A Multi-Cloud, Cloud-Native Storage-as-a-Service (STaaS) Platform: Architecture, Implementation, and Performance Analysis

Harshita Gupta (G24AI2017@iitj.ac.in), Kaushal Kushwaha(G24AI2098@iitj.ac.in), Saransh Punia(G24AI2093@iitj.ac.in), Sarthak Gupta(G24AI2057@iitj.ac.in), and Ankit Kumar Bhatnagar(G24AI2022@iitj.ac.in)

***Abstract*—The proliferation of cloud computing has led enterprises to adopt multi-cloud strategies to mitigate vendor lock-in and leverage best-of-breed services. However, this approach introduces significant operational complexity due to the heterogeneity of storage services, APIs, and security models across providers like Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. This paper presents the design, implementation, and empirical evaluation of a cloud-native Storage-as-a-Service (STaaS) platform that provides a unified abstraction layer over disparate cloud storage backends. The system is architected using a modular microservice design, provisioned via Infrastructure-as-Code (IaC), and automated with event-driven serverless workflows. It supports object, block, and file storage paradigms through a consistent RESTful API, and it enforces security using a hybrid model that combines application-level Role-Based Access Control (RBAC) with native cloud Identity and Access Management (IAM) policies. A quantitative performance analysis, conducted across AWS, GCP, and Azure, demonstrates that the abstraction layer introduces minimal overhead, with API response latencies under 150 ms and JWT validation overhead below 5 ms. The results validate the platform's ability to deliver a scalable, secure, and performant solution that simplifies multi-cloud storage management and enhances application portability.** [1]

***Index Terms*—Cloud Computing, Infrastructure-as-Code (IaC), Microservices, Multi-Cloud, Serverless, Storage-as-a-Service (STaaS).** [3]

I. INTRODUCTION

THE ADVENT of cloud computing has fundamentally reshaped enterprise IT, catalyzing a paradigm shift from rigid, on-premise infrastructure to dynamic,

on-demand services. This transition is particularly pronounced in data storage, where traditional, hardware-bound systems like Storage Area Networks (SANs) and Network-Attached Storage (NAS) are increasingly ill-suited for the demands of modern, cloud-native applications. Legacy storage models, characterized by manual provisioning, fixed capacity planning, and tight coupling with compute resources, struggle to accommodate the elastic, ephemeral, and distributed nature of microservice architectures. They exhibit inflexibility in handling bursty workloads and multi-tenant access control, often leading to significant operational overhead and performance bottlenecks.[1]

In response, enterprises are progressively adopting multi-cloud strategies, seeking to avoid vendor lock-in, optimize costs, ensure high availability, and leverage region-specific services from leading providers such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. While strategically advantageous, this approach introduces a formidable technical challenge: the profound heterogeneity across cloud platforms. Each provider exposes a unique set of storage services with distinct APIs, access protocols, performance characteristics, and security frameworks. This lack of standardization creates significant operational friction, complicates application development, and severely hinders the portability of stateful workloads between cloud environments. Developers and operations teams are forced to contend with a fragmented ecosystem, writing provider-specific code and managing disparate security policies, which negates many of the agility benefits that the cloud promises.

This paper addresses this challenge by formally defining and implementing a cloud-native Storage-as-a-Service (STaaS) platform. This platform acts as an intelligent abstraction layer, creating a unified, API-driven interface for provisioning, accessing, and managing storage resources across multiple clouds. The architectural approach itself represents a study in managing the trade-offs inherent in multi-cloud systems. The creation of any abstraction layer introduces a new system that must be engineered, deployed, and secured—an "abstraction tax" paid to gain the benefits of portability and simplified management. A central goal of this research is to demonstrate that, with a well-conceived architecture, this tax can be rendered minimal.

The contribution of this paper is threefold. First, it presents a vendor-agnostic system architecture founded on cloud-native principles,

including stateless microservices, declarative Infrastructure-as-Code (IaC), and event-driven automation. Second, it details a robust, hybrid security model that federates application-level authorization with the native Identity and Access Management (IAM) services of each cloud provider, ensuring fine-grained and auditable access control. Third, it provides a quantitative performance analysis that empirically validates the low overhead of the abstraction layer, confirming its viability for real-world, performance-sensitive applications. This work fills a notable gap in existing literature by offering a reproducible, technical implementation and cross-platform evaluation of a unified STaaS system.

II. RELATED WORK

The concept of STaaS has evolved alongside the maturation of cloud computing. This section reviews the foundational technologies and existing research that provide the context for this work, identifying the specific gap this paper aims to address.

