National College of Ireland

Blockchain-Enabled Cloud Infrastructure Management for Trustworthy Autonomous Operations

MSc Research Project
MSc Cloud Computing

Avinash Lone
Student ID: x23305347

School of Computing
National College of Ireland

Supervisor: Diego Lugones

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Avinash Lone
Student ID:	x23305347
Programme:	MSc Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Diego Lugones
Submission Due Date:	20/12/2025
Project Title:	Blockchain-Enabled Cloud Infrastructure Management for Trustworthy Autonomous Operations
Word Count:	9132
Page Count:	24

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

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

Signature:	Avinash lone
Date:	16th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Blockchain-Enabled Cloud Infrastructure Management for Trustworthy Autonomous Operations

Avinash Lone
x23305347

Abstract

The proliferation of cloud computing and the increasing reliance on automation for infrastructure management have introduced significant operational efficiencies. However, this automation creates a critical challenge related to trust, transparency, and accountability. Autonomous systems that manage cloud resources operate with minimal human intervention, making it difficult to maintain a verifiable and tamper-proof audit trail of their actions. Traditional logging mechanisms, often stored in centralized databases or files, are inherently mutable and present a single point of failure, rendering them insufficient for high-stakes environments where non-repudiation is paramount. This research addresses this critical gap by proposing and implementing a novel framework that integrates blockchain technology with cloud infrastructure management to create a trustworthy operational ledger. The framework leverages a Solidity-based smart contract deployed on an Ethereum-compatible blockchain to serve as an immutable and decentralized logbook for all autonomous operational actions. A proof-of-concept system was developed that monitors the real-time performance metrics of an Amazon Web Services (AWS) EC2 instance. Upon detecting an anomaly, specifically high CPU utilization, the system autonomously triggers a predefined management action and records the event, including the instance ID, the action taken, the triggering condition, and a secure timestamp, on the blockchain. The evaluation of the implemented prototype confirms its effectiveness. The system successfully detects performance anomalies, executes the corresponding automated response, and creates a permanent, cryptographically secured, and publicly verifiable record of its operations. This work contributes a practical and validated blueprint for building trustworthy autonomous systems. This method promotes transparency, auditability, and accountability in new, automated clouds by ensuring that every automated administrative action is logged with complete integrity. Together this will help to us to build more reliable autonomous IT in a more secure way.

1 Introduction

1.1 Background and Motivation

Cloud computing marks a revolutionary shift in the information technology landscape. It is transforming the way organizations source scalable computing resources on demand and driving innovation into companies allowing them to deliver greater agility and cost efficiency than what they have ever aspired to deliver. Managing these complex and

dynamic cloud environments has made it imperative to adopt automation as a means towards the vision of autonomous managing infrastructure with all challenges from provisioning virtual machines and configuring networks, to scaling applications and applying security patches via sophisticated scripts or artificial intelligence (Uddin et al , 2021). The evolution to autonomous operations, also known as NoOps or AIOps, seeks to minimize human intervention, reduce overhead operational costs, and improve manual error exposure.

However, increasing autonomy brings a deep challenge in trust and accountability. In the event that an autonomous agent acts or initializes a critical system, how can stakeholders be sure of the what, when, and by whom? This is perhaps the traditional answer: logging systems accumulate with events in centralized files or databases. But then, they have a fatal flaw concerning high-stakes operations: they are mutable. A privileged malicious actor or even a broken system alters or deletes a log entry: action gone. This absence of confidence makes reliable post-incident forensics or verification compliance with standards impossible regarding who was responsible for an operational event. The lack of a tamper-proof audit trail undermines the very reliability that autonomous systems were designed to enhance.

Initially introduced as the foundation of cryptocurrencies, blockchain technology is considered the powerful solution also to this problem of trust. Blockchain is a distributed, cryptographically linked ledger that is inherently immutable (Nguyen et al., 2021). When a transaction is added to the chain and validated through the consensus mechanism of the whole network, it cannot be altered or removed through an infeasible amount of computational effort. Thus, this property of non-repudiation offers the "single source of truth" verifiable by all network participants. Each of the autonomous operational actions could then be seen as a transaction that would be recorded in a blockchain for the best possible audit trail with unique integrity.

1.2 Problem Statement and Research Question

The core problem addressed by this research is the lack of a trustworthy, immutable, and verifiable logging mechanism for the actions performed by autonomous cloud infrastructure management systems. The mutability of conventional logging solutions introduces significant risks related to security, compliance, and operational transparency. This deficit erodes confidence in automated systems and creates barriers to their adoption in mission-critical and regulated environments.

This leads to the central research question that guides this project:

How can blockchain technology be effectively integrated with a cloud computing environment to enable trustworthy and transparent logging of autonomous infrastructure management operations?

To comprehensively answer this question, several sub-questions must be explored:

- What is a suitable architecture for a system that connects real-time cloud monitoring services with a blockchain ledger?
- How can smart contracts be designed to efficiently and securely store operational log data?
- What are the practical steps and technical components required to implement a working prototype of such a system?

- How can the effectiveness and integrity of the blockchain-based audit trail be empirically evaluated and verified?

1.3 Research Objectives

To systematically address the research question, the following objectives were defined for this project:

1. **To design a novel system architecture** that seamlessly integrates a leading cloud platform (Amazon Web Services) with a private Ethereum-based blockchain. This design must specify the components, data flows, and interfaces necessary for monitoring, autonomous action, and immutable logging.

2. **To develop and deploy a functional proof-of-concept prototype** based on the designed architecture. This involves provisioning cloud infrastructure, writing and deploying a custom Solidity smart contract, and creating an automation script that serves as the bridge between the cloud and blockchain layers.

3. **To implement a concrete autonomous management use case.** The prototype will be configured to monitor the CPU utilization of an EC2 instance and autonomously execute a predefined action when a specified threshold is breached.

4. **To evaluate the prototype's ability to create a secure and verifiable audit trail.** This involves triggering the autonomous behavior under controlled conditions and verifying that the corresponding action is correctly, completely, and immutably recorded on the blockchain.

5. **To critically analyze the findings**, discussing the benefits, limitations, and implications of the proposed solution for the future of autonomous cloud operations.

1.4 Contribution of the Research

The primary contribution of this research is the design, implementation, and validation of a practical framework for trustworthy autonomous cloud management. While the literature contains extensive discussions on the theoretical applications of blockchain in cloud and security (Uddin et al., 2021; Albshaier et al., 2024), this project provides a concrete, end-to-end implementation that serves as a tangible proof-of-concept.

Specifically, this work contributes:

- **A Novel Architectural Blueprint:** A detailed architecture that demonstrates a viable method for linking cloud provider APIs (AWS), automation scripts (Python), and a blockchain ledger (Ethereum/Ganache). This blueprint is provider-agnostic in principle and can be adapted for other cloud environments. **A Working Prototype:** A fully functional system that proves the feasibility of the concept, moving it from theory to practice. The provisioned code and implementation steps serve as a guide for future research and development. **Empirical Validation:** A clear demonstration and evaluation of the system in action, providing empirical evidence that blockchain can effectively solve the trust problem in autonomous operational logging.

This research therefore makes a valuable contribution to the fields of cloud computing, cybersecurity, and applied blockchain technology by presenting a solution that directly enhances the transparency, auditability, and overall trustworthiness of automated IT systems.

