degradation to the application, both temporarily and/or continuously. This paper presents an innovative framework that provides an enhanced approach to optimize resource utilization in cloud-native systems by utilizing both workload prediction and dependency relationships between functional graphs for cloudnative applications. In particular, all build and runtime activity involving containers is represented as a node in a dependencyfunction graph; therefore, the relationship of each container can be identified at any time throughout the lifecycle of the container, which offers an ability to manage all containers in the same way and facilitate deterministic cleanup decisions. Additionally, a workload-prediction model is developed to provide predicted short-term demand to indirectly guide the scaling of resources before resource saturation occurs. The proposed framework results in a reduction in the latency associated with scaling operations and minimizes the need to retain unnecessary instances of and images associated with containers, as a result of reducing the time taken for scaling operations due to a workload prediction. The experimental evaluation of the proposed framework demonstrated improved responsiveness and increased resource utilization over traditional container orchestration systems through the use of conventional reactive scaling policies in all tested workload scenarios. The experiment clearly illustrates that the utilization of predictive techniques and structured dependency graph modeling can provide a viable and effective method for increasing the efficiency of cloud-native application infrastructure.

KEYWORDS: Cloud-native application infrastructure, container orchestration, predictive resource management, functional graph modeling, workload prediction, dependency-aware scaling.

I. INTRODUCTION

Cloud-native computing is a way to deploy distributed applications on scalable and elastic infrastructures. It uses containerization and microservice-based architectures to develop applications as components that work independently but share common platform resources. Container orchestration platforms, like Kubernetes make it easier to deploy and manage applications. They handle things like scheduling and service coordination. This helps organizations keep their

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

applications running and flexible in changing environments.. Managing resources in these environments is still a big challenge.

Most orchestration systems use a method to scale resources. They. Decrease resources when certain limits are reached. For example when the CPU or memory usage goes up. This method only reacts after something changes so it might not be fast enough when workloads change suddenly. This can cause applications to run slowly or use resources than they need. Another problem is getting rid of resources. Many platforms use scans or loose rules to clean up. This means unused containers or images might stay in the system longer than necessary. They take up space. Increase costs. Over time this can affect how well the system works.

It gets more complicated when there are many services working together. Cloud-native applications are made up of containerized parts that depend on each other. They are. Run in layers.. When it comes to scaling and cleaning up these dependencies are often ignored. Without a picture of how services are connected it is hard to manage resources. This can make the system less efficient.

To solve these problems we propose a framework. It uses workload prediction and a model that understands dependencies. Container build steps and runtime operations are shown as nodes in a graph. This helps track the life of each component and how it relates to services.

We also use a module to estimate short-term workload trends. This allows the system to make scaling decisions before resources are needed. Our approach is trying to make things better by making sure we use our resources wisely and get rid of what we do not need. This way we can make our system work faster and not waste anything. It gives us a way to use our resources in the best way possible when we are working with cloud systems.

Container orchestration platforms like Kubernetes help manage applications.The proposed framework is designed to work with these platforms. It helps to improve the management of resources in native applications. Native applications and container orchestration platforms, like Kubernetes will benefit from this new framework.

A. Challenge 1: Configuration and Operational Complexity

Cloud native architectures can be really complicated to manage when you compare them to system architectures. In the past people used to deploy and manage applications in a straightforward way. They did not have to do a lot of configuration steps.. Now with modern container based environments you have to do a lot of things. You have to build container images define services, configure networking figure out how to scale and create deployment pipelines.

When people use a microservices approach, for their applications each part of the service has to be configured, deployed and maintained on its own. This makes it easier to make changes and scale. It also means you have to do a lot more work to manage everything. In life these services are all connected and they need each other to work properly.. The platforms that manage these services often treat them like they are separate without thinking about how they depend on each other.

This can cause a lot of problems. For example administrators have to do the configuration steps over and over for each service. They have to adjust how resources are allocated and manually adjust how things scale. As systems get bigger and more complicated it gets harder to keep them stable. This is especially true when services are changed or updated a lot.

It is hard to manage the parts of the system when you do not know how they depend on each other. This is why we need ways to manage these cloud native environments. We need to think about how the services depend on each other. The services depend on each other in a lot of ways. If we understand how the services depend on each other we can use that information to make management easier.

