Cloud Infrastructure as Code (IaC) Generation of SIEM and Log Collection Infrastructure

Name: Szymon Kawecki

Student Number: C00286043

Course: Year 4, Bachelor (Hons) in SE606 Cybercrime & IT Security

Project Supervisor: Hisain Elshaafi

Project Type: Student Specified Project

1. ABSTRACT

The focus of this project is the development of a modular, GUI-driven Infrastructure as Code (IaC) solution for the deployment of a Security Information and Event Management (SIEM) system and the log collection infrastructure that supports it on Microsoft Azure. The solution, which combines CustomTkinter as the front-end interface and Terraform for IaC development, allows small and medium-sized businesses to implement enterprise-grade monitoring capabilities without depending on outside consulting services.
The built system includes Microsoft Sentinel for SIEM, Azure Log Analytics for log storage, Azure Data Explorer for backup, and other elements like virtual networks, load balancers, network security groups, NAT gateways, and Azure Function Apps for security. The idea emphasizes accessibility, affordability, and adaptability for customers with little technological expertise while displaying the secure, automated, and scalable deployment of cloud-based cybersecurity infrastructure. (Microsoft, n.d.)

This project will include a multi-phase methodology: (1) designing an intuitive GUI that captures user inputs, (2) programmatically (via code) generating Terraform (IaC) templates based on modular, reusable components, (3) deploying and validating Azure infrastructure through iterative testing, and (4) implementing a Python-based Azure Function App for external log ingestion using a non-native vendor API such as AWS CloudTrail. Emphasis is placed on security focused architecture, including network segmentation, private endpoints, managed identities, and automated SIEM integration. As an exceptional deliverable I would also like to look at ingesting Windows (Security) Events via Windows Event Forwarding/Windows Event Collection (WEF/WEC).

There may be several challenges that happen throughout the design and implementation of this project. This includes ensuring the IaC generation remains fully modular and adaptable to diverse client environments; while not losing functionality, providing a user-friendly interface for users with limited cloud expertise, maintaining a secure communication path for cross-cloud log ingestion, and guaranteeing Terraform dependencies and Azure resources interactions are properly orchestrated. Additional challenges include managing state during continuous log collection, validating user inputs, and optimizing the deployment process to avoid misconfiguration.

TABLE OF CONTENTS

2. INTRODUCTION

With the ever-evolving world of cyber threats, it has become increasingly vital for organizations; particularly those with more than 50 employees; to keep cybersecurity specialists. Traditionally, companies implement a Security Information and Event Management (SIEM) system to centralize threat detection, monitoring, and incident response. However, the deployment and management of a SIEM and its supporting infrastructure are widely regarded as technically complex, necessitating the use of third-party consultants for both deployment and ongoing operations. While these services offer significant benefits, such as active monitoring and professional management, they are often prohibitively expensive, particularly when combined with the ongoing operational costs of running a SIEM.

The purpose of this project is to provide an active monitoring SIEM/SOC solution for small- to medium-sized businesses that may lack extensive cybersecurity expertise, without requiring external consultancy or professional management. The solution is designed to be cost-effective, reliable, and scalable, enabling organizations to deploy enterprise-grade monitoring capabilities efficiently.

The system allows users to interact with a CustomTkinter-based GUI to generate a fully deployable stack of Terraform templates (Infrastructure as Code). This modular Terraform stack can be tailored to fit a wide variety of client environments, and all generated resources adhere to security best practices, including proper network segmentation, access control, and identity management.

This research document examines the technical intricacies of each element of the project, including GUI design, Terraform template generation, Azure resource deployment, and log ingestion mechanisms. By providing this in-depth analysis, the document aims to support the development, validation, and reporting of the solution, demonstrating how modular, GUI-driven IaC can make enterprise-level cybersecurity monitoring more accessible, efficient, and cost-effective for small- to medium-sized organizations.

3. RELATED WORK

Based on our last iteration of the Project Specification document, there are already several key decisions that have been made. These decisions are regarding the design, architecture, and implementation of this modular, GUI-driven IaC solution for deploying a SIEM, its backend infrastructure, and log collection infrastructure. (Kawecki, 2025) CustomTkinter was selected as the front-end GUI due to it being Python-native and easy to integrate other Python code for the back end, cross-platform support (Windows, MacOs, Linux), simplified learning curve over PyQt, and fulfilling all our UI element requirements. We have also decided upon the use case a.k.a how a user will interact with the GUI and other elements, a brief overview of how the infrastructure will be created using Terraform, and the technical layout.

Microsoft Sentinel, Azure's native SIEM solution, has been selected for its seamless integration with other Microsoft standard products such as 365, and Defender, built-in SOAR playbooks, as well as Microsoft Azure's extensive global datacenter coverage, scalability, reliability, and monitoring capabilities. Terraform was selected as the Infrastructure as Code (IaC) tool over Azure ARM templates due to its modularity meaning it is reusable and easier to migrate to other cloud providers, state management across files and actual Azure resources allowing updates and modifications without overwriting critical infrastructure, the readability of Hashicorp Language (HCL) which is a lot better than JSON based ARM templates, dependency handling, and multi-vendor support.

