

Multi-Cloud Object Storage Provisioning Using Terraform

Adhi Narayanan G K¹, Dhanush Pandiyan M¹, Abdul Rasith A¹, Abdul Siddiq K¹

¹Department of Computer Science and Engineering, Dhanalakshmi Srinivasan Engineering College (Autonomous), Perambalur – 621 212, Tamil Nadu, India

Supervisor: Mrs. SR. Sowmiya M.E., (Ph.D.) | Anna University, Chennai – 600 025 | May 2026

Abstract

The rapid adoption of cloud computing has led organizations to rely on multiple cloud service providers to improve availability, scalability, and vendor independence. Managing infrastructure across heterogeneous cloud platforms, however, introduces complexity and operational overhead. This paper presents a Multi-Cloud Object Storage Provisioning system that leverages Infrastructure as Code (IaC) principles through Terraform to automate and standardize object storage provisioning across AWS, Microsoft Azure, and Google Cloud Platform (GCP). A unified Terraform configuration deploys storage services — including S3 buckets, Azure Blob Storage containers, and GCP Cloud Storage buckets — while maintaining provider-specific flexibility. The proposed system incorporates a multi-cloud data consistency framework using passive servers and non-communicating clients, ensuring reliable synchronization without centralized coordination overhead. Experimental evaluation demonstrates improvements in availability, fault tolerance, response time, and scalability compared to single-cloud architectures. The results confirm that Terraform-based IaC automation significantly simplifies multi-cloud infrastructure management and provides a scalable foundation for cloud-native applications.

Keywords — Multi-Cloud, Terraform, Infrastructure as Code (IaC), Object Storage, AWS S3, Azure Blob Storage, Google Cloud Storage, Data Consistency, DevOps, Cloud Automation

I. INTRODUCTION

Cloud computing has emerged as one of the most transformative technologies in modern information technology infrastructure. It enables organizations to store, process, and manage large volumes of data without maintaining physical hardware or on-premise data centers. Major cloud service providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) offer scalable object storage services — Amazon S3, Azure Blob Storage, and Google Cloud Storage — that are widely adopted for data storage, backup, and content delivery.

In recent years, organizations have increasingly adopted multi-cloud strategies, distributing workloads across multiple providers to improve reliability, avoid vendor lock-in, enhance performance, and ensure better disaster recovery. However, managing infrastructure across heterogeneous cloud environments is complex. Each provider uses different APIs, interfaces, and configuration procedures, making manual provisioning error-prone, time-consuming, and difficult to scale.

Infrastructure as Code (IaC) addresses these challenges by enabling declarative, version-controlled, and repeatable infrastructure definitions. Terraform, developed by HashiCorp, is among the most widely used IaC tools, supporting multiple cloud providers through provider plugins via a single unified configuration language — HCL (HashiCorp Configuration Language).

This paper presents a complete Multi-Cloud Object Storage Provisioning system using Terraform that simultaneously provisions storage resources on AWS, Azure, and GCP. Furthermore, a data consistency architecture using passive servers and non-communicating clients is proposed and evaluated to ensure reliable data synchronization across cloud nodes.

II. PROBLEM STATEMENT

Modern organizations managing multi-cloud storage infrastructure face several critical operational and technical challenges:

- **Complex Multi-Cloud Management:** Each cloud provider requires separate interfaces, SDKs, and configuration methods, creating silos and inconsistencies.
- **Manual and Time-Consuming Provisioning:** Cloud resources such as storage buckets and access policies are often created manually, increasing deployment time and human error.
- **Lack of Standardized Deployment:** Different platforms use different deployment procedures, making it difficult to maintain consistent environments.

- **Vendor Lock-in Risk:** Relying on a single cloud provider restricts flexibility and hampers workload migration.
- **Configuration Errors:** Manual configuration increases the risk of misconfigurations, security vulnerabilities, and resource wastage.
- **Limited Automation:** Traditional approaches lack automation, making it difficult to scale infrastructure efficiently.