We might be able to avoid doing the configuration steps, for the services over. This will make it easier to manage the services. Managing the services will be a lot simpler if we do this. The services will be easier to manage.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

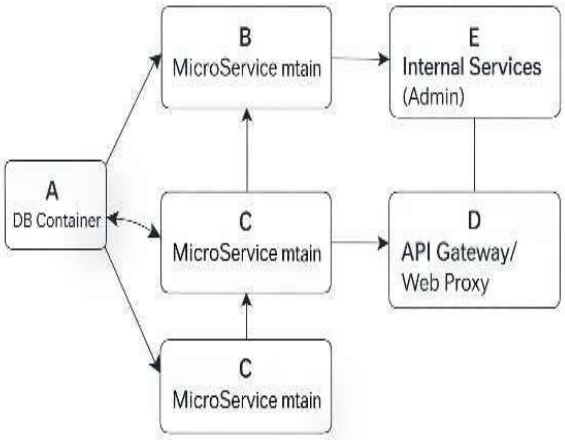


Fig.1. Simplified Microservice Dependency Structure

B. Challenge 2: Inefficient and Reactive Resource Reclamation

In cloud systems container images and the parts that run them are connected in complicated ways. The usual tools to manage these systems like Kubernetes mostly look at the system every now and then or use a method to get rid of things that are not needed anymore when they reach a certain point. This way of doing things does not always work well. Can lead to problems like holding onto things for too long using up too much space and wasting computer power.

Also when it comes to deciding how many resources to use and when to get rid of them these decisions are usually made without thinking about how the different parts of the system work. Because of this some parts might stay active for longer than they need to or resources might be gotten rid of without thinking about if they will be needed soon. This way of doing things can cost money and make the system less efficient especially when there are a lot of changes in how much work the system is doing.

To make this better we need a way of doing things that can figure out how the different parts of the system are connected without having to look at the whole system all the time. We also need it to make decisions about when to add or remove resources based on what it thinks will happen with the work the system is doing. If we use special analysis tools and a model that understands how the different parts of the system work together we can make the system run smoothly and get rid of things we do not need anymore at the right time. This means we can get rid of things we do not need don't keep copies of things and have more control over the parts of the system, in cloud systems that are based on containers.

II. RELATED WORK

Research that is relevant to this work includes things like modeling systems and managing resources in environments. People have used abstractions a lot in build systems and software environments that can be reproduced. Models that use graphs to show how things are executed make it possible to track dependencies and recalculate things in a way.

These approaches show how complicated workflows can be broken down into functional transformations that can be combined. However most of the existing work focuses on making sure things are reproducible when they are being built

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

than managing the lifecycle of containers at runtime. Other people have also looked at using analytics to scale clouds in an adaptive way. They have used models that forecast what will happen in the future and machine learning techniques to make improvements over the way of autoscaling. Even though there have been advances in this area most of the solutions are just added on to the traditional orchestration pipelines without really thinking about how the different services depend on each other. This means that the decisions about scaling are not really connected to the mechanisms that control the lifecycle. When it comes to getting resources from containers platforms like Kubernetes use strategies that collect garbage periodically.

While this works okay in situations it can take a while to get back the resources and sometimes things that are not being used are kept for too long. Most of the work on optimizing container ecosystems has focused on making images faster to start up rather than making sure that resources are cleaned up in a deterministic way that takes into account dependencies. Even though people have studied functional graph modeling and predictive scaling separately not research has combined these two ideas into one framework for orchestrating things.

The approach that is being proposed here tries to fill this gap by combining models of dependencies with predictions of what the workload will be which makes it possible to scale and get back resources in a way, in cloud-native systems. This approach uses container resource management and predictive analytics to make cloud environments work better. Container lifecycle management is a part of this approach, which is why it is a key focus of the proposed work. Cloud environments and container ecosystems are systems that require careful management, which is what the proposed approach is designed to do.

III. SYSTEM MODEL AND MATHEMATICAL FORMULATION

So when we talk about the framework we have to think about how to manage the life of a container. We do this by using a graph. This graph knows about the dependencies between the things, in the container. The framework uses this graph to understand the life of a container and its dependencies.

A. What The Graph Looks Like
The cloud environment is like a map with lots of points and lines that connect them.
G = (V, E)
Here:
* V is a set of points that represent things we do with containers.
* E is a set of lines between these points that show how they depend on each other.
Each point in the graph is like a step in a process such as:
* Building an image
* Running a container
* Connecting a service
* Stopping or cleaning up
Each of these steps is like a machine that changes the state of the system.
The graph is run in a way that respects the order of the steps so we do not get confused.

B. Keeping Track Of References
To make sure we can clean up things when they are not needed we count how many other steps depend on each one:
RC(v) = the number of steps that depend on v
When this count is zero and no other step is using it we can clean it up.
This way we do not need to check everything to see what we can clean up.

C. Predicting The Workload
We can think of the workload as a series of numbers over time:
W(t)

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

We use a model to predict what the workload will be in the future:
$\hat{W}(t + \Delta) = F(W(t. K)... W(t))$ Here:
* k is how back we look
* Δ is how ahead we predict
* F is the magic formula that makes the prediction We then figure out how many containers we need:
$C\_req(t + \Delta)$ = the smallest whole number that is bigger than or equal, to $\hat{W}(t + \Delta)$ divided by μ
Here μ is how much work one container can do.
D. What We Want To Achieve
The system tries to minimize the cost:
$\min \sum\_t (\alpha R(t) + \beta L(t))$ Here:
* R(t) is how resources we use
* L(t) is how long things take
* α and β are numbers that help us decide what is important
By predicting what the workload will be we can make things faster and cleaner.

