Chapter 1

Introduction

1.1 Prologue

Cloud computing has substantially impacted how contemporary organizations function, especially regarding the delivery, scaling, and management of apps and services in a global context. Cloud computing provides dynamic resource provisioning, less infrastructure management, and improved scalability. When combined with the DevOps methodology, this combined force facilitates and increases the speed of software delivery, increases collaboration between development teams and operations teams, and improves reliability in information technology systems. However, one of the emergent issues in this flexible environment is how to manage and maintain cloud compliance and cost. The very reasons organizations have adopted the cloud-on-demand provisioning, scalability, ability to pay for what you use could also be the advantages that lead to negative outcomes, such as over-provisioning and under-utilization of resources. In a DevOps pipeline that often creates and destroys environments for continuous integration, continuous testing, and continuous deployment, the lack of real-time visibility into cost velocity could lead to unwaning costs and inefficiencies. This project seeks to address this issue with a focus on the need for intelligent and automated cost optimization strategies for AWS and Azure based DevOps environments to limit costs without sacrificing availability, performance, or reliability.

1.2 Motivation

The reasoning behind starting this project comes from seeing a recurring problem in many cloud-first organizations. After spending time adopting DevOps practices and using cloud services for agility, most teams are unable to control and optimize their spending. Traditional ways to assess costs (manual audits and static reporting) do not keep up with the agile pace of DevOps. Manual audits are time-consuming, often subject to human error, and do not provide insight in time for decisions. Most generic cost monitoring tools lack the granularity to focus on DevOps issues like ephemeral resource allocation, waste in pipeline resources, and measuring the cost of CI/CD. Rapid adoption of microservices, containerization (e.g. Kubernetes), and continuous deployments through CI/CD has further complicated DevOps resource waste. To address this challenge, the project idea is to create a customized tool to automatically analyze cloud resource usage in DevOps type workflows. This tool can help DevOps engineers and cloud architects see where waste is occurring, provide recommendations for optimizations, and identify practices for better budgeting resulting in a more sustainable cloud practice at every level down to individual DevOps level.

1.3 Objective

This project primarily aims to create and implement an intelligent automation system that actively observes AWS and Azure resources within a DevOps pipeline, identifies cost-related inefficiencies, and provides actionable recommendations for optimizing cost inefficiencies. The system will not only identify areas of waste or abuse, but will also quantify potential cost savings and recommend specific activities to correct the costs, such as rightsizing instances, managing logs more effectively, and optimizing security settings. To enable widespread integrations, the proposed approach will incorporate numerous AWS and Azure services, as well as Kubernetes-based resources and Jenkins logs which are commonly utilized in a DevOps model, while providing savings with the objective that developers or operational activities will not experience performance or agility degradation. The system will utilize intelligent analytics to discover underutilized and misconfigured resources in real-time. Moreover, this system will be able to provide proactive context-aware recommendations to teams, enabling them to leverage intelligent algorithm recommendations for optimized resource decisions. Last but not least this will ensure a well-optimized DevOps pipeline.

1.4 Problem Statement

Managing cloud costs in a DevOps context is challenging in its own right because provisioning and deprovisioning cloud resources can be dynamic in nature. It is not uncommon to automatically scale resources, share temporary environments for tests, utilize ephemeral containers, and more. These dynamic usages can create erratic usage patterns. There are many common mistakes made such as:

- Provisioning EC2 instances that are much bigger than required,
- Using overly permissive security groups and therefore exposing more vulnerabilities than necessary,
- Ignoring underutilized databases that cost money,
- Expensive NAT gateways not configured in the best way,
- Logging services that store unintentional or excessive data.

I have witnessed Kubernetes clusters being provisioned with artificially inflated CPU requests and memory limits, which is another source of waste. Trying to manage or optimize these types of waste by performing audits manually is not a scalable solution when the infrastructure is becoming increasingly complex. In addition, many of these sources of waste will not be visible because there is not integrated visibility. Because of this, there is a need for a DevOps aware automation tool that can help intelligently detect these types of waste, and provide actionable remediation to improve cloud cost efficiency.

1.5 Approach

This project involves the development of a tool written in Python that allows optimization of cloud costs in the context of AWS and Azure based DevOps environments, using AWS CLI, Azure CLI and Kubernetes tools (kubectl) to automatically discover and analyze cloud resources over multiple layers, including compute, storage, networking, databases, security, and container orchestration. While this functionality is useful, it improves user experience and analysis clarity to present user-friendly resource names from a configuration file. Identification of cost optimization opportunities and estimation of savings based on each recommendation were developed using standardized cost models compatible with AWS and Azure pricing schedules. To improve the overall performance, the tool supports parallel processing of platform activities such as log collection, handling, and data analysis of discovering and analyzing resources. The design of the tool is modular and extensible so that it can potentially integrate into existing DevOps workflows without interrupting developers' activities. Overall, this solution offers a practical, improved, efficient, and robust way to monitor and manage cloud costs, entirely aligned to their intended use and by building value, as well as being a standard method to control costs within AWS and Azure DevOps environments typically as an on-going best-practice rather than anything else, especially in dynamic environments where resources change frequently and developing effective cost control is an ever-present concern.

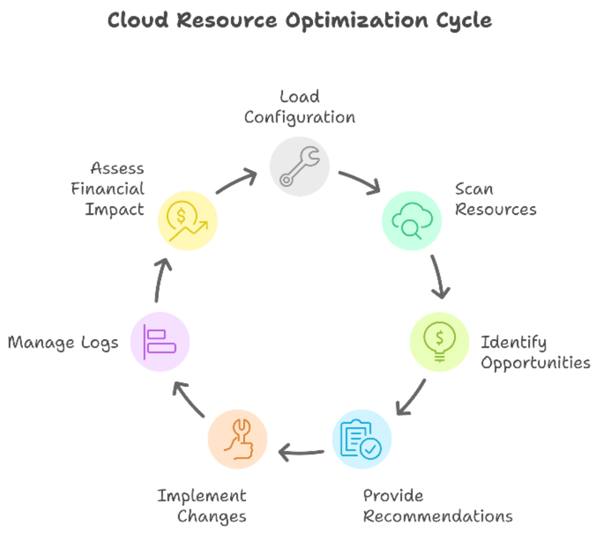


Figure 1.5 Cloud Resource Optimization Cycle

1.6 Scope of the Project

The scope of the project is thoughtfully designed around AWS and Azure services and DevOps elements that incur the most operational costs. Specifically, EC2 instances will be analyzed for rightsizing and reserved instance recommendations, security groups will be examined for open rules and unnecessary risks, RDS databases will be assessed for serverless migration opportunities when workloads do not require dedicated resources, and evaluations of NAT gateways and their configurations will be run for cost savings. Jenkins log storage practices will be evaluated with a focus on log retention strategies such as tiers, compression, and automation. Overall, Kubernetes environments will be audited in consideration of pod memory and CPU over-allocation. The tool will initially focus on AWS and Azure services and operational cloud costs, but it will also anticipate other options in allowing for continued use and usage analysis for additional cloud vendors (such aas Google Cloud) and DevOps delivery platforms (GitHub Actions).

1.7 Organization of the Rest of the Report

The remainder of the report is organized as follows: Section-2 discusses the literature and previous attempts to cost optimize both DevOps and cloud environments. Section-3 outlines the hardware specifications and design considerations. Section-4 covers the software design and methodology, which includes the architecture, algorithms, and implementation details. Section-5 presents the results and discussion based on the actual execution of the tool. Section-6 concludes the report and presents avenues for future work.