A. Foundations of Distributed Storage

The principles underlying modern cloud storage can be traced back to seminal distributed file systems. The Google File System (GFS) was a pivotal development, designed for fault tolerance, scalability, and high-aggregate throughput by managing massive datasets on clusters of commodity hardware. Concurrently, the MapReduce programming model provided a framework for processing these large datasets in parallel. These systems established the core tenets of distributing data and computation that are now fundamental to Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) offerings. This initial wave of innovation, while groundbreaking, occurred within proprietary ecosystems, setting the stage for subsequent commercial fragmentation.

B. Evolution of Cloud Storage Paradigms

Following the success of these foundational systems, major cloud providers developed their own proprietary, yet highly scalable, storage services. This led to a fragmentation of the market, but also to the refinement of distinct storage paradigms tailored to different use cases :

- **Object Storage:** Services like Amazon S3 , Azure Blob Storage , and Google Cloud Storage (GCS) have become the de facto standard for unstructured data. They are prized for their virtually limitless scalability, high durability (often promising 11 nines), and rich metadata capabilities, making them ideal for data lakes, backups, and media hosting.
- **Block Storage:** Services such as Amazon Elastic Block Store (EBS),

Azure Disk Storage, and Google Persistent Disks provide raw block-level storage volumes that can be attached to virtual machines. They are designed for high-performance, low-latency workloads like transactional databases and file systems that require persistent storage.

- **File Storage:** Fully managed network file systems like Amazon Elastic File System (EFS), Azure Files, and Google Filestore offer shared file access using standard protocols like NFS or SMB. They are commonly used for "lift-and-shift" migrations of legacy applications and for use cases requiring a shared file system across multiple compute instances.

C. Storage in Containerized Environments

The rise of container orchestration platforms, particularly Kubernetes, created a strong demand for a standardized method of managing persistent storage for stateful applications. This need for standardization drove the development of the Container Storage Interface (CSI), a specification from the Cloud Native Computing Foundation (CNCF). CSI defines a standard interface for container orchestrators to communicate with storage systems. Cloud providers have developed CSI drivers for their respective storage services, enabling Kubernetes to dynamically provision, attach, and manage volumes from S3, EBS, GCS, and others in a vendor-neutral way. Research has highlighted the critical role of CSI in enabling storage portability and automation, aligning with modern DevOps and GitOps workflows.

This progression—from proprietary innovation (GFS) to commercial fragmentation (S3, GCS) and finally to a standardizing abstraction (CSI)—reveals a recurring cycle in distributed systems. The STaaS platform presented in this paper can be viewed as an application-level extension of the same principle that motivated CSI at the infrastructure level. Just as CSI provides a unified interface for Kubernetes to orchestrate storage, this platform provides a unified API for developers and applications to consume it.

D. Security and Cost Management in STaaS

Security in multi-tenant cloud storage environments is a well-studied problem, with research focusing on threats and countermeasures related to data isolation and access control. Concurrently, cost optimization has become a critical concern, leading to studies on tiered storage strategies (hot, cold, archive) and the economic trade-offs of different usage-based billing models. While cloud providers offer native tools for security and cost

management, integrating them into a coherent multi-cloud strategy remains a significant challenge.

E. Research Gap

Despite the extensive body of work on cloud storage, a gap persists in the literature regarding comprehensive, empirical studies that detail the design and evaluate the performance of a unified STaaS abstraction layer across AWS, Azure, and GCP. Many works focus on a single provider or discuss multi-cloud strategies at a high level. This paper directly addresses this gap by presenting a detailed technical implementation and a cross-platform performance analysis of a functional multi-cloud STaaS prototype.

III. SYSTEM ARCHITECTURE AND DESIGN

The system is architected based on cloud-native principles to deliver a scalable, resilient, and secure STaaS platform. The design prioritizes modularity and abstraction to effectively manage the heterogeneity of the underlying cloud providers.

A. Overall Architecture

The platform employs a modular microservice architecture, separating the control plane, which handles API requests and orchestration, from the data plane, which involves direct data transfer between the client and the cloud provider's storage backend. This separation is crucial for scalability and security, as sensitive data does not need to pass through the central application services. Fig. 1 illustrates the high-level architecture.

