

Optimizing Container Resource Allocation by Right Sizing using Historical Timeseries Metrics in ARIMA Model

MSc Research Project
Research in Computing

Libin Tom Kuriyakose Kadavil

Student ID: 21204420

School of Computing
National College of Ireland

Supervisor: Dr. Diego Lugones

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Libin Tom Kuriyakose Kadavil
Student ID:	21204420
Programme:	Research in Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr. Diego Lugones
Submission Due Date:	14/08/2023
Project Title:	Optimizing Container Resource Allocation by Right Sizing using Historical Timeseries Metrics in ARIMA Model
Word Count:	5791
Page Count:	21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	18th September 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Optimizing Container Resource Allocation by Right Sizing using Historical Timeseries Metrics in ARIMA Model

Libin Tom Kuriyakose Kadavil
21204420

Abstract

Containers have revolutionized the deployment and management of applications in cloud computing environments, offering significant advantages such as lightweight and portable packaging, faster startup times, and reduced resource overhead. While containers excel in resource efficiency, the lack of resource limitations can lead to overconsumption or underutilization, especially in resource-constrained edge cloud environments. In edge cloud environments, which are located closer to end-users and devices, computing resources are inherently limited due to factors like physical space, power availability, and constrained network bandwidth. Optimizing resource usage in edge cloud environments is crucial to ensure smooth operation, meet quality of service requirements, and maximize performance. Right sizing containers plays a vital role in achieving resource optimization by allocating the appropriate amount of compute resources based on application requirements. Right sizing offers advantages such as cost optimization, improved performance, scalability, stability, reliability, and simplified resource management. While container orchestrators like Kubernetes provide features like Vertical Pod Autoscaler (VPA) for resource allocation optimization, relying on historical resource usage metrics for right sizing containers offers several advantages over real-time metrics. Historical metrics provide a more comprehensive and detailed view of resource utilization patterns, enabling accurate recommendations for resource requests and limits. They also allow for the identification of trends and seasonal variations, facilitating informed decision making for resource allocation. This research aims to explore approaches for right sizing container resource allocation by analyzing real historical resource usage metrics of a private company. By implementing Time Series forecasting model ARIMA (Auto Regressive Integrated Moving Average) to predicted the future CPU requirements. Findings show a potential cost reduction of up to 60% over a year compared to traditional fixed resource limits. While acknowledging the rough nature of the estimation, the study underscores the potential benefits of the approach.

Keywords

Container Orchestration, Resource Management, Time Series Analysis, Predictive Modeling, Auto Regressive Integrated Moving Average (ARIMA), Kubernetes.

Contents

1	Introduction	1
2	Related Work	2
2.1	Right-sized containers ideas and advantages	2
2.2	Resource Optimization with RUBAS Virtual Pod Autoscaler	2
2.3	Horizontal Autoscaling Approach for Optimised Resource Utilization . .	3
2.4	Fairness in Scheduling Techniques to Optimisation Resource Utilization .	3
2.5	Allocation of Virtual Machines in Data Centers using ARIMA Forecasted Scheduling	4
2.6	Resource Optimization in Edge Cloud Environments	4
2.7	Literature Gap and Research Directions	4
3	Design Specification	5
3.1	Tools and Libraries	5
4	Methodology	7
4.1	Objective	7
4.2	Research Questions	7
4.3	Data Collection	7
4.4	Workload Analysis	7
4.4.1	Time Series of CPU Utilization	8
4.4.2	Aggregated and Grouped Analysis	8
4.4.3	Seasonal Decomposition	8
4.4.4	Stationary Checking	9
5	Implementation	10
5.1	Data Transformation	10
5.2	Auto Regressive Integrated Moving Average (ARIMA) Modeling	11
5.3	Container Deployment Python Program using Kubernetes	12
5.3.1	Data Acquisition and Preprocessing	13
5.3.2	AutoRegressive Integrated Moving Average (ARIMA) Modeling .	13
5.3.3	Resource Limit Prediction	13
5.3.4	Dynamic Manifesto and YAML Generation	13
5.3.5	Deploy Container to the Kubernetes (minikube) Cluster	13
5.4	Container Deployment in Kubernetes (Minikube) Cluster	14
5.4.1	Kubernetes Cluster	14
5.4.2	Program Execution	14
5.4.3	Deployment Output	15
6	Evaluation	16
6.1	Experiment / Case Study 1 - Data Collection and Analysis	16
6.2	Experiment / Case Study 1 - ARIMA Modeling and Dynamic Resource Allocation	17
6.3	Experiment / Case Study 3 - Cost Comparison	18
6.4	Discussion	18
7	Conclusion and Future Work	20

1 Introduction

In today's dynamic and rapidly evolving world of cloud computing and containerized applications, efficient resource management is paramount to achieving optimal performance and resource utilization. According to the research utilizing data disclosed by Google, which offers insights into cluster utilization, approximately 70% of the allocation is designated for high-priority tasks, while merely 25% to 35% of the allocation matches the actual utilization Reiss et al. (2012). In another article by Amazon Web Services, states that right-sizing can cut cost about more than 36% ¹. This observation distinctly highlights the significant issue of under-utilization as a primary cost-related concern. As the demand for scalable and responsive systems continues to grow, accurate prediction and allocation of computational resources become critical factors in ensuring seamless operations. This research endeavors to address this challenge by proposing a methodology that combines the time series analysis and predictive modeling for informed resource allocation decisions within containerized environments.

Container orchestration platforms, such as Kubernetes, have gained immense popularity for their ability to manage the deployment, scaling, and management of containerized applications. Effective management of these resources, particularly CPU utilization, is essential to ensure that applications run efficiently without encountering performance bottlenecks or wastage of resources. Traditional approaches of resource allocation often rely on static limits, which may lead to either under-utilization or over-subscription of resources. The unconstrained nature of containers, however, might result in inefficient resource consumption in edge cloud settings with limited resources, which could have an impact on performance and overall effectiveness.

In this paper, experimented a comprehensive approach that leverages historical time series data of pod CPU utilization metrics to inform resource allocation decisions. Employ the Auto Regressive Integrated Moving Average (ARIMA) model, a powerful technique in time series analysis to forecast future CPU utilization. By extrapolating from historical data patterns, the ARIMA model provides valuable insights into future resource requirements, enabling more accurate and dynamic resource allocation strategies.

This paper is organized as follows: Section 2 provides an overview of related work in the domain of resource management containerized applications. Section 3, is the design of over all specification of the research and detailed list of tools and libraries used in the research. Section 4, presents the dataset used for the analysis and outline the steps for data pre-processing and elaborates on the methodology of utilizing ARIMA for predictive modeling. Then demonstrated the application of proposed approach in Section 5 by integrating the ARIMA model's predictions into the deployment of containers through dynamically deploying the manifesto using the python program into Kubernetes test environment. Finally, Section 6 discusses the experimental results and implications while Section 7 concludes the paper by summarizing the contributions and outlining potential avenues for future research.

¹AWS Cloud Enterprise Strategy Blog: <https://aws.amazon.com/blogs/enterprise-strategy/rightsizing-infrastructure-can-cut-costs-36/>

2 Related Work

Software application deployment and management have undergone a revolution after the introduction of containers in cloud computing environments Baarzi and Kesidis (2021). Containers include benefits including portable and lightweight packing, quicker start-up times, and less resource use. Right-sizing containers, however, becomes essential for effective resource allocation in resource-restricted edge cloud scenarios, where computing resources are constrained owing to physical space, power accessibility, and constrained network bandwidth. This review examines several methods resource optimization.

2.1 Right-sized containers ideas and advantages

The relevance of right-sizing containers in edge cloud systems is explored. It describes how determining the optimum number of compute resources to allocate based on application needs is known as "right-sizing." The benefits of appropriate sizing are discussed in this section, including cost reduction, enhanced performance, scalability, stability, and dependability, as well as easier resource management Li et al. (2022). Resource Allocation with Container Orchestrators: In this part, Researchers discuss how container orchestrators like Kubernetes could optimize resource allocation. It talks about how Kubernetes' Vertical Pod Autoscaler (VPA) functionality automates resource changes. Although VPA employs real-time measurements, the section stresses the drawbacks of depending just on real-time data in edge cloud systems with limited resources.

