Optimizing Trust Score Algorithms in Zero Trust IAM Systems for Multi-Cloud and Hybrid Cloud Environments

Research Project
MSc Cloud Computing

Urvashi Kamlesh Sardare
Student ID: x22235248@student.ncirl.ie

School of Computing
National College of Ireland

Supervisor: Sai Emani

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Urvashi Kamlesh Sardare
Student ID:	x22235248@student.ncirl.ie
Programme:	MSc Cloud Computing
Year:	2025
Module:	Research Project
Supervisor:	Sai Emani
Submission Due Date:	11/08/2025
Project Title:	Optimizing Trust Score Algorithms in Zero Trust IAM Systems for Multi-Cloud and Hybrid Cloud Environments
Word Count:	7200
Page Count:	20

Signature:	
Date:	26/08/2025

Optimizing Trust Score Algorithms in Zero Trust IAM Systems for Multi-Cloud and Hybrid Cloud Environments

Urvashi Kamlesh Sardare
x22235248@student.ncirl.ie

Abstract

The shift to multi-cloud architectures challenges traditional Identity and Access Management (IAM) by dissolving the network perimeter. This research addresses this through a Zero Trust Architecture (ZTA), implementing a dynamic trust score for consistent IAM in complex AWS and GCP environments. This paper details a practical system leveraging a real-time trust score calculated from contextual signals—including multi-factor authentication (MFA) status, location, and crucial threat intelligence from AWS GuardDuty and GCP Security Command Center. Parallel, cloud-native trust engines were developed using serverless components on both platforms. Evaluation via controlled case studies confirmed the system's efficacy, demonstrating dynamic responses to user context and a critical, security-first reaction to simulated threat alerts. The key finding is that a consistent Zero Trust policy is highly effective across disparate clouds when mapped to native services, providing a tangible blueprint for enhancing multi-cloud security.

1 Introduction

The widespread adoption of multi-cloud and hybrid cloud strategies has shattered traditional, perimeter-based security models, rendering network location an obsolete metric for trust (Julakanti et al., 2022; Chaudhry, 2025). In response, Zero Trust Architecture (ZTA) has emerged, operating on the principle of "never trust, always verify" for every access request, regardless of its origin (Gambo and Almulhem, 2025). However, implementing a cohesive ZTA strategy is complex in multi-cloud environments, where disparate provider services (e.g., AWS, GCP) can lead to inconsistent policy enforcement and security gaps (Emmanni, 2024). This research addresses this challenge by designing and implementing a dynamic trust score. This score serves as a real-time, quantitative measure of confidence for any access request, calculated from multiple contextual signals. Instead of a binary allow/deny decision, the score enables a nuanced, adaptive policy (e.g., Allow, MFA Required, Deny), balancing robust security with user experience. This approach aligns with the increasing significance of dynamic and AI-driven systems in modern security architectures (Ajish, 2024; Gadkari, 2024), providing a practical blueprint for consistent, context-aware access control across heterogeneous clouds.

The central research question guiding this work is: **How can dynamic trust score algorithms be designed, implemented, and optimized within a Zero Trust**

IAM framework to provide consistent and context-aware access control across disparate multi-cloud environments like AWS and GCP?

To answer this question, the following research objectives were established:

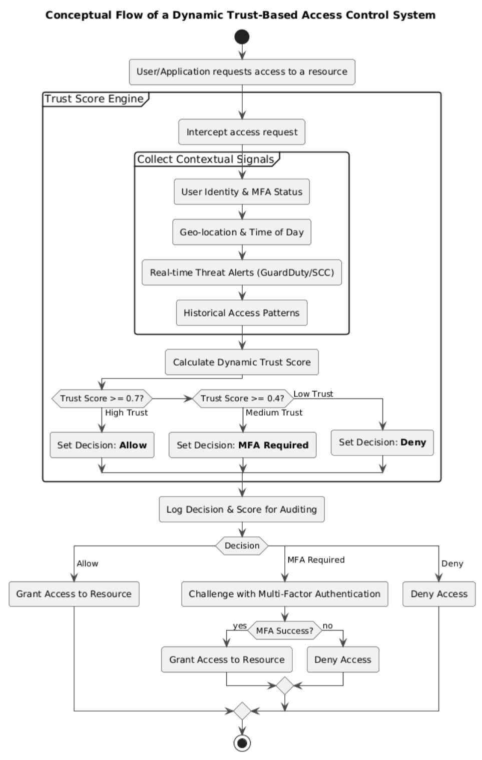
1. To conduct a critical review of the existing academic and industry literature on Zero Trust principles, multi-cloud security challenges, and dynamic access control systems.

2. To design a system architecture for a dynamic trust score engine on both AWS and GCP, leveraging their respective native serverless, database, and security monitoring services.

3. To implement two parallel, proof-of-concept trust evaluation engines, one on AWS using Lambda, API Gateway, DynamoDB, and GuardDuty, and the other on GCP using Cloud Functions, Firestore, and Security Command Center.

4. To develop a unified front-end application to demonstrate and test the interaction with both cloud-native engines from a single interface.

The primary contribution of this research is the development of a practical, hands-on blueprint for a multi-cloud Zero Trust IAM system. It moves beyond theoretical discussion to provide a tangible implementation that directly maps a unified security concept (the dynamic trust score) to the specific, and often different, services of the world's leading cloud providers. This provides valuable insight for practitioners and a solid foundation for future academic work in the field of adaptive, multi-cloud security. A conceptual overview of this adaptive access control flow is shown in Figure 1.



Conceptual Flow of a Dynamic Trust-Based Access Control System
User/Application requests access to a resource
Trust Score Engine
Intercept access request
Collect Contextual Signals
User Identity & MFA Status
Geo-location & Time of Day
Real-time Threat Alerts (GuardDuty/SCC)
Historical Access Patterns
Calculate Dynamic Trust Score
Trust Score >= 0.7?
High Trust
Trust Score >= 0.4?
Medium Trust
Low Trust
Set Decision: Allow
Set Decision: MFA Required
Set Decision: Deny
Log Decision & Score for Auditing
Decision
Allow
MFA Required
Deny
Grant Access to Resource
Challenge with Multi-Factor Authentication
Deny Access
yes
MFA Success?
no
Grant Access to Resource
Deny Access

Figure 1: Conceptual flow of a dynamic trust-based access control system.