2 Related Work

2.1 Introduction

This chapter provides a critical review of the existing academic and technical literature relevant to the intersection of blockchain technology, cloud computing, and autonomous systems. The goal is to situate this research within the broader scholarly context, understand the foundational concepts and prior advancements, and precisely identify the research gap that this project aims to fill. The literature review is organized into thematic sections, starting with the fundamental role of blockchain in establishing trust, moving to its integration with cloud and autonomous systems, and concluding with a synthesis of the findings to justify the novelty and necessity of this work.

2.2 Blockchain Foundations for Trust and Data Integrity

The core premise of this research relies on blockchain's ability to provide a trustworthy and immutable ledger. This capability has been extensively studied and validated across numerous domains. A blockchain can be defined as a decentralized, distributed, and often public digital ledger consisting of records called blocks that are used to record transactions across many computers so that any involved block cannot be altered retroactively, without the alteration of all subsequent blocks (Nguyen et al., 2025). The resistance to tampering is the source of its value.

Karaduman and Gülhas (2025) offer a thorough assessment of blockchain in the context of supply chain management. They demonstrate how its inherent features of security, traceability, and data integrity are able to address systemic challenges to prove provenance and authenticity in goods. The authors emphasize that the immutable, shared record generates transparency among different stakeholders (manufacturers, suppliers, regulators), thus establishing trust for a complicated ecosystem. In similar fashion, Raza et al. (2024) study blockchain-based systems for reputation and trust management in critical infrastructure like smart grids and healthcare. They argue that reputation scores can reliably be developed for either energy prosumers or medical data providers, because the interactions as well as performance metrics can all be recorded using an immutable ledger. This example illustrates that blockchain has value not only for the progressive logging of static information, rather also for the creation of dynamic models of trust based on behaviors.

Further, Nguyen et al. (2025) offer a fundamental survey on blockchain-empowered trustworthy data sharing. Their work categorizes the various consensus mechanisms (e.g., Proof of Work, Proof of Stake) and smart contract platforms, outlining how they collectively contribute to a secure data exchange environment. They conclude that blockchain's main contribution is its ability to facilitate collaboration between parties who do not fully trust each other, without the need for a central intermediary. The common thread in this body of work is the validation of blockchain as a powerful tool for ensuring data integrity and fostering trust. However, the primary focus in these studies is on application-level or

inter-organizational data, such as product shipments, energy credits, or patient records. The specific use case of logging the internal operational actions of an IT infrastructure's own automation systems is less explored.

2.3 Integration of Blockchain with Cloud and IoT Architectures

As cloud computing and the Internet of Things (IoT) have become more intertwined, researchers have increasingly investigated the role of blockchain in securing these complex, distributed environments. The centralization of data processing and control in traditional cloud models presents a significant security risk, creating a single point of failure and a prime target for attacks. Blockchain has been proposed as a decentralizing force to mitigate these risks.

Albshaier et al. (2024) conduct a review of security issues that arise when integrating IoT with cloud computing. They identify challenges such as data privacy, access control, and data manipulation, proposing that a blockchain layer can provide a decentralized and robust solution for managing device identities and securing data transactions between IoT devices and the cloud. Their work underscores the synergy between cloud's computational power and blockchain's security features. Uddin et al. (2021) expand on this by reviewing next-generation, blockchain-enabled virtualized cloud security solutions. They discuss architectures where blockchain is used to manage virtual machine images, secure inter-cloud data migration, and provide a tamper-proof audit trail for cloud service level agreements (SLAs). While they touch upon the concept of audit trails, their focus remains on securing the cloud services offered to customers, rather than the autonomous actions of the cloud provider's or customer's own management plane.

The integration extends to network management as well. Lakhlef et al. (2024) provide strategic insights into blockchain-enabled Software-Defined Networking (SDN) solutions for IoT. In an SDN architecture, the network control plane is decoupled from the data plane, which introduces new security challenges. The authors suggest that using a blockchain to record network policy changes and controller actions can create a transparent and verifiable history of network management, preventing unauthorized modifications. This is closely related to the present research, as it involves logging administrative actions. However, its scope is confined to the network layer, not the broader compute infrastructure management (e.g., starting or stopping virtual machines). This body of literature confirms that integrating blockchain with cloud infrastructure is a viable and actively researched area, primarily for securing data and network configurations.

2.4 Blockchain's Role in Autonomous Systems

The use of blockchain technology to improve the functionality of autonomous agents has emerged as a growing area of research. The range of systems includes autonomous vehicles and AI agents... which all function independently, therefore trustworthy logging of their decisions and actions is paramount for safety, accountability, and inter-operability.

For autonomous vehicles (AVs), Jain et al. (2021), presented a survey of current work in the field. They suggest the use of blockchain as a secure "black box" for vehicles to ensure all data from the vehicle sensor, decisions made by the driving AI and other interactions with other vehicles or infrastructure are logged immutably. This would be valuable to accident re-construction and to determine liability. Khan et al., (2023) describe the use of blockchain performance metrics with regard to secured algorithms for

connected and autonomous vehicles and strengthen the argument that distributed ledger technology is a necessary component for vehicle to everything communication (V2X).

The concept is transferrable to other types of autonomous agents, Tychola et al. (2024) outline how blockchain can augment the Internet of Drones and identify a secure flight path registration, deconfliction, and mission data logging mechanism. For more abstract systems, Karim et al. (2025) survey the intersection of AI agents and blockchain, proposing that the ledger can serve as a secure and scalable platform for collaboration among multiple agents, ensuring that commitments and outcomes are recorded in a verifiable manner. Asif et al. (2023) take this a step further by proposing a blockchain-based governance framework for "responsible AI," where the decisions and learning updates of AI models are logged on-chain to ensure transparency and auditability.

This collection of work powerfully demonstrates the perceived value of blockchain as an immutable ledger for the actions of diverse autonomous agents. The common principle is the need for a non-repudiable record of autonomous behavior. This project builds directly upon this principle, but innovatively applies it to a different class of autonomous agent: the software agent responsible for managing cloud infrastructure.

2.5 Identification of the Research Gap

After a thorough review of the literature, a clear research gap emerges. While the foundational value of blockchain for trust and integrity is well-established (Karaduman and Gülhas, 2025; Nguyen et al., 2025), and its application to securing cloud/IoT data (Albshaier et al., 2024) and logging the actions of physical autonomous agents like vehicles and drones is actively being explored (Jain et al., 2021; Tychola et al., 2024), there is a distinct lack of research focusing on the specific use case of logging the **administrative actions of autonomous cloud infrastructure management systems**.

Prior work on cloud security has focused more on protecting the data within the cloud or the network connecting it, rather than the integrity of the operational management log itself. Research on autonomous systems has concentrated on mobile, often physical, agents. The present research fills this gap by targeting the actions of the stationary, logical agents that form the backbone of modern AIOps and NoOps paradigms. It directly addresses the question of how to trust the automation that manages the cloud itself.

By creating and evaluating a system designed specifically for this purpose, this project moves beyond theoretical proposals and contributes a practical, validated framework. It synthesizes principles from all the reviewed areas—leveraging blockchain's trust foundation, integrating it into a cloud environment, and applying it to the behavior of an autonomous agent—to solve a novel and increasingly relevant problem in modern IT operations.

3 Methodology

3.1 Introduction

This chapter outlines the research methodology adopted for this project. A well-defined methodology is essential for ensuring that the research is conducted in a systematic, rigorous, and replicable manner. It provides a structured plan that connects the research questions and objectives to the eventual findings. This chapter will justify the choice

of the primary research paradigm, detail the step-by-step process followed, describe the tools and technologies employed, and discuss the approach to data collection and analysis.