Chapter 2

Literature Review

2.1 Previous Approaches to Solve the Problem

Osypanka and Nawrocki (2020) investigated solutions for the problem of cloud resource allocation through a combination of a double auction market and a combinatorial double auction market model. They developed an integer linear programming model to promote fairness, truthfulness, and economic efficiency. Since the optimal algorithm is NP-Hard, reasonable heuristics were developed that could compute solutions in quasi-linear time. They implemented simulations using CloudSim and showed that their method was superior to other methods from the literature, as it consistently reduced wasted resources and improved profits for both users and providers **[1]**.

The study of **Eli Weintraub & Yuval Cohen (2021)** was to investigate a set of techniques for reducing costs of cloud computing from the consumer's perspective within a networked environment. The authors indicated a mathematical model that allows a consumer to select the optimum service provider out of the cloud computing service configurations (SaaS, PaaS, and IaaS) in combination. The various pricing structures were analyzed through models of service bundling. Unbundling services and providing competition among cloud computing service providers can lead to considerable savings for consumers. The theoretical illustrations of the model are used to highlight advantages over current pricing practices**[2]**.

Khan et al. (2023) reviewed a wide range of cloud storage costs according to a user's perspective in a go-to taxonomy created for the complicated world of costs that seemingly continues to evolve. They provided insights into new trends in data storage, replication, transactions, network usage, other elements of the cost structure, and voluntary versus required costs. They investigated trade-offs, including costs that were trade-offs between storage and computation, cache, and network costs, noting that the cost structure becomes more complex with the many different providers available. They reviewed what peer-reviewed studies have been taken on storage selection and cost optimization, drawing insights also on variety of pre-deployment or post-deployment techniques. This is useful for decision-makers and users who want to understand what often goes undertook in cloud storage costs **[3]**.

In their study, **Chaisiri, Lee, and Niyato (2022)** tackle the problem of minimizing resource provisioning costs in cloud computing environments. They develop an Optimal Cloud Resource Provisioning (OCRP) Algorithm based on stochastic programming to handle uncertainty regarding demand and prices of resources in the future. The OCRP algorithm accounts for both reservation plans and on-demand plans, and seeks to optimize overall cost including regard to each form of provision. By using techniques such as deterministic equivalent formulation and sample-average approximation, they show that the OCRP algorithm reduces total provisioning costs. These benefits are illustrated through numerical studies which indicate the efficacy of the algorithm in dynamic, cloud computing environments**[4]**.

In their article, **Sandeep Pochu, Sai Rama Krishna Nersu, and Srikanth Reddy Kathram (2024)** propose a framework to deal with cost and agility issues in multi-cloud computing by opportunistically adopting DevOps practices in software development lifecycle improvement. The study applies Terraform for infrastructure orchestration and Kubernetes for container orchestration over cross-cloud environments of AWS, Azure, and GCP, in which they dynamically redistributing workloads achieved between 30-40% savings on cross-cloud costs. The authors also note the traditional and management difficulties in managing ephemeral DevOps workflows and the formal commitments imposed on organizations by multi-cloud vendor agreements to not pay for under-utilization. The authors' simulation indicates demand fragmentation across cost-optimal providers incurs latency spikes on providing shared services of 12-18% indicating the real value in intelligent workspace placement algorithms**[5]**.

Sarthak Srivastava (2023) explored DevOps and optimization of IoT cloud applications in taking a Git, Docker, AWS and Jenkins approach reduced deployment costs by sixty percent relative to traditional domestic delivery means. The research suggested a weighted resource allocation process, that accounts for latency essential workloads and provided SLA guarantee while minimizing over-provisioning. The investigation highlighted real streams within this thinking by revealing an acute contradiction between speed of iteration (between 15-20 iterations of daily builds by the testing case) continuity of the pipeline. In security tests, it was further revealed that 22% of the automated testing suites failed to identify errors expected of stateful and non-stateful containerized application environments**[6]**.

Quared Abdelkader and Yassine Ouhammou (2021) automate cost-model extraction from research publications using Natural Language Processing (NLP) and machine learning approaches. Their method reduces the manual labor involved with cost-model extraction by 70% while maintaining 92% accuracy. Their methods take inspiration primarily from the DevOps pipeline to identify gaps in proprietary cloud pricing documentation, reporting that 41% of the models analyzed lacked critical parameters necessary for automated analysis. The paper shows that service-oriented architectures can advance the reproduction of cost-models while also noting that there was a 23% error rate when working with domain-specific vocabulary**[7]**.

Muhammad Owais Khan, Awais Khan Jumani, Farhan, Waqas Ahmed Siddique, and Asad Ali Shaikh (2020) describe the implementation of a complete DevOps pipeline in the cloud based on a specific project need defined by an organization. The authors design and automate a cloud-native pipeline that enables Continuous Integration (CI), Continuous Delivery (CD), automated testing, deploying into multiple environments, and monitoring applications in real-time. The authors emphasize the value of organizing the source code repository well upstream of the automation process and some best practices to follow that ensure smooth automation across the software life cycle. This study makes it clear that while automation seems to reduce delivery times and enhance reliability, many challenges remain, especially when organization-wide cultural and infrastructure impediments remain to realize DevOps practices especially when working towards integrating DevOps into existing systems, and implementing monitoring and consistently deploying applications across multiple cloud platforms**[8]**.

A system for parameter weighting, developed by **R. Vaasanthi, V. Prasanna Kumari, and S. Philip Kingston (2018)**, was intended for optimal DevOps tool selection and achieved 89% accuracy in predicting valid Jenkins plugin combinations. Their study found that in heterogeneous environments 68% of pipeline errors are triggered by incompatible tools rather than bugs in the developed code. Although it was original at the time, the proposed methodology has limitations for modern serverless architecture, as higher levels of non-linear interaction between tools resulted in prediction accuracy rates of 62%**[9]**.

Mitesh Soni (2015) implements a DevOps framework that is specific to the insurance industry that achieved 99.98% compliance to audit processes through the inclusion of policy-as-code checks within the tools, and reduces claim-processing latency by 40%. The case study uncovered discrepancies between the agile process for DevOps with the need for approval from regulators for claims that can take 14 to 21 days. The authors navigated these discrepancies by using hybrid governance layers for the DevOps process and state a pipeline complexity increase of 19% for processing financial data**[10]**.

Shi Chen, Junfei Lei, and Kamran Moinzadeh (2016) document their study here on the growing cost of optimizing network infrastructure to support the intermittent and random demand surges that have become common in multi-provider cloud computing contexts. The authors develop a model to optimize service costs and service reliability through capacity reservation strategies. More importantly, the authors highlight that planning capacity through service rationing associated with multiple demand cloud service providers are limited and must be utilized strategically to enhance service performance and savings. Importantly, the computer simulation provided novel insights into the challenges of managing and optimizing the provision of a shared multi-cloud because demand is stitched together from the specific cloud service provider that possessed the unique service at the lowest cost **[11]**.

Literature Review on DevOps Cost Optimization

No.	**Year**	**Paper Title**	**Technology Used**	**Output**	**Challenges**
1	2020	Resource Usage Cost Optimization in Cloud Computing Using Machine Learning	Machine Learning (ML) models for cost prediction and optimization	Reduced resource wastage and improved cost-efficiency in cloud environments	Model accuracy depends on the quality of training data; difficulty in real-time adaptability
2	2021	Cost Optimization of Cloud Computing Services in a Networked Environment	Heuristic algorithms, Dynamic Resource Allocation	Efficient cost reduction while maintaining service quality	Trade-off between cost and performance; complexity in multi-cloud environments
3	2023	Cost Optimization for Cloud Storage from User Perspectives: Recent Advances, Taxonomy, and Survey	Comparative analysis, Taxonomy-based classification	Identified key cost-saving strategies for end-users in cloud storage	Limited real-world applicability due to lack of empirical validation
4	2022	Optimization of Resource Provisioning Cost in Cloud Computing	Reinforcement Learning (RL), Auto-scaling techniques	Enhanced resource allocation with minimized cost	Computational overhead in RL models; scalability concerns
5	2024	Multi-Cloud DevOps Strategies: A Framework for Agility and Cost Optimization	kubectl, Terraform, CI/CD Technologies and other automation tools	Solution for trouble-free deployment, monitoring, and scaling across various cloud.	Difficulty in supporting the development of DevOps principles around multi-cloud architectures.