The remainder of this report is structured as follows. Section 2 presents a critical review of the related work. Section 3 outlines the Design Science Research methodology employed for this project. Section 4 provides a detailed design specification of the system architecture and its components on both AWS and GCP. Section 5 describes the step-by-step implementation process in a narrative format. Section 6 presents a comprehensive evaluation of the system through a series of case studies and discusses the findings. Finally, Section 7 concludes the report, summarizes the key findings, acknowledges the limitations of the work, and proposes meaningful directions for future research.

2 Literature Review

This section situates the research within the existing body of academic and industry literature. The review is structured thematically to cover the foundational concepts of Zero Trust, the specific challenges of multi-cloud environments, the role of dynamic systems in modern security, and the importance of data governance. This critical analysis identifies the gap in the literature that this project aims to address: the lack of practical, implementation-focused research on building consistent, dynamic access control systems across heterogeneous cloud platforms.

2.1 The Evolution to Zero Trust Architecture (ZTA)

The concept of Zero Trust marks a significant departure from traditional, perimeter-focused security models. The classic "castle-and-moat" approach, which assumes that everything inside the network perimeter is trusted, has become untenable in the face of cloud adoption, remote work, and sophisticated cyber threats (Sarkar et al., 2022). The literature extensively documents this shift, positioning ZTA as the strategic response to the dissolution of the network perimeter.

Gambo and Almulhem (2025) provide a systematic literature review of ZTA, identifying its core principles as continuous verification, limiting the "blast radius" of breaches through micro-segmentation, and automating context collection. The authors stress that ZTA is not merely a product but a holistic strategy based on user and device identity. Similarly, Liu et al. (2024) analyze the research field and note its applicability beyond corporate networks to areas like the Internet of Things (IoT), stating that the core tenet is to enforce access policies based on confidence in the user and device, not their network location. This project directly builds on these principles by designing a system where "confidence" is explicitly quantified by a dynamic trust score, which serves as the mechanism for continuous verification.

2.2 Multi-Cloud and Hybrid Cloud Security Challenges

While ZTA provides the guiding philosophy, its implementation in multi-cloud environments introduces significant practical hurdles. Julakanti et al. (2022) explore strategies for managing security in these complex settings, identifying policy inconsistency, visibility gaps, and increased management overhead as primary challenges. They argue that a successful multi-cloud security strategy requires a centralized management plane that can abstract the underlying differences between cloud providers. This research directly engages with this problem by prototyping a system that, while using cloud-native components, is conceptually unified.

Emmanni (2024) specifically discusses implementing ZTA in hybrid cloud environments, stressing the need for seamless integration between on-premises and cloud security controls. The challenge lies in ensuring policies remain consistent regardless of where a resource resides. Furthermore, Chaudhry (2025) reinforces the idea that the abstraction and dynamism of the cloud necessitate equally dynamic and abstract security controls. The trust score algorithm developed in this project is a direct attempt to create such a control—an abstract representation of risk that can be applied consistently, even when the underlying services (like AWS GuardDuty and GCP Security Command Center) differ.

Table 1: Summary of Key Characteristics in Related Works

Reference	**Focus / Objective**	**Primary Strategy**	**Key Concepts / Technologies**	**Implementation Type**
Gambo and Almulhem (2025)	To systematically review ZTA literature and define its core principles.	Literature Review	Continuous verification, micro-segmentation, identity-centric control.	Theoretical (Systematic Review)
Julakanti et al. (2022)	To explore security management strategies for multi-cloud and hybrid environments.	Conceptual Framework	Centralized management, policy abstraction, visibility.	Conceptual Framework
Ajish (2024)	To review the significance of AI in enhancing ZTA implementations.	Literature Review	AI/ML for anomaly detection, behavioral analysis, adaptive access.	Theoretical (Review)
Nzeako and Shittu (2024)	To propose the use of AI for automating and improving IAM processes.	Conceptual Proposal	AI-driven user authentication, automated access-control decisions.	Conceptual Proposal
Jim (2024)	To describe Cloud Security Posture Management (CSPM) as a tool for automation.	Descriptive Overview	CSPM, automated risk detection, compliance checking.	Conceptual Overview

2.3 Dynamic Systems and AI in Modern Security

The static nature of traditional access control lists (ACLs) is ill-suited to the dynamic cloud environment. As highlighted in Table 1, the literature increasingly points towards Artificial Intelligence (AI) to create more adaptive security postures. Ajish (2024) provides a comprehensive review, arguing that AI can enhance ZTA by learning normal behavior patterns to detect anomalies. Similarly, Gadkari (2024) suggests that machine learning models can calculate risk scores more effectively than static, rule-based systems.

Nzeako and Shittu (2024) focus on leveraging AI for enhanced IAM, proposing its use to automate and improve authentication. While the algorithm implemented in this project is rule-based for transparency and feasibility, it is designed as a precursor to such an AI-driven system. The trust score concept provides the perfect framework into which a machine learning model's output (e.g., an anomaly score) could be integrated as another signal. This aligns with work by Jalali and Mansouri (2024), who survey fuzzy systems that deal with decision-making under uncertainty. Their work on non-binary logic supports the trust score's nuanced approach, moving away from a simple "allow/deny" to a spectrum of possibilities.

2.4 Summary and Gap Analysis

The existing literature provides a strong foundation, establishing the necessity of Zero Trust, outlining multi-cloud challenges, and pointing towards dynamic, AI-driven solutions. However, a significant gap exists between high-level strategic discussions and practical, hands-on implementation guides. As illustrated in Table 1, much of the academic work is conceptual or review-based. While many papers discuss what a multi-cloud ZTA *should* look like (Julakanti et al., 2022), few provide a detailed, replicable blueprint of *how* to build one using the native tools of different cloud providers. There is a scarcity of research that demonstrates the side-by-side mapping of a consistent security policy (like a dynamic trust score) onto the disparate services of AWS and GCP.

This research project aims to fill this specific gap. By designing, building, and evaluating a parallel trust score system on both AWS and GCP, it moves from the theoretical to the practical. It provides a concrete example of how to translate the principles of ZTA into working code and infrastructure, offering a valuable contribution to both academic understanding and industry practice.

