**High level Architecture**

**<P06>:<Anomalous Login Detection System Using ELK Stack>**

**<team member names & ids>**

| **Student ID** | **Name** |
| --- | --- |
| **26100286** | **Mohammad Mustafa** |
| **26100015** | **Muhammad Aaffan Khan Niazi** |
| **26100399** | **Mustafa Hussain** |
| **25100022** | **Shehroz Faryad** |
|  |  |

**Table of Contents**

[1.](#_dwg7p68dtnng) Introduction 3

[2.](#_rn55u8wp1c2g) Non-functional requirements/Quality attributes of the system 4

[3.](#_w1d31fza6de8) Security Requirements 5

[4.](#_tjl3duaae7u7) Project Risk Management 6

[4.1 Potential Project Risks and Mitigation Strategies 6](#_8kdmi0pln18q)

[5.](#_je81l9vb1y8w) System Architecture 7

[5.1 Architecture Diagram 7](#_gft3ce8fjds9)

[5.2 Architecture Description 8](#_f052jc9dphk)

[5.3 Justification of the Architecture 9](#_qbz3go1lnp4o)

[6.](#_b6j7z9z3n2ib) Tools and Technologies 13

[7.](#_zemgk4560igc) Hardware Requirements 14

[8.](#_vtwmq0vkrzod) Development Environment Preparation 15

[9.](#_les7ifm5va8b) Deployment Platform 15

[10.](#_nsyar0cv8nr3) Use of Generative AI 16

[11.](#_x8hxud96o721) Who Did What? 17

[12.](#_5a39da7iso3l) Review checklist 17

# Introduction

<Give an overview of the project here. The overview must highlight the overall objectives of the project and its potential users and customers.>

The project is concerned with the detection of unusual or suspicious login activity in real-time using the ELK stack (Elasticsearch, Logstash, Kibana) along with Wazuh. The primary objective of the system is to collect authentication logs and search patterns that can possibly indicate a potential security problem. Such patterns are things like multiple attempts of logging in over a short span of time or abrupt alterations in the location where the user is logging in. The system is able to detect these behaviors and minimize the chances of unauthorized access.

System administrators and security engineers are the key individuals that will utilize this system. To them, the project offers a user-friendly dashboard that lets them easily view the activity of the login at a glance, and also offer instant notification whenever something out of the ordinary occurs. This implies that there is less time wasted in searching through raw logs and that action is taken much faster when an issue arises.

In general, the project will enhance the speed at which organizations can recognize and react to abnormal login activity. Having clear visibility, real time alerts and effective monitoring, system administrators and security engineers can ensure that they protect systems better and contain potential threats.

# Non-functional requirements/Quality attributes of the system

<List down non-functional requirements of your system here. List security requirements in a separate section.>

| 1 | The system should not utilize more than 1 GB of memory at any time during its execution. |
| --- | --- |
| 2 | The system should not fail more than 3 times every 24 hours; if it does, it should recover within 5 minutes. |
| 3 | The system should be able to process at least 10,000 log entries per second without performance degradation. |
| 4 | The alerting mechanism should deliver notifications within 30 seconds of anomaly detection. |
| 5 | System dashboards must be accessible only to authorized users. |
| 6 | The system should allow simultaneous access by at least 5 users without performance degradation. |
| 7 | The system’s dashboard must refresh with new data every 10 seconds to ensure real-time data is visible. |
| 8 | The system should undergo daily health checks with results logged for administrators. |

# Security Requirements

| **Sr#** | **Security Risks** | **Potential Losses** | **Controls** |
| --- | --- | --- | --- |
| 1 | Broken Access Control | Sensitive user login data is exposed. | Only security engineers will have update rights. |
| 2 | Input Manipulation Attack | Repeated attempts can cause the system to ignore the real threats. | Simulate the fake inputs to ensure the system’s accuracy remains unaffected. |
| 3 | Data Poisoning Attack | Can cause the system’s detection quality to drop over time. | Pre-process the data. |
| 4 | Using outdated versions of client-side and/or server-side components | Exploitation of known vulnerabilities in the older versions. | Obtain components from their official links |
| 5 | Hardcoding credentials | Credentials are exposed so hackers may be able to gain access | A security testing process would take place in order to ensure credentials are not exposed in such ways. |