3.1. SIEM in the Cloud

There is a clear shift from traditional on-premises SIEM installations to virtualized and cloud-based security-as-a-service models, according to the literature I have found. The cost, scalability, and resource requirements of SIEM infrastructure maintenance are the key forces behind this progress. According to Velasquez et al. (2023), virtual environments have gradually taken the place of physical SIEM deployments, and "the latest and most recent step has been to provide this monitoring security as a service in the cloud" because vendors are incorporating cloud-native features into their products because they understand how valuable they are. (López Velásquez et al., 2023).

It emphasizes the importance of cloud computing for the advancement of SIEM. Specifically, SIEM is moving toward standard-driven, compliant, service-oriented designs; this shift is made easier by cloud solutions, quick event processing, and compliance controls. In line with more general trends like containerization, automated detection, and enhanced interoperability among security platforms, the report specifically refers to cloud SIEM as part of the broader modernization of the technology. (López Velásquez et al., 2023).

Supporting studies referenced in the review (e.g., Lee et al.) further propose SIEM delivered through cloud-based Security-as-a-Service models, emphasizing that virtualization and cloud infrastructure can strengthen threat recognition and reduce the overhead of maintaining complex SIEMs. This aligns with broader industrial trends where SIEM workloads such as log collection, normalization, correlation, and reporting are increasingly offloaded to cloud-native platforms to improve agility and responsiveness.

3.2. IaC security

nfrastructure as Code (IaC) introduces significant security considerations due to the declarative and highly automated nature of provisioning cloud environments. While IaC provides consistency, repeatability, and standardised deployment workflows, research shows that misconfigurations and insecure patterns within IaC templates can propagate rapidly across entire environments when deployed at scale. A large-scale empirical study analysing 15,232 real-world IaC scripts identified 21,201 security smell occurrences, demonstrating that insecure configurations are both common and persistent in IaC-based deployments (Rahman et al., 2019).

These “security smells” represent recurring patterns that indicate a potential vulnerability, such as hard-coded secrets, insecure file permissions, weak or missing cryptographic configurations, improper IP bindings, and the continued use of insecure

HTTP over HTTPS. Many of these issues stem from developers prioritising fast deployment over secure configuration, resulting in security weaknesses becoming embedded directly into automation workflows. The study also highlights that such weaknesses often remain in production for extended periods (up to 98 months in some cases) because IaC templates are reused across projects and repeatedly redeployed (Rahman et al., 2019).

This directly relates to log-collection infrastructure and cloud SIEM. Every security flaw identified in IaC will be replicated throughout the deployed Azure infrastructure, including excessively permissive network rules, exposed credentials, and insecure storage resources. These flaws potentially compromise essential monitoring, ingestion, and storage components for a security-sensitive implementation like a SIEM. Hard-coded secrets might grant unauthorized access to Function Apps or storage accounts, and incorrectly configured network configuration could expose ingestion endpoints to the internet. These highlight the significance of putting in place secure-by-default IaC templates, stringent GUI input validation, and automated checks to stop the introduction of unsafe configurations. (Rahman et al., 2019).

The system can reduce these risks and prevent the kinds of persistent, systemic flaws frequently found in IaC research by integrating security-focused coding techniques into the Terraform-generation procedure. This entails implementing least-privilege role assignments, utilizing HTTPS-only communication, enforcing stringent NSG rules, making sure secrets are kept in secure locations, and avoiding insecure defaults. Since any misconfiguration within IaC could be reproduced and scaled across all deployed customer settings, proactive identification and rectification of IaC security issues is crucial, as demonstrated by the literature (Rahman et al., 2019).

4. TECHNOLOGY REVIEW

This section will cover the research of all the individual technologies that will be used in the development of this project from start to finish and support the underlying solution. As mentioned in the above section, most decisions have already been made around the core technologies that will be utilized and therefore this section and research document will focus on how I should and will utilize the technology to achieve the expected outcome and solution.

4.1. Microsoft Sentinel

Microsoft Azure offers a cloud-native SIEM solution called Microsoft Sentinel. It offers an affordable, scalable SIEM platform that can be accessed through the Azure Portal. This platform allows for the development of automation and the addition of optional threat intelligence to enable active threat detection, investigation, response, and hunting. For log ingestion, monitoring, alerting, searching, investigating, and responding, it has a vast library of pre-made SIEM solutions. Additionally, the platform supports other vendors for a variety of additional goods, services, and platforms. (Microsoft, 2025)

Microsoft Sentinel allows for direct and seamless integration of security infrastructure.