IV. PROPOSED FRAMEWORK ARCHITECTURE

Figure 2 shows the design of the proposed framework. This framework puts together analytics and functional graph semantics to get the best cloud-native resource optimization.

The framework has five parts: the Monitoring Layer, Predictive Analytics Module, Functional Graph Engine, Scaling and Lifecycle Controller and Container Runtime Interface.

These parts of the framework work together to change the way things are normally done. The framework turns a process into a process that is predictive and controlled by the meaning of the data. The framework uses the Predictive Analytics Module and the Functional Graph Engine to make this happen. The Monitoring Layer and the Container Runtime Interface also play a role in the framework. The framework is, about making the cloud-native resource optimization better by using the Predictive Analytics Module and the Functional Graph Engine to make good decisions.

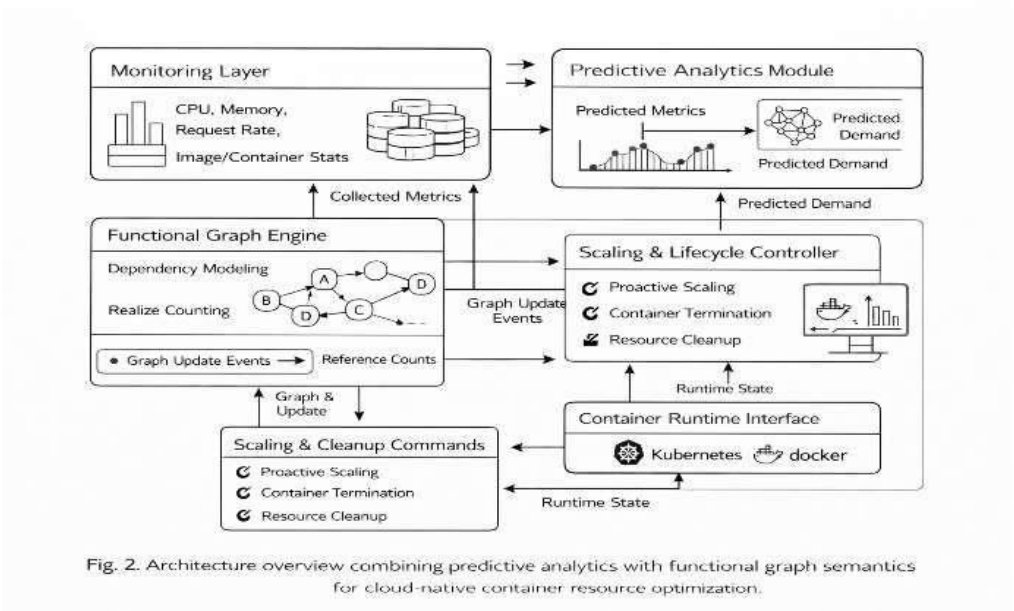


Fig. 2. Architecture overview combining predictive analytics with functional graph semantics for cloud-native container resource optimization.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

1. Monitoring Layer
The Monitoring Layer keeps track of how the container runtime environment's performing. It collects information like:

* How much CPU is being used
* How much memory is being used
* How many requests are being made
* The status of containers
* How often images are being used
This information is used for two things:

1. So we can see whats happening with the system now.
2. So we can guess what might happen with the workload in the future.
This layer is different from systems that just check if things are above or below a certain limit. Instead it sends information to a predictive module to try to figure out what might happen next.

The Monitoring Layer collects metrics from the container runtime environment.

These metrics include CPU utilization, memory consumption, request rate, container status and image usage statistics.
The Monitoring Layer uses these metrics for two purposes:
1. To provide real-time visibility into the system.
2. As input features, for estimating workload.

The Monitoring Layer does not directly trigger scaling actions. It sends metrics to the predictive module for analysis. The Monitoring Layer helps with analysis.

2. Predictive Analytics Module
The Predictive Analytics Module looks at past. Current data to guess what workload we will need soon. It uses a method called time-series forecasting to make predictions about how workload we will have in the near future. This helps us prepare for changes. The module gives us a forecast of how resources we will need. The weather forecast is then sent to the Scaling and Lifecycle Controller. The Scaling and Lifecycle Controller looks at this information. It is used for predicting changes, in the workload of the Scaling and Lifecycle Controller.

Our system can figure out when it needs to get bigger or smaller before it actually happens. This is better than reacting when things get busy. The Predictive Analytics Module helps us get ready for changes in workload. It uses past data and current data to make guesses about the future. The Scaling and Lifecycle Controller uses these guesses to make decisions. The Predictive Analytics Module and Scaling and Lifecycle Controller work together to help our system handle changes, in workload. The Predictive Analytics Module makes predictions and the Scaling and Lifecycle Controller uses those predictions.

3. Functional Graph Engine
The Functional Graph Engine is a part of the system. It helps us understand how things like building containers and running services work together. The Functional Graph Engine shows us how these services rely on each other.

