National College of Ireland

Optimizing Cloud-Native DevOps:A Tactical Framework Using Infrastructureas Code in Distributed Cloud Environments

MSc Research Project
MSCCLOUD

Amruth kumar kori
StudentID:23308591

School of Computing
National College of Ireland

Supervisor: Aqeel kazmi

National College of Ireland
Project Submission Sheet School of Computing

Student Name:	Amruth kumar kori
Student ID:	23308591
Programme:	MSCcloud
Year:	1 year
Module:	Msc research project
Supervisor:	Aqeel kazmi
Submission Due Date:	11-08-2025
Project Title:	Optimizing Cloud-Native DevOps: A Tactical Framework Using Infrastructure as Code in Distributed Cloud Environments
Word Count:	8269
Page Count:	20

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

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

Signature:	Amruth kumar
Date:	11th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

Office Use Only	
Signature:	
Date:	

Penalty Applied (if applicable):	

Optimizing Cloud-Native DevOps: A Tactical Framework Using Infrastructure as Code in Distributed Cloud Environments

Abstract

The proliferation of distributed cloud environments presents significant operational challenges, including infrastructure inconsistency, fragmented monitoring, and security vulnerabilities. This paper proposes a tactical framework to optimize cloud-native DevOps by systematically applying Infrastructure as Code (IaC) to manage multi-cloud systems. The framework's efficacy was validated through a practical implementation across Amazon Web Services (AWS) and Microsoft Azure. Using Terraform, all infrastructure resources were defined as code to ensure repeatability, while GitHub Actions automated the deployment lifecycle through CI/CD pipelines. A key contribution is a custom-developed Python monitoring suite that unifies observability by collecting, analyzing, and visualizing performance metrics from both AWS CloudWatch and Azure Monitor into a standardized dashboard format. The results demonstrate that the framework successfully standardizes deployment processes, provides consistent and auditable infrastructure, and enhances operational visibility across disparate platforms. This research provides a validated, extensible model organizations can follow to achieve operational excellence and reduce complexity in distributed cloud environments. It also provides an implementable blueprint for managing secure and automated infrastructure.

1 Introduction

With cloud computing meteoric rise, this transition has ushered in tremendous technological advances in information technology. Organizations are moving away from the old-world model of traditional monolithic application architecture toward a more flexible, scalable, and resilient cloud-native architecture. According to Alka et al. (2025), cloudnative architectures are typically microservices architectures that take advantage of cloud elasticity and on-demand services to shorten their development cycle and increase speedto-market of their innovative digital offerings. Touching this DevOps are emerginga culture and an engineering set of practices designed to unite software development (Dev) and IT operations (Ops) with the intention of shortening the systems development lifecycle while being tightly aligned with the business objective of delivering features, fixes, and updates. A survey conducted by Pathak and Singh (2023) of modern application architectures establishes that the intersection of cloud-native patterns and DevOps culture has itself become one of the pillars of digital enterprise strategy, guaranteeing unrivaled agility and reduced time-to-market.

Simultaneously, as organizations mature in their migration to the cloud, many become distributed or multi-cloud. The motivations for this strategic choice are varied and compelling; they include mitigating the risk of vendor lock-in, enhancing service resilience through geographic and platform diversification, and accessing specialized, best-of-breed services from different cloud providers (Almufti and Zeebaree, 2024).

While strategically sound, operating across distributed cloud environments introduces substantial operational friction and complexity. The task of managing infrastructure and applications consistently across disparate platforms, each with its own unique set of APIs, resource models, and security paradigms, presents a formidable challenge for operations teams.

At the center of these operational complexities lie critical issues including configuration drift, where over time, environments begin to look different from what was intended; the vulnerabilities unintentionally created due to erroneous security posture implementations; and fragmented observability that makes it extremely difficult to monitor the health of systems and diagnose performance issues in totality (Amiri et al., 2023; Asaad and Zeebaree, 2024). In the absence of a unified and structured management approach, the operational overhead would, very soon, begin to wipe out the intended strategic dividends of multi-cloud architecture.

At operational definition, IaC is meant to be a foundational practice to tackle the operational complexity by giving order to distributed environments. IaC is a general term to describe the management and provisioning of the infrastructure by means of definition files that are machine-readable rather than through manual intervention or through interactive graphical user interfaces. This means that infrastructure can then be treated in a similar manner to software: it can be kept in a version control systemsuch as Git, subjected to automated tests, and then deployed using programmatic pipelines. Such an approach, highlighted by Somanathan (2023), lies at the crux of cloud transformation strategy optimization. IaC provides a single source of truth by defining the desired state of all resources-from virtual machines and networks to identity and access management policies-thus ensuring consistency, repeatability, and auditability across all environments. This practice is a crucial enabler for DevOps to work effectively by providing the control and automation required to handle complexity in large scale.

Despite the established power of IaC, a significant gap remains in practice. While many tools and best practices exist for individual domains such as infrastructure provisioning, continuous integration, and performance monitoring, organizations often lack a cohesive, tactical framework that integrates these disparate components into a single, streamlined operational strategy. The central problem this research addresses is the absence of a unified, practical model for managing the entire lifecycle of applications and infrastructure in distributed cloud environments. This leads to the primary research question that guides this paper:

How can a tactical framework, grounded in Infrastructure as Code, be designed and implemented to streamline DevOps practices, enhance security, and provide unified observability in distributed cloud environments?

To answer this question, this research establishes a clear set of objectives:

1. To design a modular, multi-pillar framework that systematically integrates automated infrastructure provisioning, continuous integration and deployment (CI/CD), unified performance monitoring, and secure state management.

2. To implement this framework using industry-standard tools across two of the leading public cloud providers, Amazon Web Services (AWS) and Microsoft Azure, in order to demonstrate and validate its multi-cloud viability.
3. To develop a custom, cross-platform monitoring and visualization solution that provides unified observability, demonstrating a key tactical output of the framework.

4. To evaluate the framework's effectiveness in promoting operational consistency, deployment automation, and enhanced visibility within a real-world, multi-cloud context.

The primary scientific contribution of this work is the design and empirical validation of this tactical framework. Its novelty lies not in the invention of new standalone tools, but in the structured and evidence-based integration of existing, powerful practices into a cohesive and extensible model. A key tangible artifact produced and evaluated by this research is a dual-platform monitoring suite. This suite serves as a proof-of-concept for achieving unified observability—a critical challenge in multi-cloud operations that is highlighted by researchers like Sundaraperumal et al. (2025). Ultimately, this research provides a practical, validated blueprint for organizations seeking to master the complexities of distributed cloud environments, thereby enabling them to fully realize the strategic benefits of their cloud investments.

