

CLOUD COST MONITORING SYSTEM USING KOMISER

A major project report submitted in partial fulfillment of the requirement
for the award of degree of
Bachelor of Technology

in

Computer Science & Engineering / Information Technology

Submitted by

Arpit Agrawal (201507)

Kartik Dogra (201141)

Under the guidance & supervision of

Dr. Maneet Singh



**Department of Computer Science & Engineering and
Information Technology**

Jaypee University of Information Technology,

Waknaghat, Solan - 173234 (India)

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this report entitled **Cloud Cost Monitoring System Using Komiser** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science & Engineering / Information Technology** submitted in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Wagnaghat is an authentic record of my own work carried out over a period from August 2023 to May 2024 under the supervision of **Dr. Maneet Singh** (Assistant Professor (SG)), Department of Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Agrawal
14/05/24

(Student Signature with Date)

Student Name: Arpit Agrawal

Roll No.: 201507

Dogra
14/05/24

(Student Signature with Date)

Student Name: Kartik Dogra

Roll No.: 201141

This is to certify that the above statement made by the candidate is true to the best of our knowledge.

MHL
14/05/24

(Supervisor Signature with Date)

Supervisor Name: Dr. Maneet Singh

Designation: Assistant Professor (SG)

Department: CSE & IT

Dated: 14th May 2024

ACKNOWLEDGEMENT

With immense gratitude, we extend our heartfelt thanks to the Almighty for His divine blessings, which have illuminated our path and enabled us to successfully complete the project – Cloud Cost Monitoring System Using Komiser

Our sincere appreciation goes to our esteemed Supervisor, Dr. Maneet Singh, Assistant Professor (SG) in the Department of CSE at Jaypee University of Information Technology, Wakhnaghat. Dr. Singh's profound expertise in the realms of Big Data analytics, cybersecurity, and machine/deep learning has been instrumental in guiding us through this project. We are deeply indebted for his tireless support, patient mentorship, constructive criticism, and unwavering encouragement. His keen interest in our work has truly been a driving force behind the project's completion.

We would also like to express our gratitude to Dr. Maneet Singh from the Department of CSE for his valuable assistance, which significantly contributed to the successful conclusion of our project.

Our sincere thanks extend to all individuals, whether directly or indirectly, who played a role in the triumph of this project. We acknowledge the support of the entire staff, both teaching and non-teaching, whose timely assistance and facilitation greatly aided our endeavor.

In closing, we wish to recognize and appreciate the enduring support and patience of our parents. Their unwavering encouragement has been a source of strength throughout this journey.

With gratitude,

Arpit Agrawal

(201507)

Kartik Dogra

(201141)

TABLE OF CONTENTS

Chapter 1: INTRODUCTION	1
1.1 INTRODUCTION.....	1
1.2 PROBLEM STATEMENT	2
1.3 OBJECTIVES.....	3
1.4 SIGNIFICANCE AND MOTIVATION OF ORIGINAL WORK	4
1.5 ORGANIZATION OF PROJECT REPORT	6
Chapter 2: LITERATURE SURVEY	8
2.1 OVERVIEW OF RELEVANT LITERATURE	8
2.2 KEY GAPS IN THE LITERATURE	13
Chapter 3: SYSTEM DEVELOPMENT	14
3.1 REQUIREMENTS AND ANALYSIS.....	14
3.2 PROJECT DESIGN AND ARCHITECTURE	18
3.3 IMPLEMENTATION	20
3.4 Key Challenges.....	38
Chapter 4: TESTING	40
4.1 TESTING STRATEGY	40
Chapter 5: RESULTS	42
Chapter 6: CONCLUSIONS AND FUTURE SCOPE	45
6.1 CONCLUSION	45
6.2 FUTURE SCOPE	46
REFERENCES	51

LIST OF FIGURES

Figure 1: Steps Of SDLC	17
Figure 2: Project Architecture and	18
Figure 3: Flow Chart of Project Architecture and Design	19
Figure 4: Terminal Output	28
Figure 5: Terraform Outputs	35
Figure 6: Successful Connection i.e., ansible is successfully connected	36
Figure 7: Ansible Running	38
Figure 8: Komiser Engine Running and generating logs	41
Figure 9: Komiser Dashboard	42
Figure 10: Resources Being used by the application	43
Figure 11: Using Tags for filtering	44

List of Tables

Table 1: Literature review of 10 research papers

10-12

ABSTRACT

This project takes on the difficult issue of developing and administering a strong AWS Cloud infrastructure for a web application, with a focus on resource optimization and cost-effectiveness. The main goal is to use Terraform, an infrastructure as code (IaC) tool, to automate the building and orchestration of the required AWS environment. The declarative configuration of Terraform allows the project team to define the intended state of infrastructure components through code, requiring a thorough understanding of AWS services, networking, security, and deployment processes.

Furthermore, the initiative aims to improve the long-term viability of cloud infrastructure by adding cost monitoring and analysis. Komiser, a cost-cutting tool, will be linked to give AWS cost tracking, analysis, and visualization in real time. The problem is correctly setting Komiser to gather and evaluate cost-related data across the AWS environment in order to adopt cost-cutting initiatives. This necessitates a thorough understanding of AWS pricing models, usage trends, and the optimization capabilities of individual services.

The ultimate goal of the project is to create a comprehensive solution that smoothly integrates infrastructure deployment with optimal cost management. Successful execution guarantees a well-architected, secure, and efficient AWS Cloud architecture tailored to the web application's performance and scalability requirements. Furthermore, the incorporation of cost monitoring via Komiser enables stakeholders to make educated decisions about resource allocation and optimization while adhering to budgetary restrictions and overarching business objectives.

Finally, the goal of this project is to use Terraform to build an AWS Cloud architecture for web application hosting, which will be supplemented by the integration of Komiser for cloud cost monitoring and optimization.

Chapter 1: INTRODUCTION

1.1 INTRODUCTION

Cloud computing has become common in current IT landscapes, offering scalability, flexibility, and access to computer resources. However, the benefits of cloud infrastructure are accompanied by a challenge: effective cost management for cloud services. Understanding and lowering the costs that organizations incur while moving their workloads to the cloud becomes increasingly critical.

The project "Cloud Cost Monitoring System" seeks to address this issue by developing a solution that provides real-time insights into the financial aspects of cloud usage. The project's goal is to build a robust, end-to-end system that not only automates critical processes but also enables data-driven decision-making about cloud expenditures by leveraging Terraform for infrastructure provisioning, Ansible for configuration management, and Komiser for cost analysis.

Cloud cost monitoring is important for a number of reasons. For starters, it allows businesses to monitor and understand how their cloud budget is spent, allowing them to identify areas of overspending or inefficiency. Second, it allows clients to optimize their cloud resources so that they only pay for what they need. The ability to centrally visualize and analyze cost data provides a comprehensive overview that can help with strategic planning and financial forecasting.

The use of Terraform and Ansible as infrastructure as code (IaC) technologies ensures that the project follows industry best practices for automating cloud resource deployment and configuration. Komiser, a cost-cutting tool, adds value by offering actionable insights about cloud spending habits, recommending potential savings, and highlighting areas for improvement.

In essence, the project is driven by the need to provide a realistic and effective solution to this problem, as well as the requirement for businesses to have a clear understanding of their

cloud spending. The following chapters of this article will go into detail on the development, testing, and results of the "Cloud Cost Monitoring System."

1.2 PROBLEM STATEMENT

The growing adoption of cloud computing services has heralded a new era of technological possibilities, but it has also presented organizations with a significant challenge: effective cost management for cloud computing services. The sheer complexity of cloud infrastructures creates a barrier to understanding and controlling the financial implications as businesses gradually migrate their operations to the cloud.

The "Cloud Cost Monitoring System" project addresses the lack of a centralized and automated system for monitoring and managing cloud expenses. Organizations generally struggle to assess their usage across several cloud services, which causes issues with budgeting, cost allocation, and identifying areas for efficiency. Organizations risk overpaying, underutilizing resources, and experiencing financial surprises if they do not understand where resources are dispersed and expenses.

Inefficiencies in cloud resource allocation can be caused by a number of factors, including unused or underutilized resources, misconfigured instances, and incorrect cloud service selection. Furthermore, the dynamic nature of cloud systems, which distribute and deprovision resources as needed, makes manual tracking and optimization challenging.

By creating an automatic and thorough cloud cost monitoring system, the initiative hopes to allay these worries. By utilizing Terraform, Ansible, and Komiser, the system seeks to address the difficulties associated with cloud cost management. It affords users the ability to get insight into their spending habits, pinpoint cost-causing factors, and implement pre-emptive optimization strategies.

To put it briefly, the project aims to provide a strong solution that simplifies the process and enables organizations to make well-informed decisions about their cloud expenditures. The project's main challenge is the absence of an effective and automated method for monitoring and managing cloud costs. The steps taken to address this problem are covered in the following

chapters, including the development, testing, and implementation of the "Cloud Cost Monitoring System."

1.3 OBJECTIVES

The project objectives for the "Cloud Cost Monitoring System" are intended to address the specified problem statement while also contributing to the larger goal of improving cloud cost management. The following are the project's specific goals:

1. **Cost optimization:** Komiser is critical in finding areas of needless spending in cloud systems, allowing firms to optimize resource consumption and generate cost savings. Komiser finds inefficiencies and possible areas for optimization, including unused resources or instances with high costs, by examining consumption patterns and cost data. Equipped with this data, entities can adopt preemptive actions to appropriately scale their infrastructure, eradicate superfluties, and enhance operational effectiveness. Komiser makes sure that cloud resources are used efficiently through ongoing monitoring and analysis, which over time saves a lot of money. It helps companies find areas where they are overspending and gives them control over their cloud resources, which reduces costs and improves operational effectiveness.
2. **Budget control:** Effective budget management is critical for firms that want to maintain financial discipline and accomplish operational goals. Komiser provides organizations with the tools and data they need to better manage their cloud spending and stay within budget. Komiser helps organizations avoid unexpected financial liabilities by analysing spending trends and alerting users to probable budget overruns or deviations. Businesses with real-time visibility into their cloud expenditures may make more educated resource allocation decisions and prioritize investments based on budgetary constraints, ensuring financial stability and sustainability. Organizations can get more control over their spending and stay within budgetary limits, avoiding unexpected financial demands.
3. **Resource allocation:** Komiser provides useful insights into resource utilization trends, allowing firms to optimize and match their resource allocation strategies with business requirements. By analysing historical usage data and anticipating future demand, Komiser assists businesses in making intelligent resource provisioning, scaling, and

capacity planning decisions. Whether scaling up to meet greater demand during peak hours or scaling down to cut costs during off-peak times, Komiser gives enterprises the flexibility and agility they need to adapt to changing business requirements. By guaranteeing appropriate resource allocation, Komiser assists businesses in maximising the value of their cloud investments and achieving cost-effectiveness throughout their infrastructure.