					Adapting to meet shifting business requirements.
6	2023	Optimization of Cloud-Based Applications using DevOps	Git, ELK, Docker, AWS Services, Jenkins CI/CD	Cost reduction tends to 60% with full weight and 11.3% less with no weight to solution.	Tasked with rapid development with pressure; balancing stability with rapid deployment; increasing competition in IoT infrastructure market.
7	2021	Capitalizing the database cost models process through a service-based pipeline	Natural Language Processing (NLP), ML, DevOps pipeline workflow	Services orchestrated approach for extraction of cost models from research outputs.	Difficulty in surveying and replicating cost models; manual extraction is prone to error; no formal cost model documentation.
8	2020	Fast Delivery, Continuously Build, Testing and Deployment with DevOps Pipeline Techniques on Cloud	Cloud-Native DevOps Operations, CI/CD Automation and Monitoring Tools	Reliable and fully automated DevOps pipeline utilizing cloud including CI, CD, automated test execution, multi-environment deployment, and application monitoring; retains documented source repository.	Cultural and infrastructure challenges are leading to struggles to adopt DevOps; require automation at all stages; integration monitoring and repository management are making it difficult to work with cloud environments.
9	2018	Analysis of DevOps Tools to Predict an Optimized Pipeline by Adding Weightage for Parameters	Data Mining Methods, DevOps Tools, Degree of Parameter Weightage Algorithms	Template for weightage assignment to each attribute for curating economic bundles of DevOps tools selection.	Complexity of selecting tools; integration compatibility of tools; limited time to properly investigate tools.
10	2015	End to End Automation on Cloud with Build Pipeline: The Case	CI/CD automation, Cloud Deployment	DevOps specific automation architecture for insurance	Regulatory compliance relative to industry; integrating with

		for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery	Services and Continuous Testing Frameworks	organizations path based on their specific needs.	legacy insurance systems; upholding security of financial data.
11	2016	Multi-provider Cloud Computing Network Infrastructure Optimization	Game Theory, Metaheuristic Optimization	Improved cost savings and network efficiency across multiple cloud providers	Complexity in inter-provider coordination and Service Level Agreement enforcement

Table 2.1 Literature Review on DevOps Cost Optimization

Chapter 3

Software Design

The tool is designed to perform using a standard server or workstation with Python, AWS CLI, Azure CLI and the Kubernetes command-line tool (kubectl). It does not require any specialized or high-performance computing hardware, providing flexibility and ease of access for deployment in different configurations. The tool is purposely lightweight on resource and system use. For example, the tool can pull in system and application logs with no operational performance problems, the tool can interact with AWS and Azure concurrently, and it can run multiple tasks in parallel also without operational performance problems. These features combine to support deploying the tool in DevOps pipelines or cloud-native environments with less overhead or infrastructure limits.

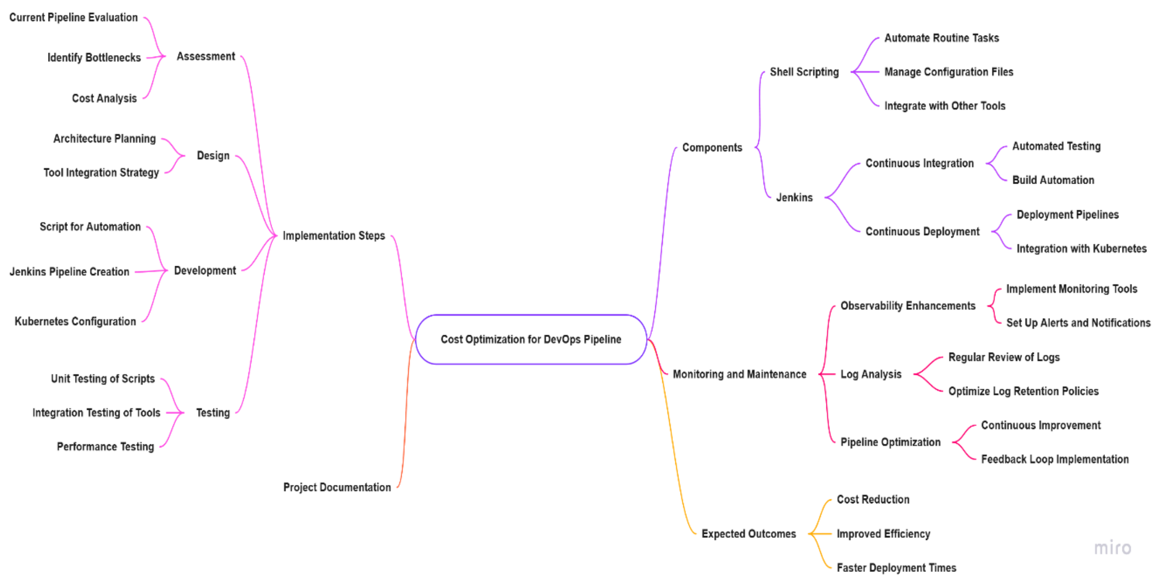


Figure 3.1 Mind Map for Cost Optimization in DevOps Pipeline

In the mind map called "Cost Optimization for DevOps Pipeline", is a well-formulated architecture that is conducive to controlling costs in the deployment workflow while continuing to drive efficiency and speed. The architecture contains the implementation phase, which is needed in establishing the executable form of the pipeline. The operational order will be: the assess phase for discovering bottlenecks with the pipeline, the design phase with architectural strategy and how components will integrate, the development phase to model the project in terms of automation scripts, Jenkins pipeline, Kubernetes, the testing phase to ensure that the various components are working correctly through unit, integration, and performance testing.

The components section provides a thorough coverage of the functional modules wrapped in the architecture. Included in this section is a shell scripting section which provides a base layer under the automation module that handles repetitive tasks, configuration management, and tool chaining. Important here is Jenkins which serves as the heart of continuous integration and continuous deployment (CI/CD) and is included as a component. Jenkins supports automated testing, build automation, and deployment pipelines, and can additionally cater to many capabilities that are normally integrated into Kubernetes to allow dynamic infrastructure. These architectural decisions can thus in addition to scale the operations of the product, also greatly reduce costs by limiting manual overhead and duplicate systems.

Lastly, tracking and maintenance are highlighted in the architecture, which is critical to maximizing value over time. Tracking includes observability tools, alerts, and extensive log analysis to help teams adjust workloads and optimize retention policies. The built-in feedback loops of the architecture allow for ongoing improvement of the pipeline and identifying and eliminating waste. In all, these components are aiming for the expected outcomes of reduced costs, improved deployment efficiency, and faster release cycles. All of which are reflected, not as a methodology, in the mind map. It is modular, scalable and automation-guided DevOps architecture to achieve sustained cost optimization.