The core of the system is a set of stateless RESTful API services, containerized using Docker. Being stateless allows these services to be horizontally scaled on demand using container orchestration platforms like Kubernetes or managed services like AWS ECS. This design adheres to fundamental cloud-native principles:

- **Dynamic Provisioning:** Storage resources like buckets and volumes are not pre-allocated but are provisioned automatically via API calls to the respective cloud providers, managed through Infrastructure-as-Code.
- **Decoupled Components:** Services for authentication, storage interaction, and event processing are designed as independent microservices, enabling them to be developed, deployed, and scaled separately.
- **Observability:** The architecture is designed for comprehensive monitoring, with hooks for centralized logging, metrics collection (e.g., using Prometheus), and tracing to provide deep visibility into system performance and health across all providers.

B. API Design for Multi-Cloud Abstraction

A well-defined API is the cornerstone of the abstraction layer. The system exposes a versioned RESTful API that conforms to the OpenAPI 3.0 specification, ensuring clarity, discoverability, and compatibility with a wide range of client tools and libraries. The API provides a single, consistent interface for all storage operations, masking the provider-specific implementation details from the client. Key characteristics of the API design include :

- **Platform Abstraction:** Endpoints like /upload or /download accept a parameter indicating the target cloud but otherwise present a uniform interface, handling the translation to the appropriate backend SDK call (e.g., S3 PutObject, GCS Upload, Azure Blob UploadBlockBlob).
- **Standardization:** The API uses standard HTTP methods (POST, GET, DELETE), status codes, and error formats, simplifying client-side development.
- **Automation Support:** The API is designed to be consumed not only by end-users via a UI but also by automated scripts and CI/CD pipelines, facilitating GitOps workflows.

TABLE II details the primary endpoints exposed by the STaaS platform's control plane. All protected routes require a valid JWT bearer token for authentication.

TABLE II. API ENDPOINT DEFINITIONS

Method	Endpoint	Description	Authentication
POST	/register	Register a new user account.	No
POST	/login	Authenticate a user and issue a JWT.	No
POST	/upload	Upload a file to a specified object storage backend.	Yes
GET	/files	Retrieve a list of the user's uploaded files.	Yes
GET	/download/<file_id>	Generate a pre-signed URL to download a specific file.	Yes
DELETE	/delete/<file_id>	Delete a specific file from object storage.	Yes

C. Unified Object Storage Backend

Object storage was selected as the primary backend paradigm due to its inherent advantages for cloud-native applications: virtually unlimited scalability, exceptional durability, and cost-efficiency through tiered storage classes (e.g., AWS S3 Glacier, Azure

Archive). The platform integrates with the following providers:

- **Amazon S3, Azure Blob Storage, and Google Cloud Storage:** These fully managed services form the core production backends, offering high availability and rich feature sets.
- **MinIO:** A lightweight, S3-compatible object storage server used for local development and testing. Its S3 compatibility allows developers to write and test code locally without incurring cloud costs, ensuring a consistent experience before deploying to a production cloud environment.

To ensure data privacy and security in a multi-tenant environment, the system implements a strict data isolation strategy. Upon registration, each user is programmatically assigned a dedicated storage bucket or a unique prefix within a shared bucket. The naming convention follows a standard format, such as st-user-<user_id>, which is then used to construct access control policies. This ensures that, by default, users can only access their own data, preventing cross-tenant data leakage.

D. Hybrid Authentication and Authorization Framework

Securing a multi-cloud application requires a sophisticated approach that balances portability with the robust security features of native cloud platforms. The system employs a hybrid authentication and authorization model to achieve this balance.

The authentication workflow is stateless and based on JSON Web Tokens (JWT). Upon successful login, the authentication service issues a signed JWT containing user claims, such as user_id and role. This token is sent with every subsequent API request and is validated by middleware at the API gateway, ensuring that only authenticated users can access protected endpoints.

The authorization strategy is where the hybrid model's power becomes apparent. It operates on two levels:

1. **Application-Level RBAC:** The application itself enforces high-level roles. For example, it can distinguish between a standard_user and an administrator, granting different API-level permissions based on the role claim in the JWT.
2. **Cloud-Native IAM Enforcement:** Instead of granting the application's service account broad permissions, the system leverages cloud-native IAM services to generate short-lived, narrowly-scoped credentials for each specific operation. For example, when a user requests to upload a file to AWS, the application authenticates the user

via JWT, then makes a call to the AWS Security Token Service (STS) to assume a role. The policy attached to this role explicitly restricts access to only the user's designated S3 prefix (e.g., s3://my-bucket/st-user-123/*). The application then returns these temporary credentials (or a pre-signed URL generated with them) to the client, which uses them for the direct upload to S3.