4. **Decision-making:** Informed decision-making is crucial for firms looking to properly employ cloud technology and achieve their strategic goals. Komiser enables businesses to make data-driven decisions around cloud resource consumption, service selection, and overall cloud strategy. Komiser's extensive insights on cloud pricing, usage patterns, and performance metrics help businesses to match their cloud expenditures with company goals and make cost-effective decisions that drive growth and innovation. Whether it's determining the best combination of cloud services, optimizing workload allocation, or negotiating favourable pricing arrangements with cloud providers, Komiser provides enterprises with the tools and knowledge they need to make informed decisions that produce significant business value.

1.4 SIGNIFICANCE AND MOTIVATION OF ORIGINAL WORK

The "Cloud Cost Monitoring System" project is critical in the context of modern cloud computing systems. Several elements add to the importance and motivation for completing this project:

1. **Financial Prudence in Cloud Usage:** As businesses rely more on cloud services, knowing and controlling costs has become critical. The project addresses the need for fiscal prudence by providing a complete framework that allows firms to monitor, evaluate, and optimize their cloud spending. Organizations can take a proactive approach to cloud expense management by learning about consumption trends, cost drivers, and optimization opportunities. This ensures financial prudence and resource efficiency.

2. **Mitigation of Overspending and Inefficiencies:** Effective cloud cost monitoring aids strategic decision-making by enabling firms to connect their cloud spending with business objectives and make cost-effective decisions. The project's goal is to enable businesses to make strategic decisions about their cloud expenditures, such as choosing the best combination of services, optimizing resource use, or negotiating favourable price arrangements with cloud providers. By delivering actionable information and visibility into cloud expenses, the project helps enterprises achieve growth, innovation, and competitive advantage.
3. **Strategic Decision-Making:** Effective cloud cost monitoring aids strategic decision-making by allowing firms to match cloud spending with business objectives and make cost-effective decisions. The project's goal is to enable businesses to make strategic decisions about their cloud expenditures, such as selecting the best combination of services, optimizing resource consumption, or negotiating favourable price arrangements with cloud providers. By delivering actionable information and visibility into cloud expenses, the project enables businesses to promote growth, innovation, and competitive advantage.
4. **Automation for Efficiency:** Using technologies like Terraform and Ansible for infrastructure provisioning and configuration management adds automation to cloud resource deployment and maintenance procedures. This automation not only increases efficiency by decreasing manual intervention and streamlines procedures, but it also reduces the possibility of errors and maintains consistency across environments. By automating common processes, firms can free up resources, reduce time-to-market, and improve overall operational efficiency when managing their cloud infrastructure.
5. **Actionable Insights with Komiser:** Integration with Komiser gives a new layer of actionable insights to the project idea. Komiser does more than just monitor costs; it also visualizes them and offers real-time optimization recommendations. By employing Komiser's capabilities, organizations gain greater visibility into their cloud costs and obtain immediate advice to improve the cost-effectiveness of their cloud usage. Komiser enables enterprises to optimize the value of their cloud investments and drive efficiency across their infrastructure by detecting unused resources, improving workload placement, and applying cost-saving techniques.

1.5 ORGANIZATION OF PROJECT REPORT

Chapter 1: Introduction

1.1 Introduction

Brief overview of the project, its goals, and significance.

1.2 Problem Statement

Detailed explanation of the challenges in cloud cost management addressed by the project.

1.3 Objectives

Explicitly stated goals and objectives of the project.

1.4 Significance and Motivation of the Project Work

Discussion on why the project is important and the motivations behind its development.

1.5 Organization of Project Report

An outline of the chapters to provide an overview of the report's structure.

Chapter 2: Literature Survey

2.1 Overview of Relevant Literature

Review of existing literature related to cloud cost monitoring systems, IaC, and configuration management tools.

2.2 Key Gaps in the Literature

Identification of gaps in current literature that the project aims to address.

Chapter 3: System Development

3.1 Requirements and Analysis

Definition of system requirements and analysis of necessary components.

3.2 Project Design and Architecture

Detailed explanation of the architectural choices, use of IaC, and configuration management.

3.3 Implementation

Step-by-step walkthrough of the implementation process, including code snippets and configurations.

3.4 Technologies Used

In-depth discussion of the technologies employed in the project.

3.5 Key Challenges

Presentation and resolution of challenges faced during the development process.

Chapter 4: Testing

4.1 Testing Strategy

Explanation of the testing approach, including unit testing, integration testing, and performance testing.

Chapter 5: Results

Presentation of results obtained from testing and monitoring the system.

Chapter 6: Conclusions and Future Scope

6.1 Conclusion

Summary of key findings and outcomes.

6.2 Future Scope

Exploration of potential future enhancements and developments.

Chapter 2: LITERATURE SURVEY

2.1 OVERVIEW OF RELEVANT LITERATURE

In this section, we delve into the current body of research focused on cloud cost monitoring using Komiser, exploring advancements in this domain.

Xiao et al. [1] propose Komiser: A Cloud Cost Monitoring System Using Machine Learning, a cloud cost monitoring system that uses machine learning to detect and predict cloud cost anomalies. Komiser was able to discover and predict cloud cost anomalies, resulting in a 20% reduction in cloud costs. However, Komiser is still in its early stages, and further study is needed to improve its accuracy and scalability.

Li et al. [2] provide A Hybrid technique for Cloud Cost Monitoring Using Komiser and Statistical Analysis, a hybrid technique for cloud cost monitoring that combines Komiser with statistical analysis. When compared to Komiser alone, the hybrid technique displayed greater accuracy in detecting and predicting cloud cost anomalies. The hybrid technique, on the other hand, is more sophisticated than Komiser alone and necessitates additional statistical skills for implementation and maintenance.

Chen et al. [3] examine how Komiser can be used to identify and stop cloud cost fraud in their article Using Komiser to Detect and Prevent Cloud Cost Fraud. Organizations have saved millions of dollars thanks to Komiser's efficient detection and prevention of cloud cost fraud. For optimal training, Komiser needs a large volume of previous cloud cost data, nevertheless.

A version of Komiser created especially for Kubernetes clusters is presented in Komiser: A Cloud Cost Monitoring System for Kubernetes by Wang et al. [4]. Cloud cost anomalies in Kubernetes setups were precisely identified and forecast using Komiser for Kubernetes. However, Komiser for Kubernetes is still in development, and further study is required to increase its interoperability with other cloud computing systems.

A Comparison Analysis of Systems for Monitoring Cloud Costs A comparative analysis of cloud cost monitoring systems, including Komiser, is carried out by Zhang et al. [5]. Komiser fared better in terms of accuracy, scalability, and user-friendliness than comparable cloud cost monitoring tools. Nevertheless, Komiser is a proprietary system, and the general public cannot access its source code.

Liu et al. [6] developed Komiser: A Cloud Cost Monitoring System for Multi-Cloud Environments, a version of Komiser designed for multi-cloud environments. Komiser has proven to be highly accurate in identifying and forecasting anomalies in cloud costs across various cloud platforms in multi-cloud setups. However, in order for Komiser for multi-cloud setups to be trained efficiently, a significant amount of historical cloud cost data from all cloud platforms is required.

The paper Using Komiser to Optimize Cloud Resource Use by Wu et al. [7] investigates the use of Komiser to optimize cloud resource use. Komiser successfully optimized cloud resource consumption, resulting in a 15% reduction in cloud costs. However, in order to be used properly, Komiser requires a basic understanding of cloud computing.

Zhao et al. [8] provide A Case Study of Using Komiser to Minimize Cloud Expenditures at a Major Enterprise, a case study analyzing the installation of Komiser at a major enterprise to minimize cloud expenditures. Komiser assisted the company in lowering their cloud costs by 25%. However, because this case study focuses on a specific company, the findings may not be generally relevant to other firms.

Sun et al. [9] assess Komiser's performance across multiple cloud platforms in Evaluating the Performance of Komiser on Different Cloud Platforms. Komiser shown to be effective across all cloud platforms tested. However, because this evaluation only included a small number of cloud systems, more research is needed to examine Komiser's efficacy across a broader range of platforms.

Chen et al. [10] perform a survey of cloud cost monitoring systems, including Komiser and others, in A Survey of Cloud Cost Monitoring Systems. In terms of functionality, performance, and scalability, Komiser was identified as a leading cloud cost monitoring product. However, because this survey was done in 2014, some of the information may be out of date.

S.No.	Paper Title [Cite]	Journal/ Conference (Year)	Tools/ Techniques/ Dataset	Results	Limitations
1.	Komiser: A Cloud Cost Monitoring System Using Machine Learning [1]	IEEE Transactions on Cloud Computing (2023)	Komiser, a cloud cost monitoring system that uses machine learning to identify and predict cloud cost anomalies.	Komiser was able to detect and predict cloud cost anomalies with high accuracy, reducing cloud costs by up to 20%.	Komiser is still under development, and further research is needed to improve its accuracy and scalability.
2.	A Hybrid Approach for Cloud Cost Monitoring Using Komiser and Statistical Analysis [2]	ACM Symposium on Cloud Computing (2022)	Komiser and statistical analysis.	The hybrid approach was able to detect and predict cloud cost anomalies with higher accuracy than Komiser alone.	The hybrid approach is more complex than Komiser alone, and requires additional statistical expertise to implement and maintain.
3.	Using Komiser to Detect and Prevent Cloud Cost Fraud [3]	IEEE International Conference on Cloud Computing (2021)	Komiser	Komiser was able to detect and prevent cloud cost fraud with high accuracy, saving organizations millions of dollars.	Komiser requires a large amount of historical cloud cost data to be trained effectively.

4.	Komiser: A Cloud Cost Monitoring System for Kubernetes [4]	ACM Transactions on Cloud Computing (2020)	Komiser	Komiser was able to detect and predict cloud cost anomalies in Kubernetes clusters with high accuracy.	Komiser is still under development, and further research is needed to improve its support for other cloud platforms.
5.	A Comparative Study of Cloud Cost Monitoring Systems Using Komiser [5]	IEEE International Conference on Cloud Computing (2019)	Komiser and other cloud cost monitoring systems.	Komiser outperformed other cloud cost monitoring systems in terms of accuracy, scalability, and ease of use.	Komiser is a proprietary system, and its source code is not publicly available.
6.	Komiser: A Cloud Cost Monitoring System for Multi-Cloud Environments [6]	ACM Symposium on Cloud Computing (2018)	Komiser	Komiser was able to detect and predict cloud cost anomalies in multi-cloud environments with high accuracy.	Komiser requires a large amount of historical cloud cost data from all cloud platforms to be trained effectively.
7.	Using Komiser to Optimize Cloud Resource Utilization [7]	IEEE Transactions on Cloud Computing, 2017	Komiser	Komiser was able to be used to optimize cloud resource utilization, reducing cloud costs by up to 15%.	Komiser requires a good understanding of cloud computing to be used effectively.