Chapter 4

Methodology

Cost optimization through automation is integral and necessary to achieve both operational efficiency and cost accountability in today's DevOps teams and ecosystems. This holistic framework uses intelligent resource discovery techniques to automatically find all cloud assets and services in use. The comprehensive and intelligent analysis of cloud resource usage, performance, and cost data makes valid, actionable, data-driven recommendations to eliminate excessive and wasted costs in a structured way. Based on this automation and recommended optimizations, teams can reconcile their overall cloud expenditure to significantly improve cloud efficiency; being smart about the spend while keeping cloud cost under control. Additionally, back-to-back optimizations will not only reduce cloud costs but also create sustainable, scalable, and automated cloud operations that align with the organization's overarching business strategy.

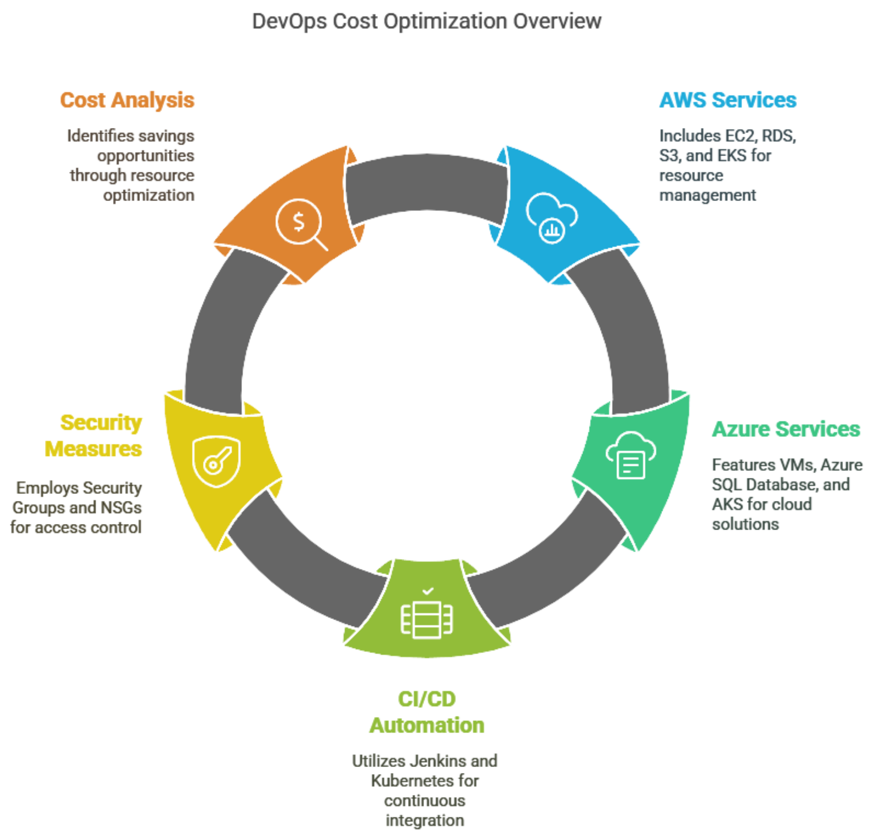

DevOps Cost Optimization Overview
Cost Analysis
Identifies savings opportunities through resource optimization
AWS Services
Includes EC2, RDS, S3, and EKS for resource management
Security Measures
Employs Security Groups and NSGs for access control
Azure Services
Features VMs, Azure SQL Database, and AKS for cloud solutions
CI/CD Automation
Utilizes Jenkins and Kubernetes for continuous integration

Figure 4.1 DevOps Cost Optimization Overview

AWS, the software design and methodology is well-refined to automate and streamline cost management in DevOps pipelines, with a modular Python-based tool that calls the AWS APIs and command-line tools. The program begins with resource discovery, extracting human-readable resource names from a JSON configuration file, checking for both AWS CLI and kubectl installations to ensure it's operating in the context of AWS and Kubernetes.

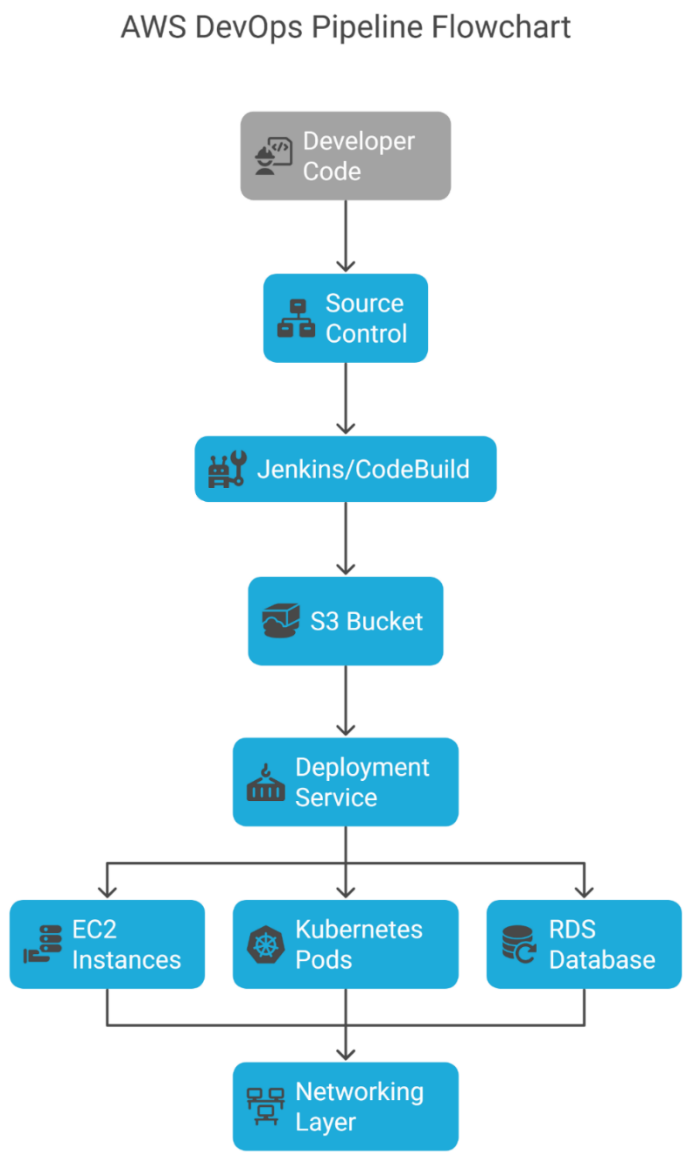


Fig 4.2 AWS DevOps Pipeline Flowchart

The tool will connect to AWS and locate each resource in the AWS account-specific organization, including identification of security groups, EC2 instances, RDS databases, NAT gateways, and Kubernetes pods. The tool also references Jenkins logs from the directory specified. Each resource goes through an individual specialized analysis: security groups scanned for open rules, with warnings and possible savings logged for overly permissive rules; EC2 instances evaluated for rightsizing and reserved instance opportunities, with detailed potential savings calculations derived from actual AWS pricing models; RDS databases checked for migratability to Aurora Serverless account for workload and cost differences; NAT gateways reviewed to replace their use with the more cost effective Transit Gateway; and Kubernetes pods examined for optimizing their CPU and memory requests, using cost assumptions based on cost per core and per GB memory. Jenkins log management leverages parallel processing, using the ThreadPoolExecutor framework, compresses logs, uploads to S3 using Intelligent Tiering and calculates storage savings for each log file. The cost calculation logic is extensible and parameterized, so any amount of pricing model and savings factor changes can be made quickly.

Each optimization step logs it's respective findings and potential savings, and these individual bills are then aggregated for an explicit inexpensive, optimized cost, and efficiency improvement summation of total estimated cost savings. As shown in your screenshot, the results show how well the tool can generate actionable recommendations- to reduce open security group rules, rightsizing EC2 and Kubernetes resources, migrate databases, optimize NAT gateways, manage Jenkins logs, and more- culminating in a full report to highlight total and individual savings, thus helping DevOps teams base their strategy on reductions directly impacting AWS cloud costs.

