2026 Volume: 02 Issue: 11 www.ijrpa.com ISSN 2456-9995 Research Article

International Journal Research Publication Analysis

Page: 01-13

MACHINE LEARNING-BASED PREDICTIVE RESOURCE MANAGEMENT AND AUTO SCALING IN CLOUD COMPUTING

***Eswari Patnana, Sai Sanjana Konapala, Shaik Mohammad Sameer, Shaik Ahmad, Kedar Pinniboyina**

Department of Computer Science and Engineering, GMR Institute of Technology Deemed to be University, Rajam, 532127, Andhra Pradesh, India.

Article Received: 15 October 2026
Article Revised: 04 November 2026
Published on: 23 November 2026

***Corresponding Author: Eswari Patnana**
Department of Computer Science and Engineering, GMR Institute of Technology Deemed to be University, Rajam, 532127, Andhra Pradesh, India.

DOI: https://doi-doi.org/101555/ijrpa.4293

ABSTRACT

Cloud computing provides flexible and scalable resources according to application workload, but improper resource allocation can cause performance issues, resource wastage, and increased costs. This project proposes an AI-Based Predictive Cloud Resource Management System that uses machine learning to predict future workloads and manage cloud resources proactively. The system gathers past and live data such as CPU usage, memory usage, user numbers and requests. This data is cleaned and used to train a machine learning model that can predict what the future workload will be and determine future resources needed. Based on the prediction, the resource management module increases, maintains, or decreases application instances. A monitoring module tracks resource utilization and verifies the scaling process, while a web dashboard displays current usage, predicted workload, active instances, and scaling decisions. This system is demonstrated by a simple contained cloud application. It can show changes to workloads and resource allocations. This system is designed to help improve application performance, optimize resources and limit the use of resources in the cloud.

KEYWORDS: Cloud Computing, Machine Learning, Predictive Resource Management, Workload Prediction, Auto Scaling, Resource Optimization, Cloud Monitoring, Containerization.

1. INTRODUCTION

The ability to predict resource utilisation on the cloud has been an area of high interest in recent years [1] [5]. In many applications, demand rarely stays constant throughout the day. There are periods of low demand during the night, increased demand in the morning and unpredictable peaks during sales or new releases. While cloud platforms allow users to scale resources up or down as they wish [6] [7] there must be a decision made on when to do it.

The majority of systems will use a simple high/low rule where when CPU usage reaches a certain high level a new server is added, and when CPU usage drops below a certain low level servers are removed [8]. This is very easy to implement however it does cause lag. As it takes a few minutes to start up a new instance, this causes delays for users. Alternatively, a number of extra servers can be kept running constantly which removes any lag issues however it means that you are paying to run idle servers.

Our proactive approach is CloudScale. It gathers historical and real-time metrics related to the workload, such as CPU and memory usage, number of users and request rates. These are preprocessed and used to train a machine learning model that can then predict the upcoming workload and required resources. We use an XGBoost regressor [15] to predict the future workload 15 minutes ahead.

The resource management module will increase, maintain or decrease based upon the prediction. The monitoring module allows us to monitor resource usage and ensure that the scaling is working correctly. Using the web-based dashboard, we can see real time data of our current resource usage, predicted load, instances running and scaling decisions. This has been demonstrated using a basic containerised application to allow us to clearly see the scaling decisions based on the changing load. We also looked at 3 alternative models, created an anomaly alert to notify us if any strange values are recorded, and have an audit log to track every decision made.

The goal is better application performance, better use of the resources we pay for, and less unnecessary cloud usage.

2. Literature Review

A highly desired aspect of cloud computing is the ability to scale up/down resources.This has been described as a key characteristic of cloud computing [6] by Armbrust et al., and is one of 5 essential characteristics in the NIST definition [7].Knowing when to use that elasticity is a different problem, and quite a lot of research has gone into it.

Lorido-Botran et al. [8] surveyed autoscaling methods for elastic applications and grouped

them into threshold rules, queueing theory, control theory, reinforcement learning and time-series analysis. From the survey we can see that rule based scaling is still the most popular. This is because it is very easy to do, although this is only reactive. A similar reactive approach was taken by Gandhi et al. [9], in multi-tier data centres.AutoScale was created to allow for over-provisioning during sudden changes. This taught us that taking servers away too quickly can cause problems so having a cool down period in our decision engine helps prevent this.