III. OBJECTIVES

- Automate multi-cloud infrastructure provisioning using Infrastructure as Code principles.
- Implement a unified Terraform configuration to define and manage storage resources across AWS, Azure, and GCP in a declarative and repeatable manner.
- Provision object storage services (S3 buckets, Azure Blob containers, GCP buckets) through a single configuration while maintaining provider-specific flexibility.
- Reduce manual configuration and deployment errors by minimizing human intervention.
- Ensure data consistency across cloud nodes using a passive-server synchronization model.

IV. SYSTEM ARCHITECTURE

The proposed system is designed as a multi-cloud distributed architecture consisting of four primary layers:

A. Client Layer

Users interact with the system through web or desktop interfaces. Clients are designed as non-communicating entities — they do not interact directly with each other. All interactions are routed through the application layer, improving security and reducing data conflicts.

B. Application Layer

The central controller processes incoming client requests, handles authentication using JWT (JSON Web Tokens), encrypts data via TLS/SSL, validates inputs, and routes requests to the appropriate cloud storage services.

C. Consistency Management Layer

This layer maintains data synchronization across all cloud nodes through controlled replication mechanisms. Passive servers store data replicas and synchronize updates only when required, reducing communication overhead. Any update is propagated to all replicas in a structured manner, ensuring eventual consistency without compromising performance.

D. Multi-Cloud Storage Layer

Data is stored across three cloud providers — AWS S3, Azure Blob Storage, and Google Cloud Storage — provisioned using Terraform. Each provider maintains a replica, ensuring high availability, redundancy, and fault tolerance. If one cloud provider fails, data remains accessible from the remaining nodes.

V. METHODOLOGY

A. Infrastructure as Code with Terraform

Terraform is used as the core provisioning engine. A single HCL configuration file declares providers for AWS, Azure, and GCP, and defines all storage resources. The *terraform init* command initializes providers, *terraform plan* previews changes, and *terraform apply* executes provisioning. Infrastructure state is tracked in Terraform state files, enabling idempotent and repeatable deployments.

B. Multi-Cloud Storage Provisioning

The following resources are provisioned across three cloud platforms through the unified Terraform configuration:

- AWS: S3 bucket (*adhi-terraform-aws-bucket-001*) in *ap-south-1* region with AES256 server-side encryption.
- GCP: Cloud Storage bucket (*adhi-terraform-gcp-bucket-448815*) in *ASIA-SOUTH1* with uniform bucket-level access enabled.
- Azure: Resource group, storage account (*adhistorageacct123*), and Blob container (*adhi-container*) in Central India with LRS replication.

C. Data Consistency Mechanism

A client-centric consistency model is implemented. When a user performs an upload or update operation, the change is propagated to all cloud replicas in a structured sequence. Conflict detection checks ensure that simultaneous updates do not create data divergence. Passive servers maintain replicas and perform synchronization only on demand, reducing unnecessary API calls and network overhead.

D. Authentication and Security

All client requests are authenticated via JWT-based mechanisms. Data in transit is encrypted using TLS/SSL. Cloud resources are configured with strict access policies — private containers on Azure, uniform bucket-level access on GCP, and bucket owner-enforced permissions on AWS. Version control of Terraform configurations is maintained through Git/GitHub.

VI. MODULE DESCRIPTION

The system is organized into eight functional modules, each with a distinct responsibility:

Module	Responsibility
Client Interface Module	Entry point for user operations (upload, retrieve, update, delete) via secure HTTPS communication.
Authentication & Security Module	Verifies user identity via JWT, encrypts connections, and prevents unauthorized access.
Data Processing Module	Validates, formats, and preprocesses incoming requests before forwarding to storage.
Consistency Management Module	Ensures data synchronization across all cloud nodes through controlled replication and conflict resolution.
Passive Server Module	Stores replicated data and synchronizes updates only when required, minimizing communication overhead.
Multi-Cloud Storage Module	Stores data across AWS S3, Azure Blob, and GCP Cloud Storage for redundancy and fault tolerance.
Database Module	Maintains metadata, user credentials, replication logs, and consistency records (MySQL / MongoDB).
Summary Module	Consolidates operation results and presents storage status, replication details, and synchronization logs to users.