2.2 Resource Optimization with RUBAS Virtual Pod Autoscaler

In a study titled "Exploring Potential for Non-Disruptive Vertical Auto Scaling and Resource Estimation in Kubernetes," the authors introduced a system named RUBAS (Resource Utilization Based Auto Scaling System). RUBAS utilizes Kubernetes' built-in Metrics Server to gather resource utilization information for each container within the cluster. The system deploys pods upon their arrival and initiates the collection of CPU and memory metrics at one-second intervals. New estimations are generated every minute. RUBAS assesses the resource utilization of individual pods against their allocated limits. If the utilization aligns with the allocation, the pods continue running. However, discrepancies trigger two instructions to Docker: one to checkpoint the container and another to create an image with the current state and data. The launch specification is then formed using the checkpoint and image Rattihalli, Govindaraju, Lu and Tiwari (2019).

The estimation strategy employed in this approach involves calculating the median of resource utilization with a buffer, where the required resource is determined as the Median of Observations with buffer. This calculation is based on the metrics collected over 60 seconds with the buffer representing the absolute deviation of absorbed values. While this approach may suit applications that can be vertically scaled at varying intervals, it may not be ideal for applications with fixed concurrency of worker specifications. Configuration changes based on the estimation could adversely affect overall application operations. Additionally, determining application worker size from resource estimation proves challenging. The approach proposed in this research employs historical data to estimate the maximum utilization placement over a defined time period, with the value

remaining constant until significant deviations arise within the application.

2.3 Horizontal Autoscaling Approach for Optimised Resource Utilization

A study centered around elastic cloud resource management strategies within the Kubernetes framework has introduced an automatic horizontal scaling system architecture. This architecture is built upon the Monitor-Analyze-Plan-Execute (MAPE) loop to address load balancing challenges that arise when removing redundant pods in capacity reduction scenarios. This is due to the limitations of a horizontal pod auto scaler's single monitoring metric, which may not accurately gauge workload demands. The proposed approach also integrates a resource deletion strategy with the horizontal scaling system Yunyun et al. (2022).

The strategy put forth in this research aims to enhance Horizontal Scaling atop Kubernetes' native solution. It introduces a design for a passive scaling strategy that relies on weighted metric thresholds. This design improves upon the core Horizontal Pod Autoscaler (HPA) by monitoring additional resource metrics, in contrast to the basic HPA's single-metric approach, and applying weight to the analysis output. Unlike the present research, this paper primarily concentrates on the accurate resource sizing for a specific category of applications within Kubernetes, prior to their management by a horizontal autoscaler.

2.4 Fairness in Scheduling Techniques to Optimisation Resource Utilization

In an endeavor to compute resource constraints while ensuring equity within Kubernetes, a solution was introduced to address the equitable allocation of pods. This approach specifically concentrates on maintaining the quality of service for pods within a multi-resource environment, accommodating diverse pod submissions with varying CPU and memory requirements. The challenge addressed pertains to the Kubernetes scheduler's behavior of prioritizing pod requests aligned with available resources rather than considering the resource limits, which could lead to resource constraints if all pod utilizations surge. In such cases, the operating system kernel disregards the set limits and allocates only the minimum requested resources Hamzeh et al. (2019).

This proposal aims to mitigate the resource allocation complexities within Kubernetes by ensuring fairness among all pods, particularly when each pod demands maximum resources. Upon scheduling pods onto specific nodes, the resource limits are advocated to be treated uniformly for all pods within that node. However, this study's focus is in accurately estimating the appropriate limits for pods that cannot be vertically auto-scaled, whereas fairness could be an additional consideration for future enhancements in managing application concurrency.

2.5 Allocation of Virtual Machines in Data Centers using AR-IMA Forecasted Scheduling

This literature review investigates the utilization of forecasting models for the purpose of selecting appropriate instruments to predict the migration of Data Center (DC) virtual machines (VMs). The study involves a comparative analysis of various forecasting methods, assessing their effectiveness in predicting VM migration within the context of Google cluster environments. The experimental results of these forecasting methods are evaluated using real-world data. The outcomes demonstrate the superior performance of the prediction and scheduling of DC VM resources leading to a reduction in overloaded and under-loaded VM situations. Consequently, the efficiency of DC IT-infrastructure resource allocation is enhanced. The article establishes fundamental prerequisites for models and methods dedicated to predicting time series patterns in DC workloads. Among these, the chosen primary model is the Auto Regressive Integrated Moving Average (AR-IMA) model. Effective parameter intervals for the model are determined through methodologies such as Auto-Correlation Function (ACF), Partial Auto-Correlation Function (PACF), and derivative function analyses Dmytro et al. (2017).

However, in contrast to the aforementioned research on forecasting the VM allocation in the large data-centers, this research will be focusing on the right-sizing the container's resource allocation thus to achieve more optimised resource utilization in the area of containerized application, where the resource is very crucial and bottleneck like in Edge Cloud environments.

2.6 Resource Optimization in Edge Cloud Environments

The following section examines the unique challenges that develop in edge cloud environments when resource limitations require efficient resource allocation strategies. It focuses on the impacts of both excessive and inefficient resource consumption in these circumstances and highlights the need of container sizing to maximize resource use. Among the key problems with edge cloud configurations is the potential for excessive resource use. If efficient resource management approaches are not implemented, containers may consume more resources than necessary, which would lead to waste and increased running costs Liu et al. (2020).

Under-using resources has negative effects as well. If containers are given access to more resources than they need to complete their tasks, important processing power is wasted. In addition to wasting resources, under-utilization restricts the number of containers that could be hosted on a given infrastructure Rattihalli, Govindaraju and Tiwari (2019). This research overcome that challenge by forecasting the future requirement by analysing the historical usage data.

2.7 Literature Gap and Research Directions

In resource-constrained edge cloud scenarios, effective container sizing becomes crucial. Right-sizing containers entails determining optimal compute resource allocation based on application needs, yielding benefits like cost reduction and enhanced performance. While container orchestrators like Kubernetes optimize resource allocation, relying solely

on real-time data may not suit resource-constrained edge systems. RUBAS introduces a system that estimates container resource needs using Kubernetes' Metrics Server, but it faces challenges with applications having fixed concurrency worker specifications. Another study proposes an automatic horizontal scaling architecture addressing load balancing during resource reduction. Unlike these, the research primarily focuses on accurate resource limit estimation for non-vertically autoscaled pods. In edge cloud settings, proper container sizing strategies are essential, mitigating the risks of both excessive and inefficient resource usage. The present research aims to address these challenges by forecasting future requirements based on historical usage data.

3 Design Specification

The study focuses on analyzing pod CPU utilization using a two-week historical dataset of CPU usage metrics provided by a private company. The research employs a time series forecasting approach, specifically utilizing the ARIMA (AutoRegressive Integrated Moving Average) model to predict future CPU utilization trends. Time Series data often exhibit temporal dependencies, where past values influence future values and ARIMA models can capture the dependencies through autoregressive and moving average components. Also this model can handle wide range of time series patterns, including trends, seasonality and cyclic behavior and hence this model is very much suitable for time series analysis. Fig 1 is the over all approach of the research conducted.