So I have this map. It is full of points. There are lots of points on this map. Each point on the map is connected to points with lines. These lines help us see how all the points are connected. The Functional Graph Engine is like a tool that helps us understand how all the points on our map are connected. It is very good at showing the connections between all these points. The Functional Graph Engine makes it easy to see how everything is linked together on our map. The Functional Graph Engine is really good, at showing us how all the points are connected to each other with lines.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

Each point on the map is a change that happens to something. The lines between the points show how these changes affect each other. The Functional Graph Engine keeps track of how each point is used. This helps it make sure everything runs smoothly. When the Functional Graph Engine decides to use a service or stop using it it updates the lines between the points automatically.

If a point is not used the Functional Graph Engine can get rid of it without checking everything all the time. This way of modeling things as a graph helps keep everything. It makes sure that things we still need are not deleted soon. The Functional Graph Engine is really good at keeping the system running correctly. It does a job of managing the Functional Graph Engine and all of its parts. The Functional Graph Engine is very important, for making sure everything works well.

4. Scaling and Lifecycle Controller
The Scaling and Lifecycle Controller is, in charge of figuring out how the workload will behave and making decisions based on that. The main thing it wants to do is make sure the system uses its resources in a way without affecting how the applications work. The Scaling and Lifecycle Controller does three things in the system:
* It gets the resources ready before the workload gets too big
* It stops the containers nicely when they are not needed anymore
* It cleans up to keep the system stable and use resources well

When the Scaling and Lifecycle Controller thinks it needs to make some changes to the scaling or the lifecycle it updates what it knows about how the system's put together. This helps the system figure out how all the parts and services are related to each other. So the system can clean up without hurting the services that still need resources.

The Scaling and Lifecycle Controller does everything together not as tasks. This means it can handle getting resources getting rid of resources and cleaning up in a way that makes sense. The Scaling and Lifecycle Controller helps the system work well and not use many resources. This is good because the Scaling and Lifecycle Controller is always looking at the Scaling and Lifecycle to make sure it is doing its job. The Scaling and Lifecycle Controller is very important for the system to work properly.

5. Container Runtime Interface
The Container Runtime Interface is what actually runs things. It talks to things like Kubernetes or Docker to get things done. This part of the system takes the decisions about how many things to run and how to clean up. It turns those decisions into commands that the system can understand. It also sends information about what's happening back, to the parts of the system that monitor and graph things.

By keeping the decisions about what to do separate from the parts of the system that actually do things the Container Runtime Interface makes the whole system work with cloud environments. This means the Container Runtime Interface and the system can work with lots of Container Runtime Interfaces and cloud environments without having to be changed.

V. ALGORITHM DESIGN

To manage scaling and lifecycle we use a framework. It works through a control loop that reacts to events. This framework helps predict workload and updates graphs based on functions. It also reclaims resources making sure to reference them The goal is to coordinate scaling and lifecycle management in a way. It does this by using events to drive its decisions. The algorithm combines predicting workload, with updating graphs. It also handles reclamation in a way that knows about references.
Algorithm 1: Predictive Dependency-Aware Resource Optimization
Input: We monitor system performance in real-time
Metrics are:
* M(t) which're real-time metrics
* G(V, E) which is a graph showing how different parts of the system depend on each other

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

* W_hist which are past workload observations
* Δ which is how far ahead we want to predict
Output:
The system applies scaling and cleanup actions while it is running

Begin

1. While the system is running do the following:
2. Step 1: Get metrics
3. Get M(t) by collecting metrics from the system now
4. Step 2: Update past workload info
5. Update W_hist with the metrics M(t)
6. Step 3: Predict future workload
7. Use W_hist and Δ to predict workload W_pred
8. Step 4: Figure out required resources
9. Use W_pred to determine how many container instances are needed, C_required
10. Step 5: Compare with current resources
11. Check how many containers are currently active C_current
12. If C_required is greater than C_current
13. Add containers, scale_up by C_required. C_current
14. Update the graph, G to show node
15. Else if C_required is, than C_current then
16. Remove some containers, scale_down by C_current.
C_required
17. Update the graph, G to remove nodes
18. End if
19. Step 6: Clean up unused resources
20. For each part of the graph G do
21. If nothing uses that part reference_count is 0
22. Then free up that resource reclaim_resource
23. End for
24. End while
25: end while

End

Algorithm Explanation
The algorithm works all the time in a loop. It starts by gathering information about how things are running and adds this information to a list of what has happened in the past. The predictive model then tries to figure out what the workload will be like in the future for a period of time. The algorithm uses the predicted workload to decide how containers it needs. If it thinks it will need more than it has now it gets more containers ready ahead of time. On the hand if it thinks it will need fewer containers it gets rid of the extra ones. Every time the algorithm adds or removes containers it updates the list of how everything's connected. When it removes something from the list it checks to see if anything is still using it. If not it gets rid of it without having to check everything all at.