3 Methodology

This project adopts the Design Science Research (DSR) methodology, a framework ideal for creating and evaluating novel technological artifacts designed to solve real-world problems. As the project's goal is to design, build, and evaluate a multi-cloud Zero Trust IAM system, DSR provides the necessary structure for a rigorous and repeatable investigation. The process moves from problem awareness through to the creation and validation of the artifact.

3.1 Research Process and Objectives

The project followed the established DSR lifecycle. The process began with **Problem Identification**, defining the challenge of inconsistent, context-unaware IAM in multi-cloud environments. This led to defining clear **Objectives for a Solution**, requiring the artifact to be:

- **Dynamic:** Calculate privileges in real-time.

- **Context-Aware:** Incorporate multiple signals like location, time, MFA, and threat intelligence.
- **Cloud-Native:** Use serverless services on AWS and GCP.
- **Actionable:** Produce a clear decision (Allow, Deny, MFA Required).
- **Auditable:** Log all decisions for compliance.

The core **Design and Development** phase involved the iterative implementation of the system using Python and serverless technologies, as detailed in Sections 4 and 5. Subsequently, the system's utility was shown in the **Demonstration** phase via a Flask web application. The artifact was then validated in the **Evaluation** phase through controlled case studies that tested its responsiveness to various risk scenarios (Section 6). This research report serves as the final **Communication** phase, documenting the process and findings. Data collection was empirical, consisting of system-generated outputs from scripted tests and logs from AWS CloudWatch and GCP Cloud Logging to verify execution and auditability.

3.2 Data Collection and Analysis

The "data" in this research consists primarily of the outputs generated by the implemented system itself. Data collection was a direct and empirical process:

- **System Test Results:** For the evaluation, specific data structures representing different user access scenarios were crafted. These were sent as HTTP requests to the system's API endpoints. The system's response, containing the calculated trust score, the final decision, and the status of threat alerts, was the primary data collected for each test case.
- **Cloud Provider Logs:** System execution logs from AWS CloudWatch and GCP Cloud Logging were reviewed to confirm function execution and troubleshoot issues. Access logs from AWS CloudTrail and the persistent databases (DynamoDB and Firestore) were also used to verify that the auditability requirement was met.

Data analysis was qualitative and comparative. The observed outputs from the system were compared against the expected outputs based on the logic of the trust score algorithms. The performance of the AWS and GCP systems were compared to assess the consistency of the multi-cloud implementation. The analysis focused on verifying that the system behaved correctly and logically in response to the various input signals, thereby validating its design.

3.3 Tools and Technologies

The implementation and evaluation of the artifact relied exclusively on industry-standard cloud services and open-source software. The choice of tools was driven by the objective to create a cloud-native, serverless solution. Table 2 provides a comprehensive list of the key tools and technologies used in this research.

This methodological approach ensures that the research is not only theoretically grounded but also practically relevant, resulting in a well-documented and rigorously evaluated artifact that directly addresses the stated research problem.

Table 2: Key Tools, Languages, and Cloud Services Used.

Category	Tool/Service	Purpose
Programming	Python 3.11	Core logic for trust algorithms and backend functions.
Web Framework	Flask	Development of the unified front-end demonstrator.
Automation	Cloud Provider CLIs/SDKs	Programmatic provisioning and management of cloud resources.
AWS Compute	AWS Lambda	Serverless execution of the trust score function.
AWS API	API Gateway (HTTP)	Public endpoint for the AWS Lambda function.
AWS Database	DynamoDB	NoSQL database for logging access decisions.
AWS Security	GuardDuty	Real-time threat intelligence signal source.
AWS Monitoring	CloudWatch	Logging, metrics, and alerting.
AWS IAM	IAM	User, role, and policy management.
GCP Compute	Cloud Functions (2nd Gen)	Serverless execution of the trust score function.
GCP Database	Firestore	NoSQL document store for logging access decisions.
GCP Security	Security Command Center	Real-time threat intelligence signal source.
GCP Monitoring	Cloud Logging	Function execution logging.
GCP IAM	IAM	Service account, role, and policy management.
Testing	HTTP Request Tools	Sending scripted HTTP requests to API endpoints.
Version Control	Git / Local	Management of source code.

4 Design Specification

This section presents the detailed architectural design of the multi-cloud Zero Trust IAM system. The design translates the abstract principles of Zero Trust and dynamic trust scoring into a concrete blueprint for implementation on AWS and GCP. The architecture is modular, serverless, and leverages cloud-native services to create two parallel yet conceptually aligned trust evaluation engines.

4.1 Overall System Architecture

The high-level architecture is designed as a distributed, event-driven system. As illustrated in Figure 2, an access request, originating from a user or application, is directed to a central point of control. In this proof-of-concept, this is represented by the unified Flask application, which acts as a Policy Decision Point (PDP) proxy. Based on the target resource, the request is then forwarded to the appropriate cloud-native Policy

Enforcement Point (PEP), which is the API endpoint in either AWS or GCP.

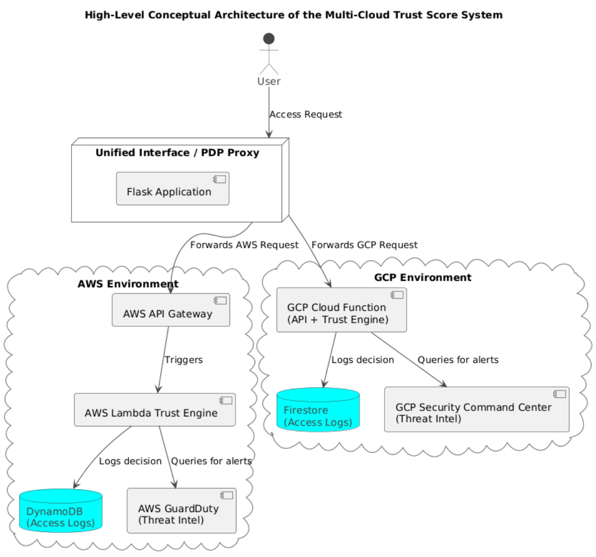


Figure 2: High-Level Conceptual Architecture of the Multi-Cloud Trust Score System.