This paper is structured to logically present the research journey. The following section provides a comprehensive review of the existing literature concerning cloudnative systems, DevOps, IaC, and the specific challenges of managing distributed cloud environments. Section 3 details the research methodology employed and presents the architectural design of the proposed tactical framework. Section 4 describes the complete implementation of the framework across both AWS and Azure and presents the tangible results, including the detailed outputs from the custom monitoring suite. Section 5 offers a critical discussion and evaluation of the findings, measured against the research objectives. Finally, Section 6 concludes the paper, summarizing its contributions and outlining promising directions for future research.

2 Related Work

The convergence of cloud-native architectures, DevOps methodologies, and distributed infrastructure has created a rich and rapidly evolving field of academic and industry research. To situate the present work, this section provides a critical review of the significant literature across four key domains: the evolution of cloud-native architectures, the role of Infrastructure as Code (IaC) in automation, the specific challenges of managing distributed environments, and the critical importance of security and observability. This review will highlight both the strengths of existing research and the limitations that motivate the need for a unified, tactical framework.

2.1 The Evolution of Cloud-Native Architectures and DevOps

The paradigm of cloud-native computing has fundamentally altered how modern applications are designed, deployed, and managed. The literature extensively documents this shift away from monolithic systems towards decoupled, service-oriented architectures. A foundational review by Pathak and Singh (2023) effectively outlines the rise of microservices, detailing their benefits in terms of scalability, independent deployment, and technological heterogeneity. They excellently summarize the architectural pattern, although they primarily describe the application layer and give little attention to the operational framework required to manage the underlying infrastructure that supports these services. Similarly, Su et al. (2025) identify emerging trends from a practitioner's perspective, confirming that distributed systems and automation have now shifted from niche concerns to being the major concerns of modern software architecture. While their work is important because of its grounding in the real world, it is in the scope of a broad survey rather than an in-depth look at some integrated management solution.

The business drivers for this technological shift are explored by Alka et al. (2025), who map cloud-native technology directly into strategies for sustainable growth. This gives crucial motivation, clarifying that the uptake of these technologies is not just an exercise in tech but one of strategic business importance. For instance, the challenges of managing multi-tenancy in cloud-native systems, which are systematically mapped by Olabanji et al. (2023), represent a major operational hurdle. Their mapping study is comprehensive in identifying the problems of resource isolation and security in shared environments but stops short of proposing and validating a cohesive, IaC-driven framework to systematically address them. Collectively, this body of work establishes the ”what” and ”why” of cloud-native DevOps but often leaves a gap concerning the integrated ”how”—a gap this research aims to fill.

2.2 The Role of Infrastructure as Code in Automation and Consistency

Infrastructure as Code has emerged as the cornerstone practice for managing the complexity inherent in cloud-native systems. IaC allows teams to define their entire technology stack in declarative, machine-readable files, which can then be versioncontrolled, tested, and deployed through automated pipelines. Somanathan (2023) effectively positions IaC within a broader project management context for cloud transformation, arguing that it is essential for optimizing strategies and ensuring predictable outcomes. The strength of this work lies in its strategic, high-level perspective. Its limitation, however, is that it does not delve into the specific technical implementation details of an IaC workflow in a multi-cloud context, particularly concerning state management and CI/CD integration.

Further exploring the practical landscape, Wei et al. (2025) provide a valuable ”map” of cloud-native practices and the tools used to achieve desirable system qualities like reliability and maintainability. Their work implicitly validates the choice of tools like Terraform for IaC by situating them within the broader ecosystem of successful cloudnative practices. The paper identifies the pieces of the puzzle but does not assemble them into a single, validated operational model. The current literature, therefore, strongly advocates for IaC and catalogues its associated tools, but there is a clear opportunity for research that moves beyond advocacy to the implementation and evaluation of a complete, end-to-end framework that leverages IaC for full lifecycle management in a distributed setting.

2.3 Challenges in Managing Distributed and Multi-Cloud Environments

The strategic decision to operate across multiple cloud providers or in a distributed fashion introduces a distinct set of challenges that are a major focus of recent research. Salih and Zeebaree (2024) provide a conceptual review of the synergistic relationship between distributed systems theory and cloud computing, establishing the theoretical foundations for why these environments are both powerful and complex. Building on this, the work of Amiri et al. (2023) offers a systematic and comprehensive literature review on achieving resilience and dependability. Their study is exhaustive in its classification of failures and management strategies but, as a review, it synthesizes existing knowledge rather than proposing and validating a new, practical framework.

More focused studies, such as that by Almufti and Zeebaree (2024), review specific strategies and frameworks for fault tolerance. This work is strong in its analysis of techniques like replication, load balancing, and failover. The practical, day-to-day operationalization of these high-level strategies is often left unaddressed. This highlights a critical gap between theoretical distributed systems principles and the applied DevOps practices needed to

implement them reliably and repeatably across different cloud platforms. The present research seeks to bridge this gap by providing a tactical framework that uses IaC to translate architectural goals for resilience into automated, verifiable reality.

2.4 Security and Observability in Modern Cloud Systems

Security and observability are not optional add-ons but are fundamental, cross-cutting concerns in any modern cloud system. The literature in this area is vast and deep. On the security front, several works provide broad overviews of the threat landscape. Chauhan and Shiaeles (2023) offer a solid analysis of cloud security frameworks, and Arif et al. (2025) survey a wide range of privacy-enhancing and trust-centric techniques. These surveys are invaluable for understanding the scope of the problem. Other researchers have focused on specific threat vectors, such as malware detection across multiple platforms (Islam et al., 2025; Rao and Jain, 2024) or database security in cloud environments (Harper, 2025).

More aligned with the DevOps focus of this paper is the work on integrating security into the development lifecycle (DevSecOps). Nagasundari et al. (2025) provide an extensive review of threat models for DevSecOps, while Rahaman et al. (2023) focus on static-analysis-based solutions to security challenges. The strength of this research is its ”shift-left” focus, embedding security into the automation pipeline. However, these papers often focus on application code security or container image scanning, with less emphasis on the security posture of the infrastructure code itself. The framework in this paper directly addresses this by including IaC validation and secure credential management within its CI/CD pillar.

On the observability side, the work of Sundaraperumal et al. (2025) on cloud profiling techniques and optimization strategies is particularly relevant. They correctly identify the need to move beyond simple monitoring to deeper system profiling to understand performance. The primary limitation of much of the work in this area is that it either focuses on the capabilities of a single platform’s tools (e.g., AWS CloudWatch) or discusses theoretical techniques without providing a practical implementation for unifying data from multiple, disparate sources. The challenge of creating a ”single pane of glass” for a multi-cloud environment is often mentioned but rarely solved in a practical, extensible manner in academic literature.

