**PROJECT REPORT OF CO-OP PROJECT at INDUSTRY (MODULE-II)(AIP252)**

**ON**

Tech Modernisation for Internal Application

**Submitted in partial fulfillment of the requirements for the award of degree of**

**BACHELOR OF ENGINEERING**

**in**

**COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE)**

**Submitted by: Supervised By:**

**Ranveer Chaudhary Dr. Niharika Thakur**

**2110993886 Assistant Professor**

**Chitkara University**

Logo

Description automatically generated

**CHITKARA UNIVERSITY INSTITUTE OF ENGINEERING & TECHNOLOGY**

**CHITKARA UNIVERSITY, RAJPURA**

**MAY 2025**

**CONTENTS**

| S.No. | Title | Page No. |
| --- | --- | --- |
|  | Declaration | i |
|  | Acknowledgement | ii |
|  | List of Figures | iii |
|  | List of Tables | iv |
|  | Abstract | v |
| 1 | Introduction to the Problem Statement | 1 |
| 2 | Research Methodology |  |
| 3 | System Flow Chart |  |
| 4 | Tools and Technologies Used |  |
| 5 | Solution Implementation |  |
| 6 | Major Findings/Outcomes/Output/Results |  |
| 7 | Social Impact of the Solution |  |
|  | Conclusion and Future Scope |  |
|  | References |  |
|  | Appendices |  |

**DECLARATION**

We hereby declare that the project work titled, “Tech Modernisation for Internal Application” submitted as part of Bachelor’s degree in CSE (ARTIFICIAL INTELLIGENCE), at Chitkara University, Punjab, is an authentic record of our own work carried out under the supervision of <Faculty Mentor>.

**Signature(s):**

**ACKNOWLEDGEMENT**

Signature…………….…..

Name: Ranveer Chaudhary

Roll No: 2110993886

**Important Information:**

* The dimension of the project report should be in A4 size.
* **List of Tables –** The list should use exactly the same captions as they appear above the tables in the text. One line spacing should be adopted for typing the matter under this head.
* **List of Figures –** The list should use exactly the same captions as they appear below the figures in the text. One line spacing should be adopted for typing the matter under this head.

| S.No. | Title | Minimum number of Pages |
| --- | --- | --- |
|  | Declaration | 1 |
|  | Acknowledgement | 1 |
|  | List of Figures | 1 |
|  | List of Tables | 1 |
|  | Abstract | 1 |
| 1 | Introduction to the Problem Statement | 3 |
| 2 | Research Methodology | 2 |
| 3 | System Flow Chart | 2 |
| 4 | Tools and Technologies Used | 3 |
| 5 | Solution Implementation | 10 |
| 6 | Major Findings/Outcomes/Output/Results | 10 |
| 7 | Social Impact of the Solution | 2 |
| 8 | Conclusion and Future Scope | 3 |
|  | References | 2 |
|  | Appendices |  |

**Abstract**

The project Tech Modernisation for Internal Application focuses on transforming a legacy internal application at **Morgan Stanley** into a modern, scalable, and secure system aligned with contemporary cloud-native practices. The initiative addresses critical limitations in the existing infrastructure, including outdated authentication mechanisms, inefficient resource utilization, and rigid deployment processes, which hindered scalability and integration with modern platforms. By adopting containerization, modern authentication protocols, and advanced security frameworks, the project aims to enhance operational efficiency, security posture, and long-term maintainability.

The existing system relies on traditional server infrastructure and Kerberos-based authentication, which restricts scalability and complicates integration with cloud environments. Additionally, the absence of containerization leads to inconsistent deployments, resource inefficiencies, and prolonged development cycles. The security model lacks granular access controls, exposing the system to potential vulnerabilities. These shortcomings necessitated a comprehensive overhaul to align the application with DevOps practices, cloud readiness, and modern security standards.

The modernization strategy centers on three pillars: containerization, authentication overhaul, and security enhancement. First, Docker containers are employed to encapsulate the application components, ensuring consistent deployments, improved portability, and optimized resource usage. The **Python-based core application** and **Java Spring Boot services are containerized**, with **Gradle** serving as the build tool for Java components. This shift reduces environment drift and accelerates deployment cycles.

Second, the legacy Kerberos authentication is replaced with **OpenID Connect (OIDC) and OAuth 2.0**, enabling secure, token-based authentication. This change simplifies integration with modern identity providers and third-party services while enhancing user experience. Complementing this, a **Zero Trust architecture** is implemented to enforce strict identity verification across the system, ensuring no entity is trusted by default, even within the network perimeter.

Third, **Role-Based Access Control (RBAC)** is introduced to manage permissions granularly. By assigning roles to users and enforcing least-privilege access, the system minimizes unauthorized access risks and improves auditability. These security upgrades collectively address compliance requirements and align with industry best practices.