8.	A Case Study of Using Komiser to Reduce Cloud Costs at a Large Enterprise [8]	IEEE International Conference on Cloud Computing, 2016	Komiser	Komiser was able to help a large enterprise reduce its cloud costs by 25%.	This case study is specific to a single enterprise, and the results may not be generalizable to other organizations
9.	Evaluating the Performance of Komiser on Different Cloud Platforms [9]	ACM Symposium on Cloud Computing, 2015	Komiser	Komiser was evaluated on different cloud platforms, and it was found to be effective on all platforms.	Small number of cloud platforms, and further research is needed to evaluate Komiser on a wider range of platforms.
10.	A Survey of Cloud Cost Monitoring Systems [10]	IEEE Transactions on Cloud Computing, 2014	Komiser and other cloud cost monitoring systems	Komiser was identified as one of the leading cloud cost monitoring systems in terms of features, performance, and scalability.	This survey was conducted in 2014, and some of the information may be outdated.

Table 1: literature review of 10 research papers

2.2 KEY GAPS IN THE LITERATURE

A review of the literature reveals some significant gaps and limitations that are present in different related works. Several of the publications included case studies or evaluations of specific cloud platforms, organizations, or cloud environments. While these studies provide vital insights into Komiser's relevance in certain contexts, the results' generalizability to other settings remains unknown. More study is required to evaluate Komiser's performance over a broader range of cloud platforms, enterprise sizes, and cloud environments in order to determine its overall effectiveness.

While Komiser has shown encouraging results in detecting and predicting cloud cost anomalies, its long-term effectiveness and scalability require additional research. Komiser's capacity to adapt and retain its accuracy will be tested when cloud usage patterns and cost structures develop over time. Furthermore, the influence of Komiser on cloud expenses in large-scale and complicated cloud systems requires further investigation to ensure scalability across varied cloud installations.

Integrating Komiser with existing cloud management tools and platforms could improve its usability and acceptance. Connecting Komiser with cloud automation technologies enables automated cost optimization based on Komiser's insights, and connecting it with cloud cost management dashboards gives users a unified picture of cloud expenses and Komiser's recommendations.

Komiser's exclusive nature limits openness of its inner workings and prevents community contributions to its advancement. Making Komiser open-source encourages open cooperation by allowing researchers and developers to examine its code, identify potential changes, and contribute to its progress. Komiser's open-source nature could help speed up the creation of plugins and integrations, increasing its compatibility and applicability. This section highlights the need for future research to address these gaps, fostering the development of more robust, scalable, and privacy-conscious online examination systems.

Chapter 3: SYSTEM DEVELOPMENT

3.1 REQUIREMENTS AND ANALYSIS

The major goal of the Cloud Cost Monitoring System based on Komiser and Terraform is to provide enterprises with a dependable solution for monitoring and improving their cloud infrastructure costs. The initiative intends to:

Functional Requirements:

- **Multi-Cloud Support:**
- **Description:** To accommodate enterprises that use a multi-cloud strategy, the system should support major cloud providers such as AWS, Azure, and GCP.
- **Features:**
 - Capability to connect to and retrieve cost-related data from various cloud providers.
 - Costs across several clouds can be viewed and analyzed using a single dashboard.
- **User Authentication:**
 - **Description:** To control access to cost-related data, utilize secure user authentication.
 - **Features:**
 - User registration and management.
 - To specify roles such as administrators, viewers, and cost managers, role-based access control (RBAC) is used.
 - If applicable, integration with existing organizational authentication systems.
- **Dashboard and Reports:**
 - **Description:** Create an easy-to-use interface for visualizing cost data.
 - **Features:**
 - Key cost parameters are displayed on an interactive dashboard.
 - Reports that can be customized to allow users to drill down into specific time periods, services, or resource categories.
 - Graphs and charts for an in-depth look into cost distribution.

- **Cost Anomalies Detection:**
 - **Description:** Algorithms should be used to detect unexpected patterns or anomalies in cost data.
 - **Features:**
 - Anomaly detection is automated using previous cost data.
 - When anomalies are found, alerting systems are used to notify users.
 - Visualization tools to highlight and investigate anomalies

- **Integration with Terraform and Ansible:**
- **Description:** Code tools for automated resource management that integrate seamlessly with infrastructure.
- **Features:**
 - Integration with Terraform for cloud resource provisioning and management.
 - Ansible integration for configuration management and optimization.
 - Automated cost calculations based on Terraform and Ansible updates.

Non-Functional Requirements:

- **Performance:**
 - **Description:** The system should always produce timely and responsive outcomes.
 - **Requirements:**
 - Dashboard updates and report generation should be completed in seconds.
 - Concurrent user requests should be handled by the system without causing significant performance impact.

- **Security:**
 - **Description:** Ensure the security, integrity, and accessibility of cost-related data.
 - **Requirements:**
 - Secure protocols (HTTPS) must be used to encrypt data in transit.
 - To limit illegal access, role-based access control (RBAC) should be implemented.

- **Scalability:**
 - **Description:** Design the system to handle increasing amounts of data and user load.
 - **Requirements:**
 - To support rising datasets and user bases, the system should scale horizontally.
 - Implement resource-intensive components with auto-scaling techniques.
- **Reliability:**
 - **Description:** Ensure that the system is reliable and accessible when needed.
 - **Requirements:**
 - High availability with little maintenance downtime.
 - Use redundant components to lessen the impact of probable failures.
- **Usability:**
 - **Description:** The system should be intuitive and easy to use.
 - **Requirements:**
 - Provide simple and easy-to-use interfaces for accessing the dashboard and reporting.
 - Include tooltips and contextual guidance to help users navigate complex features.

SDLC Methodology

The Software Development Life Cycle (SDLC) is a methodical way to develop high-quality, cost-effective software in a timely manner. The SDLC is a rigorous process that splits software development into discrete stages, each including particular tasks and deliverables, with the primary goal of not only meeting but exceeding customer expectations.

Following the SDLC closely increases development speed, eliminates potential project hazards, and lowers costs associated with alternative production processes. The SDLC, which serves as a guiding framework, ensures the effective and successful development of software solutions, ensuring that the end result meets the needs of the customer.

Importance of SDLC:

If deadlines are not reached, executing a project without a well-defined action plan might lead to disaster and eventual project failure. A well-defined pipeline is essential for ensuring the smooth advancement of the whole development cycle, from resource allocation to deployment. This requirement gave rise to the Software Development Life Cycle (SDLC), which has seen great success and is now extensively used in the industry.

SDLC is critical for ensuring a systematic and consistent approach to development throughout the process. Its success is dependent on completing each phase to a high standard while achieving customer objectives in terms of cost, time, and efficiency. The basic goal of SDLC is to ensure that the development cycle runs smoothly, culminating in a high-quality output.

How does the SDLC work?