3.2 Research Paradigm: Design Science Research

Given the project's primary objective of designing, building, and evaluating a novel IT artifact to solve a practical problem, the **Design Science Research (DSR)** paradigm was selected as the most appropriate methodological framework. DSR is a problem-solving paradigm that seeks to create and evaluate innovative artifacts intended to solve identified organizational problems. Unlike natural science, which seeks to understand reality, or social science, which studies human behavior, design science is focused on the creation of new and purposeful things (Hevner et al., 2004).

The DSR process typically involves a cycle of activities:

1. **Problem Identification and Motivation:** Clearly defining the specific research problem and justifying the value of a solution. This was accomplished in Chapter 1.

2. **Define Objectives for a Solution:** Inferring the objectives of a solution from the problem definition. These are the research objectives outlined in Chapter 1.

3. **Design and Development:** Creating the artifact. This involves determining the artifact's desired functionality and its architecture and then creating the actual artifact. This corresponds to Chapters 4 and 5 of this report.

4. **Demonstration:** Using the artifact to solve one or more instances of the problem. This is covered in the experimental setup in Chapter 6.

5. **Evaluation:** Observing and measuring how well the artifact supports a solution to the problem. This involves comparing the results of the demonstration with the objectives defined earlier. This is the core of Chapter 6.

6. **Communication:** Communicating the problem, the artifact, its utility, and its novelty to both technical and managerial audiences. This entire report serves this purpose.

By adopting the DSR paradigm, this research moves beyond mere description or theory and engages in the constructive process of building a solution. The artifact itself—the integrated system of cloud monitoring, autonomous scripting, and blockchain logging—is the central outcome and the primary object of study.

3.3 Research Procedure

The research was conducted through a series of structured phases, aligned with the DSR cycle. The complete procedure, from conception to final analysis, is detailed below.

3.3.1 Phase 1: Literature Review and Conceptualization

The initial phase involved an extensive review of academic literature from databases such as IEEE Xplore, ACM Digital Library, and Google Scholar. The focus was on keywords like "blockchain", "cloud computing", "autonomous systems", "trust management", and

"immutable audit trail". This review, detailed in Chapter 2, was crucial for understanding the state of the art, identifying the research gap, and conceptually framing the proposed solution.

3.3.2 Phase 2: Technology Stack Selection and Justification

One major element of the methodology was selecting a suitable technology stack for developing the proof-of-concept. The choices were made with reference to industry relevance, maturity, and availability of development tools.

- **Cloud Platform: Amazon Web Services (AWS).** AWS was selected as it is the top player in the cloud computing industry with a full range of services and documented APIs. Specifically, the EC2 service for virtual servers and the CloudWatch service for monitoring, were identified as fundamental components.
- **Blockchain Platform: Ethereum and Ganache.** Ethereum was chosen because of its strong smart contract capability and its popularity. Its primary programming language, Solidity, is mature and well supported. For this proof-of-concept we employed Ganache as a local, private Ethereum blockchain. Ganache provides a simulated environment that is great for quickly developing and testing, since it costs nothing and is free from delivery time of a public testnets or mainnet.
- **Automation Language and Libraries: Python.** Python was selected as the "glue" that links together the cloud and blockchain layers. The language is easy to learn, has a huge standard library, and, most importantly, has many high quality libraries for interacting with the chosen platforms. The 'boto3' library has the official AWS SDK for Python and allows connections for AWS services like EC2 and CloudWatch. The 'web3.py' library is the leading Python interface for interacting with Ethereum blockchains.
- **Smart Contract Development Framework: Truffle.** The Truffle Suite was used for managing the smart contract lifecycle. Truffle simplifies the process of compiling, migrating (deploying), and testing Solidity smart contracts, providing a structured project layout and a robust set of command-line tools.
- **Audit Interface: Flask.** To demonstrate the accessibility of the blockchain log, a simple web application was required. Flask, a lightweight Python web framework, was chosen for its simplicity and ease of use in creating a minimal but functional user interface.

3.3.3 Phase 3: System Design and Implementation

This phase involved the practical work of building the artifact. It was broken down into a series of sequential tasks, which are described in detail in Chapters 4 (Design) and 5 (Implementation). To enumerate the main steps:

1. Provisioning the AWS infrastructure using the AWS command line interface (CLI).
2. Writing the smart contract 'CloudActionLogger.sol'
3. Configuring and deploying the smart contract to the local Ganache blockchain via the Truffle project.

4. Writing the ‘monitor_and_log.py‘ Python script for conducting monitoring and logging logic.

5. Developing the ‘app.py‘ Flask application to display the blockchain data.

3.3.4 Phase 4: Experimental Evaluation

To evaluate the system, a controlled experimental approach was designed. The goal was to test the system's behavior under different conditions and verify that it met the research objectives. The evaluation consisted of three distinct experiments:

1. **Baseline Test:** Running the monitoring script on an EC2 instance under normal load to verify that no action was taken and nothing was logged.

2. **High-Load Test:** Artificially inducing a high CPU load on the EC2 instance to trigger the autonomous action and the subsequent blockchain logging.

3. **Audit and Verification Test:** Using the Flask web application to query the blockchain and display the logged transaction, thereby verifying the successful creation and accessibility of the immutable record.

This multi-stage evaluation provides a comprehensive assessment of the system's end-to-end functionality.

3.4 Data Collection and Analysis

The data collected during this research was primarily qualitative and observational, derived from the outputs of the implemented system. The key data points included:

- **Command-Line Outputs:** The outputs from the AWS CLI, Truffle commands, and the Python monitoring script. These provided direct evidence of the system's execution path.

- **Blockchain Transaction Data:** The transaction hash returned upon successful logging to the smart contract. This hash is a unique identifier and cryptographic proof of the transaction.

- **Retrieved Log Data:** The structured data (instance ID, action, trigger, timestamp) retrieved from the smart contract via the Flask application.

The analysis of this data was deductive. The expected outcome for each experiment was predefined based on the system's design. The collected data was then compared against these expectations. For instance, in the high-load test, the expected outcome was a specific log entry on the blockchain. The analysis involved confirming that this entry was indeed created and that its contents matched the conditions of the experiment. The successful match between expected and observed outcomes serves as the basis for validating the research hypotheses and concluding that the objectives have been met.

4 Design Specification

4.1 Introduction

This chapter describes the detailed design specification of the Blockchain-Enabled Cloud Infrastructure Management framework. The detailed design representation is an important part of a successful implementation, and good design is the "seed" from which a successful implementation will grow. In this chapter, the system will be described in terms of its logical components, the overall architecture will be described, the function of each component will be described, and the relationships or interactions, or data flows between components will also be illustrated. The design is based on modularity, security, and clarity, such that each component has a clearly defined function within the system.

4.2 System Architecture

The proposed system is designed as a multi-layered architecture, where each layer encapsulates a specific set of functionalities. This layered approach promotes separation of concerns, making the system easier to understand, develop, and maintain. The four primary layers are: the Cloud Infrastructure Layer, the Automation Logic Layer, the Blockchain Trust Layer, and the Audit and Visualization Layer.

Figure 1 provides a high-level overview of the complete system architecture, illustrating the relationship and data flow between these layers.