# Project Risk Management

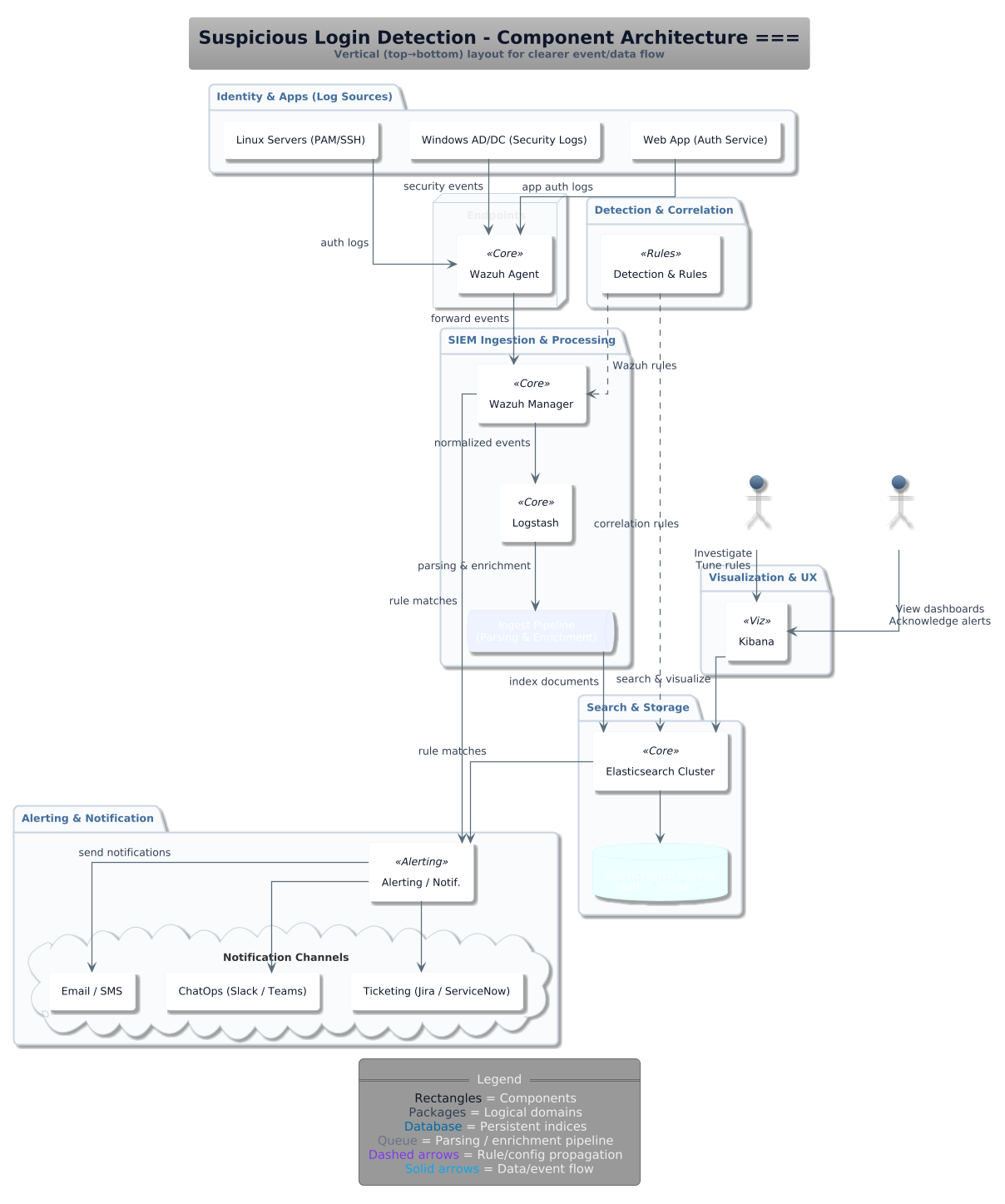
## Potential Project Risks and Mitigation Strategies

< A risk is a probability that some adverse circumstances will occur that will have negative impact on your project. Assume that the following risks may possibly occur. Suggest how you can mitigate each of the risks, i.e., what would be your strategy to avoid or minimize the effects of risks. Refer to the slides about risk management for examples.>

| **Sr.** | **Risk Description** | **Mitigation Strategy** |
| --- | --- | --- |
|  | **False positives**  The detection system might flag normal logins as suspicious ones, and this would lead to an overwhelming number of alerts. | Constantly tune detection thresholds and incorporate analyst feedback to improve the system’s accuracy over time. |
|  | **Log integration issues**  Logs from different platforms may not be properly parsed | Standardize log formats, use parsing templates, and make use of error logs to catch these issues early. |
|  | **Alert delivery failure**  The alerts may not reach the SOC team due to misconfigured settings or network problems. | Setup several redundant channels and test alert delivery regularly. |
|  | **Overloading the system**  A heavy volume of logs may overload the system and cause performance issues. | Optimize queries and enable horizontal scaling to maintain the system’s performance. |
|  | **Storage exhaustion**  Continuously saving the logs may lead to disk space filling up and halting the alerts. | Older logs should be automatically archived and monitor disk usage. |
|  | **Unauthorized rule modification**  A user might accidentally modify detection rules causing the system’s accuracy to drop. | Only security engineers will have rule editing privileges. |
|  | **Outdated detection rules**  The current detection rules may become obsolete when attackers change their methods. | Review and update the rules after a set time period (e.g. every month) to incorporate new attack patterns. |
|  | **Inadequate monitoring**  Alerts may be generated but not acted upon due to no personnel paying attention to them. | Assign SOC analysts for monitors and schedule daily summary reports for all incoming alerts. |