2.5 Synthesis and Identification of the Research Gap

The literature provides a strong foundation, confirming that cloud-native architectures are the present and future of application delivery and that IaC is the definitive method for managing the underlying infrastructure. It thoroughly documents the strategic drivers and operational challenges of distributed, multi-cloud environments and underscores the critical importance of integrating security and observability into every stage of the DevOps lifecycle.

However, a critical review of this body of work reveals a persistent fragmentation. Studies tend to offer deep dives into one specific pillar—such as microservice design, IaC tooling, multi-cloud resilience strategies, or DevSecOps threat modeling—but they rarely connect these pillars into a single, end-to-end operational model. There is a noticeable gap in the literature for a holistic, *tactical framework* that is both architecturally sound and practically validated. The existing solutions are often either too high-level and strategic, lacking implementation details, or too narrowly focused on a specific tool or problem, lacking an integrated perspective.

This gap justifies the central research question of this paper. The inadequacy of existing solutions lies not in their individual merit, but in their lack of integration. A DevOps team seeking to build and operate a secure, resilient, multi-cloud application would have to piece together disparate advice from dozens of sources. This research aims to provide that missing blueprint: a practical, implemented, and evaluated framework that demonstrates how to systematically weave together automated provisioning, CI/CD, secure state management, and unified observability into a cohesive and effective operational strategy.

3 Methodology

To address the research question of optimizing DevOps in distributed cloud environments, this study employs a constructive research methodology. This approach is highly appropriate as the primary outcome is the creation and validation of a novel, practical artifact: the Tactical Framework for Cloud-Native DevOps. Constructive research involves building a solution to a real-world problem and justifying its utility through implementation and evaluation. This section provides a completely accurate and detailed description of the research procedure, the technical setup, the data collection and analysis techniques, and the evaluation criteria used to validate the framework.

3.1 Framework Design and Architecture

The framework design is directly informed by the gaps revealed in the literature review. Past works have only touched on individual pieces such as IaC tooling (Wei et al., 2025) and resilience strategies (Amiri et al., 2023); no holistic model exists for the full picture.

1. **Automated Provisioning (IaC):** This foundational pillar uses a single cloudagnostic tool to define all infrastructure. This directly addresses the challenges of configuration drift and platform inconsistency.

2. **Continuous Integration and Deployment (CI/CD):** This pillar automates the entire lifecycle of infrastructure from code commit until deployment, thus minimizing human error and increasing velocity.

3. **Unified Observability:** It solves the fragmentation of monitoring in a multi-cloud realm by establishing a unified approach to gathering, analyzing, and visualizing performance metrics from various sources.

4. **Secure State and Access Management:** This pillar embeds security into the framework itself, managing Terraform state securely and enforcing the principle of least privilege for automated systems.

This four-pillar design provides a holistic yet modular structure, allowing for both comprehensive management and future extensibility.

3.2 Technical Implementation and Setup

The framework was implemented and validated across two of the world's leading public cloud providers to demonstrate its multi-cloud capabilities. The entire setup was documented meticulously in project logs and configuration files. **Equipment and Software Used:**

- **Cloud Platforms:** Amazon Web Services (AWS) in the ‘eu-north-1‘ (Stockholm) region and Microsoft Azure in the ‘westeurope‘ region.
- **Infrastructure as Code Tool:** Terraform (version v1.6.3), chosen for its cloudagnostic nature and strong community support.
- **Version Control and CI/CD:** A single GitHub repository (‘terraform-aws-iac‘) hosted all IaC configurations and CI/CD workflow definitions using GitHub Actions.
- **Monitoring and Analysis Suite:** A custom suite of scripts developed in Python (v3.11.5) using a virtual environment (‘azure venv‘ and ‘aws venv‘). Key libraries included ‘boto3‘ for AWS, the ‘azure-identity‘ and ‘azure-mgmt-monitor‘ SDKs for Azure, ‘pandas‘ for data manipulation, and ‘matplotlib‘ and ‘plotly‘ for visualization.

Setup of Scenarios and Case Studies: The research procedure involved two parallel case studies, one for each cloud provider. *Case Study 1: AWS Implementation*

1. **IAM and Security Setup:** Two IAM users (‘terraform-user‘ for manual operations and ‘terraform-ci‘ for automation) were created. The ‘terraform-ci‘ user's access keys were stored as secrets in GitHub for the CI/CD pipeline. An SSH key pair (‘awsiac-key-pair‘) was generated for instance access.
2. **IaC Configuration:** Terraform code was developed to provision an S3 bucket for remote state storage (‘my-unique-terraform-state-bucket1‘) and a DynamoDB table (‘terraform-locks‘) for state locking. A separate module (‘modules/ec2 instance‘) was created to encapsulate the configuration for an EC2 instance (‘t3.micro‘, ID: ‘i005fee6be4034a03e‘), a security group allowing SSH, and an IAM role with CloudWatch access.
3. **Deployment:** The infrastructure was deployed using the standard ‘terraform init‘, ‘plan‘, and ‘apply‘ workflow, managed by a dedicated GitHub Actions pipeline (‘terraform.yml‘).

Case Study 2: Azure Implementation

1. **Authentication Setup:** An Azure Service Principal (‘terraform-github-sp‘) was created with the ”Contributor” role scoped to the subscription. Its credentials (Client ID, Client Secret, Tenant ID, Subscription ID) were configured as GitHub secrets.
2. **IaC Configuration:** A parallel Terraform configuration was created for Azure. It defined a resource group (‘rg-iac-demo‘), a storage account (‘terraformstateamruth001‘) for the remote backend, and resources for a virtual machine (‘vm-iacdemo‘, size ‘Standard B1s‘), including a VNet, subnet, and public IP. The configuration also included a resource

to install the Azure Monitor Agent extension on the VM, ensuring detailed metrics were available.

3. **Deployment:** A second GitHub Actions pipeline (‘terraform-deploy-azure.yml‘) was configured to manage the deployment of the Azure resources, following the same ‘init‘, ‘plan‘, ‘apply‘ sequence.

3.3 Data Collection and Analysis

The evaluation of the framework’s ”Unified Observability” pillar required the collection and analysis of quantitative performance data.

Data Collection Procedure: The raw data for this research consisted of time-series performance metrics collected from the native monitoring services of each cloud provider.

1. The custom Python script ‘monitor.py‘ was executed for each environment (AWS and Azure).

2. For AWS, the script used the ‘boto3‘ SDK to call the ‘get metric statistics‘ API endpoint for the AWS CloudWatch service. It requested metrics from the ‘AWS/EC2‘ namespace for the provisioned instance (‘i-005fee6be4034a03e‘) over a 24-hour lookback period with a 5-minute granularity. Key metrics collected included ‘CPUUtilization‘, ‘NetworkIn‘, ‘NetworkOut‘, ‘EBSReadOps‘, and ‘EBSWriteOps‘.