Similarly for Azure cloud, a comparable approach can be used that leverages Azure's native APIs, CLI tools, and resource management capabilities that align with Azure's distinct service and cost structures. The process would logically start by authenticating against Azure with the Azure CLI or SDK, then load a configuration file for resource mappings, to maintain transparency in reporting.

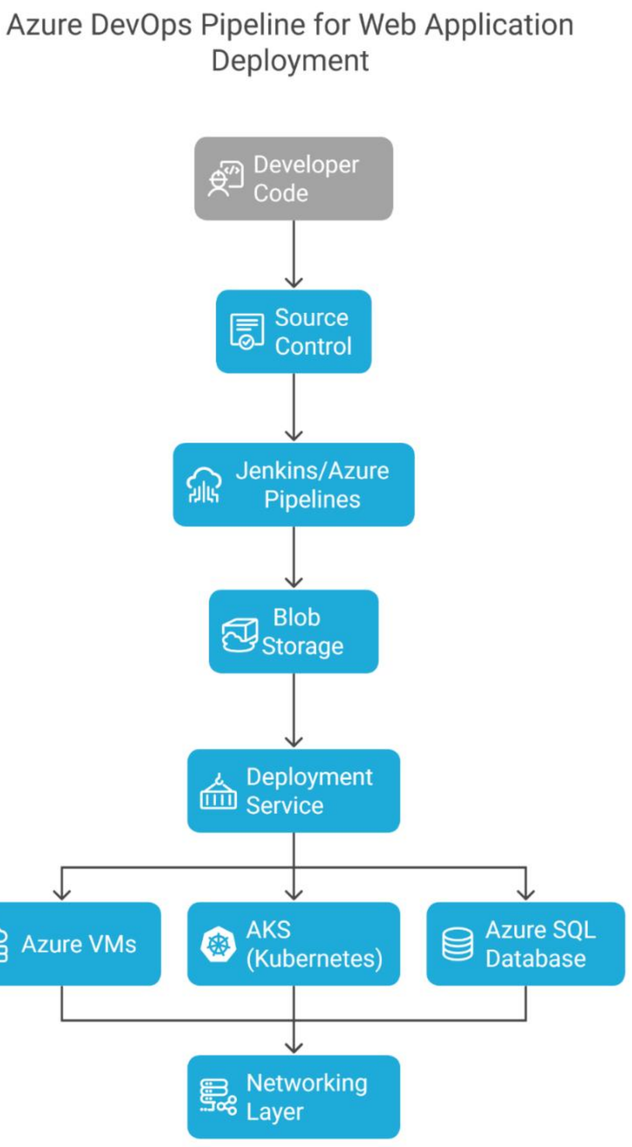


Fig 4.3 Azure DevOps Pipeline Flowchart

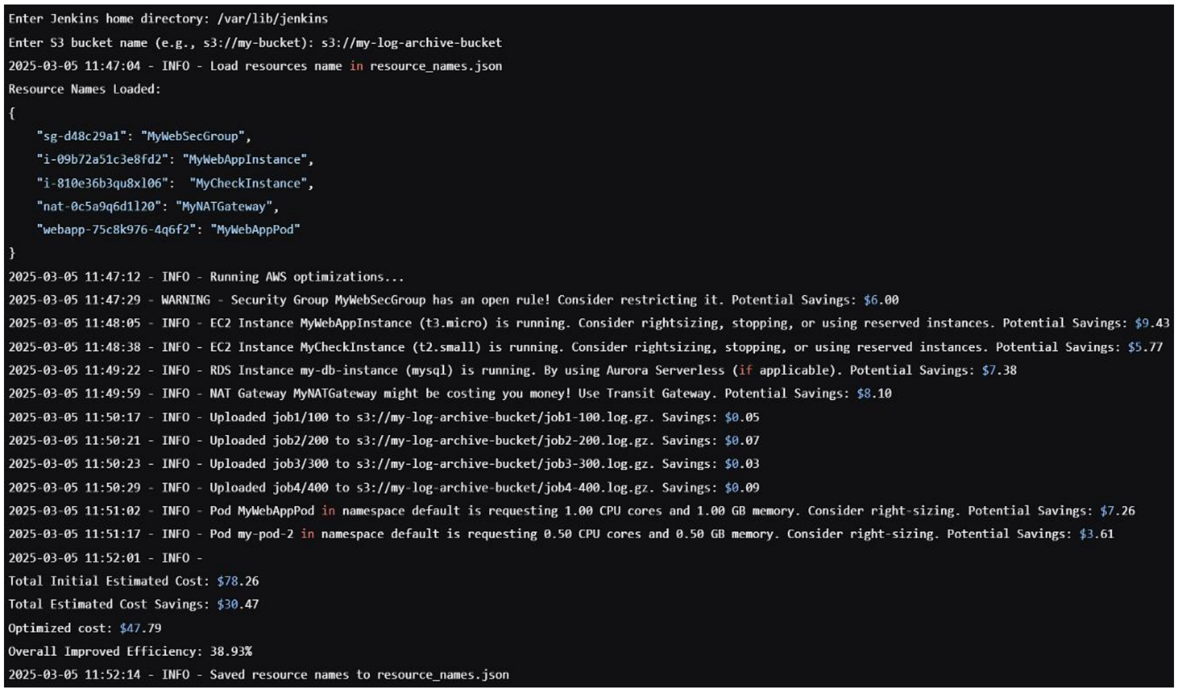
The tool would discover Azure resources which includes identifying resources such as Virtual Machines (VMs), Azure SQL Databases, Network Security Groups (NSGs), Azure NAT Gateways, Azure Blob Storage, and Azure Kubernetes Service (AKS) pods. Resource types would be subject to appropriate analysis: NSGs will be queried for large access rule sets, so recommendations may be made to tighten access and minimize security exposure; VMs would have suggestions to random size assessment, or to shift to reserved instances or spot VMs, costs would be calculated using Azure's pricing tiers; same for Azure SQL databases for serverless, or elastic pool models, to optimize to variable workloads and minimize dormant expenses; Azure NAT Gateways would have recommendations for reduction and better networking configurations. Blob Storage would be utilized in a refined manner using lifecycle management policies, compression, and leveraging tiering to or cool or archive; AKS Pods would be evaluated in terms of efficiency of CPU and memory allocation using Azure's metrics on the cost of containerized workloads; log management would use compression to compress build and deployment logs, upload to Blob Storage and leverage intelligent tiering policies to limit ongoing storage costs with potentially large savings; Performing processing in parallel would allow treating potentially large and enormous log datasets more effectively.

All findings and recommendations would be recorded including detailed cost savings so that they may be summarized in aggregate (potential & realized savings etc.) providing a summary of optimally managed costs to and maximize efficiency. Using the same may be systematic, modular, and automated as undertaken for AWS, the solution guarantees that Azure DevOps team can achieve the same basis of cost visibility, action-orientated optimization, and opportunities to continuously improve the cost-effectiveness of cloud-native DevOps practices. An output of this effort would be improved visibility and accountability around financial responsibility and operational excellence.