In addition, other works have attempted to predict the workload. A system called PRESS was developed by Gong et al. [10]. This uses signal processing techniques to find recurring patterns within the workload. If no pattern is found then a Markov chain is used. Another system called CloudScale has been described by Shen et al. [11]. This predicts demand online and includes a small amount of padding to account for any error in the prediction. It manages any conflicts between several applications attempting to use the same host. A time-series based model was developed by Roy et al. [12], where the number of resources are selected by looking a few steps ahead in the workload in order to keep costs low. Calheiros et al. [13] also used ARIMA to predict cloud workloads. They demonstrated that better predictions can lead to an increase in perceived quality of service for users whilst using fewer resources. These techniques do provide reasonably good results for slow changing demands. However, linear time-series models can struggle with sudden changes in traffic.

Newer work continues along these lines. Rossi et al. [1] work on workload forecasts that are aware of uncertainty and can carry what a model has learned over to another workload. Pothu and Kailasam [2] pair a hybrid tree-enhanced vector machine model for workload prediction with priority-based resource allocation for proactive auto-scaling. Sus and Nawrocki [3] use a signature-based approach that combines machine learning prediction of resource usage with anomaly detection, which is close to how we pair forecasting with an anomaly alarm. Meta reinforcement learning has been used on predictive auto-scaling [4], and Zanussi et al.propose a load aware framework for predictive auto-scaling [5]. This shows forecasting driven scaling remains an active research area.

Machine learning techniques have also been applied to large amounts of historical data. The Resource Central system within Microsoft Azure learns from previous virtual machine behaviour using tree based models [14]. These models make predictions which are then passed onto the resource manager. This implies that tree based learning models are suitable for use with infrastructure data. XGBoost [15] is a gradient boosted tree library well known for providing accurate results and fast predictions on tabular data. We use this for forecasting.

We compare our results to a random forest [16], Long Short Term Memory network (LSTM) [17], and ElasticNet linear model [18]. Random forests produce average results from many decision trees,meaning they work well with outliers in the data. LSTMs are designed to remember long sequences of data, making them useful for capturing daily patterns. Lastly, ElasticNet is a form of regularised linear regression,this makes it fast and easy to use as a baseline.

Any forecasting technique can only be as good as the data it is using. Chandola et al.[19] provide an overview of anomaly detection techniques, including statistical based techniques which identify values that differ significantly from the expected value. These are used here to provide a statistical envelope that if exceeded by an anomalous CPU reading will alert the administrator. This is combined into a working dashboard providing forecasting, model comparison, anomaly alerts, scaling decisions and audit trail.

3. PROPOSED METHODOLOGY

3.1 System Overview

he below Figure 1 outlines the overall process. The monitoring layer gathers host metrics using psutil and container metrics using the Docker SDK. These are pre-processed before passing to the XGBoost regressor to provide a prediction for the workload and CPU. At the scaling decision step, these predictions are contrasted against the threshold values, instance limits are considered, and cooling off rules applied.Finally at the action step, a recommendation for scale up or down is provided. Telemetry and events are recorded in SQLite using SQLAlchemy. With a dashboard created using React to display real time metrics,predictions, decisions and configuration. A feedback loop provides the results back to the monitoring layer so that subsequent data will reflect past decisions. An anomaly module and audit trail are also present. All services are run as individual containers using Docker Compose [21].