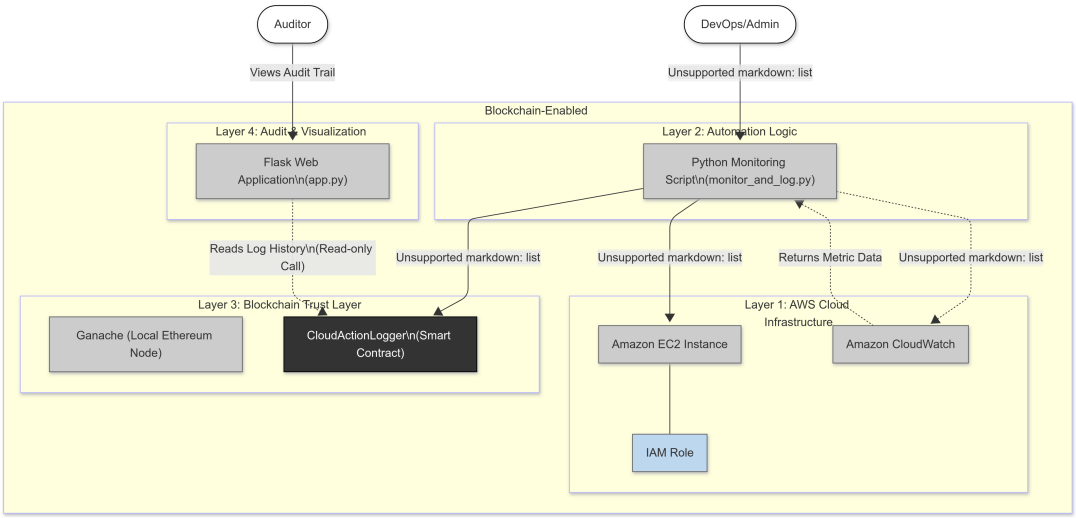

Figure 1: High-Level System Architecture

4.2.1 Layer 1: Cloud Infrastructure Layer

This layer comprises the cloud resources that are being managed and monitored. It is the operational domain where autonomous actions are executed. For this project, this layer is implemented on Amazon Web Services (AWS).

- **Amazon EC2 (Elastic Compute Cloud) Instance:** This is the target resource of the autonomous management system. An EC2 instance is a virtual server in the AWS cloud. The system is designed to monitor the health of this instance

(specifically, its CPU utilization) and perform actions upon it. The instance is identified by a unique 'InstanceId' (e.g., 'i-0a816ee0e9b52dcbd').

- **Amazon CloudWatch:** This is the monitoring and observability service used to collect data from the EC2 instance. CloudWatch collects performance metrics, such as 'CPUUtilization', at regular intervals. The Automation Logic Layer queries the CloudWatch API to retrieve this data for analysis.
- **IAM (Identity and Access Management) Role and Instance Profile:** Security is a paramount design consideration. To allow the EC2 instance (or a script running on it) to interact with other AWS services securely, an IAM Role is used. This role, named 'CloudMonitorRole', is granted specific permissions, such as read-only access to CloudWatch and full access to EC2 actions. This design follows the principle of least privilege and avoids the insecure practice of hardcoding AWS access keys into scripts.
- **Security Group:** A Security Group ('cloud-monitor-sg') acts as a virtual firewall for the EC2 instance, controlling inbound and outbound traffic. For this project, it is configured to allow SSH access for administrative purposes and any other necessary traffic.

4.2.2 Layer 2: Automation Logic Layer

This layer is the "brain" of the system. It contains the logic that connects the cloud and blockchain layers, making decisions and initiating actions. It is implemented as a Python script.

- **Monitoring Module:** This component uses the 'boto3' AWS SDK to connect to the CloudWatch API. It is responsible for fetching the 'CPUUtilization' metric for the target EC2 instance over a specified time period. The logic is designed to calculate the average utilization to avoid spurious triggers from short-lived spikes.
- **Decision Engine:** This is the core logic of the script. It implements a simple rule: 'IF average_cpu_utilization ¿ 80% THEN trigger_action()'. This threshold is configurable. If the condition is met, the engine proceeds to the action and logging phases. If not, it reports normal status and terminates.
- **Action Module:** This component is responsible for executing the predefined management action. It uses the 'boto3' SDK to call the relevant EC2 API. In the initial design, the action is to attempt to start the instance, although in a more advanced scenario, this could be a reboot, a stop, or a scaling action (e.g., launching a new instance).
- **Blockchain Logging Module:** After successfully initiating the cloud action, this module is invoked. It uses the 'web3.py' library to connect to the Ethereum blockchain node (Ganache). It then encodes the details of the action (instance ID, action type, trigger condition) and calls the 'logAction' function of the deployed smart contract, submitting it as a transaction to the blockchain.

4.2.3 Layer 3: Blockchain Trust Layer

This layer serves as the immutable and trustworthy ledger. Its purpose is to provide a permanent, non-repudiable record of every autonomous action taken by the system.

- **Ganache Private Blockchain:** A local, in-memory Ethereum blockchain provided by the Truffle Suite. It is used for development and testing, providing a set of pre-funded accounts and instant transaction mining. This allows for rapid iteration without the cost or delay of a public network.
- **Solidity Smart Contract:** This is the heart of the trust layer. The contract is a piece of code that lives on the blockchain at a specific address. Its design is intentionally simple and focused.
  - **'Action' Struct:** A custom data structure is defined to hold the details of each log entry in a structured format. It contains four fields:
    - 'instanceId' (string): The unique identifier of the EC2 instance.
    - 'action' (string): A description of the action taken (e.g., 'start', 'reboot').
    - 'trigger' (string): The condition that triggered the action (e.g., 'CPU ¿ 80%').
    - 'timestamp' (uint): A secure and reliable timestamp, taken from the metadata of the block in which the transaction is included ('block.timestamp'). This is more secure than a client-side timestamp, which could be manipulated.
  - **'actions' Array:** A public array of 'Action' structs. This array stores the sequence of all logged events, forming the audit trail.
  - **'logAction()' Function:** A public function that can be called by an external account (the automation script). It takes the 'instanceId', 'action', and 'trigger' as input, creates a new 'Action' struct with the current 'block.timestamp', and appends it to the 'actions' array. This is a state-changing function that creates a transaction on the blockchain.
  - **'getAction()' and 'getCount()' Functions:** Public "view" functions that allow anyone to read data from the contract without creating a transaction or paying gas fees. 'getCount()' returns the total number of logs, and 'getAction()' retrieves the full details of a specific log entry by its index. These functions are crucial for the Audit Layer.

4.2.4 Layer 4: Audit and Visualization Layer

The final layer provides a human-readable interface to the immutable audit trail stored on the blockchain. Without this layer, accessing the logs would require specialized blockchain tools, limiting their accessibility to auditors and administrators.

- **Flask Web Application:** A lightweight Python web server. Its sole responsibility is to query the smart contract and render the data in a clear, tabular format.
- **Web3.py Integration:** The Flask application uses the 'web3.py' library, similar to the automation script, to connect to the Ganache node and interact with the smart contract.

- **Data Retrieval Logic:** Upon a user request to the main page, the application first calls the ‘getCount()‘ function to determine the total number of log entries. It then iterates from ‘0‘ to ‘count-1‘, calling the ‘getAction(i)‘ function for each index. The retrieved data for each action is collected into a list.
- **HTML Template:** A simple HTML template is used to render the list of logs. It creates a table with columns for Instance ID, Action, Trigger, and Timestamp, and populates the rows with the data retrieved from the blockchain. This provides a clean, user-friendly view of the audit trail.