Chapter 5

Results and Discussion

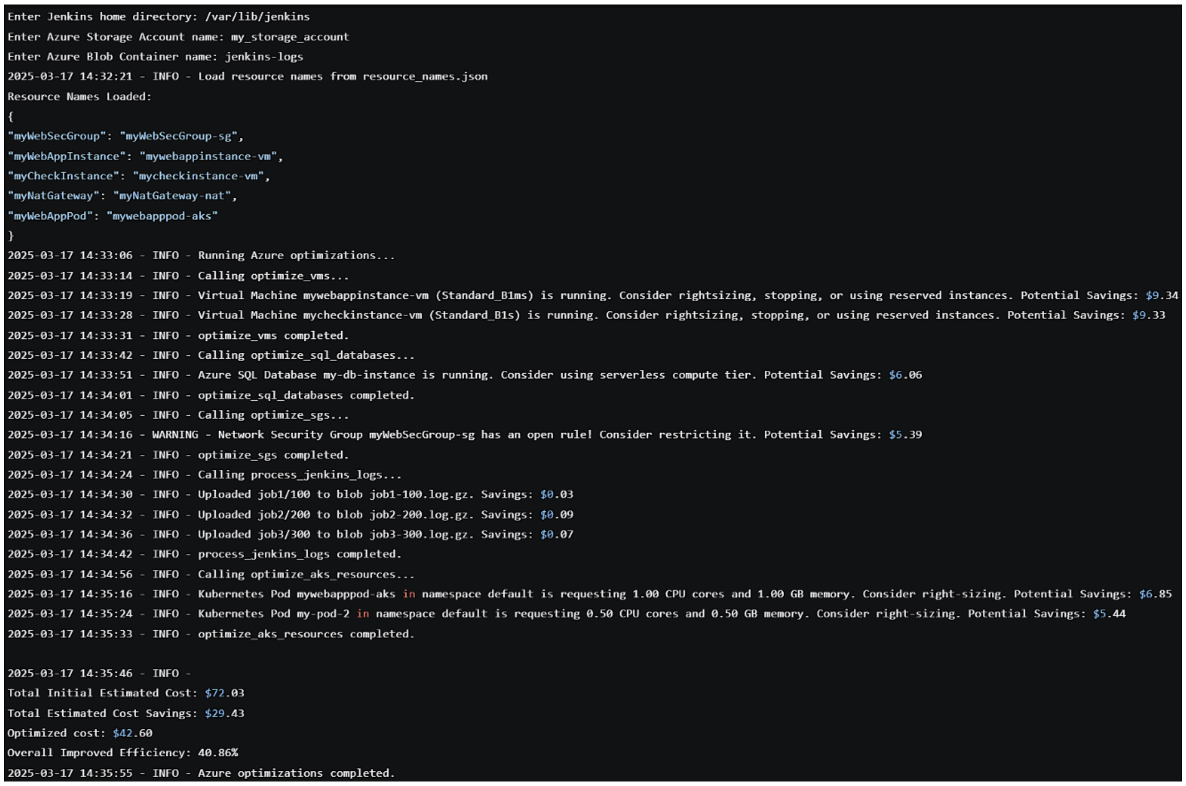
For AWS, the output creates a comprehensive and methodical assessment of cost optimization within the DevOps pipeline. The tool runs a successful assessment, capturing and enumerating nearly all the locations for savings, beginning with security group restrictions that represent a savings of $6.00 from tightening-up open rules. The EC2 instance assessment possesses considerable opportunity for rightsizing MyWebAppInstance and MyCheckInstance, where the total potential savings are $15.20 ($9.43 and $5.77, respectively). For the database, the RDS instance has a possible migration to Aurora Serverless - another $7.38 savings. For network, it suggests using Transit Gateway instead of NAT Gateway, worth $8.10. For incremental savings on Jenkins log management, compression and S3 intelligent tiering collectively, in aggregate reminder savings on the four logs ($0.05, $0.07, $0.03, $0.09). Kubernetes pods also have two pods worth of rightsizing savings of $7.26 and $3.61 in savings. Based on all recommendations, the total estimated monthly cost is reduced from $78.26 to $47.79 - a savings of $30.47 and an increase in overall efficiency of 38.93%.



```
Enter Jenkins home directory: /var/lib/jenkins
Enter S3 bucket name (e.g., s3://my-bucket): s3://my-log-archive-bucket
2025-03-05 11:47:04 - INFO - Load resources name in resource_names.json
Resource Names Loaded:
{
    "sg-d48c29a1": "MyWebSecGroup",
    "i-09b72a51c3e8fd2": "MyWebAppInstance",
    "i-810e36b3qu8xl06":  "MyCheckInstance",
    "nat-0c5a9q6d1l20": "MyNATGateway",
    "webapp-75c8k976-4q6f2": "MyWebAppPod"
}
2025-03-05 11:47:12 - INFO - Running AWS optimizations...
2025-03-05 11:47:29 - WARNING - Security Group MyWebSecGroup has an open rule! Consider restricting it. Potential Savings: $6.00
2025-03-05 11:48:05 - INFO - EC2 Instance MyWebAppInstance (t3.micro) is running. Consider rightsizing, stopping, or using reserved instances. Potential Savings: $9.43
2025-03-05 11:48:38 - INFO - EC2 Instance MyCheckInstance (t2.small) is running. Consider rightsizing, stopping, or using reserved instances. Potential Savings: $5.77
2025-03-05 11:49:22 - INFO - RDS Instance my-db-instance (mysql) is running. By using Aurora Serverless (if applicable). Potential Savings: $7.38
2025-03-05 11:49:59 - INFO - NAT Gateway MyNATGateway might be costing you money! Use Transit Gateway. Potential Savings: $8.10
2025-03-05 11:50:17 - INFO - Uploaded job1/100 to s3://my-log-archive-bucket/job1-100.log.gz. Savings: $0.05
2025-03-05 11:50:21 - INFO - Uploaded job2/200 to s3://my-log-archive-bucket/job2-200.log.gz. Savings: $0.07
2025-03-05 11:50:23 - INFO - Uploaded job3/300 to s3://my-log-archive-bucket/job3-300.log.gz. Savings: $0.03
2025-03-05 11:50:29 - INFO - Uploaded job4/400 to s3://my-log-archive-bucket/job4-400.log.gz. Savings: $0.09
2025-03-05 11:51:02 - INFO - Pod MyWebAppPod in namespace default is requesting 1.00 CPU cores and 1.00 GB memory. Consider right-sizing. Potential Savings: $7.26
2025-03-05 11:51:17 - INFO - Pod my-pod-2 in namespace default is requesting 0.50 CPU cores and 0.50 GB memory. Consider right-sizing. Potential Savings: $3.61
2025-03-05 11:52:01 - INFO -
Total Initial Estimated Cost: $78.26
Total Estimated Cost Savings: $30.47
Optimized cost: $47.79
Overall Improved Efficiency: 38.93%
2025-03-05 11:52:14 - INFO - Saved resource names to resource_names.json
```