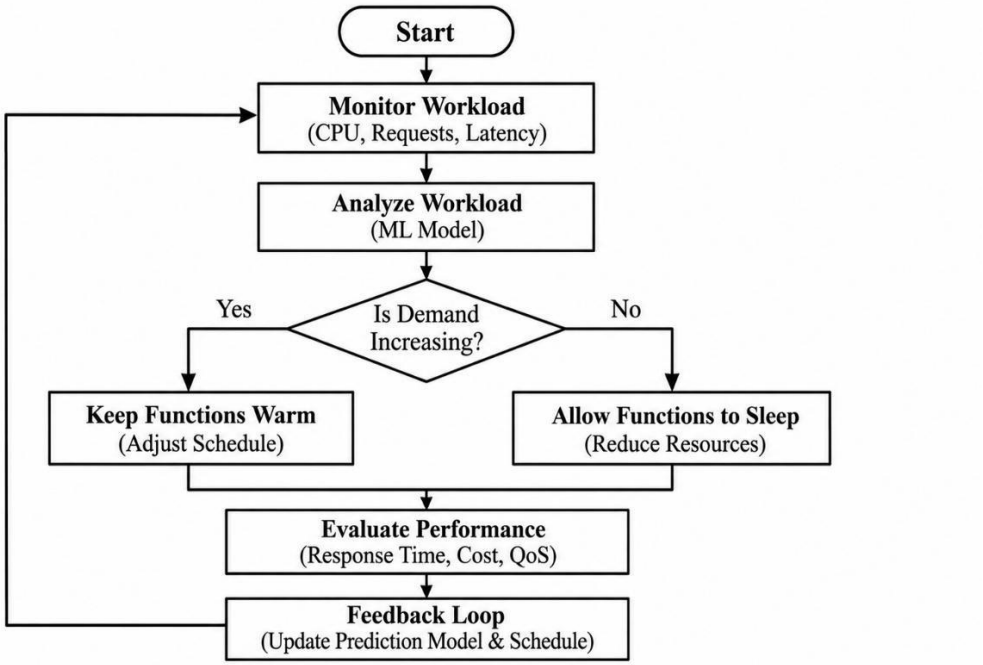


Fig. 1. System overview and workflow of CloudScale.

3.2 *Data Collection*

The monitoring system collects information about the host using psutil, and the containers using the docker SDK.The docker socket is accessed via a read only proxy. This means that it can only collect data, and can't start/stop anything. Table 1 shows what information is being collected.

Table 1. Metrics collected by the monitor.

Metric	Source	Shown on
CPU and memory usage	psutil	Live Monitoring
Request rate and throughput	Application monitor	Live Monitoring
Average, P95 and P99 latency	Application monitor	Live Monitoring
Error rate and queue depth	Application monitor	Live Monitoring
Container health, CPU, memory	Docker SDK	Container Fleet

3.3 *Forecasting Models*

The original data was first preprocessed and ordered. Features such as moving average and lag were then created.These were used to predict CPU usage and workload 15 minutes in advance. The data was divided into training data (older data) and test data (newer data). To ensure the model does not have access to future information [20],the time series data is not shuffled.

XGBoost is our production model. The algorithm uses Sequential Decision Trees, where each

new tree will correct errors made by previous trees [15]. This model works well in cases of abrupt nonlinear increases in traffic and has a small memory footprint. However, it requires quality time series features. Three other models are available on the Model Arena; a Random Forest, an LSTM network and an ElasticNet baseline. Any of these models can be promoted to production by an administrator.

3.4 *Scaling Decision*

The decision engine will compare the prediction to two thresholds. If the predicted CPU is greater than the high threshold then we recommend adding an instance. If the predicted CPU is less than the low threshold we recommend removing an instance. If it falls within these thresholds then leave the setup as it is. There are three protections in place around this rule (see Figure 2). There is a cool down timer to prevent continuous actions, limits on how many instances you can have so there is always a minimum and maximum number of instances, and also an anomaly alert to alert you to any unusual values. The thresholds in Figure 2 are just examples and can be adjusted using the System Settings page.