The outcomes of the modernization effort are substantial. Deployment speed improves significantly due to containerized workflows, reducing manual configuration efforts. Resource efficiency gains are achieved through optimized container orchestration. Security enhancements, including token-based authentication and Zero Trust principles, fortify the system against evolving threats. Furthermore, RBAC implementation streamlines access management, enabling precise control over user permissions and simplifying compliance reporting.

Looking ahead, the project lays a foundation for future advancements. The containerized architecture facilitates seamless migration to cloud platforms, promising greater scalability and cost-efficiency. Proposed enhancements include integrating an API gateway for centralized traffic management, implementing auto-scaling to handle variable workloads, and adopting policy-as-code frameworks to dynamically enforce security rules. Additionally, plans for advanced monitoring tools and secrets management systems aim to bolster operational visibility and data protection.

In conclusion, this modernization initiative represents a pivotal step toward building a resilient, agile, and future-ready internal application. By addressing technical debt and adopting cutting-edge technologies, the project not only resolves existing limitations but also positions the system to adapt to emerging technological and business demands. The improvements in security, scalability, and operational efficiency underscore the project’s success in aligning Morgan Stanley’s internal tools with modern enterprise standards.

**1. Introduction To The Problem Statement**

**1.1 Background**

In today’s rapidly evolving technological landscape, modernising enterprise applications has become a crucial priority for organisations aiming to enhance operational efficiency, security, and scalability. Legacy systems, although reliable in their time, often struggle to keep up with modern business demands and integration requirements. At Morgan Stanley, one of the world’s leading global financial services firms, internal applications play a vital role in supporting various business operations and workflows. However, many of these applications were originally designed on traditional infrastructure, relying on outdated technologies that limit their ability to scale, adapt, and integrate with contemporary systems. The internal application selected for this modernisation project was originally built using a conventional server-based architecture. While it fulfilled its functional requirements for several years, the infrastructure presented significant limitations in terms of scalability, resource efficiency, and security. The authentication mechanism employed by the system was based on Kerberos, an older, ticket-based protocol primarily suited for closed, static network environments. While Kerberos was widely adopted for internal enterprise networks in the past, it lacks the flexibility and compatibility needed for modern, distributed systems, particularly those integrated with cloud services and third-party identity providers. In addition to authentication constraints, the system’s absence of containerisation posed operational challenges. Traditional server deployments often result in inconsistent environments across development, testing, and production, leading to environment drift, deployment errors, and prolonged release cycles. Without containerisation, the application’s resource utilisation remained inefficient, making scaling difficult and maintenance cumbersome. Modern software development and operations practices, such as DevOps and continuous integration/continuous deployment (CI/CD), rely on container technologies like Docker to streamline workflows, ensure deployment consistency, and improve portability. Security was another critical concern in the legacy system. The application’s existing security model lacked fine-grained access control, making it difficult to enforce the principle of least privilege. This increases the risk of unauthorized access and complicates compliance with modern security and regulatory standards. As enterprise security paradigms have shifted towards Zero Trust models, where no user or device is trusted by default — even within the network perimeter — the existing system’s perimeter-based security model proved insufficient. Recognising these limitations, Morgan Stanley initiated a comprehensive modernisation effort aimed at addressing these challenges and aligning the application with modern technology standards. The modernisation strategy was structured around three core pillars: adopting containerisation for deployment and infrastructure management, replacing Kerberos with modern authentication protocols such as OpenID Connect (OIDC) and OAuth 2.0, and enhancing security by implementing a Zero Trust architecture alongside Role-Based Access Control (RBAC). The core application itself was primarily built using Python, with some supporting services developed in Java using the Spring Boot framework. Modernising these components involved containerising all services with Docker to enable consistent, portable, and resource-efficient deployments. The Java services adopted Gradle as a build tool to support modern build and dependency management practices. Together, these upgrades laid the groundwork for future cloud-native readiness, improved operational efficiency, and enhanced system security. By addressing these legacy limitations, the project ensures that Morgan Stanley’s internal application can not only meet current operational demands but also scale, integrate, and evolve with future technological advancements.

**1.2 Objective**

The primary objective of this project is to modernise an internal application at Morgan Stanley by transforming its legacy architecture into a scalable, secure, and cloud-ready system. This modernisation initiative addresses the technical limitations and operational inefficiencies associated with the existing infrastructure, while aligning the application with contemporary enterprise technology standards. The project aims to:

* **Transition from Legacy to Containerised Architecture:** Replace traditional server-based deployments with containerised environments using Docker to ensure consistent, portable, and efficient application deployment across various platforms.
* **Upgrade Authentication Mechanisms:** Replace the outdated Kerberos-based authentication with modern, token-based authentication protocols such as OpenID Connect (OIDC) and OAuth 2.0 to simplify integration with modern identity providers and enhance security.
* **Implement Zero Trust Security Model:** Establish a Zero Trust framework that enforces identity-first security principles, ensuring no user or system is implicitly trusted, even within the internal network perimeter.
* **Introduce Role-Based Access Control (RBAC):** Integrate RBAC to manage user permissions through predefined roles, enforcing least-privilege access, improving auditability, and strengthening system security.
* **Enhance Operational Efficiency:** Improve deployment speed, resource utilisation, and environment consistency by adopting containerisation and modern DevOps practices.
* **Lay Foundation for Cloud-Native Readiness:** Enable future cloud migrations and integrations by restructuring the application into a containerised, scalable, and secure architecture.