Each cloud environment contains a self-contained trust engine responsible for:

1. Receiving the access request context via an API.
2. Enriching the context with real-time threat intelligence from native security services.
3. Calculating a numerical trust score based on a predefined algorithm.
4. Mapping the score to a concrete access decision.
5. Logging the entire transaction for auditing and future analysis.

4.2 The Dynamic Trust Score Model

The core of the system is the trust score model. For this research, a transparent, rule-based, weighted scoring model was chosen. This approach, while simpler than a machine learning model, offers clear interpretability, making it ideal for a proof-of-concept where understanding the ”why” behind a decision is paramount. The model is based on several key contextual signals:

- **Multi-Factor Authentication (MFA) Status:** A boolean signal indicating whether the user has MFA enabled for their session or account. This is a strong indicator of identity assurance.
- **Geographic Location:** The source IP’s geographic location, compared against a predefined list of trusted regions (e.g., ”Ireland”, ”Sweden” for AWS; ”US”, ”EU” for GCP). Access from unusual locations introduces risk.
- **Time of Day:** The hour of the access request, compared against standard business hours (e.g., 9:00 to 18:00). Off-hours access can be a sign of anomalous activity.

- **Access History Score:** A floating-point number (0.0 to 1.0) representing the user's past behavior. In this implementation, it is a static input, but it is designed to be replaced by a dynamically calculated score based on historical logs. A high score indicates a history of successful, low-risk access.
- **Real-time Threat Alerts:** A boolean signal indicating the presence of active, relevant security findings from the cloud provider's threat detection service (AWS GuardDuty or GCP Security Command Center). This is the most critical signal, representing a known, immediate threat.

The final trust score is used to make a three-tiered access decision:

- **Score $\geq$ 0.7: Allow** - High confidence, grant access.
- **0.4 $\leq$ Score $<$ 0.7: MFA Required** - Medium confidence, require step-up authentication.
- **Score $<$ 0.4: Deny** - Low confidence, block access.

These specific values were chosen for this proof-of-concept to create distinct, demonstrable risk tiers. A threshold of 0.7 represents a high degree of confidence, requiring multiple positive signals to be met. The 0.4 threshold was set as a critical cutoff, below which the aggregation of multiple risk factors (or a single high-severity threat) indicates an unacceptable level of risk. In a production environment, these thresholds would be fine-tuned based on organizational risk appetite and empirical analysis of access patterns."

$$\text{Score}_{\text{AWS}} = \text{Base} + w_{\text{mfa}} \cdot S_{\text{mfa}} + w_{\text{loc}} \cdot S_{\text{loc}} + w_{\text{time}} \cdot S_{\text{time}} + w_{\text{hist}} \cdot S_{\text{hist}} + w_{\text{threat}} \cdot S_{\text{threat}} \tag{1}$$

$$\begin{aligned}\text{Score}_{\text{GCP}} = {} & (\text{Base} - (P_{\text{mfa}} \cdot S_{\text{mfa}} + P_{\text{loc}} \cdot S_{\text{loc}} + P_{\text{time}} \cdot S_{\text{time}} + P_{\text{threat}} \cdot S_{\text{threat}})) \\ & + B_{\text{hist}} \cdot S_{\text{hist}} \end{aligned} \tag{2}$$

4.3 AWS Component Design

The AWS implementation is built entirely on serverless and managed services to minimize operational overhead. The architecture is depicted in Figure 3.

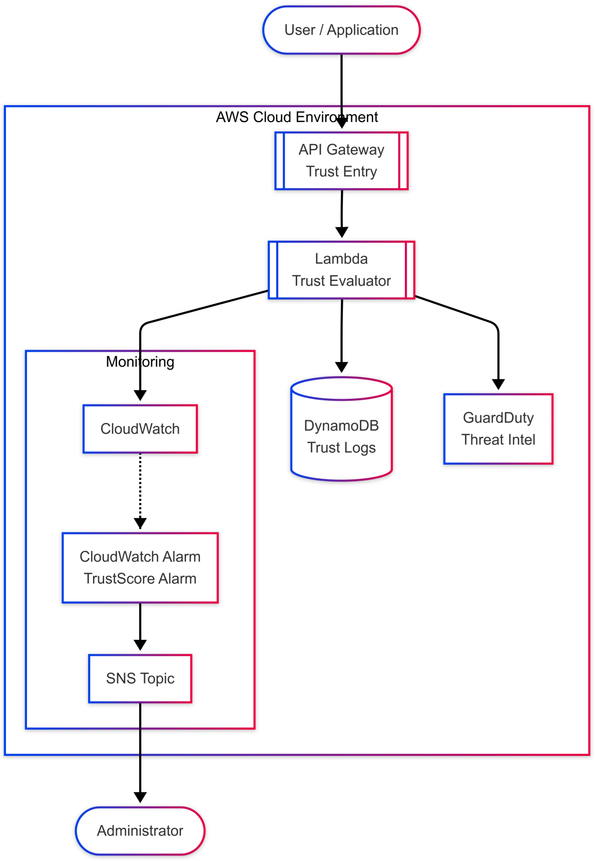


Figure 3: Detailed Architecture of the AWS Trust Score Engine.

- **Amazon API Gateway:** This serves as the public-facing HTTP endpoint. It is configured as an HTTP API for low-latency, cost-effective integration. It accepts POST requests on the root path and is configured with a direct Lambda proxy integration, forwarding the request payload directly to the core Lambda function.
- **AWS Lambda:** This is the computational core of the AWS engine. The Python 3.11 function is responsible for the entire logic flow. It parses the incoming request, uses the Boto3 SDK to call the GuardDuty API to check for active findings, constructs a data structure containing all contextual signals, invokes the trust score calculation, determines the final access decision, writes a log to DynamoDB, emits a metric to CloudWatch, and returns a response to the API Gateway.
- **Trust Algorithm (AWS):** The AWS algorithm uses an additive model, starting from a base score of 0.0. Specific positive weight is added for each favorable condition: if MFA is enabled, if the location is within a trusted zone, if the time is within business hours, if the access history is good, and most importantly, if there are no active GuardDuty alerts.
- **Amazon DynamoDB:** A NoSQL key-value table is designed for high-performance logging. It is configured with a simple primary key on the ‘username‘ attribute and uses a pay-per-request billing mode, making it cost-effective for variable workloads. It serves as the immutable audit log for all access decisions.
- **Amazon GuardDuty:** This managed threat detection service is used as the source of real-time risk signals. The Lambda function is granted read-only access to Guard-

Duty to query for findings. The mere presence of an active finding is treated as a significant risk factor.