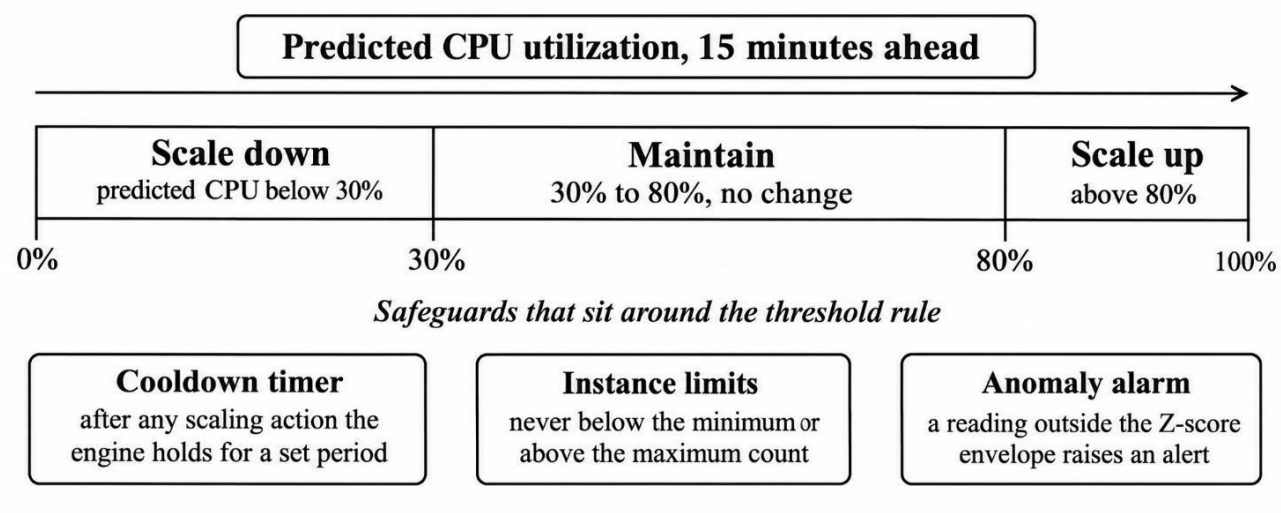


Fig. 2. Scaling decision zones and the safeguards around them. Threshold values are examples.

Table 2. Decision rules.

Condition	Action
Predicted CPU above upper threshold	Scale up
Predicted CPU below lower threshold	Scale down
Predicted CPU between the thresholds	Maintain
Cooldown still running	Hold
Minimum or maximum instance count reached	No change

3.5 *Anomaly Detection*

This module creates a Gaussian distribution based on a mean and standard deviation. If the value of the CPU exceeds the top edge of the Gaussian it is identified as an anomaly. If it exceeds three standard deviations then this is flagged as a critical event [19]. The user can select low, medium or high sensitivity to adjust the range of the envelope.

3.6 *Working Procedure*

1. Data collection from different sources using psutil and the Docker SDK: CPU usage, memory usage, network I/O, request rate and response time.

2. Preprocessing: Removing missing values, sorting data by timestamps and feature engineering.

3. ML Model: Predicting workload and CPU usage 15 minutes ahead using XGBoost Regressor.

4. Scale Decision: Compare prediction to thresholds, check instance limitations and cooldown conditions.

5. Recommendation: Producing a recommendation for scaling up or down.

6.Dashboard & Visualization: Visualizing real time metrics, comparison of prediction and real values, scaling history and configuration. Telemetry and event data are stored in SQLite.

The flowchart in Figure 3 depicts the described workflow, including decision points and repeating the process for every sampling interval.

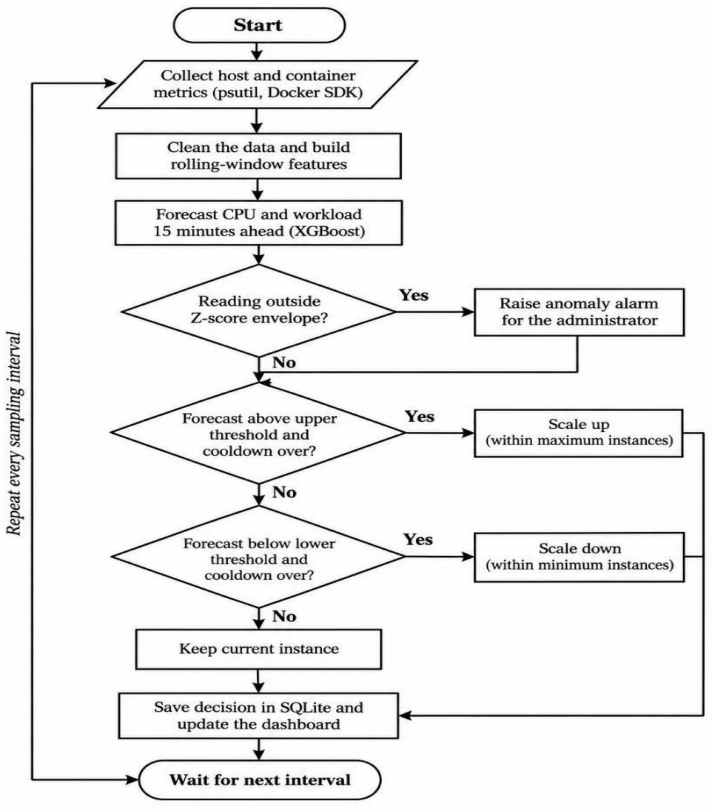


Fig. 3. Flow chart of the scaling procedure.