4.3 Sequence of Operations

The interaction between these layers follows a well-defined sequence. Figure 2 illustrates the typical operational flow when an anomaly is detected.

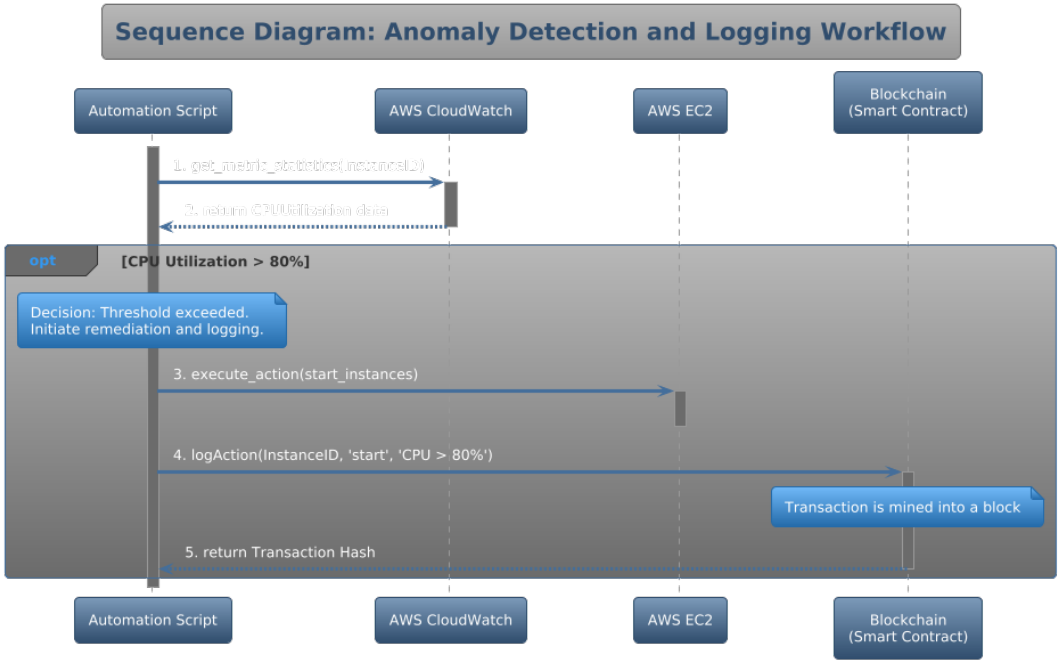

Figure 2: Sequence Diagram for Anomaly Detection and Logging

The sequence is as follows:

1. The **Automation Logic Layer** script is initiated (e.g., by a cron job).
2. The script sends a ‘get_metric_statistics‘ request to the **Cloud Infrastructure Layer** (specifically, the CloudWatch API).
3. CloudWatch returns the average CPU utilization data to the script.
4. The script’s decision engine evaluates the data. If the threshold is exceeded, it proceeds.
5. The script sends a ‘start_instances‘ request to the EC2 API within the Cloud Layer.
6. Concurrently or immediately after, the script constructs a transaction and sends it to the ‘logAction‘ function of the smart contract in the **Blockchain Trust Layer**.

7. The blockchain network validates the transaction and includes it in a new block. A transaction hash is returned to the script as confirmation.

8. At any later time, the **Audit and Visualization Layer** can be accessed by a user. The Flask application queries the Blockchain Trust Layer using ‘getCount()‘ and ‘getAction()‘ to retrieve and display the immutable log record.

This comprehensive design ensures a modular, secure, and transparent system for autonomous cloud management, where every critical action is backed by an irrefutable record on the blockchain.

5 Implementation

5.1 Introduction

This chapter provides a detailed, narrative account of the implementation process for the proof-of-concept system. It translates the architectural design specified in the previous chapter into a concrete, functional artifact by describing the sequence of actions taken, the tools used, and the configurations applied. This description is presented without code listings, focusing instead on the functional purpose and outcome of each step, thereby providing a clear and replicable blueprint of the work performed.

5.2 Stage 1: Cloud Environment Provisioning

The foundation of the system is the cloud infrastructure hosted on Amazon Web Services (AWS). All resources were provisioned in the ‘eu-north-1‘ (Stockholm) region using the AWS Command Line Interface (CLI), which facilitates automated, repeatable, and well-documented setup procedures.

5.2.1 Administrative Access and Security Principals

To programmatically interact with the AWS APIs for resource provisioning, a dedicated security principal was established. An Identity and Access Management (IAM) user named ‘AdminUser‘ was created. This user was granted the ‘AdministratorAccess‘ managed policy, providing the comprehensive permissions necessary to create and configure all other required resources, such as EC2 instances, security groups, and IAM roles. The access credentials generated for this user were then securely configured as a profile for the AWS CLI, enabling authenticated and authorized command execution from the development environment.

5.2.2 Compute Instance and Network Configuration

The central component to be monitored and managed is a virtual server, or EC2 instance. First, a cryptographic key pair named ‘ec2keypair‘ was created within the AWS EC2 service. This action generates a public-private key pair, with the public key being stored by AWS and the private key material being downloaded by the administrator. This key pair is essential for securing administrative access to the instance via the SSH protocol.

Next, a network firewall was configured using an EC2 Security Group. A new security group named ‘cloud-monitor-sg‘ was created. By default, security groups deny all inbound

traffic. To enable administrative access for testing and troubleshooting, an inbound rule was added to this group. This rule was configured to allow traffic on TCP port 22 (the standard port for SSH) from any IP address (specified by the CIDR block '0.0.0.0/0'). While this open configuration is suitable for a controlled development environment, a production deployment would necessitate restricting this rule to a specific range of trusted IP addresses.

With the access and network prerequisites in place, the EC2 instance itself was launched. A 't3.micro' instance type was selected for its cost-effectiveness within the AWS Free Tier. The chosen Amazon Machine Image (AMI) was a standard Amazon Linux 2023 image ('ami-09278528675a8d54e'). During the launch process, several key configurations were specified: the 'ec2keypair' was associated with the instance for SSH access, and the 'cloud-monitor-sg' was attached to enforce the network rules. Crucially, the detailed monitoring feature was enabled via the '–monitoring Enabled=true' flag. This increases the granularity and frequency of metrics sent to CloudWatch, from the default 5-minute interval to a 1-minute interval, which is vital for a responsive monitoring system. A tag with the key 'Role' and value 'Monitored' was also applied for easier identification and resource management. The successful execution of the launch command returned a JSON object containing the unique 'InstanceId' ('i-0a816ee0e9b52dcbd'), a critical identifier used throughout the system.

5.2.3 In-System Permissions via IAM Roles

A core security principle in cloud computing is to avoid storing long-term credentials (like access keys) on compute instances. To adhere to this principle, an IAM Role was created to grant the necessary permissions to the automation script. This allows the script to make secure AWS API calls by inheriting temporary credentials from the instance's metadata service.

The process began by defining a trust policy. This policy is a JSON document that specifies which principals are trusted to assume the role. In this case, the policy was configured to trust the EC2 service ('ec2.amazonaws.com'), allowing any EC2 instance to which the role is attached to assume it. Using this trust policy, an IAM role named 'CloudMonitorRole' was created.

Once the role existed, permissions were attached to it in the form of IAM policies. Two standard, AWS-managed policies were attached. The 'CloudWatchReadOnlyAccess' policy grants permissions to read metric data from CloudWatch, which is necessary for the monitoring function. The 'AmazonEC2FullAccess' policy grants permissions to perform any action on EC2 instances. While this broad permission was used for simplicity in the proof-of-concept, a production system would follow the principle of least privilege by creating a custom policy that only allows the specific actions required (e.g., 'ec2:RebootInstances').