# System Architecture

## Architecture Diagram



## Architecture Description

| **Subsystem** | **Purpose / Description** | **Key Interactions** |
| --- | --- | --- |

| **Identity & Apps (Log Sources)** | Generates raw authentication and security logs from Linux servers (PAM/SSH), Windows AD/DC (security logs), and web applications. | Sends security, authentication, and application logs to the **Wazuh Agent** for collection and forwarding. |
| --- | --- | --- |

| **Endpoint (Wazuh Agent)** | Installed on servers and applications to collect raw logs and forward them securely. | Receives logs from log sources and forwards events to the **Wazuh Manager**. |
| --- | --- | --- |

| **Detection & Correlation** | Applies detection logic and rules to identify suspicious logins (e.g., brute-force attempts, geo-impossible logins). | Receives normalized events from the **Wazuh Manager**; sends alerts or matches back to the SIEM pipeline. |
| --- | --- | --- |

| **SIEM Ingestion & Processing (Wazuh Manager & Logstash)** | **Wazuh Manager** normalizes logs and applies detection rules; **Logstash** enriches logs (e.g., IP geolocation, user context). | Ingests logs from **Wazuh Agents**; outputs enriched events to **Elasticsearch**; sends alerts to the Notification system; propagates detection rules back to agents. |
| --- | --- | --- |

| **Search & Storage (Elasticsearch Cluster)** | Stores normalized and enriched events in persistent indices for fast search and querying. | Receives events from **Logstash**; serves data to **Kibana** for dashboards and investigations. |
| --- | --- | --- |

| **Visualization & UX (Kibana)** | Provides dashboards and visualization for security engineers and administrators to investigate suspicious logins. | Queries **Elasticsearch** for indexed events; shows alerts and dashboards; allows analysts to refine detection rules. |
| --- | --- | --- |

| **Alerting & Notification** | Sends actionable alerts when suspicious logins are detected through channels like Email, SMS, ChatOps, and Ticketing systems. | Receives rule matches from the **SIEM**; delivers alerts to email, chat tools, or ticketing systems for incident response. |
| --- | --- | --- |

## Justification of the Architecture

<List down pros and cons of the architecture you have defined in the context of your system. Moreover, give a justification of why this architecture is appropriate for your system. Make sure that you pick each of the non-functional requirements including security requirements and discuss how your proposed architecture helps in implementation of the non-functional requirements. >

* **Pros and cons of the architecture**

| **Pros** | **Cons** |
| --- | --- |
| **End-to-end coverage**: From log collection → normalization/correlation → storage/search → alerting/response (fewer blind spots).  **Mature OSS stack**: Wazuh + Elastic + Kibana + Logstash are widely adopted, well-documented, and extensible.  **Rule + enrichment pipeline**: Normalization in Wazuh + enrichment in Logstash (geoIP, user/org context) improves signal quality.  **Fast investigation**: Elasticsearch provides low-latency queries; Kibana gives analysts dashboards and timelines out of the box.  **Clear decoupling**: Collection, processing, storage, and UX are separate—easier to scale/replace any layer independently.  **Alerting fan-out**: Email/SMS, ChatOps, and ticketing integrations align with real incident workflows and SLAs.  **Feedback loop**: Rules/config updates propagate back to agents; continuous tuning reduces false positives. | **Operational overhead**: Managing multi-component stack (agents, manager, Logstash, ES, Kibana) requires solid SRE discipline.  **Elasticsearch cost/footprint**: Storage + memory heavy; hot/warm/cold tiers and ILM policies are necessary to control spend.  **Rule maintenance burden**: Keeping rules current for new TTPs (geo-impossible, MFA bypass, SSO quirks) is ongoing work.  **False positives risk**: Without tuning and asset/user context, analysts can be alert-fatigued.  **Agent footprint & coverage**: Requires agent rollout and lifecycle; some SaaS sources need alternative collectors/API ingestion.  **Single-stack dependency**: Heavy reliance on Elastic APIs; migrations to other data lakes require planned export/ETL. |

* **Implementation of non-functional requirements in system architecture**

| **Requirement** | The system should not utilize more than 1 GB of memory at any time during its execution. |
| --- | --- |
| **Implementation in the architecture** | * **At the edge (Wazuh Agents):** Forward only the authentication and security attributes needed for detections; drop verbose payloads and duplicates before they enter the pipeline. This limits the amount of data each downstream tier must hold in memory. * **Correlation & processing (Wazuh Manager & Logstash):** Normalize events early and keep enrichment minimal and deterministic (for example, add a user or IP context once). Avoid per-event lookups that expand state in memory or create long-lived buffers. * **Storage & search (Elasticsearch):** Use compact, time-series indices with lean field sets that match what analysts actually query. Prefer numeric/keyword fields over large text blobs to keep heap usage predictable during searches. * **Visualization (Kibana):** Default dashboards to recent time windows and shared queries. This improves cache hit rates and prevents large, memory-hungry aggregations across cold data. |

| **Requirement** | The system should not fail more than 3 times every 24 hours; if it does, it should recover within 5 minutes. |
| --- | --- |
| **Implementation in the architecture** | * **Fault isolation by tier:** Collection, processing, storage, visualization, and alerting are separate. A problem in processing does not prevent agents from collecting or analysts from viewing existing data. * **Health-driven recovery:** Each tier exposes a simple notion of “healthy” (e.g., manager available, processing accepting inputs, cluster answering searches, dashboard reachable). When a tier is unhealthy, it is cycled so it can rejoin cleanly without manual steps. * **Stateless fast start:** Processing paths avoid local state so they can restart quickly and continue from the current stream. * **Operational observability:** A small health overview in Kibana shows current tier status and the time since last successful alert, helping operators verify that recovery stayed within the five-minute window. |