This way of doing things makes sure that the algorithm makes decisions, about adding or removing containers and that it does so in a way that is efficient and easy to understand. The algorithm is always trying to make sure that the containers are working well and that it is not wasting any resources. The predictive model and the algorithm work together to make sure that the workload is handled correctly and that the containers are used in the way possible. The algorithm and the predictive model are always working together to make sure that everything runs smoothly.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

VI. PREDICTIVE MODEL DESIGN

The part of the framework that makes predictions is in charge of figuring out how work the system will have to do in the short term. This helps the system make decisions about handling work before it gets busy.
The predictive model differs from approaches that only react when the system is overwhelmed.
It tries to guess how resources the system will need before it gets too full.
This way the predictive model helps the system prepare ahead of time. The system can handle work better.
The predictive model is useful for the system to get ready, for the work.

A. Input Features
The model works with time-series data from the monitoring layer. The main features that we use as input are time-series data.
CPU usage percentage
* Memory usage in megabytes
* Request rate per second
* Number of containers
* Response latency
These metrics are collected at intervals and stored in a window of size k. The feature vector, at time t is: X_t = [CPU_t, Memory_t, RequestRate_t, Containers_t] The historical data used for prediction is:
* A sequence of feature vectors: X_(t-k)... X_(t-1) X_t

B. Model Architecture
To understand workload patterns over time we use a kind of forecasting model that looks at past data. This model is called a time-series forecasting model. It helps in capturing dependencies in workload patterns. We chose a Long Short-Term Memory (LSTM) network for this task. The reason is that it is good at understanding sequences and can handle increases in traffic.
Here is how our LSTM model works:

- It has an input layer that matches the features we are looking at.
- Then it has one or two layers that use LSTM to understand the sequence.
- After that it has a connected layer that produces the output.
- We use activation for the output because we are trying to predict a number.

This model helps us predict the workload demand at a time which we call t+Δ.
Here Δ is how far ahead we are trying to predict.
The prediction is based on the data from X_(t-k), to X_t.

The model equation is:

$$W\hat{}(t+\Delta) = f\_LSTM\ (X\_(t-k)...\ X\_t).$$

The LSTM model, LSTM model and Long Short-Term Memory network are used to make this prediction. The model uses past workload patterns, workload patterns and sequential trends to make the prediction. The workload demand is predicted using the LSTM model. The LSTM model predicts the workload demand. The prediction is done using a time-series forecasting model, which is an LSTM network.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

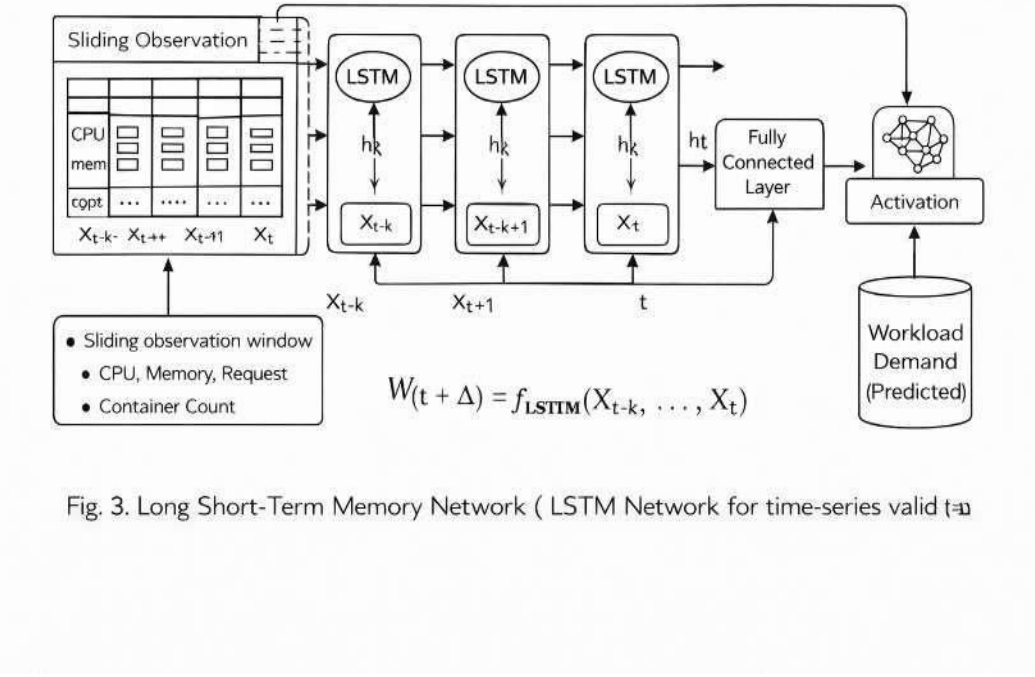


Fig. 3. Long Short-Term Memory Network (LSTM Network for time-series valid t=1

C. Training Procedure

The model is trained using workload data from where it will be used. The goal is to make it as accurate as possible by minimizing an error called mean squared error (MSE).