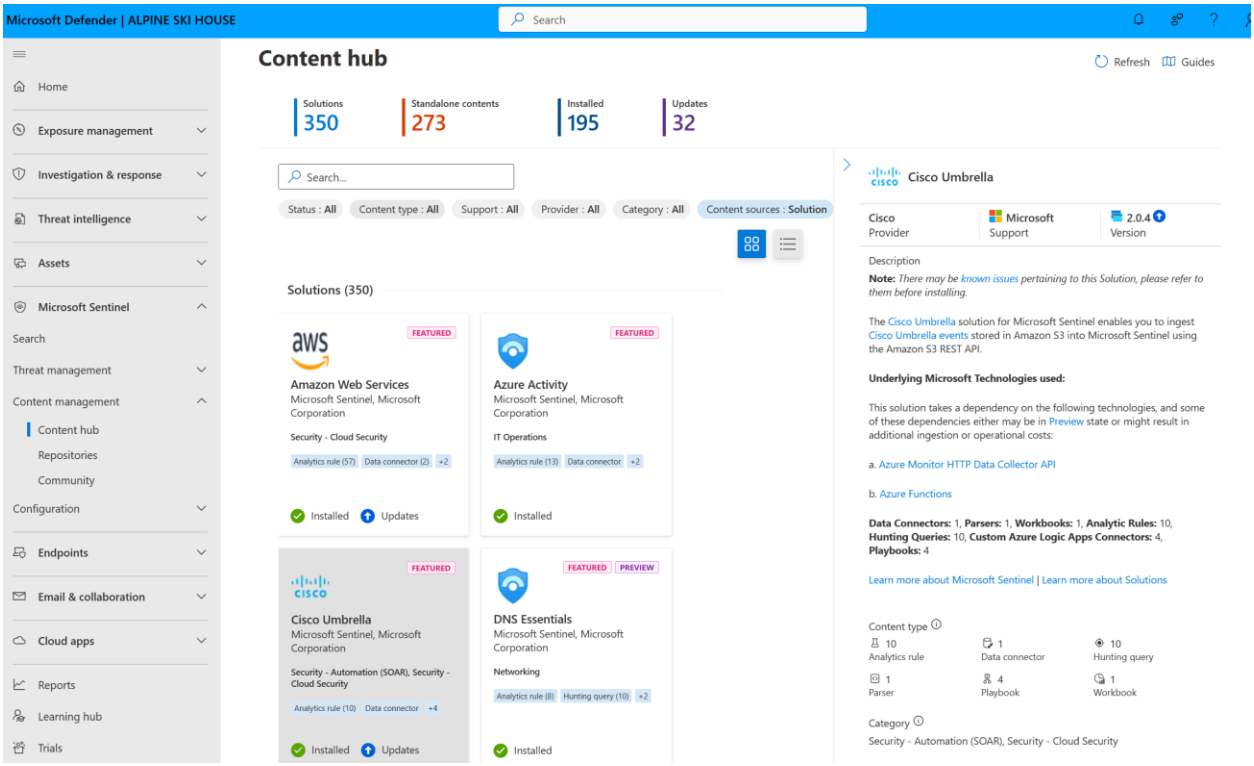


Fig. 1. Microsoft Defender Content Hub available with Microsoft Sentinel which allows for multi-vendor support