# 

| **Requirement** | The system should be able to process at least 10,000 log entries per second without performance degradation. |
| --- | --- |
| **Implementation in the architecture** | * **Parallel ingestion:** Many Wazuh Agents send in parallel to the Manager & Logstash tier, which is sized for concurrent intake and light transformations. * **Throughput-first enrichment:** Enrich only with fields that materially improve detection (for example, user role or source geo). Avoid chaining enrichments that introduce variable delays. * **Write-friendly indexing:** Store events in time-based indices optimized for frequent writes, keeping shard counts low and field mappings tight to reduce indexing overhead. * **Elastic headroom:** Because tiers are decoupled, the ingestion layer can be scaled horizontally if sustained load approaches limits, without touching storage or dashboards. |

# 

# 

| **Requirement** | The alerting mechanism should deliver notifications within 30 seconds of anomaly detection. |
| --- | --- |
| **Implementation in the architecture** | * **Detect near the source:** Wazuh Manager applies rule logic on normalized events (e.g., repeated failures, geo-impossible sign-ins), generating alerts immediately after events arrive. * **Short alert path:** Alerts flow directly from processing to the Alerting & Notification component, which is responsible for fan-out to email, chat, and ticketing. This avoids extra hops that add delay. * **Frequent query-based checks:** Where correlation requires search (e.g., sliding-window thresholds), run evaluations on tight intervals over a short look-back so new events are captured quickly. * **Measure and prove:** A Kibana panel tracks the time between event arrival and notification; if this exceeds the target, a meta-alert informs operators. |

# 

# 

| **Requirement** | System dashboards must be accessible only to authorized users. |
| --- | --- |
| **Implementation in the architecture** | * **Single entry point:** Users access only Kibana. Elasticsearch, Logstash, and Wazuh Manager remain on private networking and are not exposed to user networks. * **Least-privilege roles:** Assign roles that match user responsibilities. Analysts get read-only access to the indices behind their dashboards; only designated engineers can edit rules or saved objects. * **Data scoping:** Align index permissions with dashboards so users cannot pivot into datasets they do not need. Keep system and audit indices visible only to administrators. * **Accountability:** Log sign-ins, role changes, and access to sensitive spaces so reviews can catch misuses or misconfigurations. |

# 

# 

| **Requirement** | The system should allow simultaneous access by at least 5 users without performance degradation. |
| --- | --- |
| **Implementation in the architecture** | * **Cache-friendly dashboards:** Build dashboards on shared, reusable queries and aggregations so multiple viewers benefit from cached results in Elasticsearch. * **Responsive defaults:** Keep default time ranges recent and avoid very expensive visualizations on landing pages. Heavy, historical views can live on secondary dashboards. * **UX is isolated:** Because dashboards are served from Kibana and ingestion happens separately, several investigators can work concurrently without affecting event intake or alerting. |

# 

# 

| **Requirement** | The system’s dashboard must refresh with new data every 10 seconds to ensure real-time visibility. |
| --- | --- |
| **Implementation in the architecture** | * **Fast path from edge to UI:** Events travel a short chain—Wazuh Agent → Wazuh Manager & Logstash → Elasticsearch → Kibana—with minimal enrichment, so new documents become searchable quickly. * **Hot-data focus:** Dashboards emphasize the latest time window and use sensible chart intervals, ensuring each refresh touches small, hot indices rather than deep history. * **Time alignment:** All tiers keep clocks in sync so event timestamps match dashboard refresh cycles and operators see consistent timelines. |

# 

# 

| **Requirement** | The system should undergo daily health checks with results logged for administrators. |
| --- | --- |
| **Implementation in the architecture** | * **End-to-end synthetic:** Generate a known test event at the Wazuh edge, let it traverse processing and storage, confirm it appears in Kibana, and ensure it triggers the Alerting & Notification fan-out. This validates the full pipeline, not just individual parts. * **Recorded evidence:** Store the outcome of the daily check in a small operational dataset and surface a simple “health” dashboard showing pass/fail status and a trend over the last 7–30 days. * **Action on failure:** If the synthetic fails more than once in a row, notify administrators and create or update an operations ticket with details and suggested remediation, such as cycling a tier or reviewing recent configuration changes. |