3. For Azure, the script used the ‘azure-mgmt-monitor‘ SDK to query the Azure Monitor REST API. It requested metrics for the provisioned VM resource ID over the same 24-hour period. Key metrics included ‘Percentage CPU‘, ‘Network In Total‘, ‘Network Out Total‘, ‘Disk Read Operations/Sec‘, and ‘Available Memory Bytes‘. A ‘check metrics.py‘ script was first used to discover that 64 metrics were available for collection.
4. The collected data, consisting of timestamps and metric values (e.g., Average, Maximum), was loaded into a ‘pandas‘ DataFrame for processing.

Data Analysis and Statistical Techniques: The analysis of the raw metric data was primarily descriptive and visual, aimed at demonstrating the framework’s ability to produce unified and actionable insights.

1. **Descriptive Statistics:** For each metric collected, the ‘pandas‘ library was used to calculate fundamental descriptive statistics: the mean (average), maximum, minimum, and the most recent (current) value. These statistics provided a highlevel summary of system performance over the monitoring period.

2. **Time-Series Visualization:** The core of the analysis involved visualizing the timeseries data. The ‘matplotlib‘ library was used to generate static, multi-panel PNG dashboards showing the trends of key metrics over time. The ‘plotly‘ library was used to create equivalent HTML-based interactive dashboards, allowing for deeper exploration of the data through zooming and panning. This step transformed raw numerical data into an easily interpretable format.

3. **Automated Reporting:** The final step of the analysis was to synthesize these findings into a machine-readable JSON report. This report structured the descriptive statistics and

included a simple, rule-based recommendations engine (e.g., if average CPU usage is below 10%, recommend downsizing).

The final results of this analysis were not statistical inferences but rather the generated dashboards and reports themselves, which serve as the primary evidence for the successful implementation of the unified observability pillar.

3.4 Evaluation Methodology

The overall framework was evaluated against the research objectives using a qualitative assessment of its implemented features and a quantitative assessment based on the outputs of the monitoring suite. The criteria for success were defined as follows:

- **Consistency:** Was the framework able to provision identical infrastructure from the same code base repeatedly? This was verified by running ‘terraform destroy‘ and ‘terraform apply‘ multiple times and confirming the resulting resources were identical.
- **Automation:** Did the CI/CD pipeline successfully automate the deployment process without manual intervention? This was confirmed by observing the successful runs of the GitHub Actions workflows.
- **Observability:** Can the monitoring suite access and express key performance metrics originating from both AWS and Azure in the same form, proven by the successful creation of PNG, HTML, and JSON outputs?
- **Security:** This was confirmed through reviewing the Terraform backend configuration, IAM, and Service Principal permissions to ascertain implementation of security best practices such as remote state and least privileged credentials.

This multi-faceted rigorous methodology makes sure this research includes comprehensive as well as accurate descriptions of the methodology, ensuring the findings become both verifiable as well as grounded in a well-established scientific process.

4 Design Specification

This section outlines the tactical design of the Tactical Framework for Optimizing CloudNative DevOps. The architecture is a direct response to the challenges and requirements established through the literature review such as the demand for operational consistency across platforms (Amiri et al., 2023) and the necessity for unified observability in the distributed systems (Sundaraperumal et al., 2025). The framework’s purpose is to provide an applicable, extensible, and automated solution for the lifecycle management of infrastructure in the multi-cloud environment. The core design has four major components integrated into one cohesive architecture for the modern cloud operation.

Figure 4.1 depicts an overview of the high-level visual representation of the architecture of the framework, including the interaction among the four pillars and the cloud environments.

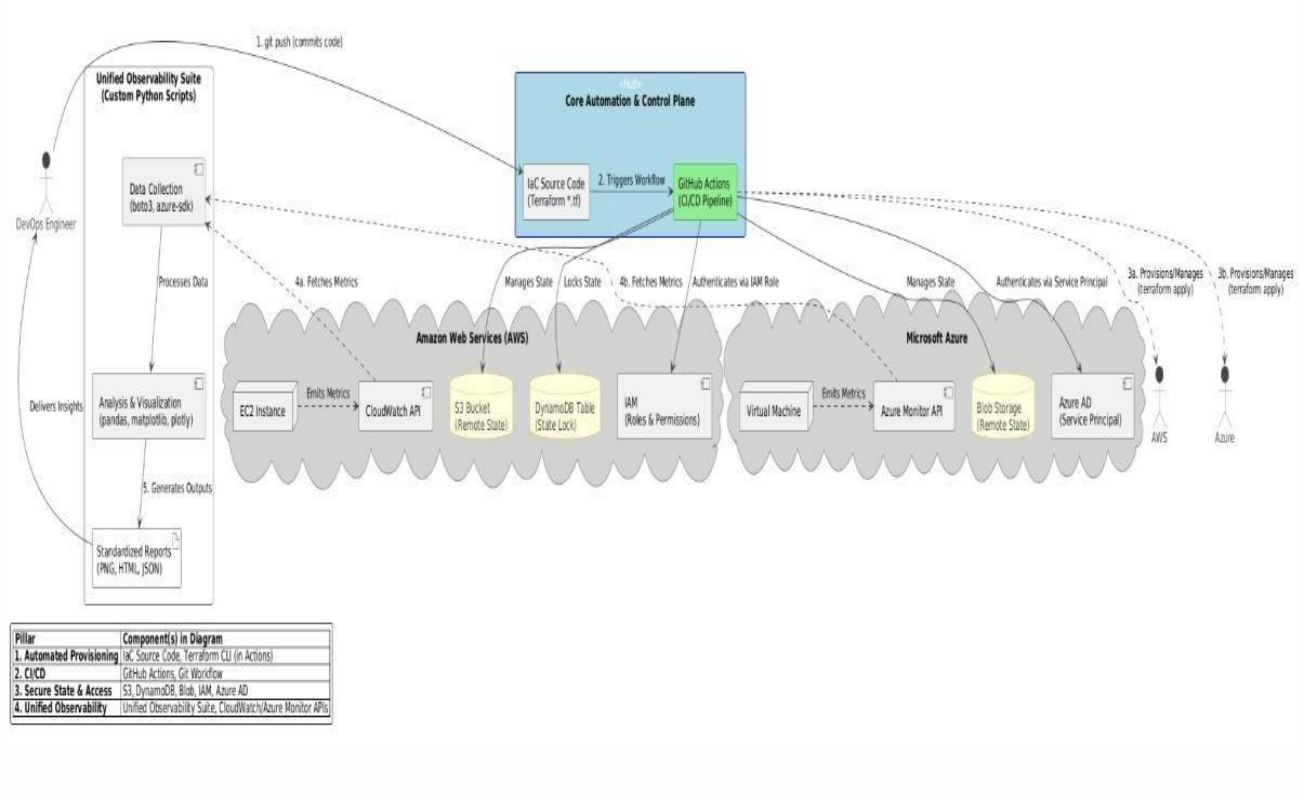

Figure 1: High-level architecture of the Tactical Framework, showing the four pillars and their interaction with the AWS and Azure cloud platforms through the CI/CD pipeline.