By addressing these objectives, the project strengthens the application’s resilience, security posture, and long-term adaptability, ensuring it remains aligned with evolving business and technological demands.

**1.3 Scope**

The scope of this project extends beyond merely updating the infrastructure of an internal application. It encompasses a comprehensive transformation aimed at enhancing the system’s scalability, security, maintainability, and integration capabilities in line with modern enterprise practices. The modernisation effort focuses on several key areas:

* **Infrastructure Modernisation:** Transition the application from a legacy server-based deployment to a fully containerised environment using Docker, enabling consistent deployments, improved portability, and optimised resource usage.
* **Authentication Overhaul:** Replace the existing Kerberos-based authentication mechanism with OpenID Connect (OIDC) and OAuth 2.0, introducing secure, token-based authentication and simplifying integration with modern identity providers and third-party services.
* **Security Enhancement:** Implement a Zero Trust security model to enforce strict identity verification and remove implicit trust within the network. Introduce Role-Based Access Control (RBAC) to manage permissions at a granular level, ensuring least-privilege access and improved auditability.
* **Operational Improvement:** Optimise deployment cycles and resource efficiency through containerisation, reducing manual configuration efforts, environment inconsistencies, and operational overhead.
* **Future-Ready Architecture:** Prepare the system for seamless migration to cloud platforms, enabling enhanced scalability, cost-efficiency, and integration with cloud-native services in future developments.

By addressing these areas, the project not only resolves existing limitations but also positions the application for future growth. It ensures that Morgan Stanley’s internal tools remain adaptable to evolving business, security, and technology requirements, supporting long-term operational and strategic objectives.

**1.4 Application**

The modernised internal application at Morgan Stanley is designed to support a range of operational and security-focused functions within the organisation. With the new architecture and enhanced features, its applications extend into several critical areas:

* **Enterprise Workflow Management:** Facilitate internal business processes by providing a secure, scalable, and efficient platform for managing enterprise operations.
* **Secure Authentication Services:** Deliver modern, token-based authentication using OpenID Connect (OIDC) and OAuth 2.0, improving security and user access management across integrated enterprise systems.
* **User Access Control and Compliance:** Implement Role-Based Access Control (RBAC) to manage user permissions effectively, ensuring least-privilege access and simplifying compliance reporting for audit and security teams.
* **Zero Trust Security Enforcement:** Enforce strict identity verification and continuous trust assessment within the internal network, strengthening protection against internal and external threats.
* **Cloud-Native Readiness:** Enable future integration with cloud services and platforms through a containerised architecture, supporting scalable, cost-effective, and resilient operations.

By modernising the internal application, Morgan Stanley not only addresses current infrastructure limitations but also establishes a reliable, secure, and adaptable system that can evolve alongside emerging business needs and technological advancements.

**2. Research Methodology**

The methodology followed in this project involves a structured, step-by-step approach designed to modernise an internal application at Morgan Stanley by addressing its architectural, operational, and security limitations. The process integrates several key components, including system analysis, infrastructure transition, security model enhancement, and authentication mechanism upgrades. Each of these stages plays a critical role in ensuring that the modernised application is scalable, secure, efficient, and aligned with contemporary enterprise standards.

The core methodology steps implemented in this project are as follows:

### 2.1 System Analysis and Problem Identification

The initial phase of the methodology involved a comprehensive analysis of the existing internal application. This step was essential to identify the limitations and shortcomings within the current infrastructure and security model. The application was originally built on a legacy server infrastructure, operating with traditional deployments that lacked flexibility and scalability. The authentication mechanism relied on Kerberos, which, while effective in the past, has become outdated in modern, distributed enterprise environments.

Additionally, the absence of containerisation led to operational inefficiencies, such as inconsistent deployments across different environments and inefficient resource utilisation. Security gaps were also evident, particularly in the system’s inability to enforce fine-grained access controls. The security model lacked features that are now considered standard in modern applications, such as Zero Trust enforcement and dynamic access control policies.

The objective of this phase was to document the existing system architecture, deployment process, authentication workflow, and security controls. This documentation provided a clear foundation for defining the areas that required improvement and modernisation.

### 2.2 Defining Modernisation Objectives and Scope

Following the system analysis, clear objectives and a defined scope for the modernisation project were established. The objectives included transitioning the application to a containerised architecture, replacing the outdated Kerberos authentication with OpenID Connect (OIDC) and OAuth 2.0, and implementing a Zero Trust security model with Role-Based Access Control (RBAC).