* **Implementation of security requirements in system architecture**

| **Security Risk** | **Broken Access Control**  Users might see or change things they shouldn’t if access rules are weak or misapplied. That can expose sensitive login data or let non-admins alter rules that shape detections. |
| --- | --- |
| **Implementation in the architecture** | * Make **Kibana** the only user-facing entry point; keep **Elasticsearch**, **Logstash**, and **Wazuh Manager** reachable only on the private network so all access is funneled through one gate. * Require sign-in on **Kibana** and assign roles so analysts get read-only dashboards while only security engineers can modify rules or saved objects. * Enforce index-level permissions aligned to the dashboards users need, and keep system/audit indices restricted to administrators. * Record access attempts and changes in audit logs and review them regularly to catch repeated denials or privilege changes. |

| **Security Risk** | **Input Manipulation Attack**  Attackers may flood the system with crafted or noisy events so genuine threats are buried. If parsing and validation are loose, the pipeline can ignore real signals or become unstable. |
| --- | --- |
| **Implementation in the Architecture** | * Filter and normalize at the **Wazuh Agent** and **Wazuh Manager** so only well-formed authentication and security events enter the pipeline; reject unexpected shapes early. * Keep **Logstash** enrichment focused (for example, IP or user context) to avoid heavy per-event work that attackers could exploit; drop malformed fields instead of propagating them. * Use **Alerting & Notification** to surface abnormal volumes or patterns (such as sudden spikes in failed logins) so responders act on manipulation attempts promptly. |

| **Security Risk** | **Data Poisoning Attack**  Adversaries try to taint data over time so detections learn the wrong “normal,” reducing quality. If unvetted events persist, future analysis and rule-tuning can drift in the wrong direction. |
| --- | --- |
| **Implementation in the architecture** | * Apply basic hygiene at **Wazuh Manager** (normalize, categorize, and apply known detection logic first) so clearly bogus records don’t reach long-term storage. * Keep **Logstash** enrichments controlled and vetted to avoid introducing untrusted external signals. * Store only fields needed for investigations in **Elasticsearch** and separate hot, recent indices from longer-term archives to minimize the blast radius of tainted data. * Review detection performance in **Kibana** and feed improvements back into **Wazuh** rules through a controlled change process. |

| **Security Risk** | **Using Outdated Client/Server Components**  Old versions of agents, processors, or the UI may carry known vulnerabilities. Attackers look for these gaps to gain footholds with little effort. |
| --- | --- |
| **Implementation in the architecture** | * Track versions of **Wazuh Agent/Manager**, **Logstash**, **Elasticsearch**, and **Kibana** and plan regular updates using official distributions to reduce exposure to known issues. * Source deployment artifacts from verified locations and align upgrades across layers so protocols and features remain compatible end-to-end (agent → manager → pipeline → storage → UI). * After each upgrade, validate basic health and alert delivery using the same synthetic checks you run daily. |

| **Security Risk** | **Hardcoding Credentials**  Secrets embedded in code or dashboards tend to leak and are hard to rotate. Once exposed, attackers can read data, change rules, or impersonate services. |
| --- | --- |
| **Implementation in the architecture** | * Keep webhook URLs, service accounts, and API keys out of pipelines, dashboards, and images; reference them through the deployment environment so they never appear in source or UI. * Rotate credentials on a routine schedule and monitor any emergency “break-glass” access, summarizing results to administrators. * Limit who can view or edit integration settings in **Kibana** and who can change rule sets in **Wazuh** to a small, trusted group. |

<Repeat the above for each of the security requirements.>

…

# Tools and Technologies

<List down tools and technologies that you plan to use for development and deployment. Make sure that you mention name and version of each tool.>

The following tools and technologies will be used for the development, monitoring, and deployment of the Anomalous Login Detection System. Each tool is selected for reliability, compatibility, and alignment with the ELK-based architecture.

| Subsystem/Layer | Tool/Technology | Version | Purpose/ Usage |
| --- | --- | --- | --- |
| Operating Environment | Ubuntu Server LTS | 24.04.3 | Primary host OS for Wazuh Manager, Logstash, Elasticsearch and Kibana services. |
| Identity & Apps(Log Sources) | Windows Server (security logs),LInux(SSH logs) | N/A | Generate authentication and system logs forwarded to Wazuh Agent. |
| Endpoint Layer | Wazuh Agent | 4.13 | Installed on endpoints to securely collect and forward logs to the Wazuh Manager. |
| Detection & Correlation | Wazuh Manager | 4.13 | Correlates events and applies detection rules to identify suspicious login behavior. |
| SIEM Ingestion & Processing | Logstash | 9.1.5 | Parses and enriches logs (GeoIP, user context) before storing in Elasticsearch. |
| Search & Storage | Elasticsearch Cluster | 9.1.5 | Stores normalized logs and supports fast search and query capabilities. |
| Visualization & UX | Kibana | 9.1.5 | Provides dashboards and visualizations for analysts to monitor login activity and alerts. |
| Alerting & Notification | Wazuh Alerting Module, Email & Slack Integrations | 9.1.5 | Sends real-time alerts to security teams for incident response. |
| Automation & Scripting | Python | 3.14.0 | Used for log simulation, custom scripts, and integration automation. |
| Configuration & Rules | JSON | N/A | Define Wazuh rules, Logstash pipelines, and Elasticsearch mappings. |
| Version Control & Collaboration | GIthub | Version | Maintains source code, pipelines, and documentation with team collaboration. |
| Containerization | Docker Engine | 28.5.0 | Enables containerized deployment for each ELK + Wazuh component to ensure portability. |
| Deployment & Hosting | Azure Cloud Infrastructure | N/A | Hosts Docker containers (Wazuh, Elasticsearch, Logstash, Kibana) on an Ubuntu VM with persistent storage and public network access. |
| Testing & Simulation | Ebryx Log Data, Wazuh Logtest | N/A | Real authentication logs from partner organization Ebryx will be used for system evaluation. |