Table I: System Module Description

VII. TOOLS AND TECHNOLOGIES

Category	Component	Technology / Specification
Hardware	Device / Processor / RAM / Storage	Laptop/Desktop, Multi-core CPU (i5+), 8 GB RAM, 256 GB+
Backend Language	IaC Language	HCL (HashiCorp Configuration Language)
IaC Framework	Provisioning Tool	Terraform v1.14.4
Cloud Providers	AWS / Azure / GCP	AWS SDK, Azure SDK, Google Cloud SDK
Object Storage	Storage Services	AWS S3, Azure Blob Storage, Google Cloud Storage
Database	Metadata & Logs	MySQL / MongoDB
Security	Encryption / Auth	SSL/TLS, JWT, AES256 (AWS S3)
Version Control	Source Management	Git, GitHub

Table II: Tools and Technologies Used

VIII. SYSTEM IMPLEMENTATION

A. Terraform Provider Configuration

The Terraform configuration block declares three required providers — *hashicorp/aws* (v~5.0), *hashicorp/google* (v~4.0), and *hashicorp/azurerm* (v~3.0). Individual provider blocks configure the AWS region (*ap-south-1*), GCP project (*crud-448815*, region *asia-south1*), and Azure features.

B. AWS S3 Bucket

An S3 bucket named *adhi-terraform-aws-bucket-001* is provisioned in the Asia Pacific (Mumbai) region with the environment tag set to Dev. AES256 server-side encryption is automatically enforced by AWS upon bucket creation.

C. Google Cloud Storage Bucket

A GCP storage bucket named *adhi-terraform-gcp-bucket-448815* is provisioned in ASIA-SOUTH1 with uniform bucket-level access enabled and *force_destroy* set to true for clean teardown. Storage class defaults to STANDARD.

D. Azure Blob Storage

An Azure resource group (*adhi-terraform-rg*) is created in Central India. A storage account (*adhistorageacct123*) with Standard tier and LRS replication is provisioned, followed by a private Blob container (*adhi-container*).

E. Data Replication Flow

When a user submits a data upload request, the client interface module forwards it securely to the application layer. After JWT authentication and TLS encryption, the data processing module validates and formats the payload. The consistency management module then triggers parallel writes to all three cloud storage nodes via their respective SDKs. Passive servers log the replication event and verify consistency across all replicas. If one cloud provider is unavailable, the system reroutes to the remaining nodes and queues the failed replica for synchronization upon recovery.

IX. RESULTS AND DISCUSSION

The system was successfully deployed and validated across all three cloud platforms. The Terraform state file confirmed provisioning of one AWS S3 bucket, one GCP Cloud Storage bucket, and one Azure Blob container with private access. Sample output screenshots confirmed resource creation on the AWS S3 console (*ap-south-1*, April 27, 2026), Azure Blob Storage (Central India, StorageV2), and GCP Cloud Storage (*asia-south1*, Standard class).

Performance analysis compared the proposed multi-cloud system against a baseline single-cloud architecture across four key metrics:

Metric	Single-Cloud System	Proposed Multi-Cloud System
Response Time	Higher (centralized load)	Reduced (distributed processing)
Availability	Moderate	High (data replication across 3 clouds)
Fault Tolerance	Low (single point of failure)	High (failover to alternate cloud)
Scalability	Limited	High (add cloud nodes seamlessly)

Table III: Performance Comparison — Existing vs. Proposed System

Security testing confirmed successful blocking of unauthorized access attempts. Authentication mechanisms ensured only valid users could access data, and encryption protected all transmissions. Fault tolerance was validated by simulating the unavailability of one cloud provider — the system successfully retrieved data from alternate nodes without interruption, demonstrating the effectiveness of the multi-cloud replication strategy.

X. DISCUSSION

The use of passive servers significantly reduces system overhead. Since synchronization occurs only on demand, unnecessary API calls and network traffic are avoided, improving overall efficiency. The non-communicating client design simplifies the architecture and reduces the risk of data conflicts and unauthorized inter-client interactions.

Terraform's declarative approach eliminates configuration drift across environments. The state file mechanism ensures that the real-world cloud infrastructure always matches the declared configuration. Version control of Terraform files through Git enables auditability and rollback capabilities, supporting DevOps best practices.

The primary challenges of the proposed system include increased management complexity when integrating multiple provider APIs and slightly higher operational costs compared to single-cloud deployments. However, the improvements in reliability, availability, and disaster recovery justify the additional overhead for production-grade systems.