This phase involved outlining the expected outcomes of the project, such as improving deployment consistency, reducing operational overhead, strengthening security, and enabling future cloud-native readiness. Defining a clear scope ensured that the modernisation effort remained focused and addressed the most critical infrastructure, security, and operational challenges.

### 2.3 Containerisation of Application Components

One of the central components of the modernisation methodology was the adoption of containerisation using Docker. The existing internal application consisted of a core application built using Python, along with supporting services developed in Java (Spring Boot framework). Traditionally, these services were deployed directly onto server environments, resulting in environment inconsistencies and slow deployment cycles.

Containerising these components provided several advantages:

* Ensured consistent deployment environments across development, testing, and production.
* Improved resource utilisation by isolating services within their own containers.
* Simplified deployment workflows by enabling developers and operations teams to manage services through Docker images and containers.

The containerisation process began with the creation of Dockerfiles for each service, specifying the base image, dependencies, and application entry points. The Java services utilised Gradle as a build tool to handle build automation, dependency management, and Docker image creation. The Python application followed a similar workflow, with Docker images built using official Python base images and custom configurations.

By containerising the entire application, deployment speed improved significantly, environment drift was eliminated, and the system became scalable and easier to maintain.

### 2.4 Replacing Kerberos with OIDC and OAuth 2.0

The next major phase involved replacing the Kerberos-based authentication mechanism with a modern, token-based authentication system using OpenID Connect (OIDC) and OAuth 2.0. Kerberos, being a ticket-based protocol designed for traditional enterprise networks, presented limitations when integrated with modern, distributed, or cloud-based systems.

The adoption of OIDC and OAuth 2.0 offered several advantages:

* Provided secure, token-based authentication suitable for distributed applications and cloud-native environments.
* Simplified integration with external identity providers and third-party services.
* Enhanced user experience through single sign-on (SSO) capabilities and improved security protocols.

The implementation process involved integrating the application’s authentication layer with an OIDC-compliant identity provider. This allowed users to authenticate through secure tokens, which could be validated by the application without requiring direct credential exchanges. OAuth 2.0 was used to manage authorization workflows, granting or restricting access to resources based on predefined scopes and permissions.

This overhaul resulted in a more secure, scalable, and interoperable authentication system, aligning the internal application with modern enterprise security standards.

### 2.5 Implementation of Zero Trust Security Model

The legacy internal application relied on perimeter-based security, where entities within the network perimeter were implicitly trusted. This model has become increasingly inadequate in modern enterprise environments, where threats can originate both internally and externally.

To address this, the modernisation methodology included the implementation of a Zero Trust security model. The Zero Trust approach operates on the principle that no user, device, or application should be trusted by default — even within the internal network. Every access request must be authenticated, authorised, and continuously validated.

The Zero Trust implementation involved:

* Enforcing strict identity verification for all users and services.
* Applying least-privilege access policies through Role-Based Access Control (RBAC).
* Segmenting the application and infrastructure to limit lateral movement in the event of a security breach.
* Continuously monitoring and logging access attempts and system activity.

By implementing Zero Trust, the security posture of the internal application was significantly enhanced, reducing the risk of unauthorised access and enabling more resilient defences against modern cyber threats.

### 2.6 Role-Based Access Control (RBAC) Integration

As part of strengthening access management within the modernised application, Role-Based Access Control (RBAC) was integrated into the system. The legacy system lacked fine-grained access controls, which posed security risks and made auditing user activity difficult.

RBAC allows administrators to assign specific roles to users, with each role containing predefined permissions for accessing different parts of the application. This ensured that users could only access resources necessary for their roles, enforcing the principle of least privilege.

The RBAC implementation involved:

* Defining roles based on job functions and operational requirements.
* Assigning permissions to roles, specifying which application resources each role could access.
* Mapping users to roles through the integrated OIDC identity management system.

This approach improved operational security, streamlined user management, and simplified compliance reporting by providing a clear audit trail of user permissions and activity.

### 2.7 Deployment Pipeline Optimisation

In addition to containerising application components, the project methodology included optimising the deployment pipeline to improve operational efficiency. Traditional deployment processes involved manual configuration and environment-specific adjustments, leading to delays and inconsistencies.

With containerisation in place, the deployment workflow was restructured to leverage automated build and deployment pipelines. The process included:

* Automated builds of Docker images for each application component.
* Container orchestration using tools such as Docker Compose or container orchestration platforms.
* Continuous integration and continuous deployment (CI/CD) practices to automate code integration, testing, and deployment.

This restructuring reduced deployment times, eliminated environment drift, and improved developer efficiency by providing consistent, replicable deployment environments.

### 

### 2.8 Preparing for Future Cloud-Native Readiness

An important part of the methodology was to lay a foundation for future cloud migrations. While the current project focused on modernising on-premises infrastructure, the new containerised architecture was designed with cloud readiness in mind.