# Hardware Requirements

<List down the hardware requirements. This should include requirements for both development machines and deployment servers>

The following hardware resources are required for both development and deployment of the *Anomalous Login Detection System* . All specifications are selected to meet the system’s non-functional goals of high throughput, real-time alerting, and concurrent dashboard access.

| Environment | Component | Minimum Specification | Recommended Specification | Purpose |
| --- | --- | --- | --- | --- |
| Developer Machine | Processor | Quad-Core CPU  (Intel i5) | Hexa-Core CPU  (Intel i7) | Required for local development, Docker image builds, and light testing. |
|  | Memory(RAM) | 8 GB DDR4 | 16 GB DDR4 | Ensures smooth execution of multiple Docker containers and IDE tools (VS Code, Git, Python). |
|  | Storage | 256 GB SSD | 512 GB SSD | For local logs, Docker images, and temporary ELK indices. |
|  | Network | Broadband connection (5Mbps +) | High-speed internet (20Mbps +) | Required for pulling Docker images and pushing code to GitHub. |
| Deployment Server (Cloud VM) | Hosting Platform | Azure Free/ VM | Azure Free VM | Hosts all Docker containers (Wazuh Manager, Logstash, Elasticsearch, Kibana). |
|  | Operating System | Ubuntu Server | Ubuntu Server | Stable Linux base for ELK and Wazuh stack. |
|  | Processor | 2 vCPUs | 4 vCPUs + (ARM or x86) | Handles ingestion of 10 000 + log entries per second without bottlenecks |
|  | Memory (RAM) | 8 GB | 16-24 GB | Supports Elasticsearch indexing and Logstash pipeline operations |
|  | Storage | 100 GB SSD | 200 GB SSD | Stores persistent indices and Wazuh archives. |
|  | Network | 10 Mbps (minimum) | 100 Mbps (preferred) | Required for real-time log streaming and alert delivery. |

**Development Environment:**

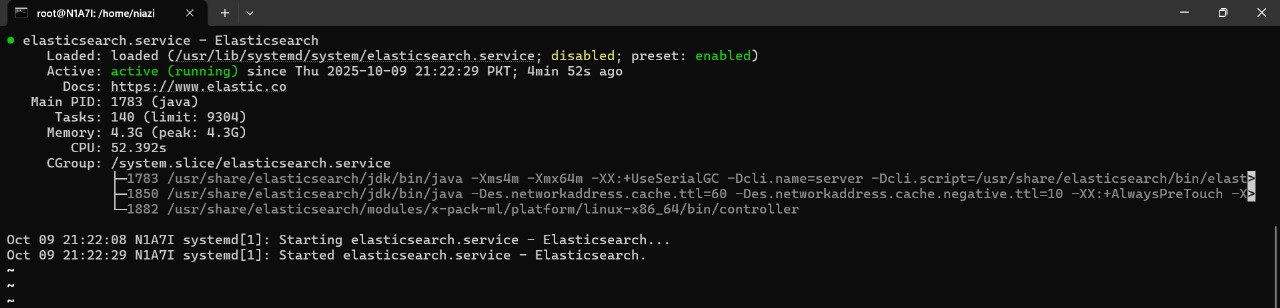
Each team member’s local machine will run Docker containers and Python scripts for configuration and testing. The recommended 16 GB RAM and SSD storage ensure smooth parallel execution of Logstash, Elasticsearch, and Kibana.