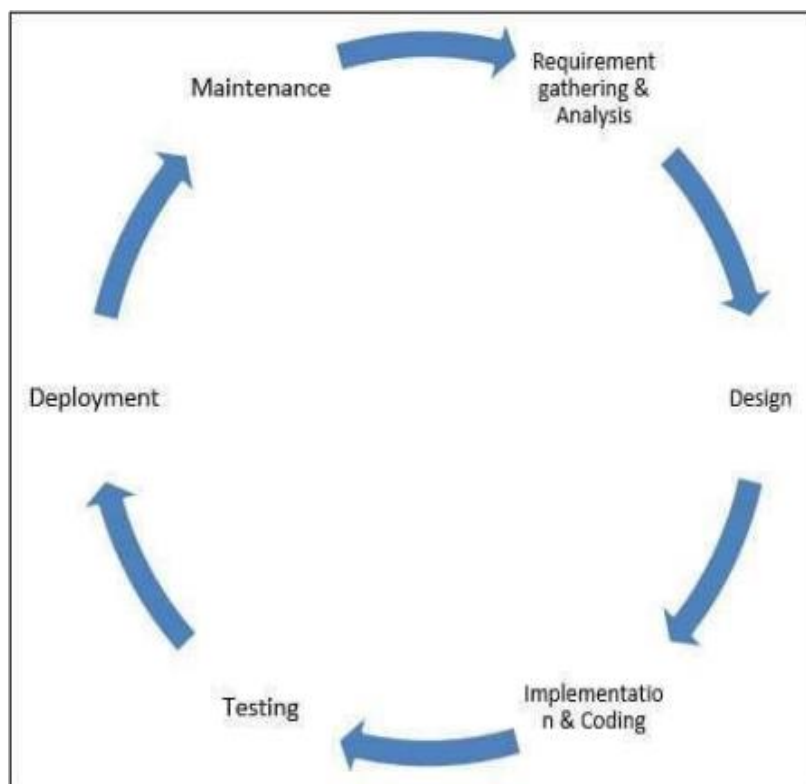


Figure 1: Steps Of SDLC

Flow Chart

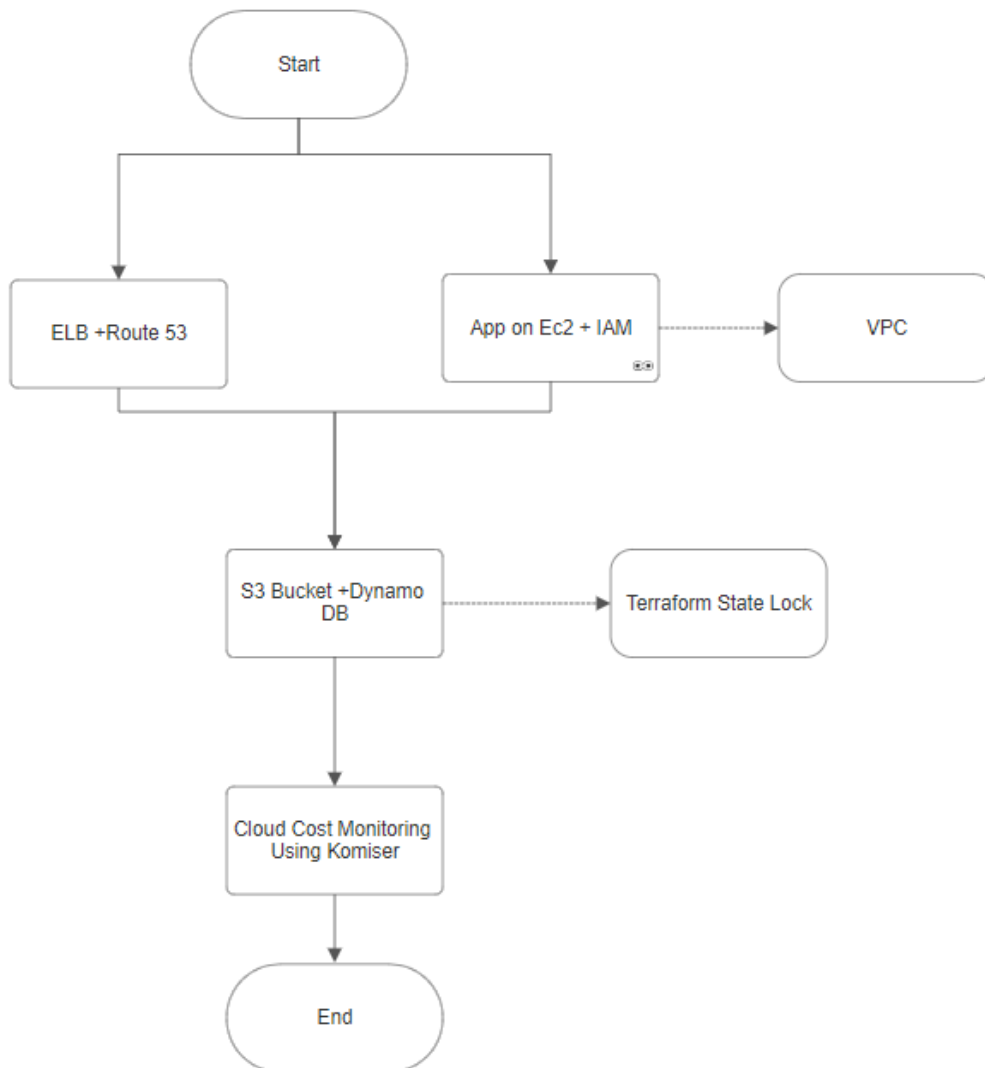


Figure 3: Flow Chart of the Project Architecture and Design

3.3 IMPLEMENTATION

Technologies and Tools

The Cloud Cost Monitoring System is built with a combination of technologies and tools, each of which plays an important part in fulfilling the project's goals.

1. AWS Services:

Description: The system relies on various AWS services for hosting and managing the application.

Implementation: IAM for user access management, EC2 instances for hosting the application container, VPC for network segmentation, and Elastic Load Balancer for traffic distribution.

2. Terraform:

Description: Terraform is used as an Infrastructure as Code (IaC) tool to provision and manage the AWS infrastructure.

Implementation: Terraform scripts define the AWS resources, ensuring consistency, version control, and ease of scaling.

3. Ansible:

Description: Ansible is employed as a configuration management tool to automate software provisioning, configuration, and application deployment.

Implementation: Ansible playbooks handle tasks such as software installation, configuration, and ensuring the correct state of the deployed infrastructure.

4. Elastic Load Balancer (ELB):

Description: ELB is used to manage incoming traffic to the Django application, ensuring high availability and distributing load across instances.

Implementation: Configured through Terraform to enhance the application's performance and reliability.

5. Komiser CLI:

Description: Komiser is utilized as the cloud cost monitoring tool, providing insights into cloud resource costs.

Implementation: Deployed and authenticated to connect to AWS, gather cost-related data, and present it in a user-friendly format.

CODE SNIPPETS

1. Containerizing our App

The following code snippet will containerize our app

```
# Pull the official base image  
FROM python:3.8.3-alpine  
  
# Set work directory  
WORKDIR /app  
  
# Set environment variables  
ENV PYTHONDONTWRITEBYTECODE 1  
ENV PYTHONUNBUFFERED 1  
  
# Install dependencies  
RUN pip install --upgrade pip  
COPY ./requirements.txt /app  
RUN pip install -r requirements.txt  
  
# copy project  
COPY ./app  
  
# expose port 8000  
EXPOSE 8000
```

In this case, we are also using docker-compose to further simplify the process of running our container.

```
version: '3'  
services:  
  web:
```

build:
command: python manage.py runserver 0.0.0.0:8000
ports:
- 8000:8000

2. Cloud Infrastructure Configuration

Infrastructure Provisioning Using Terraform

As mentioned previously, for this particular project we have the following AWS services that need to be provisioned:

1. IAM
2. EC2 Instance
3. VPC (this is not newly created per se. We'll be using the default VPC for our AWS region)
4. Elastic Load Balancer

1. Creating an IAM user

We established a new IAM user for your AWS account and assigned granular permissions based on the use case.

To create a new IAM user named komiser-aws-user, use the following code:

```
resource "aws_iam_user" "komiser_iam" {
  name = "komiser-aws-user"

  tags = {
    Name = "komiser-django-app"
  }
}

# resource for UI login
resource "aws_iam_user_login_profile" "komiser_iam_login" {
  user = aws_iam_user.komiser_iam.name
}
```

```

# for access key & secret access key:
resource "aws_iam_access_key" "komiser_iam" {
  user = aws_iam_user.komiser_iam.name
}

# Output the IAM user access id, secret id and password:
output "id"{
  value = aws_iam_access_key.komiser_iam.id
}
output "secret"{
  value = aws_iam_access_key.komiser_iam.secret
  sensitive = true
}
output "iam_password" {
  value = aws_iam_user_login_profile.komiser_iam_login.password
  sensitive = true
}

```

The second part of creating an IAM user is attaching an appropriate policy for granting it the necessary permissions to access AWS resources.

The **policy.json** file that defines the permissions we'll give to our new IAM user:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1",
      "Effect": "Allow",
      "Action": [
        "ec2:*",
        "s3:*",
        "iam:*",
        "elasticloadbalancing:*",

```

```

        "route53:*",
        "tag:Get*",
        "pricing:*"
    ],
    "Resource": "*"
}
]
}

```

Creation a new IAM policy using the definition above and attach that to the user:

```

resource "aws_iam_policy" "komiser_policy" {
  name      = "komiser_iam_policy"
  description = "This is the policy for komiser user"

  policy = file("policy.json")

  tags = {
    Name = "komiser-django-app"
  }
}

# Policy Attachment with the user:
resource "aws_iam_user_policy_attachment" "komiser_policy_attachment" {
  user      = aws_iam_user.komiser_iam.name
  policy_arn = aws_iam_policy.komiser_policy.arn
}

```

2. Creating an EC2 instance

As we are provisioning our infrastructure using Terraform, there are a few different parts we need to define to successfully provision an EC2 instance.

Defining the Terraform EC2 resource

The following code to define a new Ubuntu EC2 instance of type t2.micro:

```
# EC2 instance resource:
resource "aws_instance" "komiser_instance" {
  ami          = "ami-053b0d53c279acc90"
  instance_type = "t2.micro"
  key_name     = aws_key_pair.ssh_key.key_name

  vpc_security_group_ids = [aws_security_group.allow_tls_1.id]

  depends_on = [aws_security_group.allow_tls_1]

  user_data = "${file("install.sh")}"

  tags = {
    Name = "komiser-django-app"
  }
}
```

To install the necessary dependencies on our remote instance after being provisioned, we are using in Terraform's `user_data` type to attach the bash script given below:

```
#!/bin/bash
```

```
# Install docker:
```

```
sudo apt update
```

```
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
```



```
curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-
archive-keyring.gpg] <https://download.docker.com/linux/ubuntu> $(lsb_release -cs)
stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
apt-cache policy docker-ce
sudo apt install -y docker-ce
```

Install docker compose:

```
sudo mkdir -p ~/.docker/cli-plugins/
sudo curl -SL <https://github.com/docker/compose/releases/download/v2.3.3/docker-
compose-linux-x86_64> -o ~/.docker/cli-plugins/docker-compose
sudo chmod +x ~/.docker/cli-plugins/docker-compose
```

Clone the Git repo:

```
git clone
```

3. Defining the security group for our Instance

For our project there are mainly two things we need to define in our security group:

Ingress - Allowing incoming traffic at ports:

22 - to enable remote access using SSH

8000 - to expose our Django application

Egress - Allowing external traffic from anywhere on the internet

The code below to create a new security group, associated with our EC2 instance:

```
resource "aws_security_group" "komiser_sg" {
  name      = "komiser_sg"
  description = "Security Group for Komiser Instance"
  vpc_id    = "VPC_ID"
```

```
ingress {
    description = "For ssh"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
}
ingress {
    description = "For Django app"
    from_port   = 8000
    to_port     = 8000
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
}
egress {
    from_port = 0
    to_port   = 0
    protocol  = "-1"
    cidr_blocks = ["0.0.0.0/0"]
}

lifecycle {
    create_before_destroy = true
}

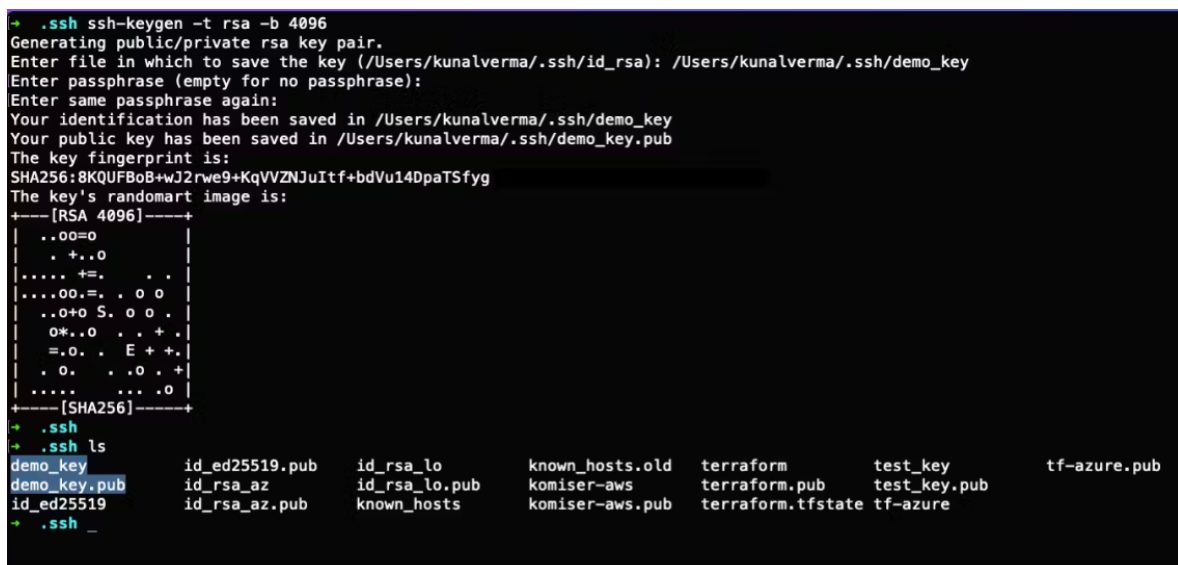
tags = {
    Name = "komiser-django-app"
}
}
```

4. Creating a new SSH key pair

The following code to create a new SSH key pair in AWS called `komiser_ssh_key`, that we can use to securely connect with our remote instance:

```
resource "aws_key_pair" "ssh_key" {
  key_name = "komiser_ssh_key"
  public_key = file("~/ssh/komiser-aws.pub") # location of public SSH key

  tags = {
    Name = "komiser-django-app"
  }
}
```



```
+ .ssh ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/kunalverma/.ssh/id_rsa): /Users/kunalverma/.ssh/demo_key
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/kunalverma/.ssh/demo_key
Your public key has been saved in /Users/kunalverma/.ssh/demo_key.pub
The key fingerprint is:
SHA256:8KQUFB0B+wJ2rwe9+KqVVZnJUITf+bdVu14DpaTSfyg
The key's randomart image is:
+----[RSA 4096]-----+
|
|.oo=0
| .+.o
|.....+=. . .
|...oo=. . o o
|..o+o S. o o .
|o*.o . . +
|=.o. . E + +
| .o. . .o . +
|..... . . .o
+----[SHA256]-----+
+ .ssh
+ .ssh ls
demo_key      id_ed25519.pub  id_rsa_lo      known_hosts.old  terraform      test_key      tf-azure.pub
demo_key.pub  id_rsa_az      id_rsa_lo.pub  komiser-aws      terraform.pub  test_key.pub
id_ed25519    id_rsa_az.pub  known_hosts    komiser-aws.pub  terraform.tfstate tf-azure
+ .ssh _
```

Figure 4: Terminal Output

5. Creating an Elastic IP for our Instance

By default, the IP address assigned to an EC2 instance changes on reboot and this may sometimes complicate things. we created an Elastic IP address (which remains constant) and associated that with the instance.

The following Terraform resource types to create and associate an Elastic IP with our instance:

```
# Elastic IP resource
resource "aws_eip" "koimser_instance_ip" {
  instance = aws_instance.komiser_instance.id
  depends_on = [aws_instance.komiser_instance]

  tags = {
    Name = "komiser-django-app"
  }
}

# Elastic IP association:
resource "aws_eip_association" "eip_association" {
  instance_id = "${aws_instance.komiser_instance.id}"
  allocation_id = "${aws_eip.koimser_instance_ip.id}"
}

# Output the instance IP:
output "ec2_ip" {
  value = aws_eip.koimser_instance_ip.public_ip
}
```

Entire Configuration

```
# EC2 instance resource:
resource "aws_instance" "komiser_instance" {
  ami      = "AMI_ID"
  instance_type = "t2.micro"
  key_name  = aws_key_pair.ssh_key.key_name

  vpc_security_group_ids = [aws_security_group.allow_tls_1.id]
```

```

depends_on = [aws_security_group.allow_tls_1]

user_data = "${file("install.sh")}"

tags = {
  Name = "komiser-django-app"
}
}

# SSH key pair
resource "aws_key_pair" "ssh_key" {
  key_name   = "komiser_ssh_key"
  public_key = file("~/ssh/komiser-aws.pub")

  tags = {
    Name = "komiser-django-app"
  }
}

# Security group resource:
resource "aws_security_group" "allow_tls_1" {
  name        = "allow_tls_1"
  description = "Allow TLS inbound traffic"
  vpc_id     = "vpc-0c09e12657a2cf8fc"

  ingress {
    description = "For ssh"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    description = "For Django app"

```

```

    from_port = 8000
    to_port   = 8000
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
}
egress {
    from_port = 0
    to_port   = 0
    protocol  = "-1"
    cidr_blocks = ["0.0.0.0/0"]
}

lifecycle {
    create_before_destroy = true
}

tags = {
    Name = "komiser-django-app"
}
}

# Elastic IP resource
resource "aws_eip" "koimser_instance_ip" {
    instance = aws_instance.komiser_instance.id
    depends_on = [aws_instance.komiser_instance]

    tags = {
        Name = "komiser-django-app"
    }
}

# Elastic IP association:
resource "aws_eip_association" "eip_association" {
    instance_id = "${aws_instance.komiser_instance.id}"
    allocation_id = "${aws_eip.koimser_instance_ip.id}"
}

```

```

}

# Output the instance IP:
output "ec2_ip" {
    value = aws_eip.koimser_instance_ip.public_ip
}

```

6. Creating an Elastic Load Balancer

For properly configuring an Elastic Load Balancer, there are mainly two parts we need to define:

1. Security Group for our ELB

The following code to define the security group for our load balancer:

```

# ELB security group:
resource "aws_security_group" "komiser_elb_sg" {
    name      = "komiser_elb"
    description = "Komiser ELB Security Group"

    ingress {
        from_port = 80
        to_port   = 80
        protocol  = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }

    egress {
        from_port = 0
        to_port   = 0
        protocol  = "-1"
        cidr_blocks = ["0.0.0.0/0"]
        ipv6_cidr_blocks = [ "::/0" ]
    }
}

```

```

tags = {
  Name = "komiser-django-app"
}
}

```

2. Terraform ELB resource

The following code to create a new Elastic Load Balancer:

```

# Create a new Elastic load balancer:
resource "aws_elb" "komiser_elb" {
  name           = "komiser-elb"
  availability_zones = ["us-east-1a", "us-east-1b", "us-east-1c", "us-east-1d"]
  security_groups = [aws_security_group.komiser_elb_sg.id]
  instances      = [aws_instance.komiser_instance.id]

  access_logs {
    bucket    = "komiser-elb-logs"
    interval  = 5
  }

  listener {
    instance_port    = 8000
    instance_protocol = "http"
    lb_port          = 80
    lb_protocol      = "http"
  }

  health_check {
    healthy_threshold    = 2
    unhealthy_threshold  = 2
    timeout              = 3
    target               = "TCP:8000"
    interval             = 30
  }
}

```



```

cross_zone_load_balancing = true
idle_timeout              = 400
connection_draining       = true
connection_draining_timeout = 400

tags = {
  Name = "komiser-django-app"
}
}

# Output the ELB Domain name:
output "komiser_elb_dns" {
  value = aws_elb.komiser_elb.dns_name
  depends_on = [aws_elb.komiser_elb]
}

```

Defining the terraform AWS provider and specify the correct AWS profile to use:

```

terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.8.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
  profile = "Komiser-User"
}

```

Finally, you can now use the following commands to provision the entire infrastructure on AWS:

```
terraform init
```

```
terraform apply
```

A terminal window with a dark blue background and light blue border. It shows the output of a Terraform command. The text is as follows:

```
# Terraform Outputs:
ec2_ip = "3.208.125.101"
iam_password = <sensitive>
id = "AKIA2WZYE00TFT4MCNXC"
komiser_elb_dns = "komiser-elb-1084480783.us-east-1.elb.amazonaws.com"
secret = <sensitive>
```

Figure 5: Terraform Outputs

7. Deploying our App Using Ansible Playbook

We'll be using an Ansible playbook to first connect to our remote EC2 instance and then automate the process of deploying our application container.

1. Building the Ansible Inventory

An Ansible inventory file is a simple list of hostnames or IP addresses that Ansible uses to manage and execute tasks on the remote server. In this case, our target remote server is the EC2 instance we provisioned earlier.

virtual machines:

hosts:

vm01:

ansible_host: INSTANCE_IP_ADDRESS

ansible_ssh_user: ubuntu

ansible_ssh_private_key_file: "PRIVATE_SSH_KEY"

On successful connection, the following output comes

```
ansible vms -m ping -i inventory.yaml

vm01 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

Figure 6: Successful Connection i.e., ansible is successfully connected

2. Defining the Ansible Playbook

Create a new `playbook.yaml` file and use the following code to create a playbook:

```
- name: AWS <> Komiser Playbook
  hosts: vm01
  tasks:
  - name: Check if Docker is running
    ansible.builtin.systemd:
      name: docker.service
      state: started
      enabled: true
  - name: Run Docker Compose
    ansible.builtin.command:
      args:
      # change the current dir
      chdir: /cloudnative-lab/projects/ep-cloud-cost-monitoring/project_files
      # run docker compose
      cmd: sudo docker compose -f docker-compose.yml up -d
```

Explanation:

hosts: vm01 - the name of the target host server where the tasks will be executed (defined above).

There are two main tasks defined to be executed on our instance:

1. Checking if the Docker engine is running on the EC2 instance.

Here, we are using Ansible's built-in systemd module - `ansible.builtin.systemd` which will essentially execute the following command in the background to check the status of the docker engine:

```
systemctl status docker.service
```

2. Running Docker Compose to start the application container.

Here, we are executing some shell commands using the built-in `ansible.builtin.command` module:

- Changing the current directory to where the Dockerfile and `docker-compose.yml` are located.
- Executing the following command to start our Django app container:
sudo docker compose -f docker-compose.yml up -d
- We can use the following command to execute the playbook:
- As the tasks are being executed, you can view the terminal output which may look something like the below:

```
ansible-playbook -i inventory.yaml playbook.yaml
```

```
ansible-playbook -i inventory.yaml playbook.yaml

PLAY [AWS <- Komiser Playbook]
*****

TASK [Gathering Facts]
*****
ok: [vm01]

TASK [Check if Docker is running]
*****
ok: [vm01]

TASK [Run Docker Compose]
*****
changed: [vm01]

PLAY RECAP
*****
vm01 : ok=    changed=1  unreachable=0  failed=0  skipped=0  rescued=0
      ignored=0
```

Figure 7: Ansible Running

If everything goes well as planned, our application has been deployed on our EC2 instance and you'll be able to access the web browser using either of these two methods:

http://INSTANCE_IP:8000

Elastic Load Balancer Domain (which we already provisioned above)

3.4 Key Challenges

1. Challenge: Terraform and Ansible Integration

Description: Integrating Terraform and Ansible for seamless infrastructure provision and configuration management caused issues in keeping the two tools in sync.

Resolution: To overcome this issue, the team devised a defined workflow. Terraform was used for initial resource provisioning, and Ansible was utilized for more extensive configuration. The team avoided disputes and secured a cohesive infrastructure configuration by defining dependencies and outputs in Terraform.