This design pattern is a pragmatic solution to a complex problem. A purely application-level security model would be portable but would require reinventing complex and critical security controls, making it less secure and harder to audit. Conversely, relying solely on cloud IAM would tightly couple the application to provider-specific identity federation mechanisms, sacrificing portability. The hybrid model leverages the best of both worlds: the application handles business logic authorization (who the user is and what their role is), while the cloud provider's mature, battle-tested IAM service handles the final, critical resource-level enforcement. This minimizes the attack surface of the custom application code and maximizes security and auditability.

IV. IMPLEMENTATION AND AUTOMATION

The architectural design was realized through a combination of modern development practices and cloud-native tooling, emphasizing automation at every stage of the lifecycle. The project was implemented using a phased agile methodology, allowing for iterative development and validation across the three major cloud platforms.

A. Infrastructure-as-Code (IaC) for Provisioning

The entire multi-cloud infrastructure was defined and provisioned declaratively using Terraform. This includes all necessary resources, such as S3 buckets, GCS buckets, Azure Storage Accounts, IAM roles and policies, service accounts, and serverless function definitions. Using IaC provides several key advantages:

- **Repeatability:** The entire environment can be torn down and recreated identically in minutes, which is invaluable for testing and disaster recovery.
- **Consistency:** It eliminates configuration drift between development, staging, and production environments, as the code is the single source of truth for the infrastructure's state.
- **Version Control:** Infrastructure changes are managed through pull requests and code reviews, just like application code, providing a full audit trail and enabling GitOps workflows.

B. Event-Driven Serverless Workflows

To enhance automation and operational efficiency, the platform heavily utilizes serverless computing for event-driven processing. Instead of running dedicated servers to handle tasks that occur in response to storage activity, the system leverages provider-native serverless functions. When a file is uploaded to a storage bucket, the provider's eventing service (e.g., Amazon S3 Event Notifications, Azure Event Grid) triggers a corresponding serverless function (AWS Lambda, Azure Functions, or Google Cloud Functions).

This pattern is used for various post-processing tasks, such as extracting metadata, generating thumbnails, scanning for viruses, or logging access events. For instance, a sample post-upload handler implemented in AWS Lambda retrieves the object key and bucket name from the event payload and logs this information to Amazon CloudWatch for auditing purposes.

The benefits of this serverless approach are significant. It reduces operational overhead to near zero, as there are no servers to provision or manage. The functions scale elastically and automatically based on the volume of incoming events, and the pay-per-invocation cost model is highly efficient, eliminating expenses for idle resources.

C. User Interface and Experience

While the platform is primarily API-centric, a lightweight web front-end was developed using HTML5, CSS3, and vanilla JavaScript to provide a user-friendly interface for demonstration and testing. The UI allows users to register, log in, and interact with their isolated storage space. It features a drag-and-drop file upload mechanism with real-time feedback, and functionalities to list, download, and delete files. As shown in Fig. 2, the interface provides a consistent experience for interacting with different cloud backends, such as AWS S3 and Azure Blob Storage, reinforcing the value of the underlying API abstraction. All interactions with the backend are handled asynchronously using the Fetch API, and secure file transfers are facilitated through the use of pre-signed URLs generated by the backend API.

V. EXPERIMENTAL EVALUATION AND RESULTS

A comprehensive testing and validation strategy was executed to empirically evaluate the STaaS platform's correctness, performance, scalability, and security across the supported cloud environments. The evaluation demonstrates that the platform meets its design goals and that the abstraction layer is robust and efficient.

A. Testbed and Methodology

The experimental evaluation was conducted using a multi-faceted approach. Performance and load testing were carried out using industry-standard tools, including Locust and Artillery.io, to simulate concurrent user activity. The test scenarios involved simultaneous file uploads, downloads, and API interactions from geographically distributed clients to measure real-world performance. The backend services were deployed across various containerized environments, including AWS ECS on Fargate, Azure Container Instances (ACI), and Google Cloud Run, to assess performance on different managed platforms.

The key metrics collected during the evaluation were :

- **API Response Latency:** The time taken for the API gateway to process a request and return a response.
- **File I/O Throughput:** The effective data transfer rate for uploads and downloads of varying file sizes (100 KB to 10 MB).
- **Serverless Trigger Latency:** The end-to-end time from a storage event occurring to the corresponding serverless function beginning execution, including any "cold start" overhead.
- **Authentication Overhead:** The additional latency introduced by the JWT validation middleware on each protected API request.