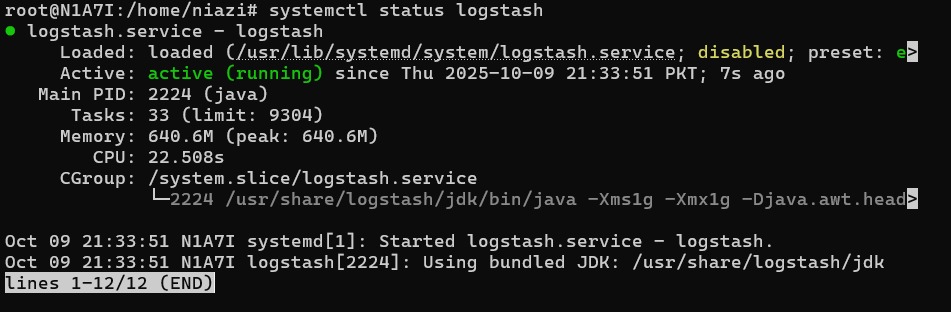
**Deployment Environment:**

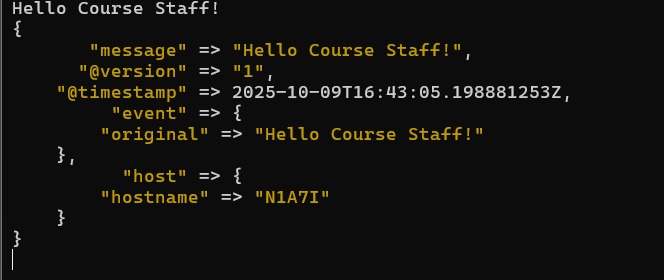
The production-like setup will run on an Azure Cloud VM hosting all ELK + Wazuh containers. SSD storage and sufficient RAM support high log throughput and rapid query response. The system can scale horizontally by adding additional Elasticsearch nodes or Logstash instances if required.

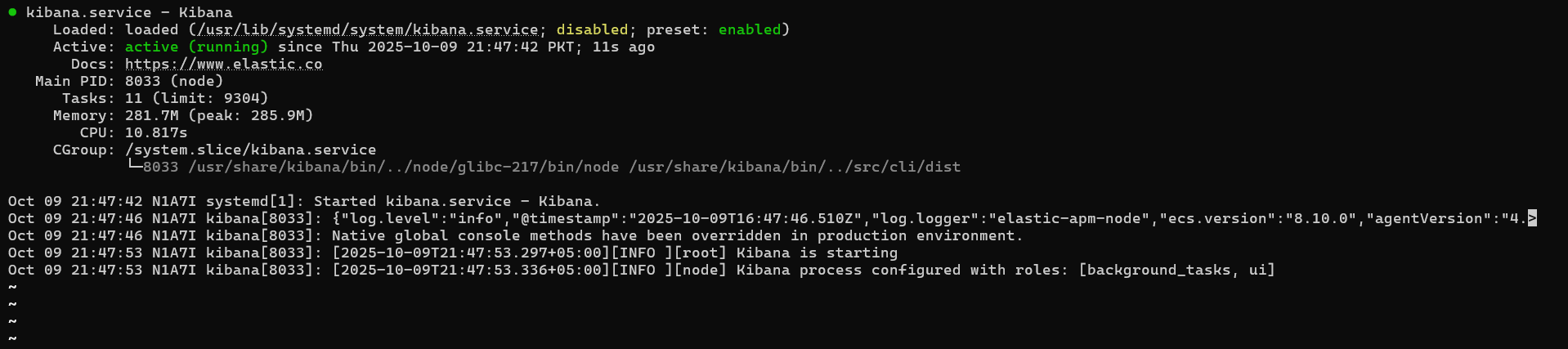
# Development Environment Preparation

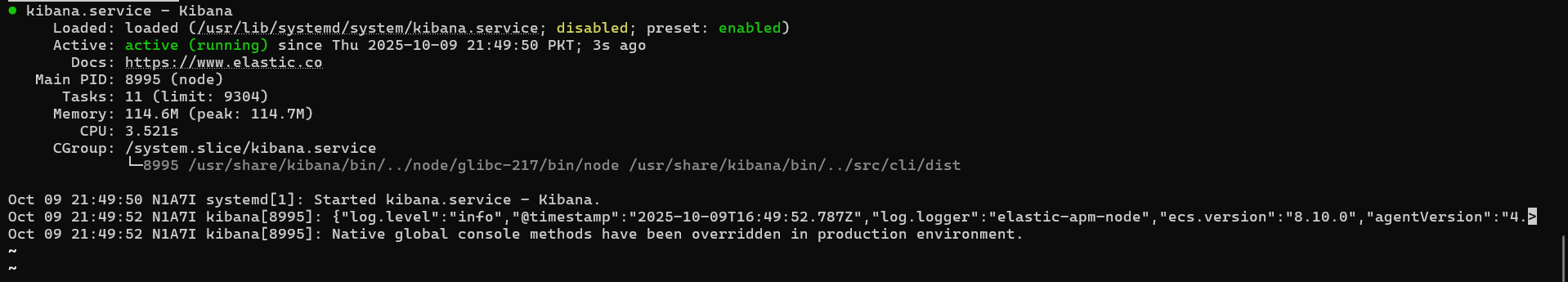
<(1) Setup the development environment on your machines and mention here that you have actually setup the environment. Include three snapshots of the tool(s) that you are going to use for development. These snapshots must be taken from the tool(s) while they are running on your development machines.>