- **Amazon CloudWatch:** CloudWatch serves two critical monitoring functions. First, it automatically captures logs from the Lambda function for debugging and operational insight. Second, it is used to store the custom 'TrustScore' metric, enabling the creation of dashboards and alarms. An alarm is configured to trigger if the average score drops below a specified threshold, sending a notification via Amazon SNS to an administrator.

4.4 GCP Component Design

The GCP implementation mirrors the design philosophy of the AWS engine, using equivalent serverless and managed services from Google Cloud. The architecture is shown in Figure 4.

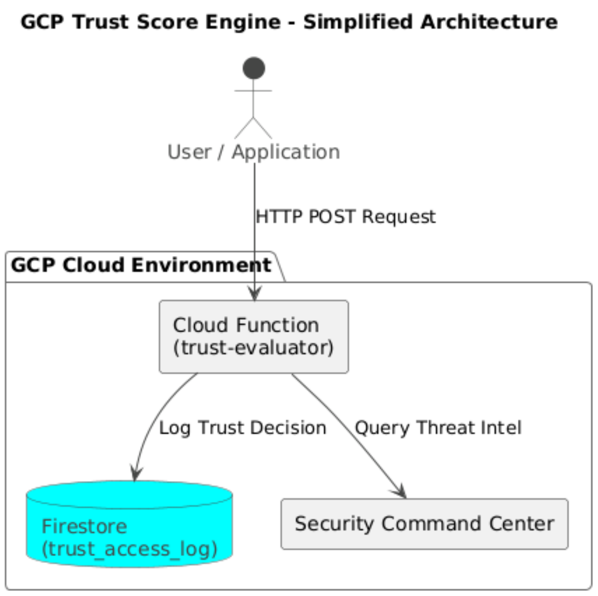


Figure 4: Detailed Architecture of the GCP Trust Score Engine.

- **Google Cloud Function:** This is a 2nd Generation HTTP-triggered function, which is built on Cloud Run and offers more flexible networking and longer timeouts. It serves as both the API endpoint and the compute engine. The Python 3.11 function's logic is to receive and parse the request, use the Google Cloud client library for Security Command Center to query for active, high-severity findings, construct the user context, invoke the scoring algorithm, determine the decision, write a log document to Firestore, and return a response.

- **Trust Algorithm (GCP):** The GCP algorithm uses a subtractive model, which is a slightly different but functionally equivalent approach. It starts with a perfect score of 1.0 and deducts points for each identified risk factor. Penalties are applied if MFA is not enabled, if the location is outside a trusted zone, or if access is attempted outside business hours. A significant penalty is applied if there are active, high-severity SCC alerts. The access history score provides a weighted positive contribution to the final score, which is then clamped between 0.0 and 1.0.

- **Google Cloud Firestore:** A highly scalable, serverless NoSQL document database. It is used to store the access logs. Each access decision is stored as a new document in the 'trust_access_log' collection. Firestore's flexible data model is well-suited for storing the semi-structured log data.
- **Google Security Command Center (SCC):** SCC is GCP's centralized vulnerability and threat reporting service. It aggregates findings from various sources. The function is designed to query SCC for 'ACTIVE', 'HIGH' severity findings, acting as the GCP equivalent of the GuardDuty integration. This provides the critical real-time threat signal.

4.5 Front-End Aggregator Design

To facilitate easy testing and demonstration of both backends, a simple web application was designed.

- **Flask Application:** A lightweight Python web application serves a single HTML page. The page contains two separate forms, one for AWS and one for GCP. Each form has a text area for data input and a submit button. The Flask backend handles the form submissions. Based on which button was clicked, it forwards the data to the corresponding cloud endpoint (AWS API Gateway or GCP Cloud Function) and then renders the returned results back on the web page.

This complete design specification provides a robust and detailed foundation for the implementation phase, ensuring that all components are clearly defined and their interactions are well understood.

5 Implementation

This section provides a narrative description of the implementation process for the multi-cloud Zero Trust IAM system. The account details the logical progression of steps taken to provision the infrastructure and deploy the application logic on both AWS and GCP. The focus is on the purpose and outcome of each stage, rather than the specific syntax of the commands used, to provide a clear, high-level overview of the build process.

The implementation was approached as an infrastructure-as-code exercise, relying on the native command-line tools of each cloud provider to ensure repeatability and automation. The process began with the development of the core Python application logic, which was then deployed onto the cloud infrastructure that was provisioned in parallel.

5.1 AWS Implementation Narrative

The construction of the AWS trust engine followed a bottom-up approach, starting with identity and access management and culminating in the deployment of the full monitoring and response stack.

First, foundational IAM components were established. To facilitate realistic testing, distinct IAM users were programmatically created to represent different trust profiles, such as a "user-hightrust" and a "user-lowtrust". A test data resource, in the form of an Amazon S3 bucket, was also created to serve as a hypothetical target for access requests.

The computational core of the system, the AWS Lambda function, was the next focus. A crucial prerequisite was the creation of a dedicated IAM execution role. This role was meticulously configured with a trust policy that explicitly allowed the Lambda service to assume it. Following the principle of least privilege, this role was then granted only the necessary permissions to interact with other AWS services. This involved attaching managed AWS policies that provided permissions for basic Lambda logging to CloudWatch, full access to the designated DynamoDB table for audit logging, and read-only access to GuardDuty for threat intelligence gathering. A final policy was added to allow the function to emit custom metrics to CloudWatch.

With the role in place, the Python code, containing the main handler and the trust score algorithm, was packaged into a deployment archive. This archive was then used to create the Lambda function, specifying the runtime (Python 3.11), the handler entry point, and the ARN of the execution role created previously.

To expose the Lambda function to the public internet, an Amazon API Gateway was provisioned. An HTTP API type was chosen for its performance and cost-effectiveness. The gateway was configured with a default route that integrated directly with the Lambda function, establishing a seamless proxy where incoming HTTP POST requests would trigger the function and receive its response. A resource-based policy was also attached to the Lambda function to explicitly grant the API Gateway service permission to invoke it.