This error is calculated like this:

$$L=1/N \sum_{i=1}^{N} (W_i - \hat{W}_i)^2$$

The data is split into two parts to make sure the model does not get too good at fitting the training data. The model is then put into the Predictive Analytics Module. It is updated regularly with new data to stay accurate.

D. Scaling Decision Mapping

$$C_{required} = \lceil (\hat{W}(t+\Delta)) / \theta \rceil$$

Here θ is how workload one container instance can handle.

When we do this mapping the system can get bigger before it gets much to handle. This helps to cut down on waiting times and problems with how the system works when a lot of people want to use it all at once. The system can do something about it early before many people want to use it and it gets overwhelmed. This is how the system avoids taking a time to start up and getting slower, over time.

The amount of work the system thinks it will have to do, which is called the predicted workload $\hat{W}(t+\Delta)$. How much work each container instance can handle which is called θ are very important.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

We figure out how container instances we need, which is called C_required by using the predicted workload $W\hat{}(t+\Delta)$ and θ.

E. Integration with Functional Graph Semantics

When the system makes decisions about scaling it does not do anything away. The Scaling and Lifecycle Controller gets these decisions. Then it updates the graph that shows how everything is connected. When you add or remove containers the system changes the graph and the connections between things. It also updates the counts.

This way the system that predicts what will happen works, in a way that makes sense. It makes sure that everything is consistent when the system is scaling up or down and cleaning up after itself. The predictive intelligence and the Scaling and Lifecycle Controller work together to keep everything running smoothly and to make sure that the system is always aware of what's connected to what.

VII. EXPERIMENTAL SETUP

We wanted to see how well the new framework for making predictions and understanding dependencies would work. So we did some tests in a cloud environment that we could control. We compared our approach to the ways of scaling and cleaning up to see which one worked better when things got really busy.

Figure 4 shows what the test setup looked like with all the parts like making workloads keeping an eye on the system making predictions and running our approach to see how it worked.

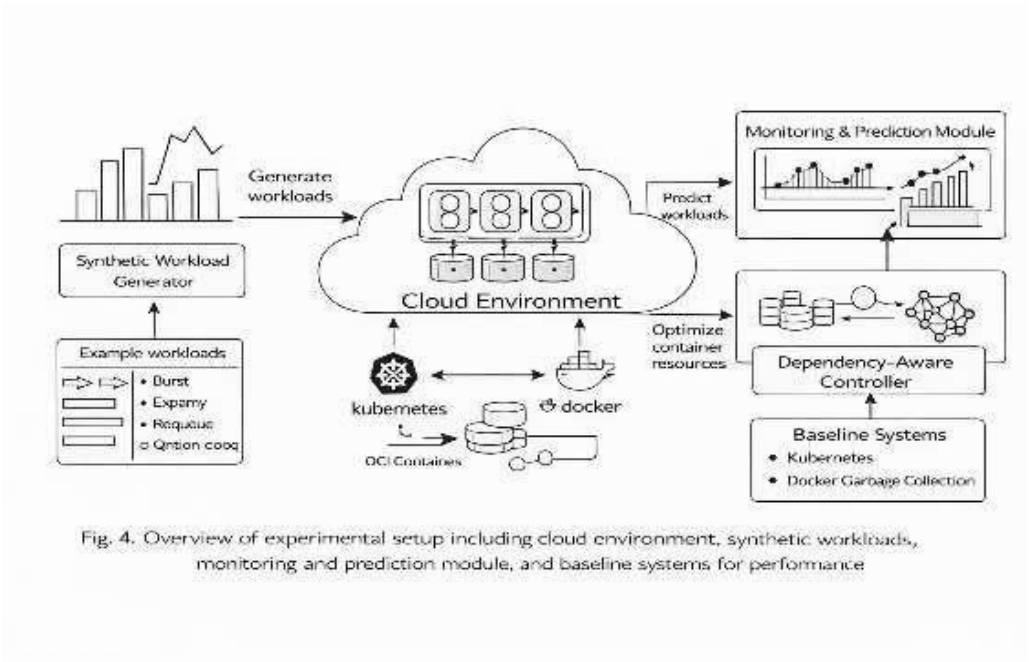


Fig. 4. Overview of experimental setup including cloud environment, synthetic workloads, monitoring and prediction module, and baseline systems for performance

A. Experimental Environment

The experiments were done on a cluster that had:

- 4 CPU cores, for each node
- 16 GB of RAM
- Ubuntu 22.04 LTS
- Kubernetes version 1.28
- Docker Engine 24.x

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

The cluster had one master node and two worker nodes. All the services were started using container images that followed the OCI rules.

The proposed framework was set up as a controller that talked to the Kubernetes API server. This way the framework worked well with the existing orchestration mechanisms and the Kubernetes runtime was not changed.