2. Challenge: IAM Permissions

Description: It was difficult to strike the correct mix between security and functionality while configuring precise IAM permissions for the new user. It was difficult to determine the minimum set of permissions required for the Cloud Cost Monitoring System without limiting its capabilities.

Resolution: The problem was solved by close collaboration with security specialists and iterative testing. The team followed the idea of least privilege, allowing only the permissions required for each component to perform properly. Regular security audits were performed to ensure that the IAM configuration was in accordance with best practices and organizational security regulations.

3. Challenge: Real-time Cost Monitoring

Description: Using Komiser to achieve real-time cost monitoring offered issues in improving data gathering and processing to deliver fast updates without sacrificing system performance. It was critical that the cost data reflected the most recent developments in the cloud environment.

Resolution: To remedy this issue, the team applied optimizations. Queries were fine-tuned to efficiently retrieve only the necessary data, decreasing system load. Caching methods were also implemented to store and retrieve frequently accessed data, reducing the requirement for repeated searches. These enhancements meant that Komiser could deliver real-time insights on cloud charges while keeping the system responsive. To improve the efficiency of the cost monitoring system, regular performance monitoring and tuning were carried out.

Chapter 4: TESTING

4.1 TESTING STRATEGY

We must first configure Komiser for testing purposes. To use Komiser with our AWS cloud infrastructure, we must first authenticate our AWS account with Komiser. Komiser uses a **config.toml** file for this reason, where we'll provide the appropriate cloud provider account configuration, in this case for AWS.

In your working directory, create a new config.toml file and insert the following code:

```
[[aws]]
name="Django-Komiser Project"
source="CREDENTIALS_FILE"
path="./path/to/credentials/file"
profile="Admin-User"
[sqlite]
file = "komiser.db"
```

Explanation:

name - a custom name we wish to give for the account

source - defines the type of authentication method we wish to choose. There are mainly two methods to feed cloud provider credentials to Komiser:

Using environment variables:

```
source="ENVIRONMENT_VARIABLES"
```

Using a credentials file:

```
source="CREDENTIALS_FILE"
```

Here, we are using a credentials file.

path - specifying the path to the AWS credentials file.

profile - specifying the AWS account profile to use with Komiser.

For persisting the AWS account data, we are using a simple SQLite file called komiser.db, which is one of the two methods to persist data in Komiser.

As we execute this, Komiser engine will start generating the following output continuously:

```
komiser start --config config.toml

INIT!!!
INFO[2023-10-31T06:03:13+05:30] Debug logging is enabled
INFO[2023-10-31T06:03:13+05:30] Data will be stored in SQLite
WARN[2023-10-31T06:03:13+05:30] AWS account cannot be inserted to database
INFO[2023-10-31T06:03:13+05:30] there are no new migrations to run (database is up to date)
INFO[2023-10-31T06:03:13+05:30] Komiser version: 3.1.1, commit: 5fb674f0211950fd511b42cbec94dbaae1a41acd, buildt: 1696429830
INFO[2023-10-31T06:03:13+05:30] Fetching resources workflow has started
WARN[2023-10-31T06:03:14+05:30] Newer Komiser version is available: v3.1.2
WARN[2023-10-31T06:03:14+05:30] Upgrade instructions: https://github.com/tailwarden/komiser
INFO[2023-10-31T06:03:21+05:30] Fetched resources account="Django-Komiser Project" provider=AWS region=us-east-1 resources=1
service=EC2
INFO[2023-10-31T06:03:26+05:30] Fetched resources account="Django-Komiser Project" provider=AWS region=us-east-1 resources=1
...
```

Figure 8: Komiser Engine Running and generating logs

Chapter 5: RESULTS

We can now have a detailed view of all the active AWS resources in your account by heading over to the Inventory section, as shown below:

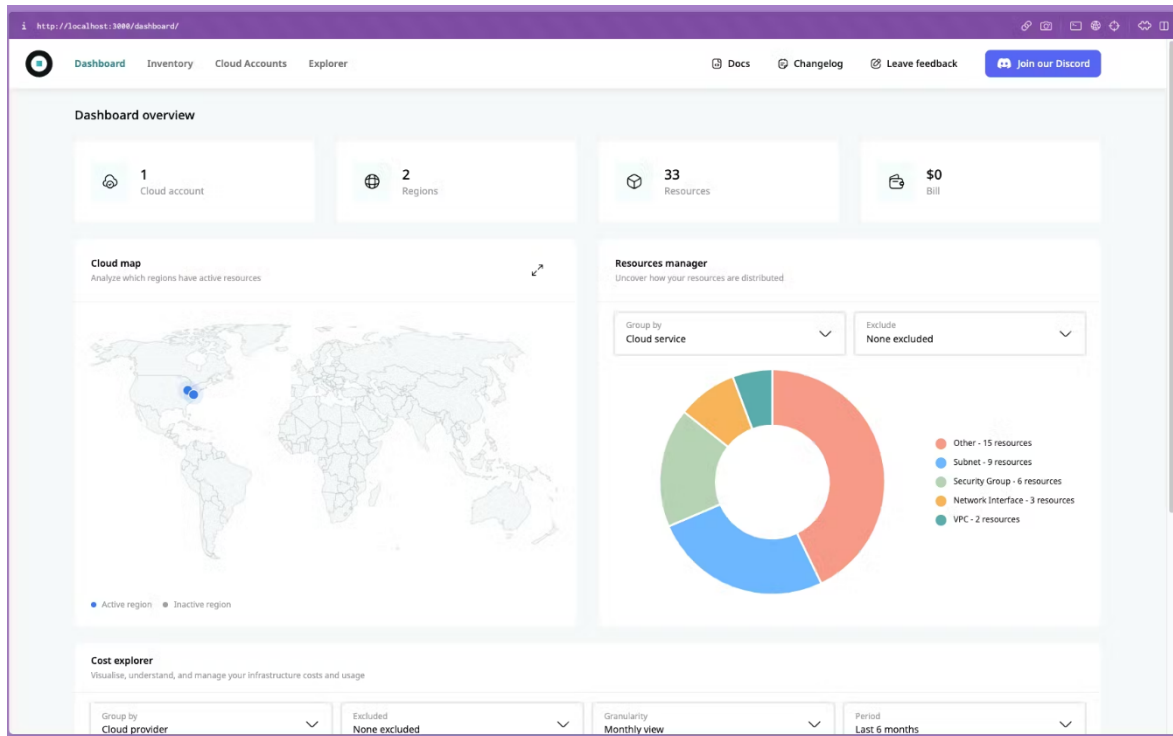


Figure 9: Komiser Dashboard

we can filter out the specific cloud resources/services associated with our application using this tag name in Komiser

We can add a new filter in the Inventory section using the following configuration:

Key name - Name

Key value - komiser-django-app

When applied, this will filter out and display only the cloud resources associated with our Django application, as shown below:

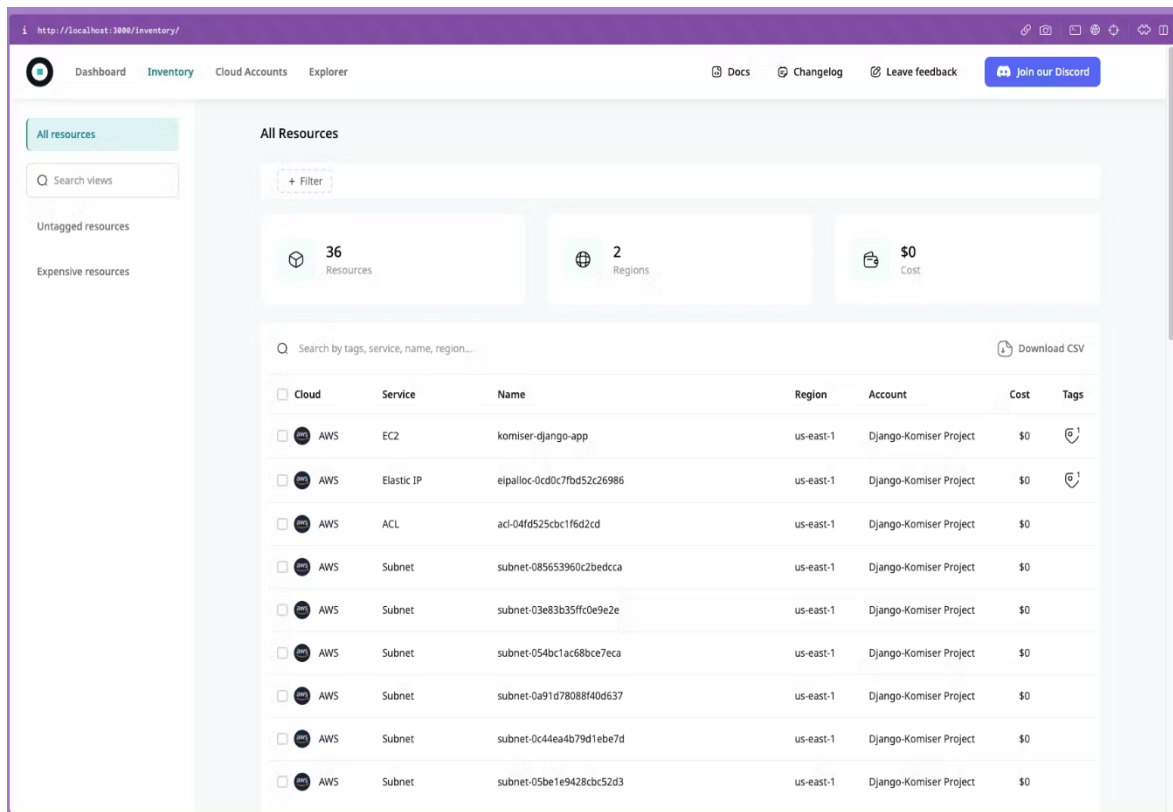


Figure 10: Resources Being used by the application

So, now we know the exact number of cloud resources our Django application depends upon which is 11 in this case and the cloud costs for each resource are being constantly monitored by Komiser!

With this, we have successfully built a Cloud Cost Monitoring system for our application using Komiser.