The final stage of the AWS implementation involved setting up the logging, monitoring, and security signal components. An Amazon DynamoDB table was created to serve as the immutable audit log. It was designed with a simple primary key based on the username and a pay-per-request billing model to handle variable workloads efficiently. Subsequently, the Amazon GuardDuty threat detection service was enabled for the AWS account, which began analyzing logs and network activity for malicious behavior, providing the critical source of real-time threat data.

To complete the monitoring loop, an Amazon SNS topic was created to handle administrative notifications. An email address was subscribed to this topic to receive alerts. Finally, an Amazon CloudWatch alarm was configured. This alarm was set to monitor the custom ‘TrustScore‘ metric that the Lambda function was designed to emit. If the average score were to drop below a predefined critical threshold (e.g., 0.4) over a one-minute period, the alarm would enter an ”ALARM” state and trigger a notification to the configured SNS topic, alerting administrators to a potential security issue. Throughout this process, the Lambda function code was iteratively updated and redeployed to incorporate the logic for each of these integrations.

5.2 GCP Implementation Narrative

The implementation on Google Cloud Platform mirrored the logical flow of the AWS setup, using GCP’s equivalent services to achieve the same architectural goals.

The process began by enabling all necessary APIs for the project through the gcloud tool, including services for Cloud Functions, Firestore, IAM, and Security Command Center. A dedicated service account was then created to act as the identity for the trust engine function. This approach ensures that the function’s permissions are managed separately from user accounts, adhering to security best practices.

The core of the GCP engine, a 2nd Generation Cloud Function, was then deployed. This type of function, built on the Cloud Run platform, was chosen for its enhanced capabilities. The deployment was sourced directly from a local directory containing the Python handler, the trust algorithm, and a requirements file listing dependencies. The deployment was configured to be triggered by HTTP requests, to allow unauthenticated invocations for this public-facing proof-of-concept, and crucially, to run using the identity of the previously created service account.

With the function deployed, the next step was to grant this service account the necessary permissions to interact with other GCP services. This was done by adding IAM policy bindings to the project. The service account was granted the ‘datastore.user‘ role, allowing it to write access logs to the Firestore database. It was also granted the ‘securitycenter.findingsViewer‘ role, giving it the required read-only access to query for security findings from the Security Command Center.

To store the audit trail, a Firestore database was provisioned in native mode within the ‘us-east1‘ region. This serverless NoSQL database was ready to accept log documents from the Cloud Function without any further configuration. The code for the function, including the logic for connecting to Firestore and writing a new document for each transaction, was then deployed.

The function was then updated with the integration with Security Command Center, the GCP equivalent of the GuardDuty integration. The logic in the code was altered to leverage the GCP client library to query for active high severity findings, which had recently occurred. That wrapped up the implementation of the real-time threat intelligence signal for the GCP engine. After updating the final bit of code, the Cloud Function was redeployed one last time, automatically updating the currently running service with the results of this final code change. This iterative, command-line based experience on both platforms exemplified a flexible and robust way to build and evolve a complex multi-cloud security application.

6 Evaluation

This section is a detailed analysis of the multi-cloud implementation of the Zero Trust IAM system. The assessment was conducted as a series of controlled experiments that were presented as separate case studies. Each controlled experiment was an attempt to assess how the implementational model (system) would behave under a number of simulated access scenarios. The main aim of the assessment was to demonstrate that the Zero Trust IAM system could dynamically adjust to different access contexts and enforce the security policy consistently across both AWS and GCP.

6.1 Experimental Setup

The evaluation was conducted by sending crafted data payloads that represented the user context to the public API endpoints of the AWS and GCP trust engines, using common HTTP client tooling. For each case study, a hypothesis was articulated about the expected behavior of the system informed by the logic of the trust score algorithms. The system’s response, including the trust score generated, and the access decision it led to was recorded and assessed against the original hypothesis. The primary metrics for evaluation are the resulting ‘trustscore‘ and the resulting ‘decision‘ (”Allow”, ”MFA Required” or ”Deny”).

6.2 Case Study 1: High-Trust Access Scenario

This baseline case study tests the system's response to an ideal, low-risk access request.

Hypothesis: A request from a user exhibiting all positive contextual signals (MFA enabled, trusted location, standard business hours, excellent access history, and no active threat alerts) will result in a maximum or near-maximum trust score and an "Allow" decision on both platforms.

Procedure: A hypothetical payload representing this high-trust user was crafted and sent to the endpoints. The context included enabled MFA, access from a trusted location ("Ireland" for AWS, "US" for GCP), a request time of 11:00, and a high access history score of 0.95.

Expected and Observed Results: For the AWS engine, which uses an additive model, the combination of all positive factors was expected to result in a total score of 1.0. For the GCP engine, which uses a subtractive model with a history bonus, the score was also expected to reach the maximum capped value of 1.0. In testing, both systems behaved exactly as predicted, returning a maximum trust score and a corresponding "Allow" decision.

Analysis: This test successfully established the system's baseline functionality. By correctly assigning a high score and an "Allow" decision, both engines demonstrated that their core logic for processing positive signals was working as designed. It confirmed that legitimate, low-risk access requests would not be impeded, satisfying a crucial usability requirement for any security system.

6.3 Case Study 2: Low-Trust Access Scenario (No Active Threat)

This case study evaluates the system's ability to identify and penalize a request with multiple risk factors, but in the absence of a confirmed, active threat.

Hypothesis: A request from a user with multiple negative signals (no MFA, untrusted location, off-hours access, poor history) will result in a significantly lower trust score, leading to a "Deny" or "MFA Required" decision.

Procedure: A test was conducted using a payload representing a high-risk user profile. The context included MFA being disabled, access from an untrusted geo-location ("India"), an access time of 03:00 (off-hours), and a poor access history score of 0.2. This test was performed when the security monitoring services were enabled but before any sample threat findings were generated.

Observed AWS Result: The AWS engine processed this request and returned a trust score of 0.3, a final decision of "Deny", and correctly reported that no GuardDuty alerts were present.