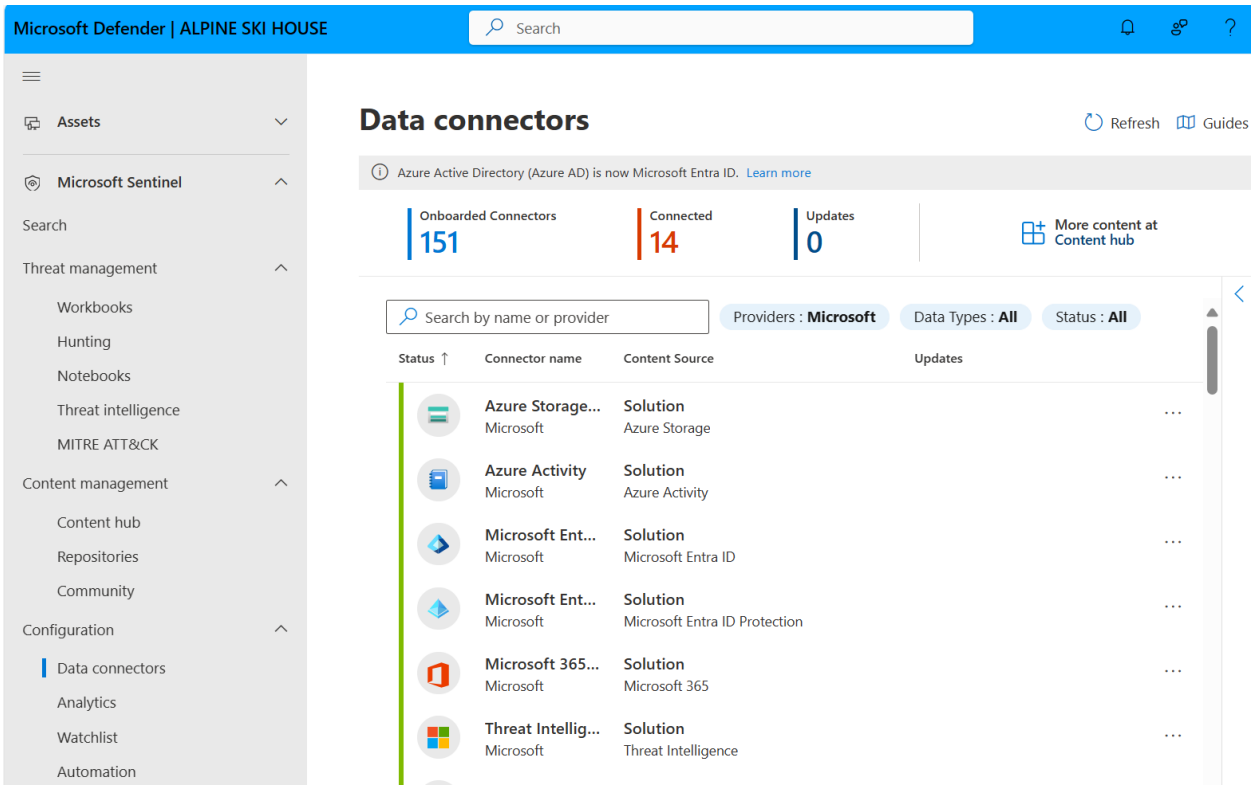


Fig. 2. Microsoft Defender Data connectors library that provides out-of-the-box solutions for ingestion logs into Microsoft Sentinel.

In this project, we will be utilizing Microsoft Sentinel itself so users can take advantage of the Microsoft Defender Content Hub for the development of automation, detections, investigations, response, and hunting. While this research document and project do not entirely plan to encompass any deployment from the available Content Hub libraries, this may happen if time allows it. This section is however here to show the function-ability of the selected SIEM for this solution and potential. Use of the Microsoft Defender Content Hub falls in line with our expectation of technically knowledge when it comes to the target audience of this solution; a way of seamless integration of automations and multi-vendor solutions into a SIEM with minimal expertise.

4.2. Terraform

(HashiCorp, 2023)

Terraform by HashiCorp is an open-source Infrastructure as Code (IaC) tool with multi-vendor support that allows for the creation of cloud and on-premises infrastructure.

The Terraform module in question that I will be using for this project is the AzureRM (ARM) module which you can see here: https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs. After defining the AzureRM provider module, we can then start creating resources using the infrastructure as code templates the module provides. This provider has 100s of modules from storage resources, DNS, load balancing, testing, networking and many more. (Hashicorp, n.d.)

Our purpose with Terraform will be to take user inputs (multiple choice) from the CustomTkinter GUI. These user inputs will also be supplemented by arguments such as naming conventions. These inputs grouped together will result in various Terraform templates featuring resources such as seen below:

Azure Resource Group that will act as the parent container holding all resources:

- azurerm_resource_group - Azure Resource Group, see above

Azure Virtual Network that will hold and isolate all resources:

- azurerm_virtual_network - Azure Virtual Network that will provide network isolation for all resources
  azurerm_subnet - a network subnet to isolate traffic
  azurerm_private_endpoint - for securing traffic to resources such as an Azure Function App, storage components, Microsoft Sentinel etc.
  azurerm_private_dns_zone - required for private endpoints

Azure Log Analytics (ALA) workspace a.k.a Microsoft Sentinel (SIEM):

- azurerm_log_analytics_workspace - Microsoft Azure Log Analytics Workspace for storing SIEM logs
  azurerm_log_analytics_solution - for enabling Microsoft Sentinel on the above ALA workspace

Azure Data Explorer (ADX) solution for backing up SIEM logs:

- azurerm_kusto_cluster - Microsoft Azure Data Explorer (ADX) Cluster itself for compute, ingestion, and capability. Storage capacity backup for SIEM logs
  azurerm_kusto_database - Microsoft ADX database to store tables
  azurerm_kusto_table - Microsoft ADX table that holds the logs with the defined ingestion mapping (schema)

Azure Load Balancer which equally distributes network traffic and workloads:

- azurerm_lb - Azure Load Balancer for ingestion of any logs coming from an external API
  azurerm_lb_backend_address_pool - Back-end server IPs that receive the traffic
  azurerm_lb_probe - Health probe to check backend availability
  azurerm_lb_rule - A rule mapping attached to the load balancer that permits traffic azurerm_public_ip - A public IP attached to the load balancer that can be used for log ingestion

Azure Network Security group that defines permit/deny traffic rules into your network:

(Microsoft, n.d.)

- azurerm_network_security_group - A Network Security Group that provides definition for permit/deny traffic rules
  azurerm_network_security_rules - The specific allow/deny rules, e.g. allowing traffic from an external log vendor API but no-one else
  azurerm_subnet_network_security_group_association - Binds a Network Security Group to a subnet

Azure Function App for handling custom vendor API log ingestion:

- azurerm_function_app - A serverless compute logic Azure Function App that runs code on demand
  azurerm_app_service_plan - Compute plan definition for an Azure Function App
  azurerm_storage_account - Required for storing any function code and triggers, also perhaps state management information
  azurerm_application_insights - Telemetry for function performance and errors
  azurerm_monitor_diagnostic_setting - For the above resource, for collecting diagnostics

Azure User Assigned Identity and Role Assignments to allow the Azure Function App to access key resources without plain-text connection strings:

- azurerm_user_assigned_managed_identity - a user assigned managed identity that allows you to isolate and control access to resources
  azurerm_role_assignment - the actual RBAC assignments for a user managed identity

Azure NAT gateway for allowing internal (private) resources communicate with the outside internet:

- azurerm_nat_gateway – For providing outbound internet access from private subnets.
- azurerm_nat_gateway_public_ip_association – To attach a public IP to the NAT Gateway.
- azurerm_subnet_nat_gateway_association – To link the NAT Gateway to your subnet(s).