The containerised setup ensures that application components can be easily migrated to cloud platforms without significant refactoring. Additionally, the adoption of token-based authentication and Zero Trust principles ensures compatibility with cloud-native security models and identity management systems.

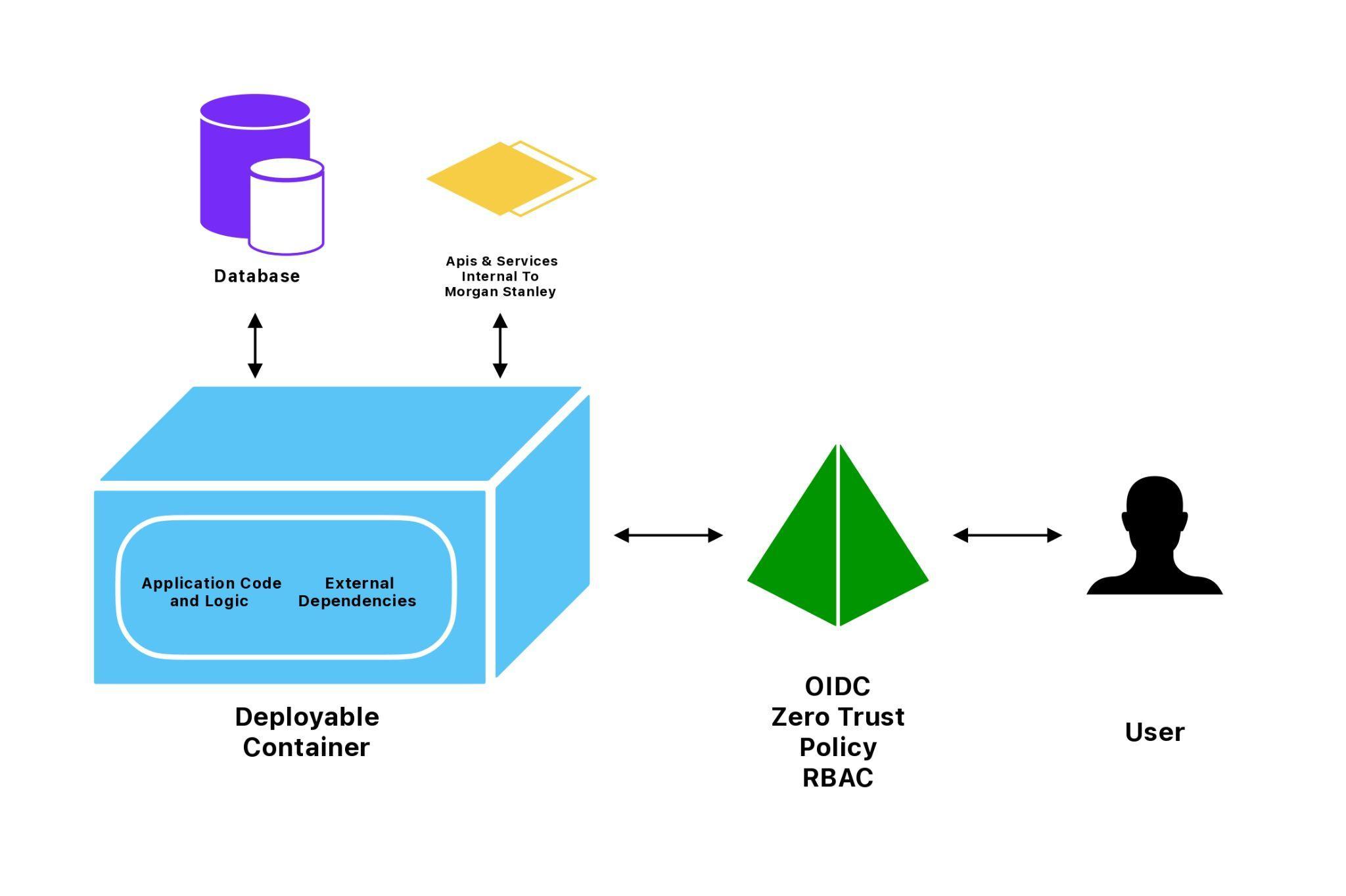
This strategic preparation positions the application for seamless cloud integration in future phases, supporting enterprise scalability, cost optimisation, and advanced orchestration capabilities.

### 2.9 Infrastructure Configuration and Container Orchestration

The final phase of the methodology focused on configuring the underlying infrastructure to support the newly containerised application environment and establishing reliable container orchestration processes. After individual application components were containerised using Docker, it became essential to manage these containers efficiently, ensuring seamless communication, scalability, and fault tolerance.

To achieve this, a structured container orchestration setup was introduced using Docker Compose for managing multi-container applications in local and on-premises environments. The orchestration configuration defined service dependencies, network configurations, environment variables, and volume mappings, enabling consistent multi-service deployments with minimal manual intervention. This approach streamlined the process of bringing up, scaling, and maintaining application services.