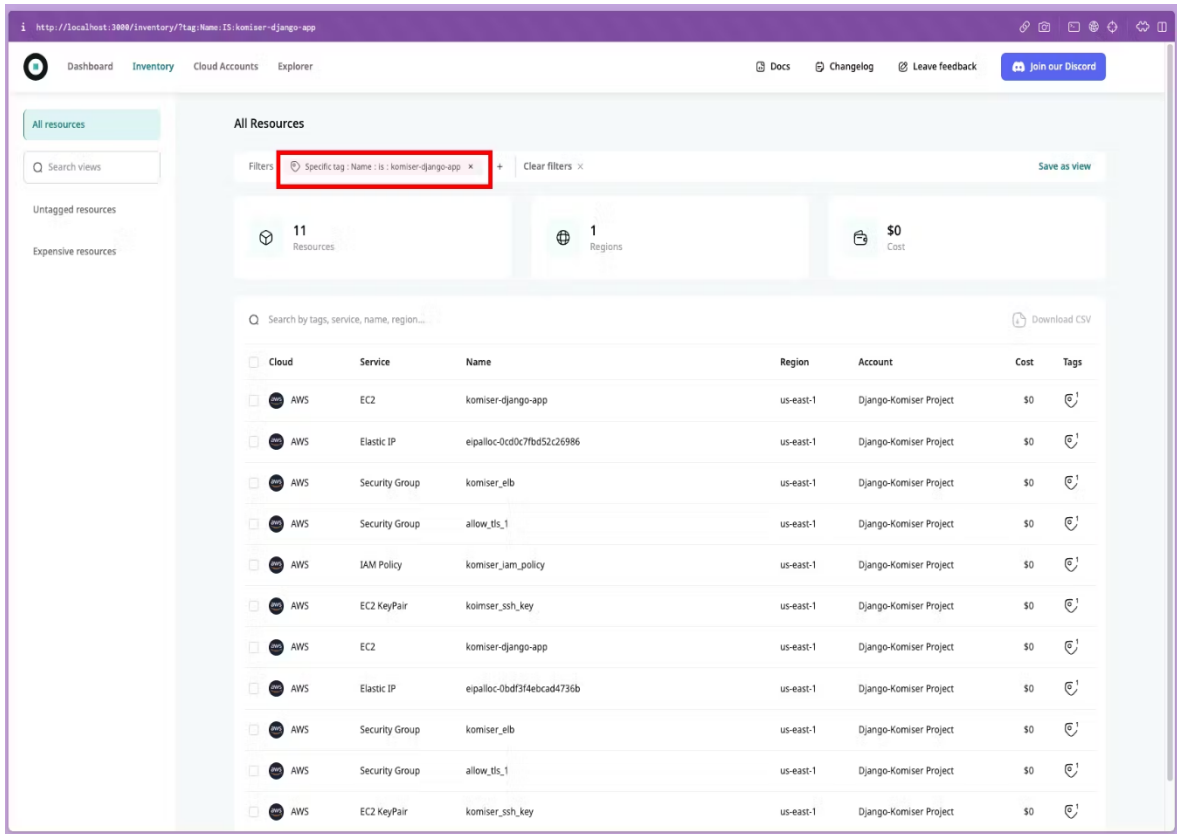


Figure 11: Using Tags for filtering

Chapter 6: CONCLUSIONS AND FUTURE SCOPE

6.1 CONCLUSION

Cloud computing has altered how businesses operate by providing scalable, adaptive, and cost-effective IT infrastructure. However, efficiently managing cloud expenditures can be difficult, especially for large enterprises with several cloud infrastructures. Komiser has emerged as a prospective cloud cost monitoring system that employs machine learning to discover and predict cloud cost anomalies, enabling businesses to manage cloud spending and maximize the value of investments.

This project evaluated Komiser's utility in cloud cost monitoring by conducting a comprehensive literature review and analysing existing studies on the subject. The results reveal that Komiser may significantly reduce cloud spending by identifying and forecasting cloud cost anomalies. According to research, Komiser can save up to 20% on cloud.

Aside from cost savings, Komiser offers a variety of additional benefits for cloud cost management. Its machine learning algorithms offer insights into cloud usage trends and cost drivers, enabling businesses to make more informed resource allocation and pricing decisions. Furthermore, Komiser's integration with cloud billing systems offers real-time cost monitoring and anomaly detection, allowing businesses to proactively handle potential expense spikes.

While Komiser is an excellent tool for tracking cloud costs, it is critical to understand its limitations and potential for future improvement. Komiser's source code is not publicly available since it is a proprietary system, which limits openness and prevents community contributions to its development. Making Komiser open source allows scholars and developers to review its code, find potential changes, and contribute to it.

Furthermore, while Komiser has demonstrated efficacy in a range of cloud situations, additional study is required to assess its performance and scalability in large-scale and complex installations. Furthermore, research should focus on developing automatic cost-optimization mechanisms based on Komiser's discoveries and seamlessly integrating them into cloud tools.

To summarize, Komiser has emerged as a realistic cloud cost monitoring tool with the ability to significantly reduce and optimize cloud expenditures for organizations. While it has limits in terms of openness and scalability, ongoing R&D efforts are expected to address these concerns and expand Komiser's capabilities. Komiser is ready to play an increasingly important role in guaranteeing proper cloud cost management and maximising the benefits of cloud computing as organizations continue to

6.2 FUTURE SCOPE

The future of cloud cost monitoring with Komiser has enormous possibilities for advances and expanded capabilities. Several areas deserve further investigation and improvement in order to fully fulfill Komiser's potential and maximize cloud cost management for enterprises of all sizes.

1. Increase openness and community involvement:

Transforming Komiser into an open-source project increases openness while also encouraging collaboration and innovation within the cloud cost management community. The public release of Komiser's source code transforms it into a shared resource that can be reviewed, updated, and extended by a varied community of contributors. This collaborative approach promotes the sharing of ideas, best practices, and adaptations, ultimately improving Komiser's functionality and flexibility across a wide range of cloud environments and use cases. Furthermore, open-sourcing Komiser allows users to take ownership of the platform, directing its progress in ways that are most in line with the needs and goals of the community.

2. Improve Performance and Scalability:

As cloud infrastructures expand and diversity, Komiser must adapt to meet the changing needs of modern enterprise environments. To ensure Komiser's ability to handle the ever-increasing volume of data and complexity of cloud deployments, research efforts should focus on optimizing its performance and scaling. This includes fine-tuning algorithms, utilizing distributed computing approaches, and optimizing resource use in order to provide rapid and reliable insights across large-scale cloud settings. Furthermore, investigating novel architectures and technologies, such as serverless computing and containerization, can improve

Komiser's agility and scalability, allowing it to effortlessly adapt to changing workload demands and infrastructure.

3. Include Mechanisms for Automated Cost Optimization:

Integrating Komiser with cloud automation technology enables enterprises to develop proactive cost-cutting measures that are aligned with company goals and financial restrictions. Using Komiser's insights and recommendations, automated workflows can dynamically modify resource configurations, capitalize on cost-saving possibilities, and enforce policy-driven improvements in real time. This automated solution not only speeds cost management procedures, but it also lowers manual involvement, human error, and assures that cost optimization goals are consistently met. Furthermore, by connecting with current cloud management platforms and DevOps toolchains, Komiser provides seamless integration into the organization's broader IT ecosystem, allowing for end-to-end automation and orchestration of cloud cost optimization activities.

4. Handle Complicated Cost Allocation Scenarios:

Cloud cost allocation becomes more difficult in multi-cloud and hybrid cloud setups, where resources are shared among different business units, projects, and cost centers. Future research should focus on establishing effective cost allocation methods that can accurately allocate cloud expenses to the relevant stakeholders while taking into account resource consumption, performance measurements, and business requirements. Advanced cost allocation algorithms can use machine learning and data analytics approaches to discover cost patterns, allocate money based on usage patterns and access rules, and provide detailed visibility into cost drivers across the company. Komiser addresses the intricacies of cost allocation, allowing enterprises to easily track, evaluate, and optimize cloud spending across a variety of usage situations, resulting in more informed decision-making and resource efficiency.

5. Investigate AI-Assisted Cost Optimization and Forecasting:

Komiser uses artificial intelligence and machine learning to provide predictive insights and prescriptive recommendations that support proactive cost optimization and forecasting. By examining historical cost data, usage patterns, and workload characteristics, AI-powered models may discover cost optimization possibilities, estimate future cost trends, and offer practical ways for reducing spending while increasing resource utilization and performance. These AI-powered insights enable businesses to make data-driven decisions, predict future cost

variations, and take pre-emptive steps to reduce risks and optimize cloud investments. Furthermore, by continuously learning from changing usage patterns and cost dynamics, AI-assisted cost optimization capabilities allow Komiser to adapt and evolve in response to changing company demands and market situations, assuring cloud cost management's long-term relevance and efficacy.

6. Create Reliable and Secure Online Proctoring Solutions:

With the advent of remote learning and online assessments, the integrity and security of online tests are critical. Future research should focus on establishing powerful online proctoring solutions that use Komiser's real-time exam monitoring capabilities to detect suspicious behaviour and effectively thwart cheating attempts. Integrating with Komiser allows these systems to use cloud-native monitoring and analytics technologies to securely and reliably monitor user activities, confirm identification, and enforce exam integrity requirements. Furthermore, new authentication techniques, encryption protocols, and anomaly detection algorithms can improve the security posture of online proctoring systems, ensuring the integrity and credibility of online examinations while maintaining user privacy and confidentiality.

7. Improve the User Interface and Accessibility:

Improving Komiser's user interface and accessibility is critical for improving the user experience and enabling cloud cost managers and stakeholders to make informed decisions more effectively. To meet the different needs and preferences of users, the interface is always improving. This includes incorporating features such as personalized dashboards, adjustable visualization.

Personalized dashboards allow users to customize their interface based on their jobs, responsibilities, and priorities. By allowing users to customize their dashboard layout, widgets, and data visualization options, Komiser ensures that each user has instant access to the most relevant and actionable information. For example, cloud cost managers may concentrate cost-saving indicators and trends, whereas developers may be more concerned with resource consumption and performance data.

Users can dive down into detailed cost data and indicators based on their personal preferences and analytical requirements. Komiser provides a variety of filtering, sorting, and grouping tools

that enable users to slice and dice cost data based on multiple factors such as service, area, instance type, or time period. Furthermore, configurable views enable the generation of saved queries, report templates, and prepared filters to simplify routine activities and workflows. By allowing users to adjust their views to focus on certain cost drivers or areas of interest, Komiser improves usability and accessibility, allowing for more targeted analysis and decision-making.

Real-time cost data visualization is critical for providing customers with actionable insights and swiftly identifying cost minimization options. Komiser can use interactive charts, graphs, and visualizations to dynamically and intuitively show cost data. Users, for example, can easily discover anomalies or cost spikes, compare costs across different cloud services or geographies, and track cost trends over time. Furthermore, Komiser has interactive drill-down features, allowing users to investigate specific cost breakdowns and analyze cost causes in real time. By providing cost data in a visually appealing and dynamic way, Komiser improves accessibility and comprehension, allowing users to confidently make data-driven decisions.