B. Performance and Scalability Analysis

The quantitative results confirm that the STaaS abstraction layer introduces minimal performance overhead while enabling significant operational benefits. TABLE III summarizes the key performance benchmarks observed under nominal load conditions.

TABLE III. PERFORMANCE BENCHMARK SUMMARY

Metric	Observed Value
Mean File Upload Latency (1MB payload)	~120 ms
Mean File Download Latency (1MB payload)	~90 ms
JWT Token Validation Overhead	< 5 ms
Average Serverless Trigger Latency	< 300 ms
Max Stable Concurrent Sessions	100+ users

The data reveals that API endpoints consistently responded in under 150 ms, and the critical JWT validation step added less than 5 ms of overhead per request, demonstrating the efficiency of the security implementation. Serverless workflows, including potential cold starts, exhibited an average trigger latency of under 300 ms, which is well within acceptable limits for asynchronous post-processing tasks.

Load testing demonstrated the platform's scalability. In containerized environments, throughput scaled linearly as more replicas were added, confirming the effectiveness of the stateless microservice design. The underlying object storage services (S3, GCS, Blob) exhibited near-constant latency even under heavy concurrent access, a testament to their highly distributed and replicated internal architectures. This empirical evidence refutes the common concern that an abstraction layer will unacceptably degrade performance. For many user-facing applications, the benefits of multi-cloud flexibility can be achieved without a significant performance penalty.

C. Functional and Security Validation

A rigorous testing regimen confirmed the platform's functional correctness and security posture. Unit tests, written using Python's pytest framework, achieved 94% code coverage across all critical modules, including authentication, file handling, and API logic. Integration tests, automated within a CI/CD pipeline, validated the end-to-end interoperability between the API layer, the different cloud storage backends, and the authentication service. All core features—including user registration, token-based authentication, secure file uploads, and per-user data isolation—were successfully validated across all three cloud platforms, ensuring behavioral parity.

Security validation included static code analysis using tools like Bandit and dynamic vulnerability scanning with OWASP ZAP. The system was tested against common web vulnerabilities, confirming that it was protected against threats like SQL injection (mitigated via ORM) and broken authentication (mitigated via strong JWT enforcement and token expiration policies). All data in transit is protected by HTTPS/TLS, and encryption at rest is enabled on the cloud storage backends.

D. Comparative Analysis

The advantages of the proposed cloud-native STaaS model become stark when compared to traditional, on-premise storage systems. TABLE IV provides a comparative evaluation across several key architectural and operational attributes, summarizing the

value proposition of the platform. The cloud-native approach transforms storage from a static, capital-intensive asset into a dynamic, programmable, and operationally efficient service.

TABLE IV. COMPARATIVE EVALUATION: TRADITIONAL VS. CLOUD-NATIVE STAAS

Feature	Traditional Storage Systems	Cloud-native STaaS (AWS/Azure/GCP)
Provisioning Time	Manual; hours or days	Instantaneous via IaC or API automation
Elastic Scalability	Limited; requires hardware upgrades	Native auto-scaling using managed services
Access Protocols	NFS, SMB, iSCSI (network-mounted)	HTTPS-based RESTful API endpoints
Cost Structure	Capital Expenditure (CAPEX)	Operational Expenditure (OPEX), pay-per-use
User Access Control	OS-level ACLs, network firewalls	Fine-grained IAM policies + app-level RBAC
Deployment Flexibility	On-premise or hybrid; limited portability	Highly portable across regions and providers
Maintenance Overhead	High (patching, backups, monitoring)	Low; offloaded to cloud providers
Disaster Recovery	Manual backup; often with downtime	Automated backups and geo-redundancy

VI. CONCLUSION

This paper has presented the design, implementation, and comprehensive evaluation of a scalable, secure, and portable cloud-native Storage-as-a-Service platform. By building a unified abstraction layer over the distinct storage ecosystems of AWS, Azure, and GCP, the system successfully addresses the critical challenge of multi-cloud heterogeneity. The architecture, founded on the principles of stateless microservices, Infrastructure-as-Code, event-driven serverless automation, and a hybrid security model, provides a robust framework for managing distributed storage resources.

The key accomplishments of this work are significant. The platform demonstrates functional parity across multiple clouds, enforcing secure, multi-tenant data isolation through a combination of application-level logic and native IAM policies. The empirical evaluation confirms that this abstraction is achieved with minimal performance overhead, delivering low-latency I/O and scaling linearly under load. This result is crucial, as it proves that the operational benefits of multi-cloud strategies—such as avoiding vendor lock-in and enhancing resilience—do not have to come at the cost of performance.