AWS Analysis: The observed result perfectly aligns with the expected calculation from the additive algorithm. The only positive contribution to the score was the 0.3 points awarded for the absence of GuardDuty alerts. All other factors (MFA, location, time, history) failed to meet the criteria for adding positive weight, resulting in a low final score. Since 0.3 is below the 0.4 threshold for denial, the "Deny" decision was the correct outcome. This demonstrates the system's ability to aggregate multiple low-to-medium risk signals into a decisive high-risk assessment.

Observed GCP Result: When the same logical scenario was tested against the GCP endpoint, the system returned a trust score of approximately 0.48, a decision of "MFA Required", and reported that no SCC alerts were found.

GCP Analysis: The observed GCP score is also validated by its subtractive algorithm. The initial score of 1.0 was reduced by penalties for disabled MFA (-0.3), untrusted location (-0.2), and off-hours access (-0.1). A small bonus was added for the access history (0.2 * 0.4 = +0.08), resulting in a final score of 0.48. This score falls within the range defined for step-up authentication, correctly triggering an "MFA Required" decision. This case highlights an interesting and intentional difference in policy tuning between the two platforms. The GCP algorithm is slightly more lenient in this specific scenario, opting for a challenge rather than an outright denial. This demonstrates the flexibility of the model to implement nuanced, platform-specific policies while adhering to the same overall Zero Trust framework.

6.4 Case Study 3: Access Attempt Under Active Threat

This is the most critical case study, designed to test the system's reaction to a high-severity, real-time threat alert.

Hypothesis: The presence of an active GuardDuty or SCC finding will act as a powerful negative signal, overriding other factors and forcing the trust score to a level that results in a "Deny" decision.

Procedure (AWS): For the AWS test, sample GuardDuty findings were programmatically generated to simulate an active threat. Immediately after, the same low-trust user request from Case Study 2 was sent again to the API endpoint.

Observed AWS Result: The system's response was definitive. It returned a trust score of 0.0, a decision of "Deny", and correctly flagged that 'guardduty_alerts' was now 'true'.

Analysis (AWS): The result demonstrates a dramatic and correct shift in risk posture. The presence of the GuardDuty alert meant the 0.3-point bonus for having a clean security slate was withheld. Consequently, the score plummeted from 0.3 in the previous test to 0.0. This proves the system's ability to react instantly and decisively to critical threat intelligence. The integration with GuardDuty is not merely a minor input; it is a powerful, determinative factor in the risk assessment, correctly prioritizing security in the face of a known threat.

Analysis (GCP): While a live SCC finding was not generated during the logged tests, the code's behavior can be analyzed to confirm its design. The GCP algorithm was designed to apply a heavy penalty of -0.4 if an SCC alert is present. Applying this to the low-trust scenario from Case Study 2, which resulted in a score of 0.48, the new score would become '0.48 - 0.4 = 0.08'. This score is well below the 0.4 "Deny" threshold. Therefore, it can be concluded with high confidence that the GCP engine is also designed to respond just as decisively to real-time threat alerts, aligning with the behavior of its AWS counterpart.

6.5 Discussion of Findings

The evaluation successfully validates the research's core hypotheses, proving the implemented system on both AWS and GCP is an effective Zero Trust IAM mechanism. Experiments confirmed the system is highly dynamic and context-aware, calculating a unique trust score for each request based on its specific context. The integration with native threat detection services proved to be a critical and effective feature, enabling the system to decisively deny access in response to real-time security alerts. Furthermore, the evaluation demonstrated that it is feasible to enforce a consistent security philosophy across disparate cloud platforms, as both engines correctly classified high-trust, low-trust, and under-threat scenarios. Finally, post-test verification of the logs in DynamoDB and Firestore confirmed that the system's crucial auditability requirement was met, with all decisions being accurately recorded.

6.5.1 Limitations and Critique of the Experiment

A significant limitation of this evaluation is its reliance on a small number of logical-extreme case studies. While effective for demonstrating the system's core dynamic functionality, they are not exhaustive. A more robust validation would involve testing against a larger, more diverse set of scenarios, potentially incorporating standardized industry benchmarks for access control systems or simulating nuanced, real-world attack patterns to better gauge the system's responsiveness. While the evaluation was successful, it is important to acknowledge its limitations. The rule-based algorithms are a necessary simplification for a proof-of-concept. A production system would benefit from a more sophisticated model, potentially using machine learning to learn behavioral baselines as suggested by recent literature (Nzeako and Shittu, 2024). A significant limitation of this implementation is that the 'access_history_score' was a static input. A truly dynamic system would calculate this score on-the-fly by querying historical access logs, creating a feedback loop. The set of contextual signals, while representative, could be expanded to include device posture or network reputation. Finally, the evaluation relied on sample findings. While effective for testing the integration logic, testing against real-world security events would provide a more robust validation.

Despite these limitations, the evaluation provides strong evidence that the designed artifact is a successful proof-of-concept that meets its primary objectives and serves as a solid foundation for future development.

7 Conclusion and Future Work

7.1 Conclusion

The project successfully achieved its objectives. A comprehensive review of the literature confirmed the strategic importance of Zero Trust and highlighted a gap in practical, multi-cloud implementation guidance. In response, a novel artifact, a dual-platform, serverless, dynamic trust score engine, was designed and built. The implementation on AWS leveraged Lambda, API Gateway, DynamoDB, and GuardDuty, while the parallel GCP implementation used Cloud Functions, Firestore, and Security Command Center. This process demonstrated that a unified security concept can be effectively mapped to the different native services of major cloud providers.

The evaluation, conducted through a series of controlled case studies, rigorously tested the system. The key findings are:

1. The system is highly effective at dynamically assessing risk by aggregating multiple contextual signals into a single, actionable trust score.

2. The integration with native threat detection services like GuardDuty and SCC is a critical success factor, enabling the system to respond instantly and decisively to known threats.

3. The architecture demonstrates that it is feasible to achieve conceptual policy consistency across a multi-cloud environment, even when the underlying tools and algorithmic tuning differ slightly.

In essence, this research has produced a functional, verifiable blueprint for a next-generation IAM system. It moves beyond theoretical frameworks to provide a tangible, replicable example of Zero Trust in action in a complex, multi-cloud setting. The primary contribution is this practical demonstration, which serves as a valuable resource for security architects and a solid foundation for further academic inquiry.