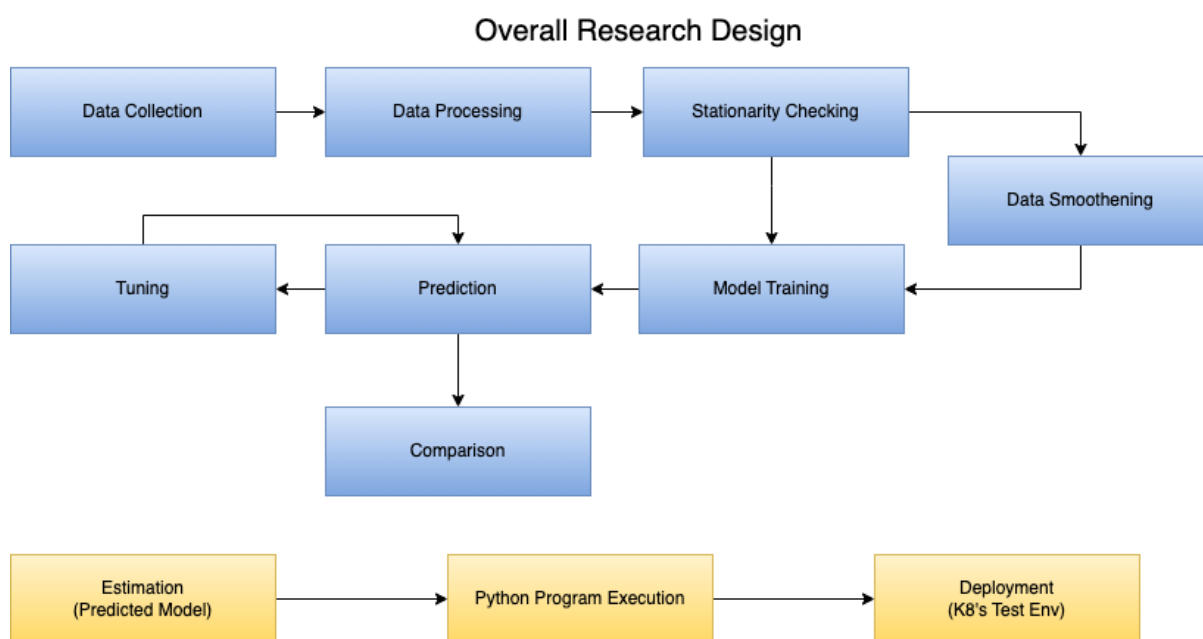


Figure 1: Research Design

3.1 Tools and Libraries

A range of tools and libraries used to conduct this proposal which helped from the data processing to deployment of the predicted model containers in the test environment.

Python: A versatile programming language used for data analysis, scripting, and application development.

Minikube: A tool for setting up and managing a local Kubernetes cluster for development and testing purposes.

Docker: A platform for creating, deploying, and managing containerized applications.

Kubernetes: An open-source container orchestration platform that automates the deployment, scaling, and management of applications.

Visual Studio: An integrated development environment (IDE) for coding, debugging, and building applications.

Google Colab: A cloud-based platform for writing and executing Python code collaboratively, particularly suited for machine learning and forecasting tasks.

Pandas Library: A Python library for data manipulation and analysis, providing data structures and tools for efficient data handling.

JSON Library: A Python library for working with JSON data, used for reading and writing JSON files.

NumPy Library: A fundamental package for scientific computing with Python, providing support for arrays and mathematical functions.

Matplotlib Pyplot Library: A Python library for creating static, interactive, and animated visualizations in a wide range of formats.

Resample Library: A Python library for resampling time series data, providing tools for adjusting time intervals.

Statsmodels TSA Seasonal Decomposition: A module within the Statsmodels library that enables seasonal decomposition of time series data into trend, seasonality, and residuals components.

Statsmodels TSA ADFuller: A module within the Statsmodels library that implements the Augmented Dickey-Fuller (ADF) test for assessing the stationarity of time series data.

Statsmodels TSA ARIMA Model (ARIMA): A module within the Statsmodels library for building and analyzing AutoRegressive Integrated Moving Average (ARIMA) models.

sklearn Metrics Mean Squared Error: A module within the scikit-learn library for calculating the mean squared error, a common metric for evaluating the accuracy of regression models.

itertools: A Python module providing functions for creating iterators for efficient looping and data manipulation.

PyYAML: A Python library for working with YAML (Yet Another Markup Language) files, often used for configuration files.

Kubernetes Python Library: Kubernetes api for interacting with a Kubernetes cluster using client tools for the automated deployment of the manifesto to the cluster from the developed program.

4 Methodology

4.1 Objective

The main objective of this research is to optimize resource allocation for containers in a cloud computing environment, using real-world pod CPU utilization metrics from a private company's production infrastructure. The research aims to determine the optimal size for pods in terms of CPU resources, considering the unique requirements and workload patterns of medical-related applications. However, determining the right CPU size is an iterative process, and it requires a good understanding of your application's behavior and performance requirements. Regular monitoring, analysis, and adaptation are key to maintaining optimal resource utilization.

4.2 Research Questions

How to optimization of container resource allocation by leveraging historical metrics to accurately forecast future resource requirements?

4.3 Data Collection

Each application resource needs would vary from one another. However, understanding the base line of the requirement and a common approach can be achieved to optimize the resource in a general accepted approach. For the experiment of this research, the dataset containing pod's CPU utilization metrics is used. This real data has been provided by a private company from their one of the medical related production application infrastructure. This dataset is provided for the soul purpose of academic research and will be handled through the data ehtics principles for the anonymity and security of the data. The dataset includes timestamped CPU utilization at 10-minute intervals for two-week period. In the Fig 2, is the current utilization graph of the pod from the private company.

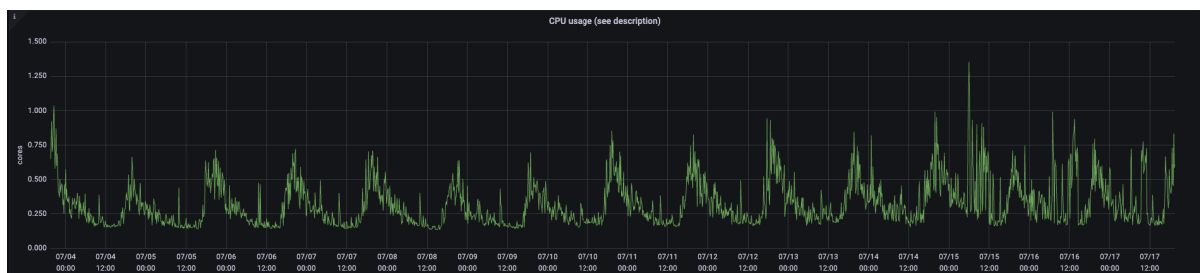


Figure 2: Pod Current Usage Metrics

4.4 Workload Analysis

Thematic analysis is used to examine the data that have been gathered. Finding patterns, themes, and categories within the data using thematic analysis enables a thorough comprehension of the study subject. The iterative analysis is used to make sure that any new topics are properly investigated. Data exploration and visualization are crucial steps in understanding the dataset and making informed decisions. Using the python's Pandas

library, dataset's data frames have been created for further analysis having timestamp and utilization as features. All the pods are allocated with 2 Core CPU for the current running workload. The Pod peak utilization is at 1.35, average 0.32 and min 0.13. Matplotlib library has been used to visualise the data for the analysis.

4.4.1 Time Series of CPU Utilization

In this line chart for time series of CPU Utilization in fig 3 plotted the Utilization and the peak value for the period of two weeks. As stated the vales in the section 4.4, a single peak over the period of the metrics has been recorded.

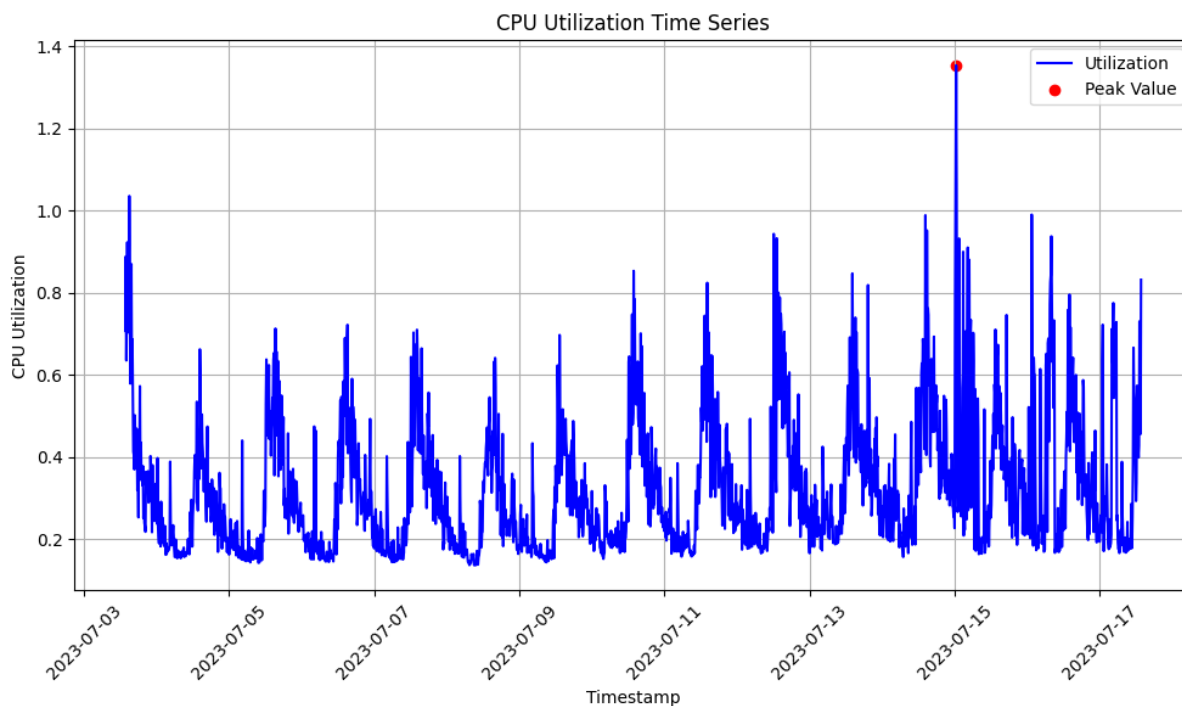


Figure 3: Time Series of CPU Utilization

4.4.2 Aggregated and Grouped Analysis

Aggregating and grouping the time series data allows to analyze patterns at different time intervals. In the Fig 4 plotted the aggregated daily max utilization starting at 1.03 and spiking to 1.34 and the drop to 0.83.