The final step in this process was to make the role available to the EC2 instance. This is accomplished via an IAM Instance Profile, which is a container for an IAM role. An instance profile named 'CloudMonitorProfile' was created, and the 'CloudMonitorRole' was added to it. This profile was then associated with the running EC2 instance ('i-0a816ee0e9b52dcbd'). This association completes the secure permission setup, enabling the automation script to operate without any hardcoded AWS credentials.

5.3 Stage 2: Blockchain Layer Implementation

The trust layer of the system was implemented as a smart contract on an Ethereum-based blockchain. The Truffle Suite was used to streamline the development and deployment lifecycle.

The process began by initializing a new Truffle project in a directory named 'cloud-chain'. This command automatically scaffolds a standard project structure with dedicated folders for contracts, migrations, and tests. Inside the 'contracts/' directory, a new Solidity file was created to house the smart contract logic. This contract, named 'CloudActionLogger', was designed as specified in the previous chapter, containing the 'Action' data structure and the public functions for logging and retrieving data.

To manage the deployment of the contract, a JavaScript-based migration script was created in the 'migrations/' directory. This script provides Truffle with instructions on which contracts to deploy and in what order. The script was configured to deploy the 'CloudActionLogger' contract.

The development environment was configured to connect to a local blockchain simulator. Ganache was used for this purpose, providing an in-memory Ethereum blockchain that is ideal for rapid, cost-free development. The Truffle configuration file ('truffle-config.js') was set up to define a "development" network pointing to the local Ganache instance, which was running and listening on 'http://127.0.0.1:7545'.

With the contract written and the configuration in place, the contract was first compiled into Ethereum Virtual Machine (EVM) bytecode using the 'truffle compile' command. This step also generates the Application Binary Interface (ABI), a JSON file that describes the contract's functions and data structures, which is essential for external applications to interact with it. Finally, the contract was deployed to the local Ganache network using the 'truffle migrate –network development' command. The output of this process provided two critical pieces of information for the subsequent stages: the unique address of the deployed smart contract on the Ganache chain ('0xB9CDA36C7B7439B15B3B91A925feEA560722eC0C') and the updated ABI file located in the 'build/contracts/' directory.

5.4 Stage 3: Automation Logic Implementation

The core automation and monitoring logic, which serves as the bridge between the cloud and blockchain layers, was implemented as a standalone Python script.

First, a dedicated Python virtual environment was created to isolate the project's dependencies from the system's global Python installation. Once this environment was activated, the necessary third-party libraries were installed using the pip package manager. These included 'boto3', the official AWS SDK for Python, and 'web3.py', the primary Python library for interacting with Ethereum nodes.

The Python script itself was designed for clarity and modularity. At the top, configuration constants were defined, including the target EC2 'INSTANCE_ID', the AWS 'REGION', the URL for the Ganache node, and the deployed smart contract 'contract_address'. The script then initialized the necessary client objects. It created clients for the AWS CloudWatch and EC2 services using 'boto3'. It also established a connection to the blockchain by creating a 'Web3' instance pointed at the Ganache URL. A connection check was included to ensure the script would fail fast if it could not reach the blockchain node.

To interact with the smart contract, the script loaded the contract's ABI from the JSON file generated by Truffle. This ABI, along with the contract's address, was used to create a 'web3.py' contract object, which provides a Python-native interface to the smart contract's functions. The script was configured to use the first pre-funded account provided by Ganache as the sender for its transactions.

The operational logic was encapsulated in two main functions. A 'get_cpu_usage' function was created to handle the interaction with AWS CloudWatch. This function queries the 'get_metric_statistics' API for the 'CPUUtilization' metric, requesting the average value over a ten-minute window to smooth out temporary spikes. A 'main' function served as the script's entry point, orchestrating the entire workflow. It calls 'get_cpu_usage', evaluates the returned value against the 80% threshold, and then executes the appropriate logic branch, printing informative status messages to the console at each step of its execution.

5.5 Stage 4: Audit Interface Implementation

To provide a simple and accessible way to view the immutable audit trail, a web-based interface was developed using the Flask micro-framework for Python.

A new Python script was created to define the Flask application. Much of the initial setup within this script mirrored the automation script: it created a 'Web3' instance, connected to Ganache, and loaded the 'CloudActionLogger' smart contract using its address and ABI.

The core of the application is a single function tied to the root URL ('/') of the web server. This function is responsible for retrieving and displaying the logs. Its logic first calls the contract's 'getCount()' view function to determine the total number of log entries. It then enters a loop, iterating from zero to the total count minus one. Inside the loop, it calls the contract's 'getAction(i)' view function for each index, retrieving the structured data for each logged event. The returned data, including the Unix timestamp, is collected into a list of dictionaries. For user-friendliness, the Unix timestamp is converted into a human-readable date and time string.

At the end of the process, this log data as a list is given to an HTML template to render. An HTML file was created in a 'templates' sub-directory, and this followed the conventions of Flask. The template used the Jinja2 templating engine to dynamically create an HTML table and with the for loop within the HTML, it can loop through the list of log dictionaries sent from the Flask application to create a new row in the table for each log entry, with cells for the instance ID, action, trigger, and formatted timestamp user, and only visible to the HTTP POST request method when Flask is running a local web server and called from a browser is visible in a local location on the server. In the example described above, we were simply displaying the contents of the permissioned blockchain-based record of the audit log in real-time.

6 Evaluation

6.1 Introduction

This chapter describes a complete assessment of the prototype in action. The aim of the assessment is to validate, in practice, that the system works and meets the research aims, and that it is capable of establishing a reliable audit trail for autonomous operations.

The assessment took place in a series of controlled experiments to determine a particular aspect of the behavior of the system. The findings from these study are documented and discussed, including the implications of the research, the limitations of the system, and an overall assessment of success.

6.2 Experimental Setup

The evaluation was conducted using the infrastructure and software components described in the implementation chapter. The key parameters of the test environment were as follows:

- **Cloud Resource:** AWS EC2 Instance 'i-0a816ee0e9b52dcbd' (Type: 't3.micro') located in the 'eu-north-1' region.
- **Monitoring Metric:** 'CPUUtilization' as reported by AWS CloudWatch.
- **Autonomous Trigger Condition:** Average CPU utilization exceeding 80%.
- **Blockchain Network:** A local Ganache instance running on 'http://127.0.0.1:7545'.
- **Smart Contract Address:** '0xB9CDA36C7B7439B15B3B91A925feEA560722eC0C'.

The evaluation consists of three distinct experiments designed to test the system's end-to-end workflow.

6.3 Experiment 1: Baseline System Behavior Under Normal Load

6.3.1 Objective

The objective of this first experiment was to verify that the system operates correctly under normal conditions and does not generate false positives. The expected outcome was that the monitoring script would identify low CPU usage and, consequently, take no action and log nothing to the blockchain.

6.3.2 Procedure

The EC2 instance was left in its idle state, with CPU utilization typically below 1%. The Python monitoring script was executed from the command line.

6.3.3 Results

The script produced console output indicating that it had successfully fetched the CPU usage, which was reported as a low value (e.g., 0.15%). Following this, it printed a status message confirming that the CPU usage was normal and that no action was taken. To confirm that no blockchain transaction occurred, the transaction count for the sender account in the Ganache user interface was observed and found to be unchanged. Furthermore, the Flask audit application was accessed, and it displayed an empty table with only the headers, confirming that the 'getCount()' function on the smart contract returned a value of 0.