# 3. System Flow Chart

****

**OIDC User Authentication (Identity Verification)**The legacy Kerberos protocol was fully replaced with OpenID Connect (OIDC), a modern identity layer built on OAuth 2.0. It enables secure, token-based authentication using JWTs and facilitates seamless integration with distributed containerised services and cloud platforms. This shift modernised identity verification while supporting Single Sign-On (SSO) capabilities and scalable authentication flows.

**Zero Trust Policy & Role-Based Access Control (RBAC)**A Zero Trust security framework was implemented to enforce strict, continuous authentication and authorisation for every access request, irrespective of network origin. Coupled with Role-Based Access Control (RBAC), permissions are explicitly mapped to predefined roles such as Admin, Developer, or Analyst. This ensures resource access is granted solely based on operational necessity, significantly reducing lateral movement risks.

**Internal Applications & Microservices**The internal application ecosystem comprises essential business-critical services and supporting microservices, containerised for improved scalability and resilience. Services communicate securely through API gateways and internal service discovery mechanisms, with each microservice independently deployable and adhering to standardised organisational security and performance policies.

**Application Code and Logic**The core business logic layer was retained in Python, handling transaction workflows, data transformation, and backend integration services. Performance-sensitive workloads and enterprise-grade features, such as multithreading and advanced dependency injection, were delegated to Java Spring Boot microservices. Middleware integration layers ensured reliable, secure interaction between these heterogeneous services.

**Centralised Database Services**A relational SQL database was maintained as the system’s primary data store, offering ACID compliance for transaction integrity and operational consistency. All data at rest is encrypted, while communication between services and the database is protected via TLS encryption. RBAC-driven access controls further ensure that only authorised users can access or modify sensitive data based on role-specific privileges.

**Containerised Deployable Environment**All application components, runtime dependencies, and configuration files are encapsulated within Docker containers, ensuring consistency across development, staging, and production environments. Custom Docker images were created for both Python and Java services, with Docker Compose orchestrating multi-container services during development and on-premises deployments. This approach significantly streamlined the deployment pipeline and reduced environment-specific issues.

# 

# 

# 4. Tools and Technologies Used

This project utilised a comprehensive set of modern tools, frameworks, and technologies to successfully modernise the internal application at Morgan Stanley. The selection of each tool was based on its ability to address specific limitations in the legacy system while supporting scalability, security, maintainability, and cloud-native readiness. The following subsections provide a detailed overview of the core tools and technologies incorporated during the modernisation process.

1. **Python**

The core application components of the legacy system were built using Python, a high-level, interpreted programming language widely recognised for its simplicity, readability, and versatility. Python was retained as the primary language for application logic due to its extensive library support, ease of integration, and strong developer community.

In the modernised system, Python continues to handle business logic, data processing, and integration tasks. Its compatibility with modern containerisation platforms like Docker made it an ideal candidate for containerising application components without significant modifications. The language's flexibility also facilitated the integration of modern authentication mechanisms, secure communication protocols, and improved logging functionalities within the updated system.

1. **Java and Spring Boot**

Certain supporting services within the internal application were developed using Java, specifically leveraging the Spring Boot framework. Spring Boot is a popular Java-based framework known for simplifying application development by reducing boilerplate configuration and enabling rapid deployment.

During the modernisation, existing Java services were containerised and restructured for cloud-native compatibility. Spring Boot’s embedded server model and seamless integration with OAuth 2.0 made it suitable for adopting token-based authentication workflows. Additionally, its robust dependency injection and modular configuration capabilities supported the application’s transition to a microservices-aligned deployment strategy, laying a solid foundation for scalable, maintainable service management.

1. **Gradle**

The build process for Java-based services was managed using Gradle, a modern and flexible build automation tool. Gradle offers efficient dependency management, incremental builds, and customisable build scripts, making it a suitable choice for enterprise-grade applications.

In the modernised architecture, Gradle was employed to automate the build, testing, and packaging processes for the Java services. Its seamless integration with Docker allowed for the creation of container images as part of the build pipeline. This integration streamlined deployments, improved consistency across environments, and enhanced developer productivity by automating repetitive tasks.

1. **Docker**

A critical component of the modernisation effort was the adoption of Docker, an open-source containerisation platform. Docker enables applications and their dependencies to be packaged into lightweight, portable containers that run consistently across different computing environments.

Docker was used to containerised both Python and Java-based application components. Custom Dockerfiles were created to define the build environment, dependencies, and runtime configurations for each service. This approach eliminated environment drift, simplified deployments, and optimize resource utilisation. Docker Compose was further utilised to orchestrate multi-container applications during development and testing, ensuring seamless communication and service discovery between dependent services.

1. **OpenID Connect (OIDC)**

As part of the authentication modernisation, OpenID Connect (OIDC) was introduced to replace the outdated Kerberos authentication protocol. OIDC is an identity layer built on top of the OAuth 2.0 protocol, designed to provide secure, token-based authentication for modern applications.