4.4.3 Seasonal Decomposition

Seasonal decomposition allows to separate the time series data into different components: trend, seasonality, and residuals Gweon and McLeod (2013). This helps in understanding underlying patterns and trends. The period parameter has been set to 24×6 as the data is at 10 minute intervals for a daily seasonality, this specifies the number of data points in one complete season. Also the daily aggregated seasonal decomposition has been plotted. In Fig 5 plotted the Observed, Trend, Seasonal and Residual using the Statsmodels

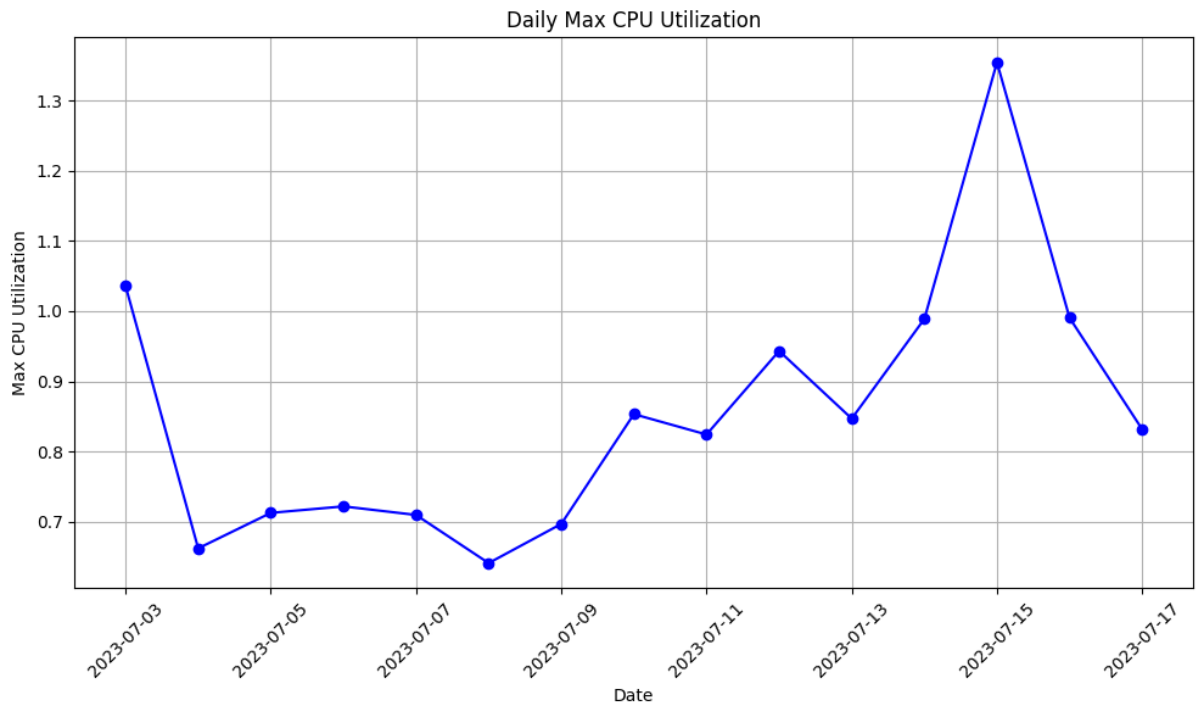


Figure 4: Daily Aggregated Max Utilization

seasonal_decompose library for both the actual and aggregated data. In this plotting, it is evident that this time series has seasonality and upward trend.

4.4.4 Stationary Checking

The primary assumptions time series forecasting is that the data should stationary to achieve the forecasting. By removing the trend and seasonality, the time series would be made stationary. Removing is the process of separating the trend and seasonality component leaving the errors(Residual).

Rolling Statistics are commonly used in time series analysis and other fields where data points are collected over time. It is particularly useful for smoothing out fluctuations and identifying trends or patterns in noisy or volatile data. In the Fig 6 plotted the Rolling Statistics and it is clear that the data has fluctuations and further smoothing has to be carried out for better forecasting.

However, for further more comparison in the statistical point, another method, Augmented Dickey-Fuller (adfuller) has been performed on the same data for later comparison after running the smoothing techniques. Adfuller is a statistical test used to determine whether a given time series is stationary or not. The ADF test works by comparing the autoregressive (AR) model of the time series with a model that includes a unit root. The null hypothesis of the ADF test is that the time series has a unit root and is non-stationary. The alternative hypothesis is that the time series is stationary. It produces a test statistic and a p-value. If the p-value is below a chosen significance level (such as 0.05), it would reject the null hypothesis. Adfuller is a component of statsmodels tsa stattools.

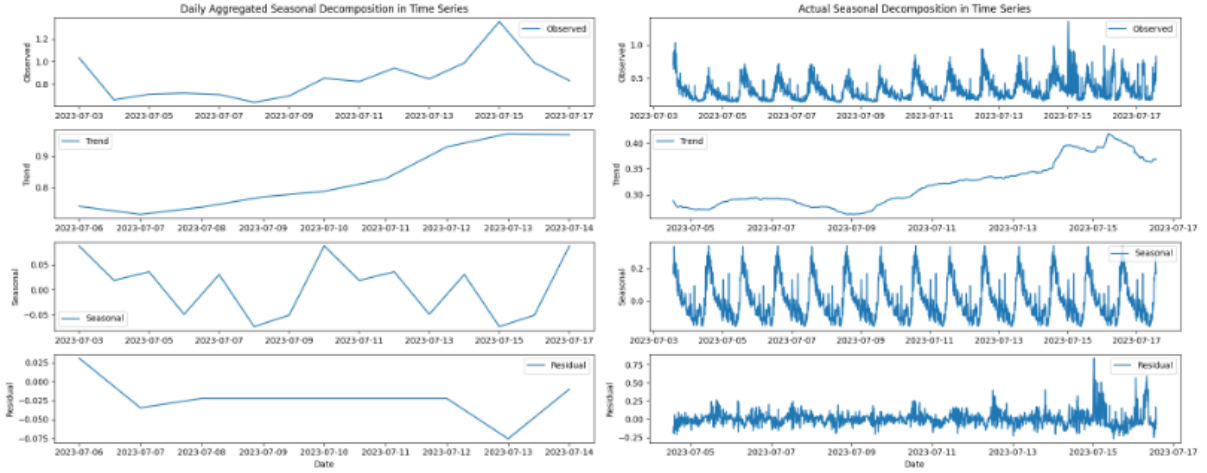


Figure 5: Seasonal Decomposition Actual and Aggregated

The p value of adfuller test for the daily aggregated data is : 0.8455405909803948 which is above 5 % and the data is non-stationary as per the null hypothesis of the ADF test.