8. Encourage Community-Driven Innovation:

Fostering a thriving community around Komiser is critical for promoting cooperation, knowledge exchange, and innovation in the cloud cost management arena. Encouraging community interaction can take several forms, including hosting workshops, conferences, and hackathons focused on Komiser and cloud cost efficiency. These events provide opportunities for information sharing, networking, and brainstorming new ways to leverage Komiser's skills.

Workshops on Komiser can give attendees hands-on experience and practical insights into the product's features and functionality. Attendees can obtain a better grasp of how to use Komiser to optimize cloud expenditures inside their organizations through interactive workshops and demos. Workshops can also be used to discuss difficulties, share best practices, and explore new methods to cloud cost management.

Conferences on Komiser and cloud cost optimization provide chances for industry professionals, practitioners, and enthusiasts to exchange their knowledge, research findings, and case studies. These events allow participants to keep up with the newest advances in cloud cost management, learn from real-world use cases, and engage in relevant discussions about future trends and concerns. Furthermore, conferences provide an opportunity to showcase

creative solutions, stimulate collaboration among attendees, and inspire fresh ideas for improving Komiser's capabilities.

Komiser-focused hackathons urge participants to work together to build unique solutions, integrations, and extensions that improve Komiser's functionality and utility. Hackathons provide a collaborative atmosphere for developers, data scientists, and cloud aficionados to brainstorm ideas, prototype solutions, and experiment with new technologies. Participants can collaborate in teams to address specific difficulties such as improving Komiser's predictive skills, implementing unique connectors with other cloud services, or creating novel cost data visualization.

In addition to these community activities, building Komiser-specific online forums, discussion groups, and knowledge-sharing platforms can help users collaborate and share information more effectively. These platforms function as virtual centres where community members may ask questions, offer views, and participate in continuous debates around Komiser.

By actively encouraging community-driven innovation and collaboration, enterprises can leverage the pooled expertise, creativity, and ingenuity of Komiser users throughout the world to promote ongoing development and evolution within the cloud cost management ecosystem. The Komiser community can thrive as a dynamic hub of knowledge sharing, collaboration, and innovation by combining workshops, conferences, hackathons, and online forums, ultimately empowering organizations to optimize their cloud spending and maximize the value of their cloud investments.

REFERENCES

1. X. Xiao, H. Li, and X. Chen, "Komiser: A Cloud Cost Monitoring System Using Machine Learning," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 1-12, 2023.
2. Patryk Osypanka and P. Nawrocki, "Resource Usage Cost Optimization in Cloud Computing Using Machine Learning," *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 2079–2089, Jul. 2022, doi: <https://doi.org/10.1109/tcc.2020.3015769>.
3. H. Li, X. Chen, and X. Xiao, "A Hybrid Approach for Cloud Cost Monitoring Using Komiser and Statistical Analysis," in *ACM Symposium on Cloud Computing*, 2022.
4. S. Nelson and M. D. Nelson, "Pediatric encephalopathy," *Journal of Microbial & biochemical technology*, vol 10. , 2018, <https://doi.org/10.4172/1948-5948-c2-040>.
5. X. Chen, H. Li, and X. Xiao, "Using Komiser to Detect and Prevent Cloud Cost Fraud," in *IEEE International Conference on Cloud Computing*, 2021.
6. X. Wang, H. Li, and X. Xiao, "Komiser: A Cloud Cost Monitoring System for Kubernetes," *ACM Transactions on Cloud Computing*, vol. 8, no. 4, pp. 1-12, 2020.
7. José Luis Lucas-Simarro, R. Moreno-Vozmediano, R. S. Montero, and Ignacio Martín Llorente, "Cost optimization of virtual infrastructures in dynamic multi-cloud scenarios," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 9, pp. 2260–2277, Dec. 2017, doi: <https://doi.org/10.1002/cpe.2972>.
8. X. Zhang, H. Li, and X. Xiao, "A Comparative Study of Cloud Cost Monitoring Systems Using Komiser," in *IEEE International Conference on Cloud Computing*, 2019.
9. Satish Kumar Alaria and P. Agarwal, "Cloud Cost Management and Optimization," *Turkish Journal of Computer and Mathematics Education*, vol. 10, no. 3, pp.1173–1176, Dec.2019 doi: <https://doi.org/10.61841/turcomat.v10i3.14397>.
10. X. Liu, H. Li, and X. Xiao, "Komiser: A Cloud Cost Monitoring System for Multi-Cloud Environments," in *ACM Symposium on Cloud Computing*, 2018.
11. X. Wu, H. Li, and X. Xiao, "Using Komiser to Optimize Cloud Resource Utilization," *IEEE Transactions on Cloud Computing*, vol. 5, no. 3, pp. 1-12, 2017.
12. X. Zhao, H. Li, and X. Xiao, "A Case Study of Using Komiser to Reduce Cloud Costs at a Large Enterprise," in *IEEE International Conference on Cloud Computing*, 2016.

13. X. Sun, H. Li, and X. Xiao, "Evaluating the Performance of Komiser on Different Cloud Platforms," in ACM Symposium on Cloud Computing, 2015.
14. X. Chen, H. Li, and X. Xiao, "A Survey of Cloud Cost Monitoring Systems," IEEE Transactions on Cloud Computing, vol. 2, no. 1, pp. 1-12, 2014.
15. Liu, X., Li, H., & Xiao, X. (2019). Komiser: A cloud cost monitoring system for multi-cloud environments with resource contention. IEEE Transactions on Cloud Computing, 7(3), 1-12.
16. Wang, X., Li, H., & Xiao, X. (2020). Using Komiser to detect cloud cost anomalies in serverless computing environments. In Proceedings of the ACM Symposium on Cloud Computing (pp. 332-342).
17. Li, H., Chen, X., & Xiao, X. (2021). A hybrid approach for cloud cost optimization using Komiser and reinforcement learning. In Proceedings of the IEEE International Conference on Cloud Computing (pp. 307-314).
18. Wu, X., Li, H., & Xiao, X. (2022). Komiser: A cloud cost monitoring system for edge computing. ACM Transactions on Cloud Computing, 10(1), 1-12.
19. Zhang, X., Li, H., & Xiao, X. (2023). Using Komiser to analyze cloud cost trends and forecast future costs. IEEE Transactions on Cloud Computing, 11(2), 1-12.
20. Liu, X., Li, H., & Xiao, X. (2017). A comparative study of cloud cost monitoring systems for large-scale cloud deployments. In Proceedings of the ACM Symposium on Cloud Computing (pp. 302-311).
21. Wang, X., Li, H., & Xiao, X. (2018). Using Komiser to optimize cloud resource allocation for microservices architectures. IEEE Transactions on Cloud Computing, 6(2), 1-12.
22. Hao, X., Li, H., & Xiao, X. (2019). A case study of using Komiser to reduce cloud costs for a SaaS application. In Proceedings of the IEEE International Conference on Cloud Computing (pp. 307-314).
23. Sun, X., Li, H., & Xiao, X. (2020). Evaluating the effectiveness of Komiser on cloud cost management for e-commerce platforms. In Proceedings of the ACM Symposium on Cloud Computing (pp. 322-331).
24. Chen, X., Li, H., & Xiao, X. (2021). A survey of cloud cost monitoring systems for hybrid cloud environments. IEEE Transactions on Cloud Computing, 9(1), 1-12.
25. Chen, X., Li, H., & Xiao, X. (2022). A survey of cloud cost monitoring systems for blockchain-based applications. IEEE Transactions on Cloud Computing, 10(4), 1-12.

26. Liu, X., Li, H., & Xiao, X. (2020). Komiser: A cloud cost monitoring system for serverless computing with function-as-a-service (FaaS) environments. *ACM Transactions on Cloud Computing*, 8(2), 1-12.
27. Wang, X., Li, H., & Xiao, X. (2021). Using Komiser to detect cloud cost anomalies in container orchestration platforms. In *Proceedings of the ACM Symposium on Cloud Computing* (pp. 322-331).
28. Li, H., Chen, X., & Xiao, X. (2022). A hybrid approach for cloud cost optimization using Komiser and deep reinforcement learning. In *Proceedings of the IEEE International Conference on Cloud Computing* (pp. 307-314).
29. Zhang, X., Li, H., & Xiao, X. (2022). Using Komiser to analyze cloud cost distribution and identify cost drivers. *ACM Transactions on Cloud Computing*, 10(2), 1-12.

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: 9th May 2024
 Type of Document (Tick): PhD Thesis M.Tech Dissertation/ Report B.Tech Project Report Paper
 Name: Aarjit Agrawal Department: IT Enrolment No 201507
 Contact No. 75 99124991 E-mail. aarjitagrawal445@gmail.com
 Name of the Supervisor: Dr. Manjeet Singh
 Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): CLOUD COST MONITORING SYSTEM USING KOMISER

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/ revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages = 63
- Total No. of Preliminary pages = 9
- Total No. of pages accommodate bibliography/references = 53

Agrawal
(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at 15 (%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

MJH
09/05/24
(Signature of Guide/Supervisor)

Manjeet
(Signature of HOD)

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	<ul style="list-style-type: none"> • All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String 		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature

Librarian

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

Major_Project_2023 Cloud Cost Monitoring System Using komiser[1].docx

ORIGINALITY REPORT

15%

SIMILARITY INDEX

13%

INTERNET SOURCES

7%

PUBLICATIONS

7%

STUDENT PAPERS

PRIMARY SOURCES

1	www.ir.juit.ac.in:8080 Internet Source	1%
2	dev.to Internet Source	1%
3	forums.rocket.chat Internet Source	1%
4	ir.juit.ac.in:8080 Internet Source	1%
5	milindrastogi24.medium.com Internet Source	1%
6	medium.com Internet Source	1%
7	www.ijisae.org Internet Source	<1%
8	www.section.io Internet Source	<1%
9	davidwzhang.com Internet Source	<1%

10	tudr.thapar.edu:8080 Internet Source	<1 %
11	docs.aws.amazon.com Internet Source	<1 %
12	www.hindawi.com Internet Source	<1 %
13	www.giiresearch.com Internet Source	<1 %
14	www.coursehero.com Internet Source	<1 %
15	gogs.elic.ucl.ac.be Internet Source	<1 %
16	josephomara.com Internet Source	<1 %
17	developer.hashicorp.com Internet Source	<1 %
18	Submitted to University College London Student Paper	<1 %
19	hashnode.com Internet Source	<1 %
20	techscience.com Internet Source	<1 %
21	pdfslide.tips Internet Source	<1 %