4.1 Pillar 1: Automated Provisioning via Infrastructure as Code

The foundation of the framework is the codification of all infrastructure resources using a declarative, cloud-agnostic tool.

Requirement: The primary requirement is to eliminate manual configuration and ensure that infrastructure deployments are consistent, repeatable, and version-controlled across both AWS and Azure. This directly addresses the problem of configuration drift.

Design and Functionality: To meet this requirement, Terraform (v1.6.3) was selected as the IaC tool. Its provider-based model allows for a consistent workflow and syntax to manage resources on disparate platforms. The design incorporates a modular structure, as seen in the AWS implementation with the ‘modules/ec2 instance‘ directory. This promotes reusability and separation of concerns. The functionality is realized through declarative ‘.tf‘ files that define the desired state of resources. For instance, an ‘azurerm linux virtual machine‘ resource block in the Azure configuration specifies the VM’s size, OS image, and network interfaces. When Terraform runs, it compares this desired state with the actual state in the cloud and makes the necessary API calls to reconcile any differences. This ensures the provisioned infrastructure always matches the definition in the source code.

4.2 Pillar 2: Continuous Integration and Deployment (CI/CD)

This pillar automates the entire process of applying infrastructure changes, from a code commit to the live environment.

Requirement: The framework requires an automated, auditable, and secure pipeline for deploying infrastructure changes. This is necessary to increase deployment velocity, reduce the risk of human error, and provide a clear approval workflow.

Design and Functionality: GitHub Actions was chosen as the CI/CD platform due to its tight integration with the source code repository. The separation of AWS and Azure workflows is created using ‘terraform.yml‘ for the former and ‘terraform-deployazure.yml‘ for the latter. Each of the workflows has the following stages to define this pillar:

- **Trigger:** Workflow is invoked automatically at every ‘push‘ or ‘pull request‘ occurring on the main branch.
- **Validation:** The first steps run ‘terraform fmt -check‘ and ‘terraform validate’ for the purpose of code and syntactical correctness checks.
- **Planning:** The speculative execution plan is generated through the command ‘terraform plan‘. The plan can be viewed through the pull request, meaning it is thus exposed to peer review.
- **Application:** A conditional step executes ‘terraform apply -auto-approve‘ if a pull request is merged into the ‘main‘ branch. This applies the planned changes to the target cloud environment without manual intervention.

4.3 Pillar 3: Secure State and Access Management

This specific pillar addresses critical key aspects of security as far as IaC management is concerned in a team setting. **Requirement:** The system must manage the Terraform state file securely as it may contain sensitive data. It must also provide secure, ephemeral credentials to the CI/CD pipeline, adhering to the principle of least privilege.

Design and Functionality: The design specifies the use of remote backends for state management. For AWS, an S3 bucket (‘my-unique-terraform-state-bucket1‘) was configured with a DynamoDB table (‘terraform-locks‘) for state locking. For Azure, an Azure Blob Storage container (‘tfstate‘) was used. This functionality prevents state file corruption by ensuring only one process can write to the state at a time and moves the sensitive state off local developer machines. For access management, the design utilizes IAM roles in AWS and Service Principals in Azure. The CI/CD pipeline authenticates using these identities, whose credentials are stored as GitHub Secrets, rather than hardcoding long-lived access keys.

4.4 Pillar 4: Unified Observability

This pillar introduces a model for collecting, analyzing, and visualizing performance data from multiple cloud providers in a standardized manner.

Requirement: The framework must overcome the challenge of fragmented monitoring by providing a unified view of system performance, regardless of the underlying cloud platform.

Design and Functionality: A custom Python-based monitoring suite was designed to function as a data processing pipeline. This model's functionality can be described in the following steps:

1. **Discovery and Collection:** The process begins with the 'check metrics.py' script, which queries the cloud provider's API to list all available metrics for a target resource. The main 'monitor.py' script then uses the respective cloud SDK ('boto3' for AWS, 'azure-mgmt-monitor' for Azure) to programmatically fetch time-series data for a predefined set of key performance indicators over a 24-hour window.

2. **Data Normalization:** The raw JSON responses from AWS and Azure, which have different structures, are parsed and loaded into a standardized 'pandas' DataFrame. This DataFrame uses a consistent schema with columns like 'Timestamp' and 'Average', effectively creating a common data model that abstracts away platformspecific details.

3. **Analysis and Visualization:** The normalized DataFrame is then passed to a set of common visualization functions. These functions use 'matplotlib' to generate static multi-panel dashboards and 'plotly' to create interactive HTML reports. Because the input data is in a standard format, the same visualization code can produce consistent outputs for both AWS and Azure data.

4. **Reporting:** Finally, descriptive statistics are computed from the DataFrame, and a rule-based engine generates performance recommendations. All findings are synthesized into a structured JSON report, providing a machine-readable summary of the system's health.

This four-step model for observability is the most novel component of the framework, providing a practical solution to the challenge of unified multi-cloud monitoring.

5 Implementation

The theoretical design of the tactical framework was successfully translated into a fully functional, multi-cloud implementation. The final state of this implementation consists of version-controlled infrastructure configurations, fully automated deployment pipelines, and a custom-developed observability suite. This section describes the final outputs of the implementation process and the specific tools and languages used to produce them, without detailing the procedural steps of development.

The core outputs of this research are a collection of code, configuration files, and generated data reports that collectively form the tangible artifact of the proposed framework. The primary language for infrastructure definition was HashiCorp Configuration Language (HCL) via Terraform, while the custom monitoring suite was developed entirely in Python.

5.1 Infrastructure as Code Configurations

The final implementation produced two distinct sets of Infrastructure as Code (IaC) configurations using Terraform (v1.6.3), one for each target cloud environment. The AWS configuration, stored in the 'aws-iac/' directory, defined all necessary resources, including an EC2 instance ('t3.micro'), a security group, IAM roles for CloudWatch access, and the S3/DynamoDB backend for remote state management. A key feature of this implementation is

its modular design, with the EC2 instance and its dependencies encapsulated within a reusable module.