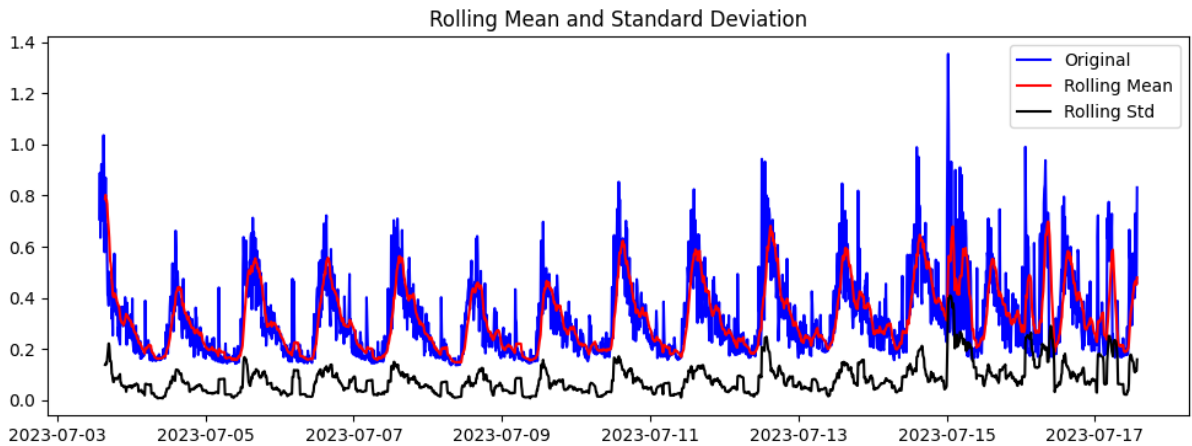


Figure 6: Rolling Mean and Standard Deviation

5 Implementation

The implementation of the proposed methodology involves several key steps, ranging from data acquisition and pre-processing as described in section 4 to the integration of predictive modeling into resource allocation. This section provides a comprehensive overview of the technical details involved in realizing our approach.

5.1 Data Transformation

In time series analysis, data transformation using logarithm and differencing is a common practice to stabilize and make the data more amenable for analysis. The logarithm transformation, often applied to data with exponential growth patterns or varying scales,

compresses large values while expanding smaller ones, reducing the impact of extreme values and making the data distribution more symmetrical. This can aid in achieving homoscedasticity and improving the fit of models.

Differencing involves computing the differences between consecutive observations, aiming to remove trends or seasonality. First-order differencing eliminates linear trends, while seasonal differencing can mitigate seasonality effects. These transformations can help convert non-stationary time series data into stationary ones, which are more suitable for applying various statistical techniques and forecasting in the ARIMA model.

As discussed and plotted in the 4.4.4, it is evident that the dataset has seasonality and trend. Through log and differencing transformations, it will help further modeling to mitigate data irregularities, stabilize variance, and create more manageable data for modeling, ultimately enhancing the accuracy and reliability of forecasts and insights drawn from the time-dependent data. In the Fig 7 plotted the data after the transformation and adfuller resulted with p value of 0.

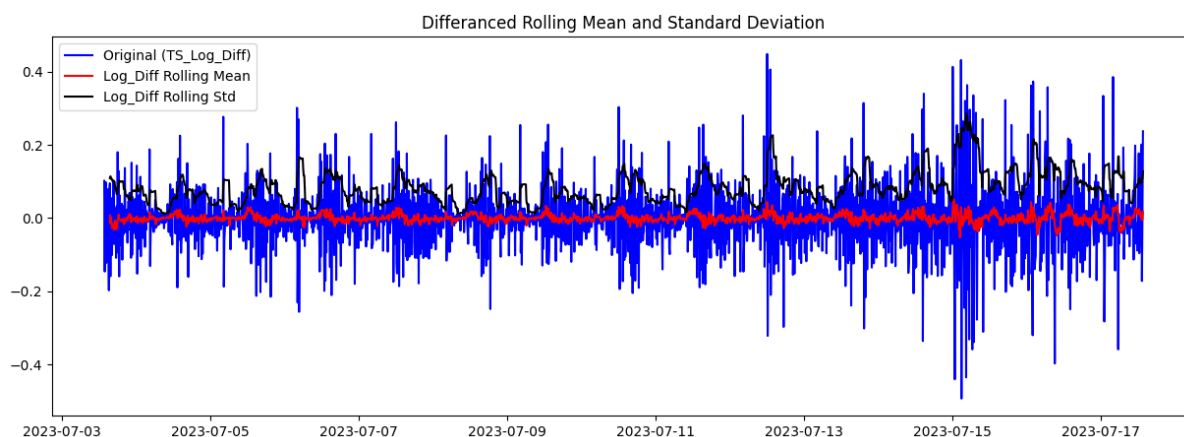


Figure 7: Transformed Data

5.2 Auto Regressive Integrated Moving Average (ARIMA) Modeling

As stated in the section 3, The ARIMA (Auto Regressive Integrated Moving Average) model building involves selecting appropriate parameters (p , d , q) for auto regressive, differencing, and moving average components, respectively. It aims to capture temporal patterns, stationarity, and seasonality in time series data, facilitating accurate forecasting and analysis.

There are many methods and techniques to selecting the appropriate (p , d , q) parameters. ACF (AutoCorrelation Function) measures correlation between a time series and its lagged values, helping identify potential patterns Dmytro et al. (2017). PACF (Partial AutoCorrelation Function) reveals direct relationships between values at different lags,

aiding in determining optimal lags for forecasting models like ARIMA. However, in this research, a manual identification of the best parameters has been detected by running the prediction in a for loop by leveraging the `itertools` library to parse the range from 0 to 8 and compared the root mean squared error RSME (from the `sklearn.metrics` library) with the predicted results of 120 possible combinations with a lowest RSME of 0.1790705366 with the parameters (0,0,6).

An ARIMA model was constructed to forecast future CPU utilization, extending the time series data's length. The average prediction yielded 0.8477 CPU utilization, while the root mean square error (RMSE) measured 0.1629. This model demonstrates its capability to approximate forthcoming trends, with the RMSE illustrating the closeness of predictions to actual values. These results underline the model's potential in aiding decision-making, resource allocation, and proactive system management by providing insightful estimations of CPU utilization patterns. Plotted the actual and the predicted values in Fig 8.

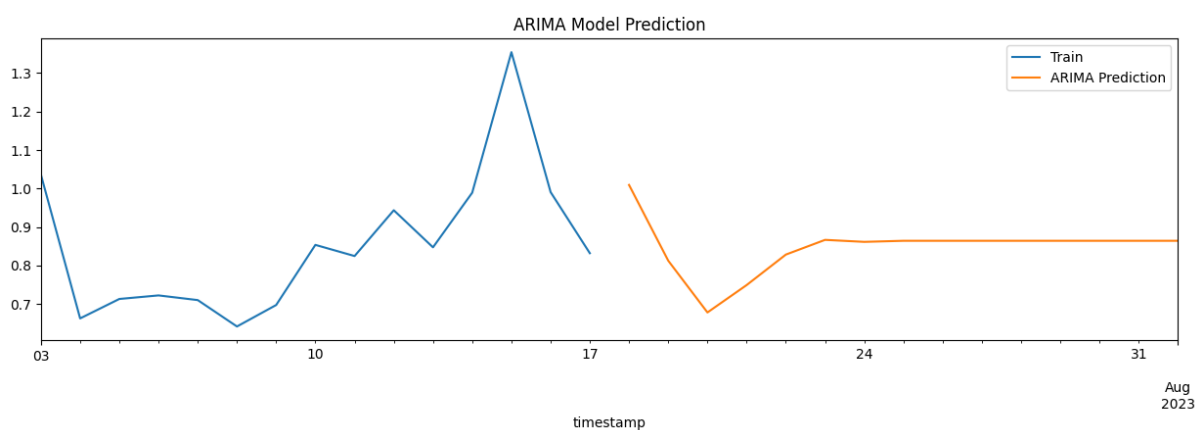


Figure 8: Actual and ARIMA Model Prediction

5.3 Container Deployment Python Program using Kubernetes

Utilizing the trained ARIMA model, we predict the future CPU utilization values based on the observed historical patterns. The predicted CPU utilization is then utilized as a dynamic resource limit for containerized applications. To ensure realistic limits, observed mean CPU utilization has been accounted, preventing predicted values from falling below the observed peak.

A python program has been developed to deploy the pods. To operationalize the predicted CPU utilization, this program dynamically generates a deployment manifesto configuration for Kubernetes deployments. The manifesto embeds the predicted CPU utilization value as the resource limit for the deployment of the pods. The program also utilises the python's Kubernetes client library and deploys the pod in the kubernetes cluster and also generates a manifesto YAML for the configuration backup.