OIDC allowed the application to integrate with contemporary identity providers, facilitating features like single sign-on (SSO), secure token issuance, and user identity verification. It also enabled improved interoperability with third-party services and cloud-based platforms. OIDC’s JSON Web Token (JWT) based authentication made it compatible with distributed, containerised environments, addressing the limitations posed by the legacy authentication system.

1. **OAuth 2.0**

Complementing OIDC, OAuth 2.0 was implemented as the core protocol for managing authorisation workflows within the application. OAuth 2.0 is an industry-standard protocol for authorisation, widely adopted for securing APIs and distributed applications.

OAuth 2.0 facilitated secure, scoped access to application resources by issuing time-bound access tokens based on predefined permissions and roles. This token-based system provided fine-grained control over resource access, improving security and compliance with modern enterprise security standards. OAuth 2.0 also enabled seamless integration with cloud-based identity providers and third-party services, enhancing the overall flexibility of the modernised system.

1. **Zero Trust Security Model**

The security architecture of the modernised application was built around the Zero Trust security model. Zero Trust operates on the principle of “never trust, always verify,” ensuring that no user, device, or application is implicitly trusted, even if located within the internal network perimeter.

The Zero Trust model enforced continuous identity verification, strict access controls, and segmentation of application services. It complemented the OIDC and OAuth 2.0-based authentication workflows by ensuring that each access request was authenticated and authorised before granting resource access. This model significantly enhanced the application’s resilience against internal threats, unauthorised access, and potential lateral movement within the infrastructure.

1. **Role-Based Access Control (RBAC)**

To manage user permissions and enforce least-privilege access, Role-Based Access Control (RBAC) was integrated into the modernised application. RBAC assigns permissions to users based on their roles within the organisation, ensuring that users only have access to resources necessary for their job functions.

RBAC was implemented through the identity management system integrated with OIDC. Roles were predefined based on operational requirements, and permissions were mapped to each role, providing granular access control and improving compliance.

1. **Docker Compose**

For managing multi-container applications during development and on-premises deployments, Docker Compose was utilised. Docker Compose simplifies container orchestration by allowing developers to define and manage multi-service applications through a single configuration file.

Using Docker Compose, dependent services such as the Python application, Java services, and supporting databases could be deployed, scaled, and managed consistently.

# 5. Solution Implementation

## 5.1 Assessment and Architectural Planning

The implementation of the modernisation strategy for the internal application at Morgan Stanley began with a detailed assessment of the existing system. The legacy environment consisted of a monolithic application deployed on traditional server infrastructure, using Kerberos for authentication and lacking containerisation or fine-grained access control. These architectural and security limitations were clearly identified during the preliminary analysis phase.

The first objective of the implementation was to understand the existing architecture’s constraints in depth. This included documenting how components were deployed, understanding network dependencies, evaluating the performance of existing services, and identifying operational inefficiencies. Special attention was given to how the legacy authentication system (Kerberos) was configured, as it was one of the main blockers for integration with modern platforms.

After gathering system insights, a revised architecture was designed based on containerisation principles. This new architecture was planned around Docker containers for application packaging, OpenID Connect (OIDC) and OAuth 2.0 for modern authentication, and the introduction of Zero Trust and Role-Based Access Control (RBAC) for security enhancement. The new architecture aimed to transition away from tightly coupled, server-bound deployments to a more modular, scalable, and portable container-based setup.

Alongside architectural planning, a service mapping exercise was conducted to determine how each component of the core application (Python-based services) and auxiliary services (Java/Spring Boot) would be decoupled and containerised. Each component was evaluated for its deployment readiness, resource usage, external dependencies, and compatibility with container environments.

## 5.2 Containerisation of Application Components

With the revised architecture finalised, the next stage of implementation focused on containerising the existing application components. The internal application consisted primarily of a core Python application, supported by Java-based services developed using the Spring Boot framework. Previously deployed on traditional servers, these components required significant restructuring to align with container-based infrastructure.

The containerisation process began with the creation of individual **Dockerfiles** for each service. These configuration files defined the base images, application dependencies, build instructions, environment variables, and container entry points for each component. The Python application utilised official Python base images, while the Java services adopted OpenJDK-based images tailored for Spring Boot deployments.

For the Java services, **Gradle** was integrated into the build process to automate the packaging of application binaries and container images. Gradle’s support for Docker plugins allowed the build process to create Docker images as part of its automated workflow, streamlining the transition from code to container image. Dependencies were declared within Gradle build scripts to ensure consistent environment configurations within each container.

Additionally, the project adopted **Docker Compose** to manage and orchestrate multi-container environments during development and testing. This configuration allowed developers to define service dependencies, internal networks, volume mappings, and environment-specific variables within a single YAML configuration file. Docker Compose enabled rapid provisioning of a complete, isolated environment replicating production-like conditions, ensuring consistency across different stages of deployment.