A lot of these resources contain dependencies on other resources or values of said resources after creation. For example, an Azure Function App will wait for the Azure Storage Account to be created so it can inherit and reference its resource ID. Terraform handles this seamlessly.

Challenges that could arise:
While there are many advantages to using Terraform, such as automation, consistency, and repeatability, there are also several security issues that must be carefully considered when deploying sensitive infrastructures, such as log-collection pipelines and SIEM systems. These hazards originate from poor settings, management of credentials, disclosure of state files, and the difficulty associated with maintaining cloud resources on a big scale, such as:

- Disclosure of Private Information (such as Secrets) in State Files

Terraform's state file (terraform.tfstate) contains the current state of all managed resources. This file may contain extremely sensitive data such as resource IDs, keys and secrets, connection strings, and IAM role definitions.
If this file is not adequately protected; particularly in remote backends like Azure Storage, it could result in privilege escalation or unauthorized access to essential resources within the SIEM framework.

For this project, guaranteeing state encryption, storing in a secure backend, and implementing role-based access control are vital. This is going to be done by Terraform’s recommended HCP application that allows encryption of state when it is not being used, and more.

- Misconfigured Resources Due to Human Error

IaC introduces the risk of deploying insecure infrastructure at scale. A single mistake in a Terraform module, such as:
- Allowing overly permissive NSG rules (localhost, 0.0.0.0/0)
- Misconfigured load balancer rules
- Incorrect private endpoint assignments
- Incorrectly assigned identities or RBAC roles
  (can be applied across an entire environment instantly).

Because Terraform is declarative, once the configuration is applied, vulnerabilities propagate automatically. This risk increases when end-users generate their own

Terraform templates using the CustomTkinter GUI, making input validation and secure defaults critical; which will be a key focus of the GUI creation.

- Over-Privileged Identities and Role Assignments

Terraform automatically provisions Azure resources, which often requires service principals or managed identities. Risks include:

- Assigning overly broad roles such as Owner or Contributor
- Allowing the Terraform identity to create high-privilege resources
- Forgetting to remove temporary deployment permissions

For a SIEM system, identity misconfiguration could result in unauthorized access to security logs, ADX storage, or the Azure Function App used for log ingestion. Therefore, we will have to carefully review Microsoft's IAM recommendations to properly assign the correct privileges without leading to security risks within the generated infrastructure.

- Dependency Errors Leading to Insecure Defaults

IaC depends heavily on correct ordering and module logic. Unmet dependencies may cause:

- Resources being created before security components (e.g., Function App before NSG or Private Endpoint)
- Temporary public exposure while resources deploy
- Incorrect referencing of IDs, keys, or access policies

Although Terraform typically handles dependencies automatically, custom-generated templates may introduce ordering issues if not carefully validated. I will have to carefully structure code templates to prevent this from occurring.

- Risks When Using Community Modules or Unverified Code

While this project uses the official AzureRM provider, in other IaC deployments there is a risk in:

- Downloading unverified Terraform modules
- Using outdated provider versions
- Pulling insecure examples from GitHub

Unverified modules may include insecure defaults such as public access, unencrypted storage, or excessive role permissions. In this case; we will avoid using said community modules or any unverified code.

4.3. CustomTkinter

The GUI (graphical user interface) for this project will be developed using CustomTkinter as it will allow users to interact with a front-end interface to aid with the configuration of infrastructure. This app will dynamically build Terraform resources using the AzureRM Terraform module mentioned in the above section based on user selections, allowing for inputs such as naming conventions, resource tags, resource selection, NSG rules, and more. The application will also provide validation to prevent misconfiguration. This will start with a base code featuring just a simple customtkinter.CTk app defined with a geometry variable for resolution. The CtkLabels will be used to provide a label for each entry, supported by a CtkButton for buttons, CtkEntry for a text box to accept user entry, CtkSwitch for toggle on and off options for optional resources, same with CtkCheckBox for multiple choice options, CtkButton for a visual button to allow for the code generation copying and so on. There will also be other code features to configure the appearance of the application such as Set_Appearance_Mode, TabView(s) which I intended to hold the main configuration menu alongside a help menu defining everything. User inputs will be collected from the elements such as CtkEntry, CtkSwitch and so on which will allow us to run the Terraform code generation based on multiple templates. This will occur at the back end after the user presses the visual CtkButton to submit their options for generation. The generated Terraform code will be exported into a sub-directory where the application exists. Alongside a read-me documentation file; a user will be able to connect to an IDE of their choice, set up authentication against Microsoft Azure, and simply push the code which will start the code creation.