In this research, for the purpose of the demonstration, used `minikube` a kubernetes sandbox application to create the cluster in the local environment. This seamless integration

between predictive modeling and container orchestration allows for agile and responsive resource allocation. The subsequent sections delve into the pseudocode detailing each step of the implementation process. Through this approach, we enable data-driven and adaptable resource management strategies, enhancing the efficiency and performance of containerized applications.

5.3.1 Data Acquisition and Preprocessing

- Load JSON dataset from file
- Extract CPU utilization values from dataset
- Convert the timestamp to pandas date and time with ms unit
- Convert the dataset to dataframes
- Set timestamp as the index
- Resample the daily max utilization

5.3.2 AutoRegressive Integrated Moving Average (ARIMA) Modeling

- Define ARIMA parameters: order (p, d, q)
- Train ARIMA model with the resampled CPU utilization data

5.3.3 Resource Limit Prediction

- Predict future CPU utilization using ARIMA model
- Ensure predicted value = mean(observed CPU values)

5.3.4 Dynamic Manifesto and YAML Generation

- Define YAML template with predicted_cpu placeholder
- Replace predicted_cpu with predicted value
- Record other deployment details through keyword argument
- Generate pod deployment YAML file
- Print("Pod deployment YAML with predicted CPU as resource limit generated.")

5.3.5 Deploy Container to the Kubernetes (minikube) Cluster

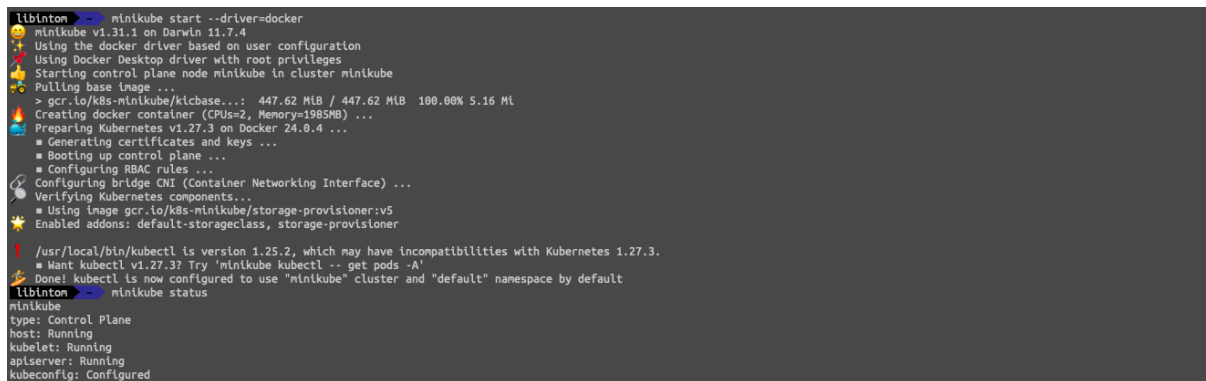
- Import the kubernetes config and client
- Deploy pod using the dynamically generated manifesto through Kubernetes client api
- Print api response

5.4 Container Deployment in Kubernetes (Minikube) Cluster

5.4.1 Kubernetes Cluster

A Minikube Kubernetes sandbox environment has been created as discussed in section 5.3. Minikube is a tool that sets up a local single-node Kubernetes cluster in the local environment, which allows to develop and test applications in a Kubernetes-like environment. This is particularly useful for local development and testing purposes. In this setup, Minikube utilizes the Docker daemon underlying it to manage and orchestrate the Kubernetes control plane and worker nodes.

Fig 9 and Fig 10 are the cluster details of the Minikube and Docker environment.



```
libinton ~ - minikube start --driver=docker
minikube v1.31.1 on Darwin 11.7.4
Using the docker driver based on user configuration
Using Docker Desktop driver with root privileges
Starting control plane node minikube in cluster minikube
Pulling base image ...
> gcr.io/k8s-minikube/kicbase...: 447.62 MiB / 447.62 MiB 100.00% 5.16 Mi
Creating docker container (CPUs=2, Memory=1985MB) ...
Preparing Kubernetes v1.27.3 on Docker 24.0.4 ...
  # Generating certificates and keys ...
  # Booting up control plane ...
  # Configuring RBAC rules ...
  # Configuring bridge CNI (Container Networking Interface) ...
  # Verifying Kubernetes components...
  # Using image gcr.io/k8s-minikube/storage-provisioner:v5
Enabled addons: default-storageclass, storage-provisioner

! /usr/local/bin/kubectl is version 1.25.2, which may have incompatibilities with Kubernetes 1.27.3.
  # Want kubectl v1.27.3? Try 'minikube kubectl -- get pods -A'
Done! kubectl is now configured to use 'minikube' cluster and 'default' namespace by default
libinton ~ - minikube status
minikube
type: Control Plane
host: Running
kubelnet: Running
apiserver: Running
kubeconfig: Configured
```

Figure 9: Minikube Environment



```
libinton ~ - sudo docker info
Password:
Client:
Context: default
Debug Mode: false
Plugins:
buildx: Docker Buildx (Docker Inc., v0.9.1)
compose: Docker Compose (Docker Inc., v2.12.2)
dev: Docker Dev Environments (Docker Inc., v0.0.3)
extension: Manages Docker extensions (Docker Inc., v0.2.13)
sbom: View the packaged-based Software Bill Of Materials (SBOM) for an image (Anchore Inc., 0.6.0)
scan: Docker Scan (Docker Inc., v0.21.0)
```

Figure 10: Docker Dev Environment

5.4.2 Program Execution

The Python program orchestrates the deployment of Kubernetes pods within the cluster, utilizing keyword arguments to input data into various functions within the program. These keyword arguments serve as parameters that guide the program's behavior and configuration. As the program executes, it generates a deployment manifest, which is a structured configuration file that defines how the pods and related resources should be deployed and managed in the Kubernetes cluster. This manifest includes details such as the container image, resource requirements, number of replicas, networking settings, and more. To initiate the program execution, the following command is used.

```
# command to execute the deployment with the predicted cpu limit.
$python3 predicted-pod-depolymnt.py --ts_data prometheus-dataset-1.json
--pod_name research-app-1 --image nginx:1.15.4 --cpu_request 500m
--memory_request 500Mi --memory_limit 1Gi --no_replicas 2 --port_num 80
```

Here, predicted-pod-depoyment.py represents the name of the Python program file, the -ts_data is timeseris utilization metrics file path, -pod_name is the desired name for the application pod, -image is the container image name, -cpu_request is the CPU minimum request, -memory_request is the memory minimum request, -memory_limit is the max memory limit, -no_replicas is the number of pod replicas, and -port_num is the pod networking port number flags along with their corresponding values provide the necessary input for the program's functions." Here the CPU limit will be the model prediction output provided as input for the dynamic manifesto generation.

This Figure 11 captures the response and output generated after executing the Python program. It showcases relevant information, such as the progress of pod deployment, successful completion, or any potential error messages. The response may include details about the dataset description before grouping, after grouping and pods creation status from the response of the Kubernetes client api which is higlighted in the figure. This visual representation offers insights into the program's interaction with the Kubernetes cluster and deployment process.

```

(reproject) libintom <~/Documents/ms_projects/sem3/research_project/k8-pods> python3 predicted-pod-deployment.py --ts_data prometheus-dataset-1.json --pod_name research
--no_replicas 2 --port_num 80
Imported the Dataset Succussfully!

Time Series Dataset Details Before Grouping:
count      2017.000000
mean       0.321518
std        0.161581
min        0.135765
25%        0.195673
50%        0.274132
75%        0.401373
max        1.354289
Name: utilization, dtype: float64
Time Series Dataset Details After Grouping:
count      15.000000
mean       0.854279
std        0.107944
min        0.641196
25%        0.711257
50%        0.831692
75%        0.966169
max        1.354289
Name: utilization, dtype: float64
Time Series Data Head 2
utilization
timestamp
2023-07-03    1.035846
2023-07-04    0.662199
The predicted CPU limit is : 0.8477036838542542 : 847m
Pod deployment YAML file generated as [pred_pod_deployment.yaml]
Pod created. Status='research-app-1'

```

Figure 11: Deployment Program Output

5.4.3 Deployment Output

In Figure 12, observed the result of executing the kubectl get pod command. This command provides a concise overview of the current state of the deployed pods within the Kubernetes cluster. The output showcases the successful deployment of pods, with the requested number of replicas being 2 in this specific scenario. The 'STATUS' column indicates that both replicas are in a 'Running' state, indicating that they are active and operational within the cluster. This visual representation confirms the accomplishment of the deployment goal, and the desired pod replicas are up and running as expected.