Figure 5.1 Optimized Cost for AWS Cloud

For Azure, the findings had a similar process, optimized for the Azure cloud ecosystem. The tool starts with logs for virtual machines. It made rightsizing recommendations for two of my virtual machines, both Standard_B1ms (mywebappinstance-vm and mycheckinstance-vm). There were potential savings of $9.34 and $9.33, respectively, if I were to implement those recommendations. The Azure SQL Database was a candidate for migration to the serverless compute tier, with an estimated savings of $6.06. Added security was also possible by finding open rules in the Network Security Group, which had a potential savings of $5.39 once restricted. I also optimized Jenkins log processing by compressing logs from Jenkins and uploaded the logs to Azure Blob Storage, for savings of $0.03, $0.09, and $0.07 for each log. The analysis went up to Azure Kubernetes Service (AKS). It could optimize the recommendation for rightsizing of two pods, mywebapppod-aks and my-pod-2, which had potential find savings of $6.85 and $5.44 respectively. Altogether, the total estimated cost of $72.03 was reduced to $42.60, with total savings of $29.43 and overall efficiency improvement of 40.86%.



```
Enter Jenkins home directory: /var/lib/jenkins
Enter Azure Storage Account name: my_storage_account
Enter Azure Blob Container name: jenkins-logs
2025-03-17 14:32:21 - INFO - Load resource names from resource_names.json
Resource Names Loaded:
{
"myWebSecGroup": "myWebSecGroup-sg",
"myWebAppInstance": "mywebappinstance-vm",
"myCheckInstance": "mycheckinstance-vm",
"myNatGateway": "myNatGateway-nat",
"myWebAppPod": "mywebapppod-aks"
}
2025-03-17 14:33:06 - INFO - Running Azure optimizations...
2025-03-17 14:33:14 - INFO - Calling optimize_vms...
2025-03-17 14:33:19 - INFO - Virtual Machine mywebappinstance-vm (Standard_B1ms) is running. Consider rightsizing, stopping, or using reserved instances. Potential Savings: $9.34
2025-03-17 14:33:28 - INFO - Virtual Machine mycheckinstance-vm (Standard_B1s) is running. Consider rightsizing, stopping, or using reserved instances. Potential Savings: $9.33
2025-03-17 14:33:31 - INFO - optimize_vms completed.
2025-03-17 14:33:42 - INFO - Calling optimize_sql_databases...
2025-03-17 14:33:51 - INFO - Azure SQL Database my-db-instance is running. Consider using serverless compute tier. Potential Savings: $6.06
2025-03-17 14:34:01 - INFO - optimize_sql_databases completed.
2025-03-17 14:34:05 - INFO - Calling optimize_sgs...
2025-03-17 14:34:16 - WARNING - Network Security Group myWebSecGroup-sg has an open rule! Consider restricting it. Potential Savings: $5.39
2025-03-17 14:34:21 - INFO - optimize_sgs completed.
2025-03-17 14:34:24 - INFO - Calling process_jenkins_logs...
2025-03-17 14:34:30 - INFO - Uploaded job1/100 to blob job1-100.log.gz. Savings: $0.03
2025-03-17 14:34:32 - INFO - Uploaded job2/200 to blob job2-200.log.gz. Savings: $0.09
2025-03-17 14:34:36 - INFO - Uploaded job3/300 to blob job3-300.log.gz. Savings: $0.07
2025-03-17 14:34:42 - INFO - process_jenkins_logs completed.
2025-03-17 14:34:56 - INFO - Calling optimize_aks_resources...
2025-03-17 14:35:16 - INFO - Kubernetes Pod mywebapppod-aks in namespace default is requesting 1.00 CPU cores and 1.00 GB memory. Consider right-sizing. Potential Savings: $6.85
2025-03-17 14:35:24 - INFO - Kubernetes Pod my-pod-2 in namespace default is requesting 0.50 CPU cores and 0.50 GB memory. Consider right-sizing. Potential Savings: $5.44
2025-03-17 14:35:33 - INFO - optimize_aks_resources completed.

2025-03-17 14:35:46 - INFO -
Total Initial Estimated Cost: $72.03
Total Estimated Cost Savings: $29.43
Optimized cost: $42.60
Overall Improved Efficiency: 40.86%
2025-03-17 14:35:55 - INFO - Azure optimizations completed.
```

Figure 5.2 Optimized Cost for Azure Cloud

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

Organizations are increasingly deploying cloud computing platforms such as AWS and Azure to deliver dynamic digital products and services that can scale and adapt with the organization. Unfortunately, as organizations adopt cloud technology at increasing rates, the complexity of managing costs in these ever-changing environments increases to levels few can initially foresee. This project aims to directly address many of the astonishments in cloud technology management cost issues with the provision of on demand, fully automated Cost Optimization Targeting capabilities specific to DevOps environments utilizing AWS and Azure services. The aim of this project was to deliver a clear and practical approach to managing cloud costs using a combination of advanced automation, smart analytics, and design modularity, while creating a costing solution that allows organizations to find and eliminate waste and thus maximize the value of their cloud services.

The various projects examined of automatic cost optimization integrated into the DevOps pipelines and platforms in AWS and Azure have shown a powerful and quantifiable change in how modern organizations control their cloud spending. By asking the projects to consider automatic tools for assessment that used intelligent, automated controls into the DevOps workflow, we ensured every layer of the cloud infrastructure security, compute, storage, networking, logging, and container orchestration in accordance with DevOps principles was going to be reviewed for efficiency, and targeted savings through recommendations and optimizations. The results yielded are indicative of major wins: in AWS the targeted, focused optimization programs created to rightsizing EC2 instances, tightening security group rules, migrating RDS to Aurora Serverless, reducing network and log management recommendations reduced each monthly cost by almost 39%. In Azure, rightsizing virtual machines, optimizing SQL Database tiers, tightening network security, and rightsizing AKS pods experienced almost a 41% deceleration per month spend. The results demonstrated the advantages of a systematic, datacentric approach in which everything works together in a structured manner to allow for both incremental and significant changes which can provide significant overall savings with no delays to performance, availability, or that ever-increasing expectation of continuous delivery agility.

This success is attributed to several key tenets of modern cloud cost optimization: first, continuous rightsizing and proactive resource management- with automation and analytics - are still the strongest methods for eliminating waste and matching cloud resources to real workload requirements. The project's automated discovery engine and intelligent analysis modules were able to identify underutilized or misconfigured resources while also providing actionable, prioritized recommendations that allowed DevOps teams to make informed and timely decisions. Robust reporting and visualization tools also helped bridge the gap between the technical community and the financial world, creating an environment of shared responsibility and visibility around cloud spending. Finally, integration of these optimization routines into the day-to-day DevOps routine allows organizations to view cost optimization efforts not as a one-off initiative but instead as an ongoing iterative process of resource management that can adapt in real time to changes in usage patterns and business needs.

In conclusion, this example of cost fulfilment indicates how automation, modular designs, and advanced analytics can cultivate durable financial discipline for cloud operations. The achieve the potential for repeatable, significant savings within AWS and Azure environments-without jeopardizing service quality or deployment speed-the adds a confidence and competitive position for organization to network and scale at pace in a constantly changing digital ecosystem. As cloud adoption accelerates, embedding these types of tools and practices in the DevOps lifecycle, IT can not only maximize ROI but also create a resilient and low-cost IT operation in alignment with what comes next.

6.2 Future Scope

Even with the success of this initial version, there are many ways to improve the implementation and for future development and scaling. For example, machine learning algorithms could be employed to add a predictive cost optimization feature. By analyzing past usage patterns and detecting potential anomalies, the tool could predict future spike, or reductions in demand, and recommend precede actions such as upsizing or downsizing resources, or recommending a change in deployment strategy. This type of predictive functionality would further reduce the role of humans in the process and, ideally, prevent potential budget overruns for organizations.

Another area of possible improvement would be to build one or more automated remediation features. As mentioned previously, the tool simply presents actionable suggestions. Future iterations could improve upon this by allowing the tool to implement certain optimizations, such as contesting idle resources or pulling workloads in real time. This automation would improve the time to action, would further remove human involvement, and would help organizations ensure that cost-saving solutions were being implemented consistently - and, potentially critically, quickly. In the future, adding coverage for more services, third-party, and hybrid or multi-cloud environments could also increase value for customers with varied and complex infrastructures.

The addition of historical trend analysis would also be beneficial, helping teams to visualize cloud spending to show how things change over time, and potentially relate cost changes to specific events, deployments, and business cycles. These insights would be very useful for long term budget forecasting, and accountability, as well as strategic planning. Identifying the causes of spending changes will equip organizations with better information about their ability to allocate cloud resources and invest in cloud technologies.