5. EXTERNAL LOG INGESTION

As a mandatory deliverable of this project; I will be creating an additional, non-native log collector that will feature Python 3.11 code running via an Azure Function App. This will be an optional choice in the CustomTkinter GUI. (Kawecki, 2025)

This was a decision made as it could be expanded into a library of said non-native collectors in the future for the end user to pick from. Not always do you find native collectors in the Microsoft Defender Content Hub for external vendors; however, Microsoft does already have an expansive library of Azure Function App code you can download and deploy to achieve log ingestion from external APIs. You can find the GitHub library here: https://github.com/Azure/Azure-Sentinel/tree/master/DataConnectors.

To build and highlight this non-native log collector; we need to choose an example vendor that will allow us to generate logs and access their API to poll these logs. Due to this being a university project, this must be done for free. The vendor and the API should also be hosted online; this means that we are not hosting a Docker container or otherwise locally which would hinder the showcase of how we are able to safely poll external APIs in the World Wide Web and securely bring them into our SIEM.

Due to a lackluster number of results when looking for free, API-pollable platforms that generate logs by themselves from user actions, security logs, and such, I decided to go with **AWS CloudTrail**.

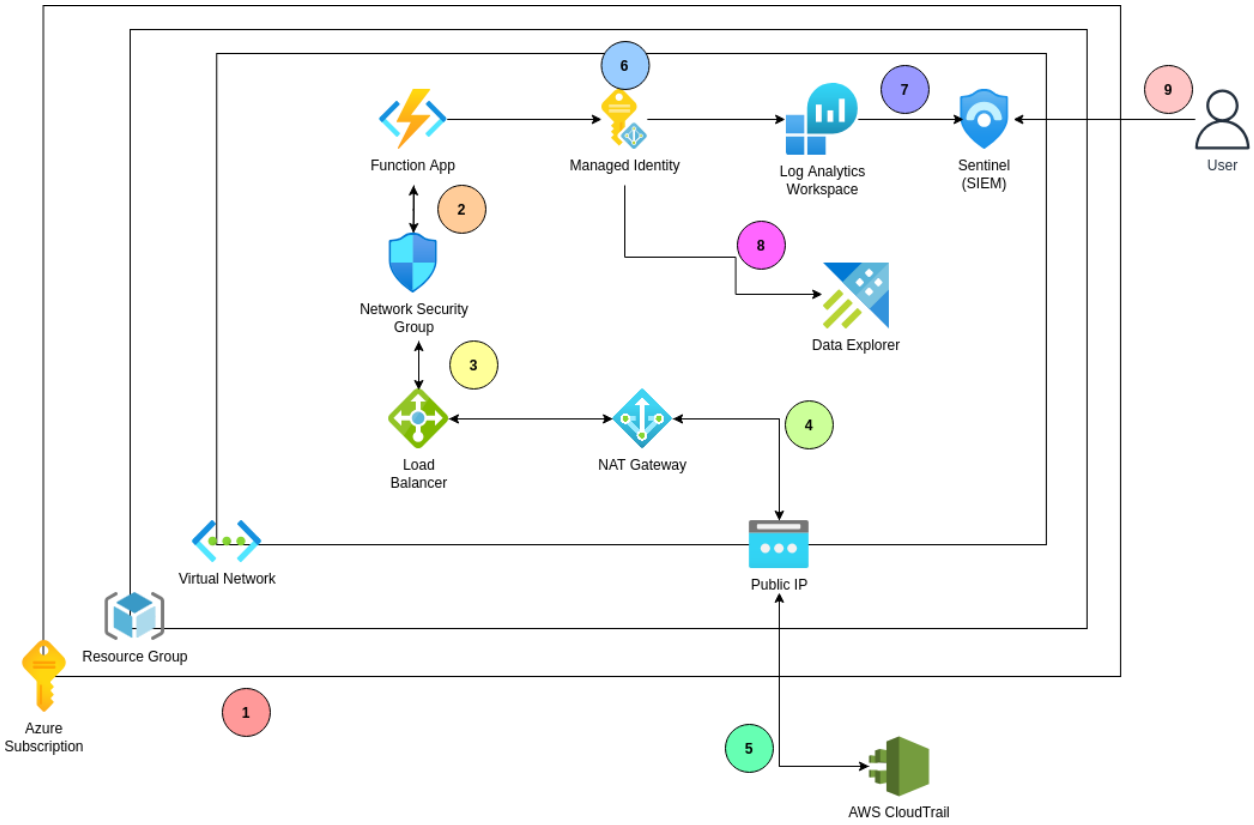
AWS CloudTrail is one of the only options I have found, but also appears to be a great option because:

- **Free management event logging:** AWS CloudTrail automatically logs management events for free, allowing us to generate extensive events during testing without incurring any costs. These events include actions such as creating or deleting resources, role changes, and API calls.
- **LookupEvents API:** CloudTrail provides the **LookupEvents API**, which allows us to poll logs programmatically at no cost. This satisfies the requirement of having an external API that can be safely integrated into our SIEM.
- **Cross-platform relevance:** Using AWS CloudTrail demonstrates that even when working in environments like Microsoft Azure, it is possible to log and monitor notable events occurring in external cloud platforms. This highlights the duality and flexibility of integrating cross-cloud security monitoring.