B. Workload Generation
A tool was created to mimic traffic patterns. It was used to test three types of traffic:
- Stable traffic. Where the number of requests stays the same
- Burst traffic. Where there are sudden spikes, in traffic
- Traffic that increases slowly. Where the number of requests grows bit by bit

Each test was run for 30 minutes. We repeated each test five times to get accurate results.

C. Predictive Model Training
The model that uses LSTM was trained with information from things that happened before.
Training configuration is like this:
- The sequence length is twenty time steps
- We want to predict what happens five time steps later
- The model has sixty four hidden units
- The model uses two LSTM layers
- We use thirty two things at a time for training
- The learning rate is zero point zero zero one
- We use the Adam optimizer ∟ The loss function is Mean Squared Error ∟ The model was trained for fifty epochs.

We split the information into two parts: eighty percent for training and twenty percent, for checking.
After we trained the model we put it in the Predictive Analytics Module so it can be used to make predictions in time with the LSTM model and the Predictive Analytics Module working together with the LSTM model.

D. Baseline Systems
The new framework was compared to an other things:
1. Kubernetes Horizontal Pod Autoscaler, which is based on how the computer is being used
2. Docker garbage collection, which happens on a schedule

All the systems were tested on the same kind of computer and, with the same kind of workload.

E. Evaluation Procedure
For each workload scenario:
For each workload scenario the framework did a things:
- It recorded when the containers were scaled up or down
- It kept track of how the computer was being used ∟ It wrote down when the cleanup happened
- It measured how long it took for things to happen

The framework did all these things times and then it averaged the results to make sure they were accurate. The framework was tested with Kubernetes Horizontal Pod Autoscaler and Docker garbage collection to see how it worked.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

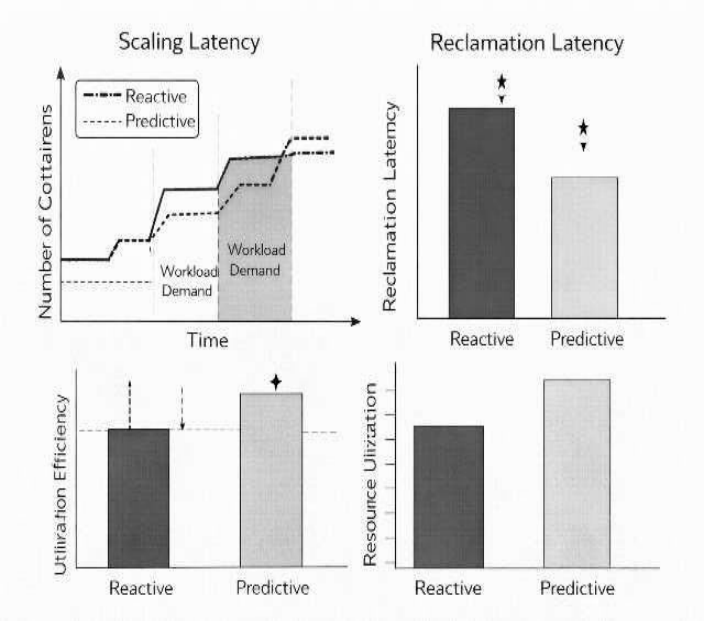
Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

VIII. PERFORMANCE METRICS DEFINITION

To see how well the proposed framework works we defined four metrics. These metrics look at how responsive the system's how efficiently it reclaims resources and how well it uses resources when the workload changes.
* Figure 5 shows how the proposed framework compares to reactive methods.

The proposed framework is predictive. Takes dependencies into account. It behaves differently under workload conditions. The conventional reactive orchestration does not consider predictions or dependencies. This makes the proposed framework more efficient, in terms of resource utilization.



A. Scaling Latency
Scaling latency is the time it takes for the system to settle on the number of containers after a change in workload. T_scale = T_stable. T_event where:
T_event's when the workload change happens
T_stable is when the number of containers is just right and stays that way.
Systems that react to changes usually take a time to settle, T_scale because they only adjust when something gets too high or too low.

The predictive framework we are suggesting reduces this delay by scaling up or down before the change happens based on what we expect the demand to be. It does this to make sure the system has the number of containers.
This way the system can handle changes in workload better.The goal is to make T_scale as small, as possible. The predictive framework helps to achieve this.

B. Reclamation Latency
Reclamation latency is the time it takes to clean up resources after they are no longer being used.
Reclamation latency is calculated by subtracting the time when a resource like a container or image becomes unused from the time when the resource is actually removed.
The formula for this is: T_reclaim = T_cleanup - T_inactive
Here is what these terms mean:

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

* T_inactive is the time when a container or image is no longer being used
* T_cleanup is the time when the resource is removed
There are garbage collection mechanisms that can cause delays that we cannot predict.
On the hand the reference-aware graph mechanism allows for cleanup to happen as soon as it is needed which reduces the reclamation latency or the time it takes to clean up resources after they are no longer being used which is the reclamation latency.