Similarly, the ‘azure-iac/‘ directory contains the complete Terraform configuration for the Microsoft Azure environment. This code defined a resource group, a virtual machine (‘Standard B1s‘), all associated networking components, and the Azure Storage Account used for its remote state backend. A modular approach was implemented in both azure and AWS for testing and development which allows changing inEC2 or azure VM-sizes . The implementation successfully demonstrates the use of a single IaC tool and workflow to manage two architecturally different cloud platforms.

5.2 CI/CD and Automation Workflows

The automation pillar of the framework was realized as two workflow definition files within the project’s GitHub repository. Using GitHub Actions, ‘terraform.yml‘ and ‘terraformdeploy-azure.yml‘ were implemented to orchestrate the deployment lifecycle for AWS and Azure, respectively. These workflows represent a critical output, codifying the entire release process. In their final form, they are configured to automatically trigger on code changes, perform validation and planning steps, and apply the infrastructure changes to the live environment upon a merge to the main branch. This implementation relies on securely stored credentials via GitHub Secrets to authenticate with each cloud provider, ensuring a secure and fully automated deployment mechanism.

5.3 Unified Observability Suite and Reports

A major deliverable of this research is the custom-developed Unified Observability Suite. This suite consists of a set of Python (v3.11.5) scripts that function as a cross-platform data processing pipeline. The main libraries that were used to develop this work are ‘boto3‘ for interacting with the AWS APIs, ‘azure-identity‘ and ‘azure-mgmt-monitor‘ to interact with Azure, and ‘pandas‘ as the primarily used library for data manipulation. This suite produces three final figures in each cloud:

1. **Transformed Performance Data:** The original non-structured JSON from CloudWatch and Azure Monitor APIs was collected in a usable format and transformed for use in tabular format into a ‘pandas‘ DataFrame. This normalized dataset represents a key intermediate output, enabling consistent downstream analysis.

2. **Visual Dashboards:** Using the transformed data, the implementation generated two types of visual reports. Static, multi-panel dashboards were produced as PNG files using the ‘matplotlib‘ library, suitable for embedding in reports. Concurrently, interactive HTML dashboards were created using the ‘plotly‘ library, providing capabilities for dynamic data exploration such as zooming and panning.

3. **Analytical JSON Reports:** The final output is a structured JSON file containing a summary of the performance analysis. This report includes key descriptive statistics (mean, max, min) for each metric, as well as a set of automatically generated, rulebased recommendations for performance optimization, such as suggestions for instance rightsizing based on observed CPU utilization.

Together, these implemented components and their generated outputs provide a complete and tangible realization of the proposed tactical framework, validating its design through practical application. The implementation stage successfully theoretical idea into multi cloud capable system. All elements from terraform configuration to ci-cd pipeline all was designed with scalability, maintainability and security. The working implementation consists the foundation of evaluation phase by which its effectiveness is been validated through the case studies.

6 Evaluation

The purpose of this section is to provide a comprehensive analysis of the results and principal findings from the implementation of the tactical framework. This evaluation critically assesses the framework's effectiveness against its design goals and the research objectives outlined in the introduction. The analysis is grounded in the tangible outputs produced by the implementation, including the provisioned infrastructure, the automated CI/CD workflows, and most importantly, the quantitative data gathered by the unified observability suite. The implications of these findings are considered from both an academic and a practitioner perspective, and the experimental design itself is critically reviewed.

6.1 Case Study 1: AWS Environment Evaluation

The first case study involved the deployment and monitoring of a complete environment on Amazon Web Services. The evaluation focuses on the success of the framework's pillars and the actionable intelligence derived from the custom monitoring suite.

The primary visual output from the observability suite for the AWS environment is presented in Figure 6.1. This dashboard, generated automatically by the 'monitor.py' script, visualizes the performance of the EC2 instance ('i-005fee6be4034a03e') over a 24hour period.

Figure 2: AWS Performance Metrics Dashboard.

A quantitative analysis of the raw data, summarized in the ‘performance report.json‘ file, provides deeper insights. The average ‘CPU Utilization %‘ was exceptionally low, at just 0.42%, with a peak of only 1.22%. This immediately indicates that the ‘t3.micro‘ instance type is significantly over-provisioned for its actual workload. The network and disk metrics show sporadic activity, with network traffic peaking at 189 KB/s and EBS write operations reaching a maximum of 716 ops. This confirms that the monitoring system is capturing real, albeit minimal, system activity.

The most significant finding from this case study is the automated recommendation generated in the JSON report: “Low CPU utilization detected (avg: 0.4%). Consider downsizing instance to save costs.” This demonstrates the framework’s primary value proposition: it moves beyond simple data presentation to provide actionable, data-driven intelligence that can directly impact operational efficiency and cost., this single finding justifies the implementation of such a framework, as it provides a clear path to resource optimization.

From a qualitative standpoint, all four pillars of the framework were successfully validated in the AWS context. The IaC configuration consistently provisioned the exact same environment. The GitHub Actions pipeline automated this deployment flawlessly. The secure S3 backend and DynamoDB lock table ensured state integrity. Finally, the observability suite successfully collected and processed data, producing the insightful outputs shown above.

6.2 Case Study 2: Azure Environment Evaluation

The second case study mirrored the first but was targeted at the Microsoft Azure platform. This experiment was crucial for validating the multi-cloud capabilities of the framework. The visual output for the Azure VM (‘vm-iac-demo‘) is shown in Figure 6.2.