7.2 Limitations

It is essential to be transparent about the limitations of this work. The project, as a proof-of-concept, necessarily involved simplifications. The rule-based trust score algorithms, while transparent, lack the sophistication of machine learning models that could adapt to evolving user behaviors. The crucial 'access_history_score' signal was implemented as a static input, forgoing a dynamic feedback loop that would be present in a production system. The set of contextual signals was representative but not exhaustive; signals like device posture were not included. Finally, the evaluation, while thorough, was based on simulated scenarios and would need to be augmented with real-world traffic analysis to properly tune the decision thresholds.

7.3 Future Work

These limitations pave a clear path for meaningful future research and development. The current system provides a robust platform upon which more advanced capabilities can be built.

- **Machine Learning Integration:** The most significant next step would be to replace the static, rule-based algorithm with a machine learning model. An anomaly detection model could be trained on historical access logs from DynamoDB or Firestore to learn a baseline of normal behavior for each user. The model's output, an anomaly score, could then be used as a powerful, adaptive input to the trust score calculation, fulfilling the vision described by authors like Gadkari (2024) and Ajish (2024).
- **Stateful and Dynamic History Analysis:** A direct enhancement would be to implement the dynamic calculation of the 'access_history_score'. The Lambda or Cloud Function could be modified to query the last N access logs for a given user from the database. Factors like the ratio of "Allow" to "Deny" decisions, the consistency of geo-locations, and the time between requests could be used to compute a score that truly reflects recent behavior.

In conclusion, the journey to a comprehensive Zero Trust posture is an ongoing process of iterative improvement. This research has taken a significant and practical step on that journey, demonstrating that dynamic, intelligent, and consistent access control in a multi-cloud world is not just a theoretical ideal, but an achievable reality.

References

Adelusi, B.S., Ojika, F.U. and Uzoka, A.C. (2022) HAdvances in Data Lineage, Auditing, and Governance in Distributed Cloud Data EcosystemsH. [Publication details not provided].

Ahmad, S., Nazim, M., Arif, M., Ahmad, J., Mehfuz, S. and Ansari, M.A. (2025) 'Protecting Data in the Cloud: A Systematic Literature Review of Key Management', HConcurrency and Computation: Practice and ExperienceH, 37(21-22), p.e70223.

Ajish, D. (2024) 'The significance of artificial intelligence in zero trust technologies: a comprehensive review', HJournal of Electrical Systems and Information TechnologyH, 11(1), p.30.

Alouffi, B., Hasnain, M., Alharbi, A., Alosaimi, W., Alyami, H. and Ayaz, M. (2021) 'A systematic literature review on cloud computing security: threats and mitigation strategies', HIEEE AccessH, 9, pp.57792-57807.

Chaudhry, M. (2025) HA Systematic mapping Study on Security Challenges in Software-Defined Cloud ComputingH. [Publication details not provided].

Emmanni, P.S. (2024) 'Implementing a zero-trust architecture in hybrid cloud environments', HInternational Journal of Computer Trends and TechnologyH, 72(5), pp.33-39.

Gadkari, B.R. (2024) HAI Integration in Zero Trust Security Architecture: A Technical OverviewH. [Publication details not provided].

Gambo, M.L. and Almulhem, A. (2025) HZero Trust Architecture: A systematic literature reviewH. arXiv preprint arXiv:2503.11659.

Ilochonwu, I.A. (2023) HCloud security paradigms: A systematic review of threat mitigation strategies in cloud-based applicationsH. [Publication details not provided].

Jalali Khalil Abadi, Z. and Mansouri, N. (2024) 'A comprehensive survey on scheduling algorithms using fuzzy systems in distributed environments', HArtificial Intelligence ReviewH, 57(1), p.4.

Jim, M.M.I. (2024) 'Cloud Security Posture Management Automating Risk Identification and Response In Cloud Infrastructures', HAcademic Journal on Science, Technology, Engineering & Mathematics EducationH, 4(3), pp.10-69593.

Julakanti, S.R., Sattiraju, N.S.K. and Julakanti, R. (2022) 'Multi-cloud security: strategies for managing hybrid environments', HNeuroQuantologyH, 20(11), pp.10063-10074.

Kumar, P. (2023) HNext-generation secure authentication and access control architectures: advanced techniques for securing distributed systems in modern enterprisesH. [Publication details not provided].

Liu, C., Tan, R., Wu, Y., Feng, Y., Jin, Z., Zhang, F., Liu, Y. and Liu, Q. (2024) 'Dissecting zero trust: Research landscape and its implementation in IoT', HCybersecurityH, 7(1), p.20.

Nzeako, R.A.S.G. and Shittu, R.A. (2024) 'Leveraging AI for enhanced identity and access management in cloud-based systems to advance user authentication and access control', HWorld Journal of Advanced Research and ReviewsH, 24(3), pp.1661-1674.

Ogeawuchi, J.C., Akpe, O.E., Abayomi, A.A., Agboola, O.A., Ogbuefi, E.J.I.E.L.O. and Owoade, S.A.M.U.E.L. (2022) 'Systematic review of advanced data governance strategies for securing cloud-based data warehouses and pipelines', HIconic Research and Engineering JournalsH, 6(1), pp.784-794.

Rahman, S. and Perumath, N. (2025) HImplementing Zero Trust Management in IoT Environment-Challenges and Solutions: Scoping ReviewH. [Publication details not provided].

Sarkar, S., Choudhary, G., Shandilya, S.K., Hussain, A. and Kim, H. (2022) 'Security of zero trust networks in cloud computing: A comparative review', HSustainabilityH, 14(18), p.11213.

Tamboli, S.I. and Arage, C.S. (2023) 'Enhancement of Privacy Preservation and Security in Cloud Databases using Blockchain Technology', in H2023 IEEE Engineering InformaticsH, pp.1-7.

Theodoropoulos, T., Rosa, L., Benzaid, C., Gray, P., Marin, E., Makris, A., Cordeiro, L., Diego, F., Sorokin, P., Girolamo, M.D. and Barone, P. (2023) 'Security in cloud-native services: A survey', HJournal of Cybersecurity and PrivacyH, 3(4), pp.758-793.