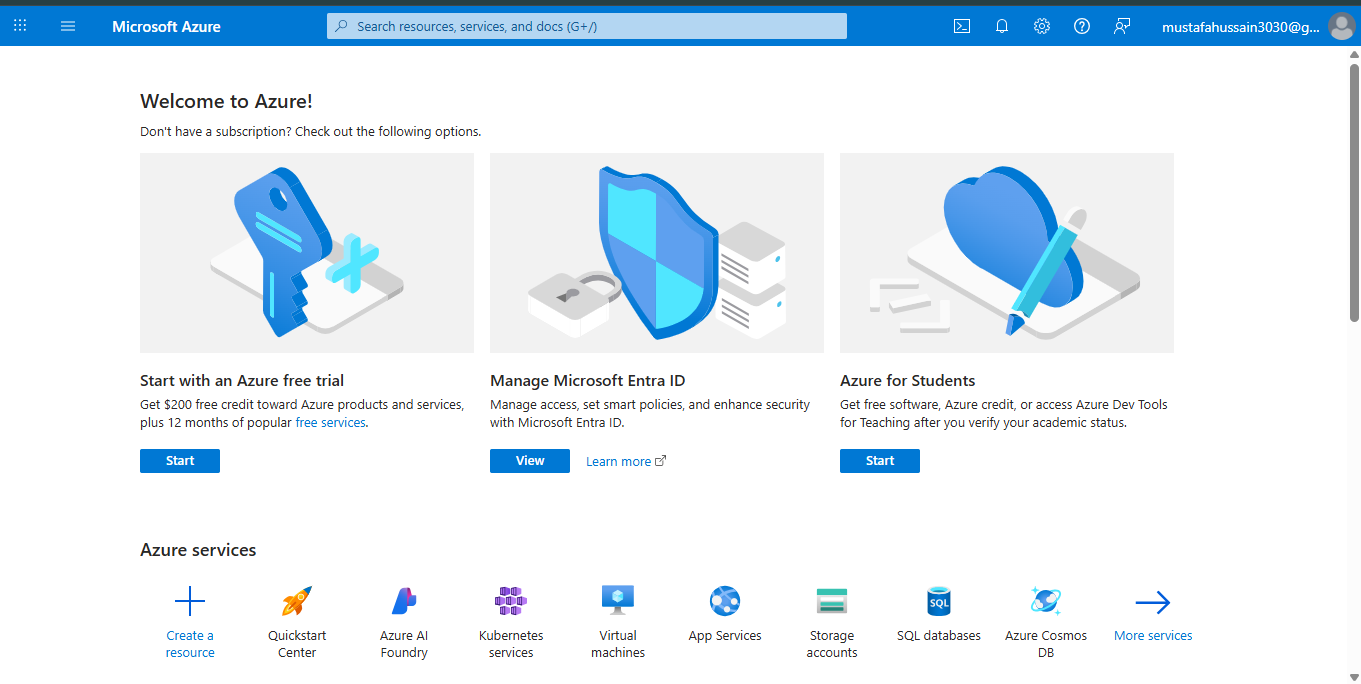






# Deployment Platform

<Find a free hosting service where you can deploy your system for anyone to access online. This service will be used for deployment of prototype, sprints and final system.>



The *Anomalous Login Detection System*  will be deployed on **Microsoft Azure Cloud Platform**, which provides a reliable and scalable environment for hosting the system’s containerized services.  
 The deployment uses Azure Virtual Machines (VMs) running Ubuntu Server, configured with Docker Engine and Docker Compose to manage all components of the ELK + Wazuh stack.

Each major subsystem Wazuh Manager, Logstash, Elasticsearch, and Kibana, runs as an independent Docker container. This containerized setup simplifies configuration, ensures environment consistency, and allows quick updates during prototype iterations and sprint deployments.

The Azure environment provides:

* **Persistent virtual machines** for continuous uptime,
* **Public IP access** for remote log ingestion and dashboard viewing,
* **Integrated monitoring and scaling tools** for performance optimization,
* **Secure network management** to restrict unauthorized access to system dashboards.

The system dashboard is hosted through the Azure portal, allowing authorized team members and evaluators to access the Kibana interface online for demonstration and testing purposes.  
 Azure’s built-in resource monitoring tools will also be used to track CPU, memory, and network utilization during load testing and anomaly detection scenarios.

For future scalability, the same Docker-based configuration can easily be extended to Azure Kubernetes Service (AKS), enabling automatic scaling and high availability for production-grade deployments.

# Use of Generative AI

<Mention here how generative AI was used in preparation of this document.>

# No generative AI has been used in this document. (change if required)Who Did What?

| **Name of the Team Member** | **Tasks done** |
| --- | --- |
| Muhammad Aaffan Khan Niazi | System Architecture |
| Mustafa Hussain | Tools and Technologies, Hardware Requirements, Deployment |
| Shehroz Faryad | NFR in System Architecture, Development Environment Preparation, Development Platform, Security Requirements |
|  |  |

# Review checklist

Before submission of this deliverable, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

| **Section** **Title** | **Reviewer Name(s)** |
| --- | --- |
| 1,2,3,4,5,6 | Muhammad Aaffan Khan Niazi |
| 7,8,9 | Mustafa Hussain |
| 9,10 | Mohammad Mustafa |
| 5,6,8,9 | Shehroz Faryad |