4. RESULTS

This section displays the output of the running system. All data shown has been captured via screenshots of the dashboard on 8th November 2026.

4.1 *Live Monitoring*

The Live Monitoring page is updated every 5 seconds. This can display data for a period between the last hour and 7 days. At the time of taking the screenshot, the system was not heavily loaded. CPU usage was 5.4% and memory was 43.6%. Requests were being handled at a rate of 9.91 requests per second with no errors being logged. The average latency was 38.8ms with 72.0ms at the 95th percentile and 99.2ms at the 99th.There was no requests in the queue.

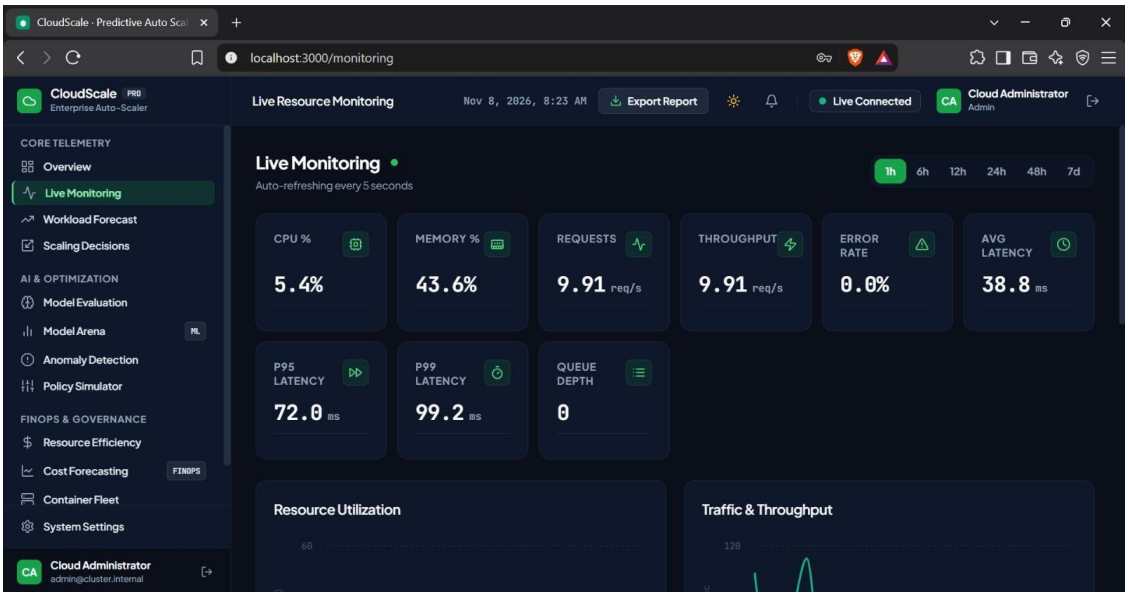


Fig. 4. Live Monitoring page.

Table 4. Snapshot from the Live Monitoring page.

Metric	Value	Metric	Value
CPU	5.4%	Average latency	38.8 ms
Memory	43.6%	P95 latency	72.0 ms
Requests	9.91 req/s	P99 latency	99.2 ms
Error rate	0.0%	Queue depth	0

4.2 *Model Comparison*

In Figure 5, the models are shown in the Model Arena. With the smallest mean prediction error of 3.71%, XGBoost provided answers in 14.2ms. The LSTM performed similarly, with only an increase in prediction error to 3.93%. However, the time taken increased to 41.5ms, almost three times longer than XGBoost. The random forest model also performed well, with a prediction error of 4.12% and a response time of 28.6ms. The fastest model was ElasticNet

with a response time of 3.8ms however the prediction error was much larger at 6.86%. This is nearly twice as much as XGBoost and expected due to ElasticNet's baseline nature not being able to adapt to sharp peaks. As seen in Table 5 and Figure 6 XGBoost has a good balance between accuracy and speed. Therefore, XGBoost is highlighted as the model to be used in production.