Beyond its immediate functional delivery, this project serves as a foundational blueprint for future research and enterprise applications. The architecture is extensible and can be enhanced with more advanced features, such as AI-driven data lifecycle management for automatic storage tiering, integrated billing analytics for usage-based cost allocation, and formal audit logging to meet stringent compliance requirements like GDPR and HIPAA. In essence, this work bridges the gap between traditional, infrastructure-centric storage and the programmable, software-defined paradigms required by modern applications. It affirms the role of STaaS as a vital enabler for the next generation of distributed systems in an increasingly multi-cloud world.

ACKNOWLEDGMENT

The authors would like to thank the Department of Computer Science at the Indian Institute of Technology Jodhpur for providing the resources and support for this project as part of the Postgraduate Diploma in Data Engineering program. [1]

REFERENCES

- Amazon Web Services, Inc., Amazon S3 Documentation. [Online]. Available: https://docs.aws.amazon.com/s3/ [Accessed: Oct. 26, 2023].
- Microsoft Corporation, Azure Blob Storage Documentation. [Online]. Available: https://learn.microsoft.com/en-us/azure/storage/blobs/ [Accessed: Oct. 26, 2023].
- Google LLC, Google Cloud Storage Documentation. [Online]. Available: https://cloud.google.com/storage/docs [Accessed: Oct. 26, 2023].
- M. Armbrust et al., "A view of cloud computing," Commun. ACM, vol. 53, no. 4, pp. 50–58, Apr. 2010.
- S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in Proc. 19th ACM Symp. Oper. Syst. Princ. (SOSP '03), 2003, pp. 29–43.
- J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Commun. ACM, vol. 51, no. 1, pp. 107–113, Jan. 2008.
- Cloud Native Computing Foundation, Cloud Native Definition v1.0. [Online]. Available: https://github.com/cncf/toc/blob/main/DEFINITION.md [Accessed: Oct. 26, 2023].
- N. Chawla, G. S. Aulakh, and S. K. Sood, "A comprehensive review on container-based virtualization for cloud environment," in Proc. Int. Conf. Comput. Commun. Informat. (ICCCI), 2020, pp. 1–7.
- K. Zhang, X. Zhou, and Y. Chen, "A survey on security and privacy of online social networks," IEEE Commun. Surv. Tutor., vol. 14, no. 3, pp. 928–941, Third Quarter 2012.
- F. Liu, P. Shu, and J. Li, "Data-intensive applications in cloud computing: A survey," ACM Comput. Surv., vol. 47, no. 4, article 59, Jul. 2015.
- Amazon Web Services, Inc., AWS Lambda Developer Guide. [Online]. Available: https://docs.aws.amazon.com/lambda/ [Accessed: Oct. 26, 2023].
- Microsoft Corporation, Azure Functions Documentation. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-functions/ [Accessed: Oct. 26, 2023].
- Google LLC, Cloud Functions Overview. [Online]. Available: https://cloud.google.com/functions [Accessed: Oct. 26, 2023].
- Amazon Web Services, Inc., Identity and Access Management (IAM) User Guide. [Online]. Available: https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html [Accessed: Oct. 26, 2023].
- MinIO, Inc., MinIO Documentation. [Online]. Available: https://min.io/docs

[Accessed: Oct. 26, 2023].
- M. Fowler, “Microservices: A definition of this new architectural term,” martinfowler.com, Mar. 25, 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html [Accessed: Oct. 26, 2023].

Works cited

1. Cloud-native Storage-as-a-Service (STaaS).pdf
2. Learn IEEE Research Paper Format Here: A Comprehensive Guide, accessed July 20, 2025, https://www.sharkpapers.com/blog/research-paper-writing-guides/ieee-research-paper-format
3. IEEE General Format - Purdue OWL, accessed July 20, 2025, https://owl.purdue.edu/owl/research_and_citation/ieee_style/ieee_general_format.html
4. pg4-sample-word-template.docx - IEEE PES, accessed July 20, 2025, https://ieee-pes.org/wp-content/uploads/2023/01/pg4-sample-word-template.docx
5. IEEE Conference Template - Overleaf, Online LaTeX Editor, accessed July 20, 2025, https://www.overleaf.com/latex/templates/ieee-conference-template/grfzhhncsfqn