Figure 13 provides an insightful description of the deployed pods. This detailed description is accessible by executing the kubectl describe pod command, followed by the name of the specific pod. The figure's focus is on highlighting the pod's predicted CPU limit, which is an essential aspect of resource allocation and management in Kubernetes.

The highlighted section underscores the allocated CPU resources for the pod, helping to understand how the pod is configured in terms of computing power.

```
libintom ~~/Documents/ms_projects/sen3/research_project/k8-pods$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
research-app-1-779db85dd6-59vzs    1/1     Running   0           6m12s
research-app-1-779db85dd6-c8vsw    1/1     Running   0           6m12s
```

Figure 12: Successful Pod Deployment

```
libintom ~~/Documents/ms_projects/sen3/research_project/k8-pods$ kubectl describe pod research-app-1
Name:                               research-app-1-779db85dd6-59vzs
Namespace:                           default
Priority:                               0
Service Account:                       default
Node:                                  minikube/192.168.49.2
Start Time:                            Fri, 11 Aug 2023 23:03:35 +0100
Labels:                                 app=research-app-1
                                         pod-template-hash=779db85dd6
Annotations:                            <none>
Status:                                 Running
IP:                                     10.244.0.19
IPs:                                    IP: 10.244.0.19
Controlled By:                         ReplicaSet/research-app-1-779db85dd6
Containers:
  research-app-1-cont:
    Container ID:   docker://2251a82e08dd4c3a198e91f71c1f670a1c08e3ee6457d75bbd3175431ceeab19
    Image:          nginx:1.15.4
    Image ID:       docker-pullable://nginx@sha256:e8ab8d42e0c34c104ac60b43ba60b19af08e19a0e6d50396bdfd4cef0347ba83
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Fri, 11 Aug 2023 23:03:37 +0100
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:          847m
      memory:       1Gi
    Requests:
      cpu:          500m
      memory:       500Mi
    Environment:   <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-9r754 (ro)
```

Figure 13: Deployed Pod Description

Overall the Python program coordinates pod deployment by utilizing keyword arguments to configure its behavior. These arguments drive the creation of a deployment manifest, a structured configuration file detailing resource deployment within the cluster. Executed via command line, the program interfaces with the Kubernetes cluster to deploy pods with specified attributes. The successful execution is reflected in the cluster’s deployment, achieving the desired number of replicas. The program also interacts with Docker for containerization and orchestration. Moreover, detailed pod attributes, like predicted CPU limits, aid resource allocation optimization. Overall, this orchestrated interplay underscores Kubernetes development and testing the implementation of forced efficient exploration of Kubernetes capabilities in a contained setting.

6 Evaluation

6.1 Experiment / Case Study 1 - Data Collection and Analysis

The data collection process was successful in obtaining real-world pod CPU utilization metrics from a private company’s production infrastructure. The analysis conducted in section 4.4 provided valuable insights into the workload patterns and behavior of the medical-related applications. The time series analysis, aggregated analysis, and seasonal decomposition revealed underlying trends, seasonality, and fluctuations in the CPU utilization data. The identification of non-stationarity using the Augmented Dickey-Fuller test

highlighted the need for data transformation. Figure 14 is the adfuller result before and after the transformation. The two transformation methods used are Log and Difference which smoothen the metrics from 0.8455 to 0.0 P value. This evaluation demonstrated a thorough understanding of the collected data's characteristics, setting a strong foundation for subsequent steps.

```

15] from statsmodels.tsa.stattools import adfuller ##### Original Timeseries Data #####
##### Original Timeseries Data #####
51] ##### Original Timeseries Data #####
# Perform Augmented Dickey-Fuller test:
# ADF Test - nul hypothesis - non-stationary - i adfuller_result = adfuller(ts_sqrt_diff.dropna(),
adfuller_result = adfuller(ts_daily, autolag='AI print(f'ADF Statistic: {adfuller_result[0]}')
print(f'ADF Statistic: {adfuller_result[0]}') print(f'p-value: {adfuller_result[1]}')
print(f'p-value: {adfuller_result[1]}') for key, value in adfuller_result[4].items():
for key, value in adfuller_result[4].items(): print('Critical Values:')
print('Critical Values:') print(f' {key}, {value}')
print(f' {key}, {value}')

ADF Statistic: -0.7046531182361528 p-value: 0.0
p-value: 0.8455405909803948 Critical Values:
Critical Values: 1%, -3.433605925774539
1%, -4.137829282407408 Critical Values:
Critical Values: 5%, -2.862978297026843
5%, -3.1549724074074077 Critical Values:
Critical Values: 10%, -2.5675356871295394
10%, -2.7144769444444443

```

Figure 14: Adfuller Result Before and After Transformation

...

6.2 Experiment / Case Study 1 - ARIMA Modeling and Dynamic Resource Allocation

The implementation of the ARIMA modeling in section 5.2 showcased the capability to forecast future CPU utilization based on historical patterns. The selection of ARIMA parameters through a comprehensive search process yielded a promising model. The generated ARIMA model demonstrated its effectiveness by producing accurate predictions and maintaining a low root mean square error (RMSE), shown the results in the Figure 15 . These results affirm the suitability of the ARIMA model for predicting CPU utilization patterns, contributing to the optimization of resource allocation strategies.

The implementation of the dynamic resource allocation strategy using Kubernetes and the Python program was successful in orchestrating the deployment of pods based on the predicted CPU utilization. The program's ability to dynamically generate a deployment manifesto and utilize predicted resource limits showcased the integration between predictive modeling and container orchestration. The deployment to the Minikube Kubernetes cluster demonstrated the feasibility of the approach in a controlled environment. The successful deployment of pods with predicted CPU limits verified the potential for adaptive and data-driven resource allocation.

...

```

from sklearn.metrics import mean_squared_error

pred_error = np.sqrt(mean_squared_error(test,pred_model))
pred_error

0.1636538338189795

test.mean(), np.sqrt(test.var())

(0.39618020600714215, 0.030894911036565935)

```

Figure 15: Root Mean Square Error

6.3 Experiment / Case Study 3 - Cost Comparison

In order to gauge the tangible benefits of the proposed resource allocation strategy, conducted a cost comparison between the original configuration and the configuration utilizing the predicted CPU limits. In the original setup, we had 20 pods running at 2 cores each, resulting in significant underutilization of resources. On the other hand, our predictive modeling approach suggested a more efficient allocation, with an average predicted CPU limit of 0.8 cores per pod.

Considering a cost of 1 euro per core per day, calculated the annual expenses for both scenarios. In the original underutilized configuration, the total cost over the course of 356 days amounted to 14,600 euros. In contrast, the implementation of predicted CPU limits resulted in a substantially reduced cost of 5,840 euros for the same duration. This stark contrast highlights the potential for substantial savings. By employing predictive modeling and dynamic resource allocation, we achieved a remarkable 60% reduction in costs, as depicted in Chart 16. It's important to note that this cost reduction is a preliminary estimate and could be further optimized in real-world scenarios.

While this cost comparison provides a compelling snapshot of the potential benefits, it is accepted that additional factors need to be considered for a comprehensive evaluation. Factors such as application performance, response time, system throughput, and the potential impact of over-utilization should all be part of the equation when refining this approach for future implementations.

...