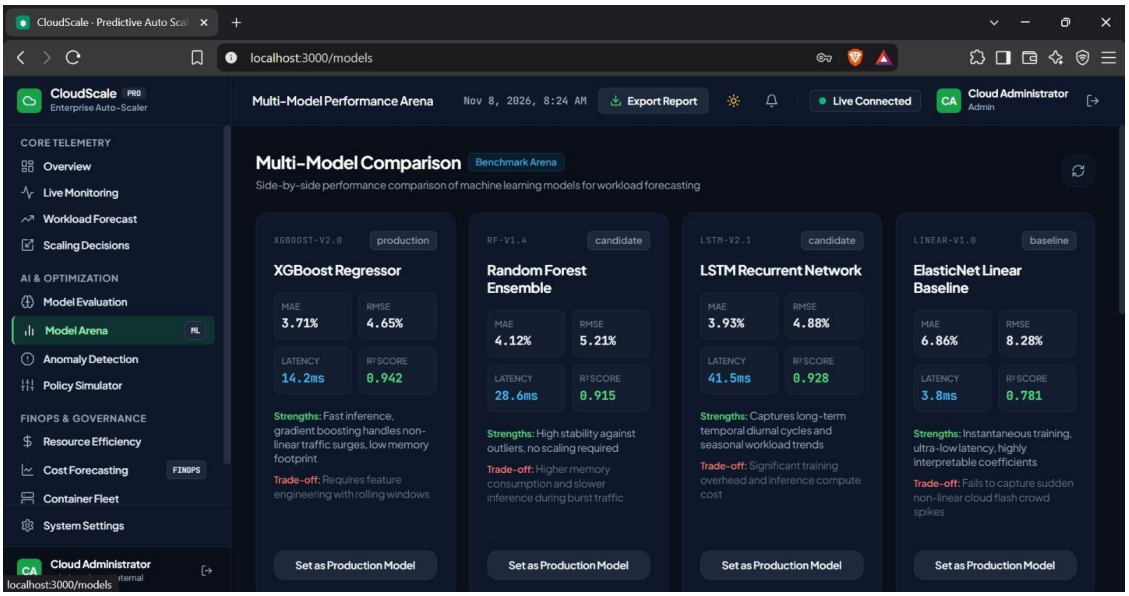


Fig. 5. Model Arena page.

Table 5. Forecasting models compared.

Model	Role	Avg. error	Latency
XGBoost Regressor	Production	3.71%	14.2 ms
Random Forest	Candidate	4.12%	28.6 ms
LSTM Network	Candidate	3.93%	41.5 ms
ElasticNet	Baseline	6.86%	3.8 ms

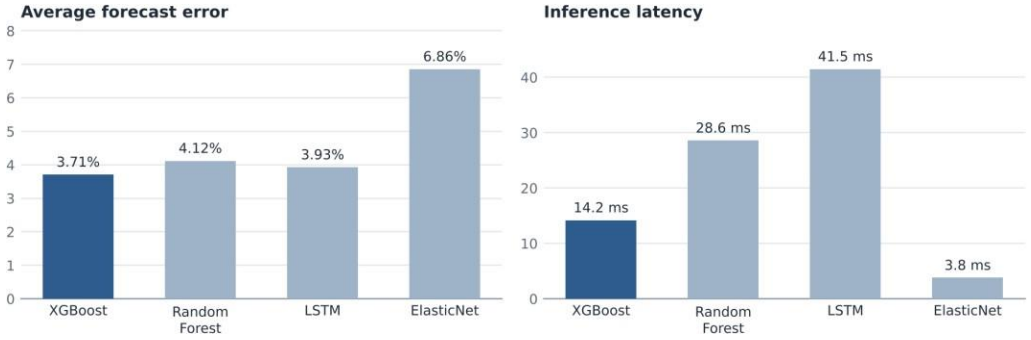


Fig. 6. Average forecast error and inference latency of the four models.

4.3 *Anomaly Detection*

In the Anomaly Detection page (Figure 7) we plot the real CPU values against the dynamic

threshold. We can see that there were 68 anomalies found in the last 24h; of those 51 exceeded the 3σ cut off and were therefore marked critical. The health score was 94%. In the tooltip we can see a data point at 07:11 AM where the CPU was only 64.7% compared to the upper limit of approximately 73.0% so we can see that this is still within the envelope. We can also see that most of the red points are clustered at the start of the day with a few individual spikes. The width of the envelope was 2.1σ at medium sensitivity.