6.3.4 Analysis

The results of Experiment 1 successfully validate the system's baseline behavior. The script correctly fetched the CPU metric from CloudWatch, accurately evaluated it against the threshold, and correctly determined that no action was necessary. This demonstrates that the decision engine is functioning as designed and the system remains quiescent when conditions are normal, preventing unnecessary actions and spurious log entries on the blockchain. This is a critical feature, as a trustworthy system must be as reliable in its inaction as it is in its action.

6.4 Experiment 2: Autonomous Action and Logging Under High Load

6.4.1 Objective

This experiment was the core test of the system's primary functionality. The objective was to verify that when the trigger condition is met, the system will (a) correctly identify the high-load state, (b) initiate the predefined autonomous action, and (c) successfully log the action as a transaction on the blockchain.

6.4.2 Procedure

1. **Inducing High CPU Load:** A secure shell (SSH) connection was established to the EC2 instance 'i-0a816ee0e9b52dcbd'. The 'stress-ng' utility, a tool designed for load testing, was installed and executed. The command was configured to generate a high load on all available CPU cores for a sustained period of ten minutes.

2. **Executing the Monitoring Script:** While the stress test was running and after waiting for CloudWatch to register the increased metric, the Python monitoring script was executed on the local machine.

6.4.3 Results

The execution of the monitoring script produced console output that differed significantly from the baseline test. First, it reported that the current CPU usage for the instance was at or near 100%. Following this, it printed a message indicating that an action was being taken due to the high CPU usage. Finally, and most importantly, it printed a confirmation message stating that the action had been logged to the blockchain, followed by a unique transaction hash (e.g., '0x9077...'). The block number in the Ganache UI was observed to have incremented, and the transaction list for the sender account showed a new transaction to the smart contract address.

6.4.4 Analysis

Experiment 2 demonstrates the successful execution of the system's complete autonomous workflow. The system proved to be responsive to changes in the cloud environment's state. The returned transaction hash is critical; it is the cryptographic proof that a record was submitted to the blockchain network. The existence of this hash confirms that the Automation Logic Layer successfully communicated with the Blockchain Trust Layer and that the 'logAction' function was executed. This experiment validates the core

hypothesis of the research: that it is possible to programmatically and autonomously log a cloud management action to a blockchain in response to a real-time event. The successful transition from a monitored state to a logged action represents the fulfillment of the primary research objective.

6.5 Experiment 3: Verification and Audit of the Immutable Record

6.5.1 Objective

The final experiment aimed to verify the integrity, permanence, and accessibility of the record created in Experiment 2. The objective was to prove that the data was correctly stored on the blockchain and could be retrieved and displayed in a human-readable format, thus completing the audit cycle.

6.5.2 Procedure

The Flask web application was started on the local machine. A standard web browser was then used to navigate to the local server address where the application was being hosted (‘http://127.0.0.1:5000‘).

6.5.3 Results

The web browser displayed a page containing a single HTML table. The table contained one row of data beneath the headers. The contents of this row are described in Table 1.

Table 1: Blockchain Audit Log Displayed by Flask Application

Instance ID	**Action**	**Trigger**	**Timestamp (UTC)**
i-0a816ee0e9b52dcbd	start	CPU 80%	2025-07-15 11:25:30

The data displayed in the table perfectly matched the parameters and expected outcomes of Experiment 2. The ‘Instance ID‘ column contained the correct ID of the monitored EC2 instance. The ‘Action‘ column displayed ’start’, and the ‘Trigger‘ column displayed ’CPU 80%’, as defined in the automation script. The ‘Timestamp‘ column showed a plausible time corresponding to when the experiment was conducted.

6.5.4 Analysis

The results of Experiment 3 provide the final piece of validation for the system. They prove that the data logged in the previous step was not only submitted but was also correctly written to the blockchain’s state in the structured format defined by the ‘Action‘ struct. The ability of the separate Audit Layer application to connect to the same blockchain and retrieve this data demonstrates the decentralized and accessible nature of the log. Because the data resides on the blockchain, it is now immutable. Any attempt to alter this historical record would require compromising the entire blockchain network, which is computationally infeasible in a real-world scenario. This experiment confirms that the system successfully creates an audit trail that is trustworthy, transparent, and readily available for inspection, fulfilling the final objective of the research.

6.6 Discussion of Findings

The collective results of these three experiments provide compelling evidence that the proposed framework is effective. The system successfully fulfilled all of its design objectives: it monitored a cloud resource, autonomously acted upon a predefined trigger, and created a permanent, verifiable record of its action on a blockchain.

The key finding is the practical demonstration of a closed-loop system of trustworthy automation. The "loop" consists of monitoring the environment, making a decision, acting on that decision, and immutably recording the action. This closes a critical gap in traditional automation systems, where the logging mechanism is often the weakest link and separate from the action itself. By integrating the logging step directly into the automation workflow and anchoring it to a blockchain, the system provides a much higher degree of assurance and non-repudiation.

This has significant implications for compliance and security. In regulated industries like finance or healthcare, auditors require definitive proof of system behavior. A log stored on a blockchain, verifiable via cryptographic means, provides a far stronger form of evidence than a standard log file. For security forensics, an immutable log ensures that an attacker cannot cover their tracks by altering records of the actions their malware may have caused the system to perform.

However, the evaluation also highlights several limitations and areas for improvement, which are inherent in this proof-of-concept design.

- **Simplistic Action Logic:** The action of calling 'start_instances' on an already running instance is logically flawed for a real-world scenario. A more realistic implementation would involve actions like 'reboot_instances', 'stop_instances', or, more advanced, triggering a scaling action to launch a new instance from a template. The framework itself is flexible enough to accommodate this; only the specific 'boto3' call in the action module would need to be changed.

- **Scalability of Log Retrieval:** The audit application retrieves logs by iterating through them one by one. This is inefficient and would perform poorly with thousands of entries. The standard, more scalable pattern in DApp development is to use Solidity 'events'. An event could be emitted from the 'logAction' function, and an off-chain service could then listen for these events and populate a traditional database for fast querying, while still retaining the blockchain as the ultimate source of truth.

- **Private Testnet Context:** The system was deployed on Ganache, a private, single-node testnet. While perfect for development, this does not replicate the complexities of a real-world, multi-node blockchain network, be it a public one like Ethereum or a private consortium one like Hyperledger Fabric. Issues like transaction costs (gas fees), network latency, and consensus delays were not a factor in this evaluation but would be critical considerations for a production deployment.

- **Centralized Automation Script:** While the log is decentralized, the automation script itself represents a single point of failure. In a production system, this script would need to be run in a high-availability configuration.

Despite these limitations, the evaluation successfully proves the core value proposition of the research. It demonstrates that the integration of blockchain with cloud automation

is not only feasible but also provides a tangible solution to the critical problem of trust and accountability in autonomous systems.

7 Conclusion and Future Work

7.1 Conclusion

This research set out to address a critical trust deficit in the domain of autonomous cloud infrastructure management. The central research question was: **How can blockchain technology be effectively integrated with a cloud computing environment to enable trustworthy and transparent logging of autonomous infrastructure management operations?** Through the design, implementation, and rigorous evaluation of a proof-of-concept system, this project has provided a definitive and affirmative answer.