Finally, further integration into CI/CD pipelines would have a great impact by establishing cost governance as a cultural feature in the software delivery process. By embedding cost checks and optimization processes in the deployment process, organizations can ensure that no release is cost-inefficient, as they are continuing to adhere to budget constraints and financial examinations. This ongoing cost management approach ensures that budgets are not exceeded and ingrains an ethos of efficiency and financial responsibility in DevOps teams.

Overall, this project carries many first steps towards automation and intelligence in cloud cost optimization for DevOps environments. Its extensive feature set, modular design, and actionable insights make it a powerful tool for any organization looking to reduce the cost of cloud expenses while maintaining performance and agility. As this tool grows in strength, through adding advanced analytics, automated remediation, or deeper embeddedness in the DevOps toolchain - the cloud efficient and cost control journey is becoming a safer bet and easier pursuit.

References

Research Papers

[1] P. Osypanka and P. Nawrocki, “Resource usage cost optimization in cloud computing using machine learning,” *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 1–1, 2020.

[2] E. Weintraub and Y. Cohen, “Cost optimization of cloud computing services in a networked environment,” *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 4, pp. 1–7, 2021.

[3] M. Liu, H. Pan, and Y. Shen, “Cost optimization for cloud storage from user perspectives: recent advances, taxonomy, and survey,” *ACM Computing Surveys*, vol. 55, no. 3, pp. 1–36, 2023.

[4] S. Chaisiri, B. S. Lee, and D. Niyato, “Optimization of resource provisioning cost in cloud computing,” *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 164–177, 2022.

[5] Pochu, S., Nersu, S. R. K., & Kathram, S. R. Multi-cloud DevOps strategies: A framework for agility and cost optimization, “*Journal of Advanced Information and Global Studies*”, 7(1), 2024.

[6] Buttar, A. M., Khalid, A., Alenezi, M., Akbar, M. A., Rafi, S., Gumaei, A. H., & Riaz, M. T. Optimization of DevOps transformation for cloud-based applications. Electronics, 12(2), 357. 2023.

[7] Gadani, N. N. Optimizing software development processes in cloud computing environments using agile methodologies and DevOps practices, “*Asian Journal of Research in Computer Science”*, 17(7), 75–83. 2021

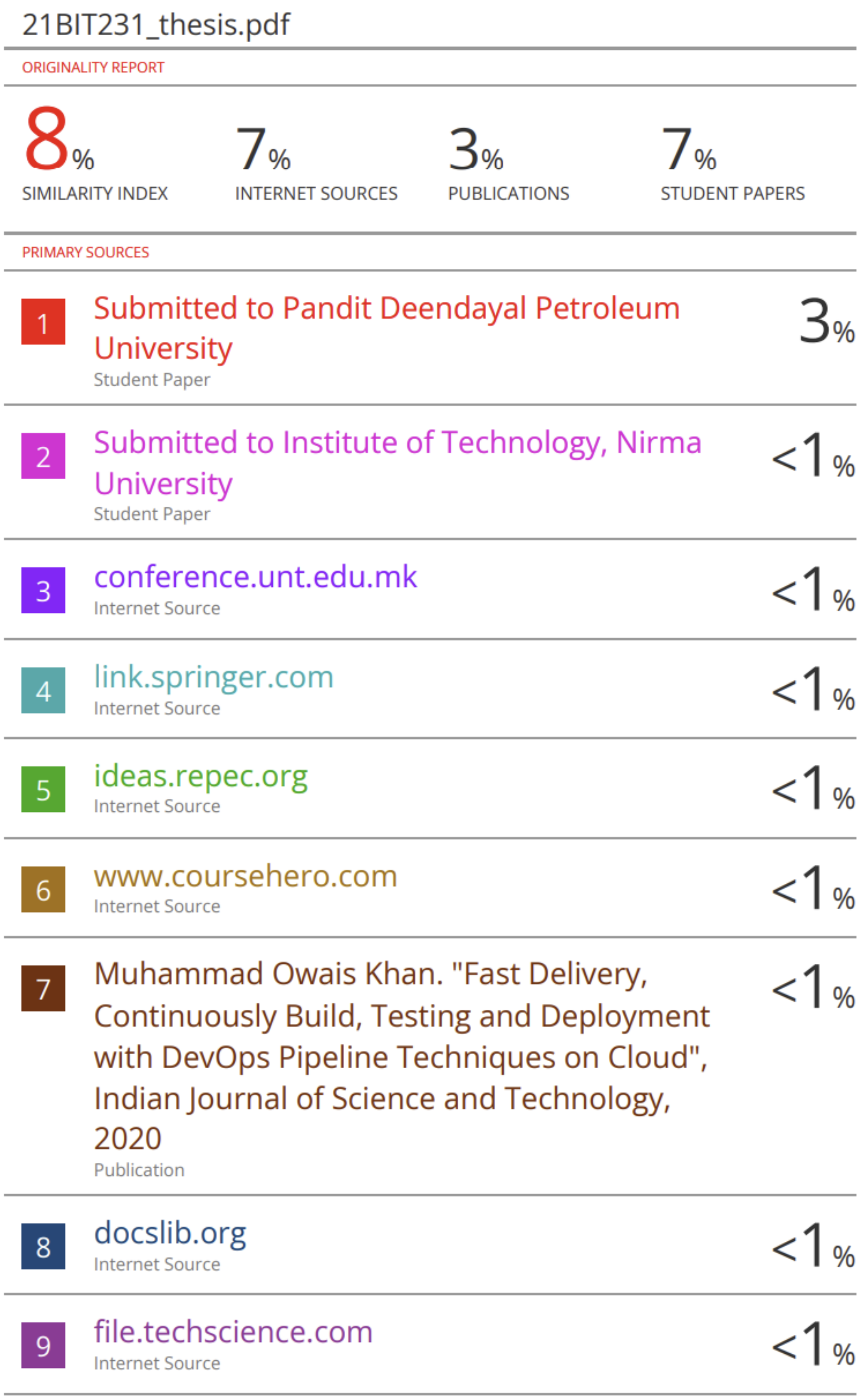
[8] Khan, M. O., Jumani, A. K., Farhan, W. A. S., & Shaikh, A. A. Fast delivery, continuously build, testing and deployment with DevOps pipeline techniques on cloud, “*Indian Journal of Science and Technology*”, 13(5), 552–575, 2020.

[9] Vaasanthi, R., Kumari, V. P., & Kingston, S. P. Analysis of DevOps tools to predict an optimized pipeline by adding weightage for parameters, “*International Journal of Computer Applications”*, 181(33), 33–35, 2018

[10] Soni, M. End to end automation on cloud with build pipeline: The case for DevOps in insurance industry, continuous integration, continuous testing, and continuous delivery. "*Proceedings of the International Conference on Computing, Communication and Automation (ICCCA)",* 1–6, 2015.

[11] S. Chaisiri, B. S. Lee, and D. Niyato, "Multi-provider cloud computing network infrastructure optimization," *Future Generation Computer Systems*, vol. 54, pp. 1–14, 2016.

Turnitin Plagiarism Report



21BIT231_thesis.pdf

ORIGINALITY REPORT

8%	7%	3%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Pandit Deendayal Petroleum University Student Paper	3%
2	Submitted to Institute of Technology, Nirma University Student Paper	<1%
3	conference.unt.edu.mk Internet Source	<1%
4	link.springer.com Internet Source	<1%
5	ideas.repec.org Internet Source	<1%
6	www.coursehero.com Internet Source	<1%
7	Muhammad Owais Khan. "Fast Delivery, Continuously Build, Testing and Deployment with DevOps Pipeline Techniques on Cloud", Indian Journal of Science and Technology, 2020 Publication	<1%
8	docslib.org Internet Source	<1%
9	file.techscience.com Internet Source	<1%