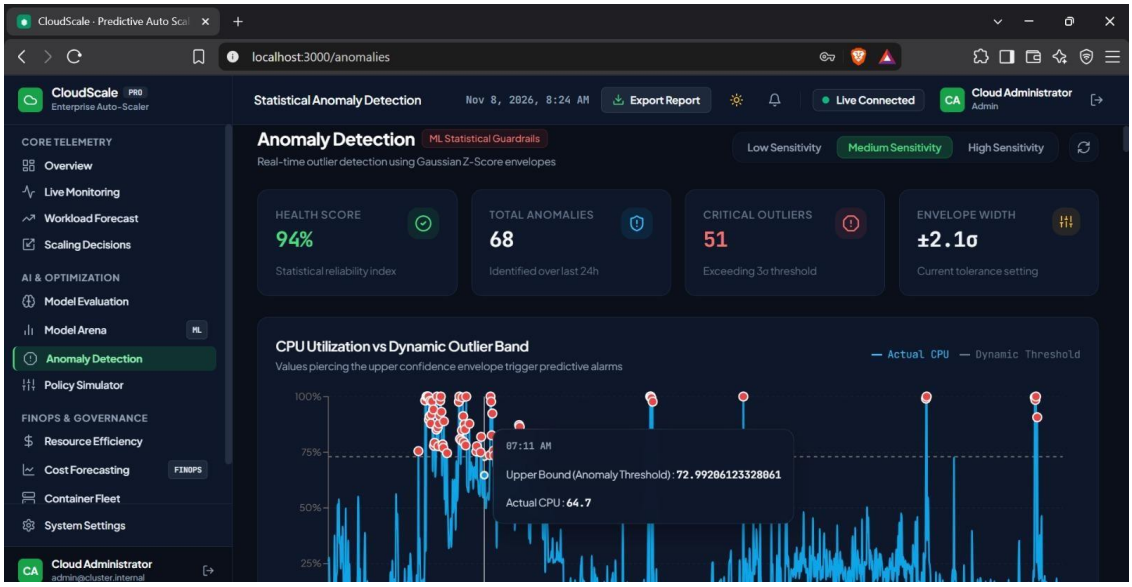


Fig. 7. Anomaly Detection page.

4.4 *Audit Trail*

All decisions regarding scaling, logins and actions are logged within the Audit Trail (Figure 8). There were 32 in total, with 25 successful actions and 7 failed actions. Each of these can be viewed further into, Figure 6 shows the three most recent items. The scale up decision made at 2:47AM had an error associated with it (ERR_CLUSTER_QUOTA_EXCEEDED) due to the cluster being unable to start another instance. The next action taken at 2:51AM was to leave the system as is. This demonstrates how useful the log can be for the administrator, they can see what decision has been taken by the engine and the reason for why it wasn't applied.