The work successfully demonstrated that by treating autonomous actions as transactions to be recorded on a distributed, immutable ledger, a new level of trust and accountability can be achieved. The project's key objectives were met. A novel, four-layer architecture was designed, separating the concerns of cloud infrastructure, automation logic, blockchain trust, and auditing. A functional prototype was built using industry-standard tools—AWS, Python, Solidity, and Flask—proving the practical feasibility of the design. The system was evaluated through a series of controlled experiments that confirmed its ability to monitor a cloud resource, autonomously respond to an anomaly, and, most importantly, create a permanent, verifiable, and easily auditable record of its actions on a blockchain.

The primary contribution of this research is a validated framework that serves as a blueprint for building next-generation autonomous systems where transparency is not an afterthought but an integral part of the design. By anchoring operational logs in the cryptographic certainty of a blockchain, this approach mitigates the risks associated with mutable, centralized logging systems. It provides the non-repudiation necessary for high-stakes environments, enabling robust post-incident forensics, simplifying compliance audits, and ultimately fostering greater confidence in the automation that underpins modern IT. While the implementation was a proof-of-concept, it successfully established the viability and value of this approach.

7.2 Future Work

This research opens up several promising avenues for future work, building upon the foundation established by this project.

- **Implementation of Advanced Autonomous Scenarios:** Future research could expand the system's logic to handle more complex scenarios. This could involve multi-metric triggers (e.g., CPU, memory, and network I/O), predictive-failure triggers based on machine learning models, and more sophisticated self-healing actions, such as automatically migrating services away from a failing instance.
- **Adoption of Event-Driven Architecture:** To address the log retrieval scalability issue, the smart contract should be re-architected to emit a 'LogActionEvent' whenever an action is recorded. A separate off-chain service could then be built to subscribe to these events and populate a conventional, indexed database (like

PostgreSQL or Elasticsearch) for fast, efficient querying and analytics, while the blockchain remains the ultimate guarantor of data integrity.

- **Deployment on Consortium Blockchains:** For enterprise use cases, deploying this framework on a permissioned or consortium blockchain like Hyperledger Fabric or Quorum would be a logical next step. This would allow a group of trusted organizations (e.g., a company, its clients, and its auditors) to participate in the network, providing a shared, immutable view of operations while maintaining privacy from the public.

By pursuing these future directions, the principles demonstrated in this project can be developed into mature, robust, and scalable systems that fundamentally enhance the trustworthiness of our increasingly automated world.

References

[1] Alam, T. (2024). Data privacy and security in autonomous connected vehicles in smart city environment. *Big Data and Cognitive Computing*, 8(9), p.95.

[2] Alaya, H., Ben Letaifa, A. and Rachedi, A. (2025). State of the art and taxonomy survey on federated learning and blockchain integration in UAV applications. *The Journal of Supercomputing*, 81(5), p.655.

[3] Albshaier, L., Budokhi, A. and Aljughaiman, A. (2024). A review of security issues when integrating IoT with cloud computing and blockchain. *IEEE Access*.

[4] Asif, R., Hassan, S.R. and Parr, G. (2023). Integrating a blockchain-based governance framework for responsible AI. *Future Internet*, 15(3), p.97.

[5] Das, D., Banerjee, S., Chatterjee, P., Ghosh, U. and Biswas, U. (2023). Blockchain for intelligent transportation systems: Applications, challenges, and opportunities. *IEEE Internet of Things Journal*, 10(21), pp.18961-18970.

[6] Datta, S., Namasudra, S., Moparthi, N.R., Kumari, S. and Crespo, R.G. (2025). Transforming Healthcare With Artificial Intelligence and Blockchain: A Secure, Transparent and Energy-Efficient Approach. *Expert Systems*, 42(8), p.e70101.

[7] Hevner, A.R., March, S.T., Park, J. and Ram, S. (2004). Design science in information systems research. *MIS quarterly*, 28(1), pp.75-105.

[8] Jain, S., Ahuja, N.J., Srikanth, P., Bhadane, K.V., Nagaiah, B., Kumar, A. and Konstantinou, C. (2021). Blockchain and autonomous vehicles: Recent advances and future directions. *IEEE Access*, 9, pp.130264-130328.

[9] Karaduman, Ö. and Gülhas, G. (2025). Blockchain-Enabled Supply Chain Management: A Review of Security, Traceability, and Data Integrity Amid the Evolving Systemic Demand. *Applied Sciences (2076-3417)*, 15(9).

[10] Karim, M.M., Van, D.H., Khan, S., Qu, Q. and Kholodov, Y. (2025). AI agents meet blockchain: A survey on secure and scalable collaboration for multi-agents. *Future Internet*, 17(2), p.57.

[11] Khan, R., Mehmood, A., Maple, C., Curran, K. and Song, H.H. (2023). Performance analysis of blockchain-enabled security and privacy algorithms in connected and autonomous vehicles: A comprehensive review. *IEEE Transactions on Intelligent Transportation Systems*, 25(6), pp.4773-4784.

[12] Lakhlef, H., Lerner, T., Kebir, A., El Atia, N., Du, X. and Ingardin, V. (2024, June). Blockchain-Enabled SDN Solutions for IoT: Advancements, Discussions, and Strategic Insights. In *2024 IEEE Symposium on Computers and Communications (ISCC)* (pp. 1-6). IEEE.

[13] Li, X., Wang, Z., Leung, V.C., Ji, H., Liu, Y. and Zhang, H. (2021). Blockchain-empowered data-driven networks: A survey and outlook. *ACM Computing Surveys (CSUR)*, 54(3), pp.1-38.

[14] Nguyen, T.L., Nguyen, L., Hoang, T., Bandara, D., Wang, Q., Lu, Q., Xu, X., Zhu, L. and Chen, S. (2025). Blockchain-empowered trustworthy data sharing: Fundamentals, applications, and challenges. *ACM Computing Surveys*, 57(8), pp.1-36.

[15] Pandey, A., Obaidat, M.S., Kushwaha, A., Dwivedi, S.K. and Amin, R. (2025, July). Exploring Blockchain in the Metaverse: Review of State-of-the-Art Frameworks and Future Research Challenges. In *2025 International Conference on Computer, Information and Telecommunication Systems (CITS)* (pp. 1-6). IEEE.

[16] Raza, A., Badidi, E., Hayajneh, M., Barka, E. and El Harrouss, O. (2024). Blockchain-based reputation and trust management for smart grids, healthcare, and transportation: a review. *IEEE Access*.

[17] Shinde, N.K., Seth, A. and Kadam, P. (2023). Exploring the synergies: a comprehensive survey of blockchain integration with artificial intelligence, machine learning, and iot for diverse applications. In *Machine Learning and Optimization for Engineering Design* (pp.85-119).

[18] Tychola, K.A., Voulgaridis, K. and Lagkas, T. (2024). Beyond flight: Enhancing the Internet of Drones with blockchain technologies. *Drones*, 8(6), p.219.

[19] Uddin, M., Khalique, A., Jumani, A.K., Ullah, S.S. and Hussain, S. (2021). Next-generation blockchain-enabled virtualized cloud security solutions: review and open challenges. *Electronics*, 10(20), p.2493.

[20] Yapa, C., De Alwis, C. and Liyanage, M. (2021). Can blockchain strengthen the energy internet?. *Network*, 1(2), pp.95-115.

[21] Zhong, B., Pan, X., Ding, L., Chen, Q. and Hu, X. (2023). Blockchain-driven integration technology for the AEC industry. *Automation in Construction*, 150, p.104791.