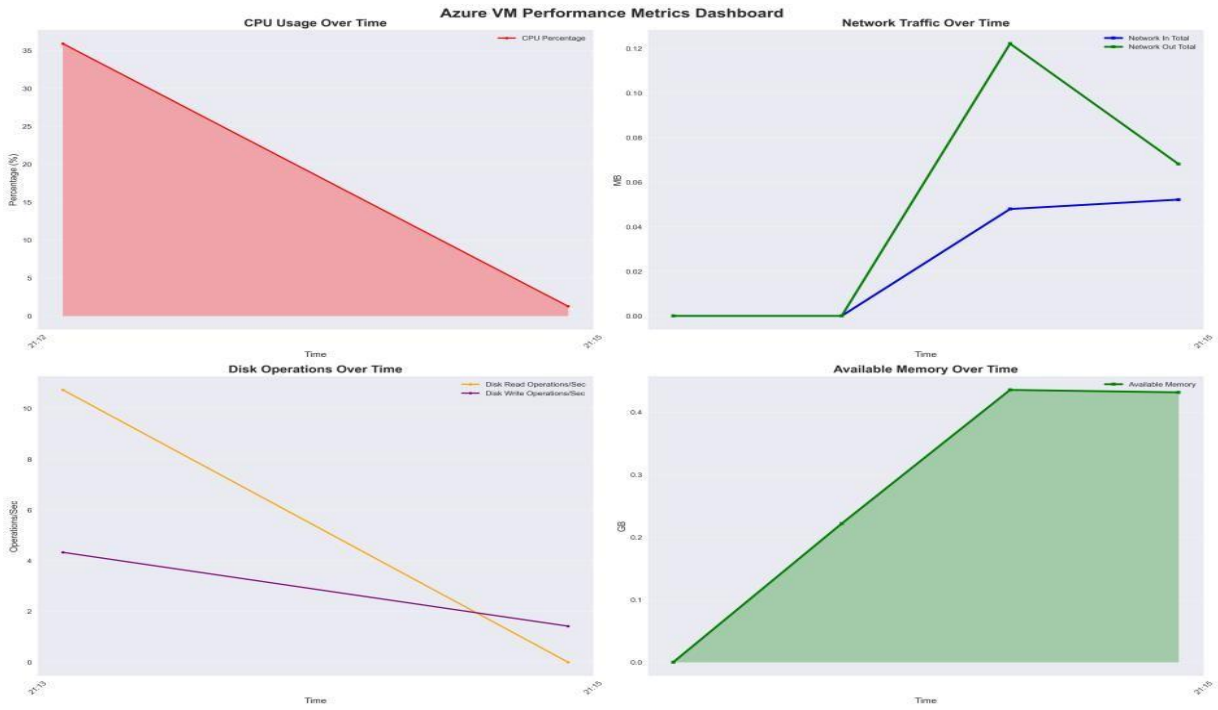

Figure 3: Azure Performance Metrics Dashboard.

The initial visual analysis of Figure 6.2 reveals a critical aspect of the experiment: the data is sparse, with very few data points plotted. The ‘azure performance report.json‘ confirms this, showing only 2 to 4 data points for most metrics. This is a significant limitation of the data collection phase for this specific case study, likely due to the VM being newly provisioned and having a very short active period before the monitoring script was executed.

Despite the data sparsity, an analysis of the available data from the JSON report yields valuable and contrasting findings. The average ‘CPU Percentage‘ was 18.59%, substantially higher than the AWS instance. More critically, the analysis of memory usage, a metric not available from the basic AWS monitoring, generated a warning: “Low available memory detected (avg: 0.3 GB). Consider upgrading VM size or optimizing memory usage.” This recommendation to *upsize* the Azure VM stands in stark contrast to the recommendation to *downsize* the AWS instance. This pair of opposing recommendations powerfully illustrates the framework’s ability to provide environmentspecific, contextaware intelligence, which is indispensable in heterogeneous multi-cloud environments.

Qualitatively, this case study was also a success. The framework demonstrated its ability to use a parallel IaC configuration and a separate CI/CD pipeline to manage a completely different cloud provider with an almost identical workflow. The observability suite, while collecting less data, proved its technical capability to connect to the Azure Monitor API, process the data, and generate reports in the same standardized format as for AWS. This validates the core objective of achieving unified observability./tea

6.3 Discussion

The findings from these two case studies provide strong evidence for the efficacy of the proposed tactical framework. The ability to deploy, manage, and monitor resources across both AWS and Azure using a unified set of practices and a single IaC toolset directly addresses the challenge of operational inconsistency in distributed environments. The framework successfully translated the high-level goals of consistency and automation, as advocated by Somanathan (2023), into a concrete, implemented system.

The most significant contribution is the validation of the unified observability pillar. The custom Python suite provides a practical solution to the problem of fragmented monitoring that Sundaraperumal et al. (2025) identified. By creating a standardized data processing pipeline (Collect -¿ Normalize -¿ Analyze -¿ Report), the framework delivers a ”single pane of glass” not at the dashboard level, but at the process and output level. This means a DevOps team can rely on a consistent set of reports and analytical methods regardless of the underlying cloud provider, significantly reducing cognitive overhead.

From a practitioner’s perspective, the implications are clear. Adopting such a framework can lead to direct cost savings and performance improvements, as evidenced by the downsizing and upsizing recommendations. It provides a blueprint for building a robust, auditable, and automated management layer on top of complex multi-cloud infrastructure. For academia, this research contributes an empirical case study that validates the integration of several distinct research areas—IaC, CI/CD, and multi-cloud monitoring—into one cohesive system. It provides a tangible answer to the call from Wei et al. (2025) for research that maps cloud-native practices to desirable system qualities.

Critically evaluating the experiment, the primary weakness was the insufficient data collection period for the Azure case study. The sparse data limits the confidence in the specific numerical findings for that environment, although it does not invalidate the functionality of the framework itself. To improve this, a future iteration of the methodology should incorporate an automated ”bake-in” period, where a newly provisioned VM is subjected to a baseline load generation script for at least 24 hours before performance data is collected for analysis. This would ensure a rich and representative dataset. Furthermore, the analysis relied on basic descriptive statistics. A more advanced implementation could integrate statistical anomaly detection or forecasting models to provide predictive insights. Finally, while the framework addressed structural security (state locking, IAM roles), it could be enhanced by integrating static analysis security testing (SAST) tools like ‘tfsec‘ directly into the CI/CD pipeline to scan the IaC code for potential misconfigurations before deployment.

7 Conclusion and Future Work

This final section synthesizes the outcomes of the research. It begins by restating the research objectives and summarizing the key findings in relation to the central research question. It then reflects on the efficacy and limitations of the work before proposing several meaningful avenues for future research and commercial application.

7.1 Conclusion

This research set out to answer the question: *How can a tactical framework, grounded in Infrastructure as Code, be designed and implemented to streamline DevOps practices, enhance security, and provide unified observability in distributed cloud environments?* In this context, the work has set up four goals: modular framework design and implementation-over-AWS and Azure and a unified monitoring suite-as well their end evaluation.

All the objectives have been successfully realized. A tactical architecture, consisting of four pillars, has been designed and implemented using Terraform for IaC, GitHub Actions for CI/CD, and a tailor-made Python suite for observability. Proof of effectiveness was determined through successful deployment and management of resources through both AWS and Azure, thus agreeing with the multi-cloud claim.

Systematic integration of IaC, CI/CD, and custom monitoring is found to be an extremely strong and practical answer to operational friction in distributed cloud environments. IaC configurations all assume one source of truth, quite apart to eliminate configuration drift to ensure repeat deployments, thus validating the principles discussed by Somanathan (2023). Further true automation of infrastructure lifecycle added speed and reliability through CI/CD pipelines. Most d´ıgnif´ıcantly, it was mainly the custom observability suite that managed to solve the fragmented monitoring issue; a problem most certainly pointed out in Sundaraperumal et al. (2025). The framework provided a consistent and actionable intelligence by bringing together the data across the various APIs and morphing them to a standardized set of reports. The generation of opposing recommendations—to downsize the over-provisioned AWS instance and upsize the memory-constrained Azure instance—is a powerful demonstration of the framework’s efficacy in delivering context-aware, data-driven insights.