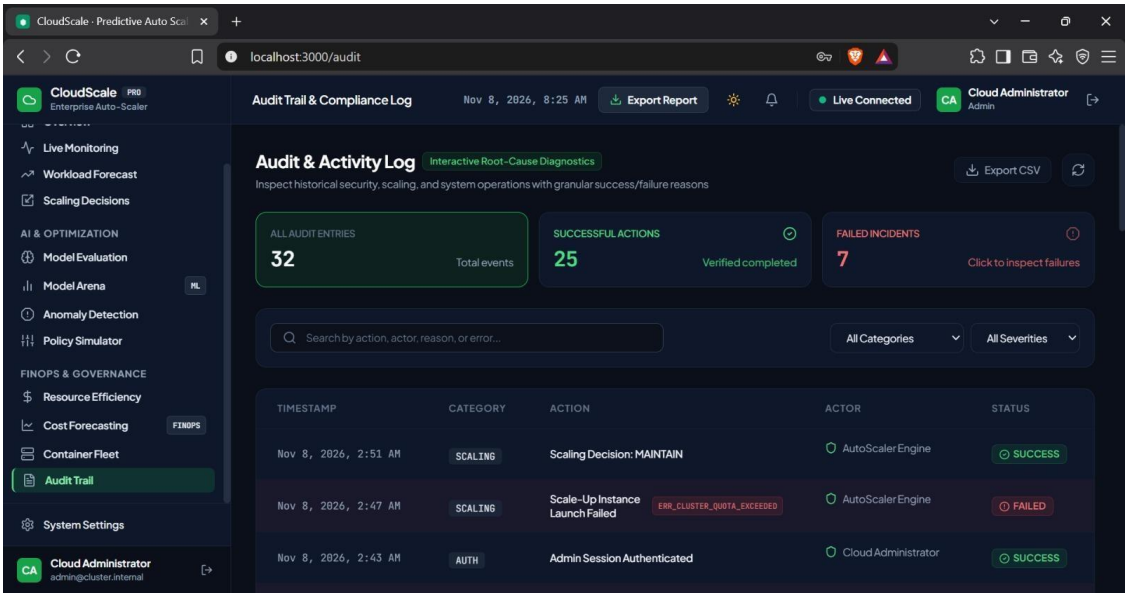


Fig. 8. Audit Trail page.

Table 6. Recent audit entries.

Time	Category	Event	Status
Nov 8, 2:51 AM	Scaling	Scaling decision: MAINTAIN	Success
Nov 8, 2:47 AM	Scaling	Scale-up instance launch failed (quota exceeded)	Failed
Nov 8, 2:43 AM	Auth	Admin session authenticated	Success

5. CONCLUSION

This paper presented CloudScale, a system which uses the XGBoost algorithm to predict workloads 15 minutes in advance and makes decisions about scaling. XGBoost produced the best balance between accuracy and speed when compared to the four different models. An anomaly module and audit trail help visualise what is happening within the system and the dashboard provides a single point of monitoring, prediction and decision making.

The results are collected from a low load system; we have not measured any cost savings and/or change in response times. One scale up event was also rejected in the audit logs due to a cluster quota, meaning that the engine is dependent on quotas within the Cloud account. We want to evaluate CloudScale with high and burst loads, measure cost saving over a static number of servers, and link it with AWS CloudWatch and EC2 Auto Scaling.

REFERENCES

1. Rossi, A. Visentin, D. Carraro, S. Prestwich, and K. N. Brown, “Forecasting workload in cloud computing: Towards uncertainty-aware predictions and transfer learning,” *Cluster Computing*, vol. 28, no. 4, p. 258, 2025.
2. S. N. Pothu and S. Kailasam, “Effective priority-based resource allocation for proactive

auto-scaling framework in workload prediction using hybrid tree-enhanced vector machine model," *Discover Sustainability*, vol. 5, no. 1, p. 391, 2024.

3. W. Sus and P. Nawrocki, "Signature-based adaptive cloud resource usage prediction using machine learning and anomaly detection," *Journal of Grid Computing*, vol. 22, no. 2, p. 46, 2024.
4. S. Xue et al., "A meta reinforcement learning approach for predictive autoscaling in the cloud," in *Proc. 28th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, 2022, pp. 4290–4299.
5. L. Zanussi, D. Tessera, L. Massari, B. Bermejo, C. Juiz, and M. Calzarossa, "Load-aware predictive auto-scaling framework for cloud environments," *Cluster Computing*, vol. 29, no. 3, p. 197, 2026.
6. M. Armbrust et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
7. P. Mell and T. Grance, "The NIST definition of cloud computing," NIST Special Publication 800-145, 2011.
8. T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, pp. 559–592, 2014.
9. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch, "AutoScale: Dynamic, robust capacity management for multi-tier data centers," *ACM Transactions on Computer Systems*, vol. 30, no. 4, Art. 14, 2012.
10. Z. Gong, X. Gu, and J. Wilkes, "PRESS: PRedictive Elastic ReSource Scaling for cloud systems," in *Proc. Int. Conf. on Network and Service Management (CNSM)*, 2010, pp. 9–16.
11. Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: Elastic resource scaling for multi-tenant cloud systems," in *Proc. ACM Symposium on Cloud Computing (SoCC)*, 2011.
12. N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *Proc. IEEE Int. Conf. on Cloud Computing (CLOUD)*, 2011, pp. 500–507.
13. R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications' QoS," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2015.
14. E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini,

"Resource Central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, 2017, pp. 153–167.

15. T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.
16. L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
17. S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
18. H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society: Series B*, vol. 67, no. 2, pp. 301–320, 2005.
19. V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, Art. 15, 2009.
20. Bergmeir and J. M. Benítez, "On the use of cross-validation for time series predictor evaluation," *Information Sciences*, vol. 191, pp. 192–213, 2012.