6.4 Discussion

The findings of the research indicate promising results in optimizing resource allocation based on predicted CPU limits. The utilization of real-world pod CPU utilization metrics provided a valuable dataset for analysis. Thematic analysis, time series exploration, and ARIMA modeling contributed to understanding the workload patterns, identifying trends, and generating accurate predictions for future CPU utilization. The integration of predictive modeling into container deployment demonstrated the feasibility of dynamically adjusting resource limits based on forecasted utilization. The cost comparison revealed substantial cost reductions, with the predictive approach achieving a 60% reduction in expenses compared to the underutilized original configuration. This finding underscores the potential of this methodology in achieving efficient resource utilization

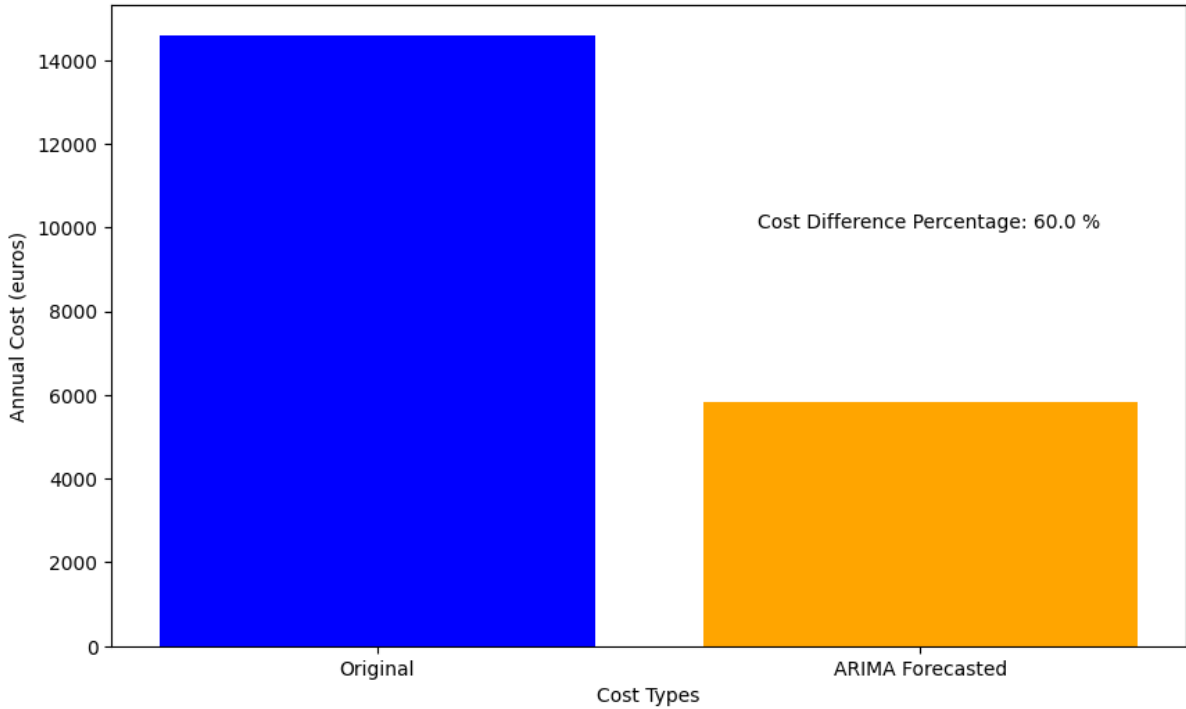


Figure 16: Comparison of Original vs. ARIMA Forecasted Annual Costs

and cost savings.

The detailed analysis of the methodology encompassed various stages, from data collection and preprocessing to ARIMA modeling and dynamic deployment. The transformation of the CPU utilization data through logarithm and differencing improved stationarity, facilitating accurate modeling. The ARIMA modeling process was successful in predicting future CPU utilization, which was further utilized as dynamic resource limits for container deployment. Furthermore, the integration of Kubernetes and the Python program provided a practical approach to dynamically allocate resources based on predictions. The successful deployment of pods with predicted CPU limits validated the viability of this approach in achieving adaptive resource allocation.

While the experimental design exhibited strong points, such as comprehensive data analysis and successful implementation, there are areas that warrant scrutiny and potential enhancements. One limitation of the research was the focus on a single utilization metric and a relatively short two-week timeframe. To enhance the generalizability of this findings, future research could involve forecasting multiple usage metrics (e.g., memory, network) over longer time periods. This would provide a more comprehensive view of resource allocation optimization. Additionally, while ARIMA modeling yielded accurate predictions, exploring alternative time series models like Seasonal ARIMA could provide a benchmark for comparison. Comparing the effectiveness of different models in predicting CPU utilization patterns would enhance the robustness of the findings.

In the realm of future improvements, several key enhancements are recommended. Firstly, to address the variability of application workloads comprehensively, integrating predictive

models across multiple usage metrics is proposed. This would entail developing predictive models for memory and network usage, allowing for a more holistic approach to resource allocation optimization. Additionally, extending the duration of data collection and prediction could enhance the accuracy of forecasts by revealing longer-term trends and seasonal patterns. Moreover, considering alternative time series models like Seasonal ARIMA and exploring machine learning techniques may enhance prediction precision, and a comparative assessment of these models would provide insights into the most effective resource optimization approach. In terms of contextualizing this research within existing literature, it fills a significant gap by offering a data-driven solution for optimizing resource allocation within fixed limits, a distinct departure from the predominant focus on vertical resource scaling. This approach aligns more effectively with the constraints of diverse application scenarios, ultimately advancing the field of cloud computing resource management.

7 Conclusion and Future Work

In conclusion, this research has demonstrated a data-driven approach to optimize resource allocation for containerized applications in a cloud computing environment. By leveraging historical CPU utilization metrics and ARIMA forecasting, proposed a strategy to dynamically adjust CPU resource limits for pods. The results showcased a substantial reduction in resource underutilization, leading to cost savings of around 60% and improved resource efficiency. Through comprehensive workload analysis, time series modeling, and Kubernetes integration, we illustrated the feasibility and potential benefits of this approach.

Future Work: Looking ahead, there are several exciting directions to build upon the progress made in this study. One promising avenue is to expand the predictive approach beyond CPU utilization to include other vital metrics like memory and network usage. This would create a more comprehensive strategy for optimizing different aspects of resource allocation. Additionally, it is good delve into the realm of advanced techniques like Seasonal ARIMA and ACF, PACF smoothing techniques to refine the predictions further. By testing the proposed methodology with a broader range of application scenarios and considering longer duration dataset, it is possible to gain more comprehensive understanding of its real-world applicability. Exploring dynamic resource management through real-time adjustments and exploring how the approach interacts with auto-scaling mechanisms would add another layer of sophistication to the strategy. Ultimately, ongoing research in this field holds the promise of revolutionizing how we manage cloud resources, making applications more efficient, cost-effective, and responsive to changing demands.

References

- Baarzi, A. F. and Kesidis, G. (2021). Showar: Right-sizing and efficient scheduling of microservices, *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21, Association for Computing Machinery, New York, NY, USA, p. 427–441.
URL: <https://doi.org/10.1145/3472883.3486999>
- Dmytro, K., Sergii, T. and Andiy, P. (2017). Arima forecast models for scheduling usage of resources in it-infrastructure, *2017 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)*, Vol. 1, pp. 356–360.
- Gweon, H. and McLeod, A. I. (2013). Seasonal decomposition for geographical time series using nonparametric regression.
URL: <https://api.semanticscholar.org/CorpusID:31041250>
- Hamzeh, H., Meacham, S. and Khan, K. (2019). A new approach to calculate resource limits with fairness in kubernetes, *2019 First International Conference on Digital Data Processing (DDP)*, pp. 51–58.
- Li, P., Wang, X., Huang, K., Huang, Y., Li, S. and Iqbal, M. (2022). Multi-model running latency optimization in an edge computing paradigm, *Sensors* **22**(16).
URL: <https://www.mdpi.com/1424-8220/22/16/6097>
- Liu, Y., Peng, M., Shou, G., Chen, Y. and Chen, S. (2020). Toward edge intelligence: Multiaccess edge computing for 5g and internet of things, *IEEE Internet of Things Journal* **7**(8): 6722–6747.
- Rattihalli, G., Govindaraju, M., Lu, H. and Tiwari, D. (2019). Exploring potential for non-disruptive vertical auto scaling and resource estimation in kubernetes, *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 33–40.
- Rattihalli, G., Govindaraju, M. and Tiwari, D. (2019). Towards enabling dynamic resource estimation and correction for improving utilization in an apache mesos cloud environment, *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 188–197.
- Reiss, C., Tumanov, A., Ganger, G. R., Katz, R. H. and Kozuch, M. A. (2012). Heterogeneity and dynamicity of clouds at scale: Google trace analysis, *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, Association for Computing Machinery, New York, NY, USA.
URL: <https://doi.org/10.1145/2391229.2391236>
- Yunyun, Q., Chen, P. and Tan, D. (2022). Research on elastic cloud resource management strategies based on kubernetes, *2022 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*, pp. 441–448.