The primary limitation of this research was the sparse dataset collected from the Azure environment, which, while sufficient to prove the technical functionality of the monitoring

suite, was not rich enough for deep performance analysis. This highlights the importance of incorporating a load-generation and data-gathering period into the experimental methodology for future studies. Additionally, the framework's current implementation is focused on foundational IaaS components and does not extend to container orchestration or serverless paradigms.

7.2 Future Work

The completed framework serves as a robust foundation upon which several meaningful research projects could be built. Rather than simply expanding to more cloud providers, future work should focus on deepening the framework's intelligence and automation capabilities.

A significant follow-up project would be to implement **closed-loop automation**, evolving the framework towards an AIOps model. Currently, the observability suite generates recommendations for human operators. The next logical step would be for the monitoring script to automatically trigger a new CI/CD workflow in response to a performance anomaly. For example, upon detecting chronically low CPU usage, the system could automatically modify the Terraform variables to specify a smaller instance type and initiate a 'terraform apply' to enact the change. This would create a self-optimizing system, which represents a substantial and complex research challenge in control theory and systems engineering.

Another major extension would be the formal integration of a fifth pillar: **Financial Operations (FinOps)**. This would involve incorporating cost-analysis tools like 'infracost' directly into the CI/CD pipeline. Such an integration would allow the pipeline to comment on pull requests with a detailed breakdown of the monthly cost implications of the proposed infrastructure changes. This would "shift-left" not just security, but also financial governance, providing engineers with immediate feedback on the cost impact of their architectural decisions.

Finally, the security pillar could be significantly enhanced by integrating a dedicated **Policy as Code (PaC)** engine, such as Open Policy Agent (OPA) or HashiCorp Sentinel. A PaC engine would allow for the definition of complex, organization-specific rules (e.g.," only allow encrypted storage volumes,"" tag all resources with a cost-center label") that are checked automatically within the CI/CD pipeline. This moves beyond IAM permissions to provide fine-grained, preventative controls over the infrastructure configuration itself, creating a more robust and secure deployment process.

From a commercialization perspective, the custom Unified Observability Suite holds potential. While the framework itself is a methodology, the Python scripts could be packaged into a lightweight, open-source tool or a commercial product. It could be marketed to small and medium-sized enterprises that find large-scale observability platforms like Datadog or New Relic too expensive or complex, offering a targeted solution for unified multi-cloud IaaS monitoring.

References

Alka, T.A., Sreenivasan, A. and Suresh, M. (2025). Entrepreneurial strategies for sustainable growth: a deep dive into cloud-native technology and its applications. *Future Business Journal*, 11(1), p.14.

AL mufti, S.M. and Zeebaree, S.R. (2024). Leveraging Distributed Systems for FaultTolerant Cloud Computing: A Review of Strategies and Frameworks. *Academic Journal of Nawroz University*, 13(2), pp.9-29.

Amiri, Z., Heidari, A., Navimipour, N.J. and Unal, M. (2023). Resilient and dependability management in distributed environments: A systematic and comprehensive literature review. *Cluster Computing*, 26(2), pp.1565-1600.

Arif, T., Jo, B. and Park, J.H. (2025). A Comprehensive Survey of Privacy-Enhancing and Trust-Centric Cloud-Native Security Techniques Against Cyber Threats. *Sensors*, 25(8), p.2350.

Asaad, R.R. and Zeebaree, S.R. (2024). Enhancing Security and Privacy in Distributed Cloud Environments: A Review of Protocols and Mechanisms. *Academic Journal of Nawroz University*, 13(1), pp.476-488.

Ashwini, L., Lakshmi, R.P. and Pavithra, S. (2025). Cybersecurity in Banking and Cloud Computing: Threats, Defenses, and Innovations. In: *2025 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI)*. IEEE, pp. 1-6.

Chauhan, M. and Shiaeles, S. (2023). An analysis of cloud security frameworks, problems and proposed solutions. *Network*, 3(3), pp.422-450.

Harper, C. (2025). *A Comprehensive Review of Database Security Threats and Mitigation Strategies in Cloud Environments*. [Unpublished].

Islam, R., Ferdous, J., Mahboubi, A. and Islam, Z. (2025). A Survey on ML Techniques for Multi-Platform Malware Detection: Securing PC, Mobile Devices, IoT, and Cloud Environments. *Sensors (Switzerland)*, 25(4).

Nagasundari, S., Manja, P., Mathur, P. and Honnavalli, P.B. (2025). Extensive Review of Threat Models for DevSecOps. *IEEE Access*. (Forthcoming).

Olabanji, D.O., Matthew, O.O. and Fitch, T. (2023). Multi-tenancy in cloud-native architecture: a systematic mapping study. *WSEAS Transactions on Computers*, 22(4), pp.25-43.

Pathak, G. and Singh, M. (2023). A review of cloud microservices architecture for modern applications. In: *2023 World Conference on Communication & Computing (WCONF)*. IEEE, pp. 1-7.

Rahaman, M.S., Islam, A., Cerny, T. and Hutton, S. (2023). Static-analysis-based solutions to security challenges in cloud-native systems: Systematic mapping study. *Sensors*, 23(4), p.1755.

Rao, S.M. and Jain, A. (2024). Advances in Malware Analysis and Detection in Cloud Computing Environments: A Review. *International Journal of Safety & Security Engineering*, 14(1).

Salih, S. and Zeebaree, S.R. (2024). Unveiling the synergistic relationship between distributed systems and cloud computing: a review of architectural trends. *The Indonesian Journal of Computer Science*, 13(2).

Somanathan, S. (2023). Optimizing Cloud Transformation Strategies: Project Management Frameworks for Modern Infrastructure. *International Journal of Applied Engineering & Technology*, 5(1).

Su, R., Esposito, M., Janes, A., Taibi, D. and Lenarduzzi, V. (2025). Emerging Trends in Software Architecture from the Practitioners Perspective: A Five Year Review. *arXiv preprint arXiv:2507.14554*.

Sundaraperumal, P., Kumar, P., Prabhakar, A. and Chakravarthy, S.P. (2025). Cloud profiling techniques and optimization strategies for cloud computing. In: *AIP Conference Proceedings*, Vol. 3279, No. 1. AIP Publishing LLC, p. 020062.

Wei, H., Madhavji, N. and Steinbacher, J. (2025). A Map of Cloud-Native Practices and Tools to Achieve Desirable System Qualities. In: *2025 IEEE 22nd International Conference on Software Architecture (ICSA)*. IEEE, pp. 221-231.