C. Resource Utilization Efficiency
Resource utilization efficiency is about how the resources that are given to something are actually used when it is running. This is calculated by doing this:

Ueff = The amount of resources that are really used / The amount of resources that're available to use.
When the utilization is higher it means that there are not many extra resources just sitting there not being used. If you look at Figure 5 which's at the bottom left you can see that when things are scaled in a proactive way the resources that are allocated are closer, to what is actually needed for the workload and that makes things more efficient. Resource utilization efficiency is really important because it helps to make sure that resources are being used in the way possible.

D. Overall Resource Utilization
When we look at how the system uses its resources we see that it is using a lot of the computers brain power and memory. This is what is happening when we do the experiment. If we look at Figure 5 which's, at the bottom right we can see that our way of predicting things helps the system keep using its resources in a good way. This means that the system does not get slower and can keep working. The system uses more of its resources when we use our framework.

IX. CONCLUSION

This paper is about a way to make cloud computing work better. It combines two things: analytics and functional graph semantics. The goal is to make cloud computing resources work efficiently. The new approach solves two problems in cloud computing: scaling and getting rid of resources that are not needed anymore. It does this by looking at how containers work and predicting what resources will be needed. This means that the system can get ready for what's coming and get rid of things it does not need without having to check everything all the time. The system has two parts: one that looks at how containers depend on each other and one that predicts what resources will be needed. This helps the system get resources ready when they are needed and get rid of them when they are not needed anymore. The cloud computing system is made to work with the cloud computing systems that're already out there. This means the cloud computing system can be used in life with the existing cloud computing systems without having to change the way the underlying system works. The people who created the cloud computing system want to keep working on the cloud computing system. They want the cloud computing system to work with cloud computing systems and make the cloud computing system even better at figuring out what resources the cloud computing system will need. They also want to make sure the cloud computing system works well when a lot of people are using the cloud computing system at the time. They will try out the cloud computing system with a lot of users to see how well the cloud computing system works.
The new cloud computing system is better, than systems because the cloud computing system can figure out what resources the cloud computing system will need and get them ready. This makes the cloud computing system more efficient and easier to control. Cloud computing systems that use this approach will work better. Be more efficient. The Cloud Native Resource Optimization system is an improvement. Cloud Native Resource Optimization is what this system is, about. It makes Cloud Native systems work better.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504174|

REFERENCES

[1] Cloud Native Computing Foundation, Cloud Native Survey Report 2024. Cloud Native Computing Foundation, 2024.
[2] B. Burns, B. Grant, D. Oppenheimer, E. Brewer and J. Wilkes "Kubernetes and Container Management Systems: What We Learned," Communications of the ACM vol. 59, No. 5 Pp. 50–57, 2016.
[3] Kubernetes Documentation, "Garbage Collection " Kubernetes Official Documentation, 2024. You can find it here: https://kubernetes.io/docs/
[4] A. Mokhov, N. Mitchell and S. Peyton Jones "Build Systems: A New Look," Proceedings of the ACM on Programming Languages, vol. 2 ICFP, 2018.
[5] U. A. Acar, G. E. Blelloch and R. Harper "Adaptive Functional Programming," ACM Transactions on Programming Languages and Systems vol. 28 No. 6 Pp. 990–1034 2006.
[6] E. Dolstra, A. Löh and N. Pierron "NixOS: A Linux Distribution," Journal of Functional Programming vol. 20 No. 5–6 Pp. 577–615, 2010.
[7] D. Malka, N. Palix and J. Lawall " Builds: Better Software Supply Chains," IEEE Software, vol. 36 No. 5 Pp. 78–84, 2019.
[8] D. Malka, N. Palix and J. Lawall "Reproducibility in Large Software Ecosystems " Empirical Software Engineering, vol. 25 Pp. 1–30, 2020.
[9] P. Vasconcelos, J. Sousa and M. Florido "Functional Programming in Cloud Computing " Journal of Systems and Software vol. 170 2020.
[10] N. Mitchell, "Shake: A Build System " Proceedings of the Haskell Symposium, 2012.
[11] Meta Engineering, "Buck2: A Modern Build System " Meta Open Source, 2023
[12] P. Odersky et al. "Unison: A Programming Language " Unison Computing, 2021.
[13] G. Gousios, M. Pinzger and A. Van Deursen "Trends in Build Systems " IEEE Software, vol. 37 No. 4 Pp. 76–83 2020.
[14] Open Container Initiative, OCI Image Specification v1.1, 2023.
[15] Amazon Web Services "Container Image Management," AWS Documentation, 2023.
[16] Microsoft Azure, "Azure Container Registry " Microsoft Docs, 2023.
[17] S. Harter et al. "Slacker: Distribution, with Docker Containers " Proceedings of USENIX ATC, 2016.
[18] containerd Project, "stargz-snapshotter," 2022.
[19] Alibaba Cloud, "Nydus: A File System," 2022.
[20] AWS, "Seekable OCI," 2023.
[21] A. Merino et al. "Lazy-Loading Container Snapshotters," Future Generation Computer Systems, 2021.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details