We will use AWS CloudTrail to log user actions such as CreateBucket (creating an S3 bucket), RunInstance (EC2), CreateUser (IAM) and more. These logs will then be ingested by our Azure Function App into Microsoft Sentinel and Data Explorer as a showcase that the app itself is able to autonomously (on a cron scheduled job) download logs while maintain state and more. This setup will highlight the ability to safely collect and analyse external cloud platform events within a centralized SIEM. (Amazon Web Services, 2024) (Redmond, 2021) (Microsoft, 2023a) (Microsoft, 2023b)

6. INFRASTRUCTURE LAYOUT

The diagram in this section shows the intended infrastructure layout because of the IaC generation in a client's Azure environment including all the way up to the subscription and resource group. It features robust outbound and inbound traffic handling via a network security group (think permit/deny statements), a load balancer, NAT gateway (for private to public traffic processing) and out to the internet. It also covers the traffic flow into ALA and ADX as well into Sentinel (SIEM solution). (Microsoft, n.d.) (Redmond, 2021) (Microsoft, 2023a) (Microsoft, 2023b)



1. Azure Subscription, Resource Group, and Virtual Network. The core 3 nested resource containers in which the infrastructure will rest.
2. Azure Function App's inbound and outbound traffic is secured under strict permit/deny statements by the Network Security Group which handles what can come in and what can come out. (Microsoft, n.d.)
3. Traffic passes through a Load Balancer before arriving at the Network Security Group. This allows for efficient distribution of traffic to not stress resource compute.

4. Traffic in and out of a NAT Gateway is subject to IP address resolution to determine the source and destination regardless of whether the traffic is of private or public derivative.
5. A Public IP guards the edge of the virtual network as the primary access point. This is what resources (Azure Function App mainly) will use to communicate with the internet.
6. Tagged with the Public IP, API calls will be made to AWS CloudTrail to retrieve logs and bring them back into the Virtual Network to eventually end up in the Function App.
7. The Function App utilizes a Managed Identity with pre-defined permissions to communicate with ADX and ALA.
8. Logs from the ALA will be carried to Microsoft Sentinel using a Data Connection Endpoint and Data Connection Rule.
9. A backup set of logs will also be stored in ADX, however without the need for a DCR or DCE.
10. An end-user can now log into Microsoft Sentinel as their SIEM solution and review logs, build automatics, and review analytics.

6.1. Strengths of the Infrastructure

In this section, we will discuss the strengths of the infrastructure researched, planned, and outlined here in 6. Infrastructure Layout.

1. Strong Network Isolation and Traffic Control

The use of a Virtual Network (VNET), Network Security Groups (NSGs), NAT Gateway, and Private Endpoints enforces a high degree of network segmentation.

Key benefits:

- Minimizes exposure of critical resources such as Function Apps, ADX, and Log Analytics.
- Reduces attack surface by controlling outbound and inbound flows.
- Ensures predictable egress IPs via the NAT Gateway, which is essential when integrating with external platforms like AWS CloudTrail.

This design aligns with a zero-trust model where each subnet and resource is strictly controlled.

2. Scalable and Modular Deployment Using Terraform

Terraform modules allow:

- Reuse of infrastructure components across clients
- Automated provisioning, reducing human misconfiguration
- Consistent architecture across environments

This modularity also supports future expansion (e.g., adding additional log collectors or SOAR components).

3. Cloud-Native SIEM with Multi-Vendor Support

Microsoft Sentinel provides:

- Global availability and scalability
- Native integration with Azure's security ecosystem
- Support for external vendor ingestion through Function Apps and Data Connectors

This approach makes the solution universally applicable to SMEs without requiring specialist SIEM engineering knowledge.

4. Reliable Backup and Long-Term Log Storage via Azure Data Explorer

Using ADX provides:

- High ingestion throughput
- Efficient querying for analytical investigations
- Cheaper long-term retention compared to Log Analytics Workspace

Storing logs in both Sentinel and ADX enhances resilience and incident investigation capability.

5. Automated External Log Ingestion Pipeline

The Function App provides:

- Scheduled or event-driven external log ingestion
- A secure place to run custom Python collectors
- Integration with Managed Identities for secure access

This avoids dependency on third-party collectors.

6.2. Weaknesses of the Infrastructure

In this section, we will discuss the weaknesses of the infrastructure researched, planned, and outlined here in 6. Infrastructure Layout.

1. Cost Overhead for Small Organizations

While scalable, the architecture includes components that may be expensive for exceedingly small clients:

- ADX clusters
- NAT gateways (per-hour charges)
- Load balancers
- Log Analytics ingestion fees

Although designed for SMEs, cost optimization must be carefully considered.

2. Increased Complexity for Non-Technical Users

Even with a GUI-driven IaC generator, SMEs may still face challenges:

- Understanding Azure authentication
- Managing Terraform state
- Monitoring the Function App
- Maintaining SIEM rules and analytics

The architecture simplifies deployment but not long-term operations.

3. Potential Bottlenecks in the Network Path

Routing traffic from AWS → Public IP → Load Balancer → NSG → Function App introduces multiple hops.

Potential drawbacks:

- Added latency
- More failure points
- More components to maintain

For high-throughput log ingestion, this might become a limitation.