XI. LIMITATIONS

- The system is validated in a development environment; real-world deployment requires additional testing with production-scale workloads.
- Multi-cloud deployment increases operational costs due to egress charges and cross-cloud data transfer fees.
- Managing provider-specific API changes may require periodic Terraform configuration updates.
- The current implementation does not include AI-based dynamic cloud selection or automated cost optimization.

XII. FUTURE SCOPE

- **AI-Based Optimization:** Integrate artificial intelligence to dynamically optimize data placement, replication strategies, and cloud provider selection based on cost and latency.
- **CI/CD Pipeline Integration:** Automate Terraform deployments through continuous integration pipelines for zero-downtime infrastructure updates.
- **Kubernetes Integration:** Extend the framework to provision compute and networking resources, enabling complete cloud-native application deployment.
- **Advanced Security:** Integrate blockchain-based audit trails or zero-trust architecture for enhanced data protection.
- **Real-Time Monitoring:** Add dashboards for live infrastructure health, cost tracking, and anomaly detection across cloud providers.

XIII. CONCLUSION

This paper presented a comprehensive Multi-Cloud Object Storage Provisioning system using Terraform, successfully deploying storage resources on AWS S3, Azure Blob Storage, and Google Cloud Storage through a single, unified IaC configuration. The system demonstrates that Terraform-based automation significantly simplifies multi-cloud infrastructure management, eliminates manual configuration errors, and provides a scalable, repeatable deployment process.

The proposed data consistency architecture using passive servers and non-communicating clients ensures reliable synchronization without centralized coordination overhead. Performance evaluation confirmed improvements in fault tolerance, availability, response time, and scalability over traditional single-cloud systems. Security mechanisms including JWT authentication, TLS encryption, and provider-level access policies successfully protected all data operations.

The project underscores the critical importance of IaC tools in achieving efficient, secure, and maintainable multi-cloud deployments, providing a strong foundation for future research in AI-driven cloud optimization, automated CI/CD pipelines, and advanced multi-cloud security architectures.

REFERENCES

- [1] M. Armbrust et al., "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
- [2] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," NIST Special Publication 800-145, 2011.
- [3] HashiCorp, "Terraform Documentation," <https://developer.hashicorp.com/terraform/docs>, 2024.
- [4] N. Mhaisen and Q. M. Malluhi, "Data Consistency in Multi-Cloud Storage Systems with Passive Servers," *IEEE Access*, 2025.
- [5] J. Zheng et al., "Intent-Based Multi-Cloud Storage Management Powered by a Fine-Tuned Large Language Model," *IEEE Access*, 2025.
- [6] Raju P. B., K. Chandrasekaran, and V. Vijayakumar, "Efficient Multi-Cloud Storage Using Online Dynamic Replication," *IEEE Access*, 2023.
- [7] D. C. Diningrat, Suhardi, and B. Rahardjo, "Security Issues in Multi-Cloud: A Systematic Literature Review," *IEEE Access*, 2025.
- [8] M. A. Altahtat, T. Daradkeh, and A. Agarwal, "Optimized Encryption-Integrated Strategy for Containers Scheduling in Multi-Cloud Data Centers," *IEEE Access*, 2024.

- [9] S. Fugkeaw et al., "FPRESSO: Fast and Privacy-Preserving SSO Authenticate with Dynamic Load Balancing for Multi-Cloud-Based Web Applications," *IEEE Access*, 2024.
- [10] S. Khan, A. Raza et al., "A Comprehensive Review on Scaling Machine Learning Workflows Using Cloud Technologies and DevOps," *IEEE Access*, 2024.
- [11] G. DeCandia et al., "Dynamo: Amazon's Highly Available Key-Value Store," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 205–220, 2007.
- [12] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *ACM SIGACT News*, vol. 33, no. 2, 2002.
- [13] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality," *Future Generation Computer Systems*, vol. 25, no. 6, 2009.
- [14] E. M. Mendes and J. R. de Almeida Junior, "Hybrid Data Lake Entities Localization Optimization Based on Complex Networks," *IEEE Access*, 2026.
- [15] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *ICLR*, 2015.