4. Reliance on Terraform State Management

State file corruption, mismanagement, or unauthorized access poses security and operational risks.

Even with remote state backends, organizations must ensure:

- Proper access control
- Encryption
- Versioning and backups

Without this, deployments can become inconsistent or insecure.

This is typical in SME environments but should be acknowledged.

7. CONCLUSION

The project aims to effectively show how enterprise-level SIEM and log collection infrastructure deployment and management can be made easier using a modular, GUI-driven IaC solution. The solution aims to offer small to medium-sized businesses a safe, scalable, and affordable framework for implementing active cybersecurity monitoring by integrating CustomTkinter for user interaction and terraform for automated resource development. Important conclusions include the capacity to:

- Create and implement a full Azure infrastructure stack while following best practices for identity management, network segmentation, and access control.
- For monitoring and backup, import logs into Microsoft Sentinel and Azure Data Explorer from both internal Azure sources and external cloud platforms (such AWS CloudTrail).
- Use automated IaC workflows to improve reproducibility, get rid of manual configuration errors, and simplify operations.

In the future, the platform may be extended for hybrid or multi-cloud installations, automated SOAR playbooks may be integrated, and the library of non-native log collectors may be expanded. This project aims to demonstrate how enterprise-grade cybersecurity monitoring can be made accessible, effective, and sustainable for businesses with little internal experience using modular IaC and GUI-driven tooling.

Several possible vulnerabilities may emerge while developing these tools and must be addressed. These include the danger of sensitive information being exposed via Terraform state files, misconfigured network security rules due by improper or incomplete GUI inputs, overly liberal role assignments, and unsecured user-defined parameters that may result in unintended access paths. Additionally, unvalidated configuration logic in the CustomTkinter GUI or dependency-ordering problems within Terraform modules could introduce vulnerabilities. On the other hand, by enforcing secure defaults, utilizing private endpoints, implementing least-privilege IAM assignments, verifying configuration inputs, and implementing secure state management techniques, the system is made to reduce a number of typical dangers. When taken as a whole, these safeguards help prevent vulnerabilities including unsecured deployment pipelines, cloud resource exposure, privilege escalation, and credential leakage.

8. REFERENCES

Kawecki, S. (October 24th, 2025) Project Specification: Cloud Infrastructure as Code (IaC) generation of SIEM and log collection infrastructure. Available at: https://blackboard.itcarlow.ie/ultra/courses/_29920_1/cl/outline?legacyUrl=~2Fwebapps~2Fgradebook~2Fdo~2Fstudent~2FviewGrades%3Fcourse_id%3D_29920_1%26callback%3Dcourse (Accessed: October 26th, 2025)

Microsoft, (n.d.) What is Microsoft Sentinel security information and event management (SIEM)? Microsoft Corporation. Available at: https://learn.microsoft.com/en-us/azure/sentinel/overview?tabs=defender-portal (Accessed: Nov 18th, 2025)

Hashicorp, (n.d.) Azure Provider. Hashicorp Terraform Registry. Available at: https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs (Accessed: Nov 24th, 2025)

HashiCorp, (2023) Terraform Security Best Practices. HashiCorp Documentation. Available at: https://developer.hashicorp.com/terraform/language/security (Accessed: Dec 1st, 2025)

Microsoft, (n.d.) Network security groups (NSG). Microsoft Corporation. Available at: https://learn.microsoft.com/en-us/azure/virtual-network/network-security-groups-overview (Accessed: Dec 1st, 2025)

Amazon Web Services, (2024) AWS CloudTrail User Guide: Logging and Monitoring API Activity. AWS Documentation. Available at: https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-user-guide.html (Accessed: Dec 2nd, 2025)

Redmond, E., (2021) Serverless Security: Securing Functions in Cloud Environments. O'Reilly Media. Available at: https://www.oreilly.com/library/view/serverless-security/9781492083461/(Accessed: Dec 2nd, 2025)

Microsoft, (2023) (a) Azure Data Explorer Documentation – Best Practices for Log Ingestion. Microsoft Corporation. Available at: https://learn.microsoft.com/en-us/azure/data-explorer/data-explorer-best-practices (Accessed: Dec 2nd, 2025)

Microsoft, (2023) (b) Azure Functions Developer Guide. Microsoft Corporation. Available at: https://learn.microsoft.com/en-us/azure/azure-functions/functions-reference (Accessed: Dec 3rd, 2025)

López Velásquez, J.M. et al., (January 2nd, 2023) Systematic review of SIEM technology: SIEM-SC birth. International Journal of Information Security. Available at: https://link.springer.com/article/10.1007/s10207-022-00657-9 (Accessed: Dec 6th, 2025)

Lee, J.H, et al., (2017) Toward the SIEM architecture for cloud-based security services. 017 IEEE Conference on Communications and Network Security. Available at: https://doi.org/10.1109/CNS.2017.8228696 (Accessed: Dec 6th, 2025)

Rahman, A., et al., (2019). The Seven Sins: Security Smells in Infrastructure as Code Scripts. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). Available at: https://ieeexplore.ieee.org/document/8812041 (Accesed: Dec 7th, 2025)