## 5.3 Authentication Modernisation with OpenID Connect and OAuth 2.0

Once the containerisation of application components was in place, the next crucial phase involved replacing the legacy Kerberos-based authentication system with a modern, token-based authentication framework. The chosen solution integrated **OpenID Connect (OIDC)** layered over **OAuth 2.0**, enabling secure, scalable, and interoperable identity management across distributed application services.

The implementation process began with integrating the application with a compliant **OpenID Connect Identity Provider (IdP)**. This provider was configured to handle user authentication, issue access and ID tokens, and manage session state securely. The core application and supporting services were modified to authenticate users by redirecting them to the IdP’s login interface, where credentials were verified.

Upon successful authentication, the IdP issued **JSON Web Tokens (JWT)**, which contained user identity claims and authorisation information. These tokens were passed back to the application and validated before granting access to protected resources. Each service within the containerised environment was equipped with JWT validation logic to independently verify token integrity, expiry, and scope, ensuring decentralised and stateless authentication management.

**OAuth 2.0** was integrated to handle authorisation workflows, managing access permissions to specific application resources based on predefined scopes and client permissions. Access tokens issued through OAuth 2.0 provided secure, time-limited authorisation, mitigating risks associated with long-lived credentials and simplifying integration with third-party services and future cloud-native platforms.

## 5.4 Zero Trust Security Model Implementation

With modern authentication mechanisms established, the project advanced to reinforcing the application’s security architecture through the implementation of a **Zero Trust security model**. Unlike traditional perimeter-based security approaches that implicitly trust internal users and systems, Zero Trust operates on the principle of “never trust, always verify.” This strategy was essential for securing the modernised, distributed, containerised environment.

The implementation began by reconfiguring service-to-service communication within the Docker container network to restrict traffic based on explicit identity verification and policy enforcement. Each containerised service was isolated within logically segmented **virtual networks**, preventing unrestricted lateral movement within the infrastructure.

In alignment with Zero Trust principles, **strict identity verification** was enforced at every access point. Every API request or internal service communication was authenticated and authorised using the **JWT tokens** issued by the OIDC Identity Provider. Services were configured to validate token claims, expiry, and authorisation scopes before processing any request.

Additionally, application-level and infrastructure-level security policies were established. These policies specified which services could interact with each other, under what conditions, and with what permissions. Traffic flow was restricted by default, and permissions were granted explicitly based on operational requirements.

This Zero Trust implementation not only secured the containerised environment but also ensured continuous identity validation, reducing risks posed by internal threats, misconfigurations, and unauthorised access attempts.

## 5.5 Role-Based Access Control (RBAC) Integration

To further strengthen access management within the modernised application, **Role-Based Access Control (RBAC)** was introduced. The legacy system lacked fine-grained access control, relying on static, broad permissions that increased security risks and made user management inefficient. Integrating RBAC allowed permissions to be managed based on assigned user roles, enforcing least-privilege access throughout the application.

The RBAC system was designed around operational job functions and business requirements. Each role was defined to include a set of permissions corresponding to the specific tasks and application resources necessary for that role. For example, administrative users were granted permissions to manage service configurations and access sensitive logs, while standard users were limited to core application functions.

Roles and permissions were managed centrally through the **OIDC Identity Provider**. When a user authenticated, their assigned roles were embedded within the issued **JWT token claims**. The application and containerised services validated these role claims during each access attempt, authorising or denying access based on the associated permissions.

This integration improved security by preventing unauthorised access to sensitive resources and simplifying compliance reporting. Audit logs captured each access attempt, including user identity, role, and action performed, supporting traceability and accountability. RBAC also reduced administrative overhead by enabling administrators to manage access at the role level rather than maintaining individual user permissions.

## 5.6 Deployment Pipeline Optimisation and Infrastructure Management

Following the containerisation of application components and implementation of security enhancements, the next stage focused on improving the application’s deployment workflows and infrastructure management. The legacy system relied on manual, server-based deployment processes, which were prone to environment inconsistencies, human error, and lengthy release cycles. The modernised architecture required an automated, reliable, and scalable deployment process aligned with DevOps practices.

To address these challenges, a **Docker-based deployment pipeline** was established. This pipeline streamlined the process of building, testing, and deploying application services within containerised environments. Each service’s **Dockerfile** defined its build instructions, dependencies, and runtime configurations, ensuring consistency across development, testing, and production environments.

During development and testing phases, **Docker Compose** was utilised to orchestrate multi-container environments. Compose configurations specified service dependencies, internal networking, persistent volumes, and environment-specific variables, enabling developers to replicate production-like environments on local systems and staging servers.

The deployment workflow was further optimised by integrating **Gradle** for the Java-based services. Gradle’s Docker plugin allowed for automated container image builds and versioning, reducing manual build effort and maintaining consistency between source code changes and container images.

Additionally, **infrastructure configurations** were standardised using container networking policies, virtual network segmentation, and security rules. Each container was assigned to a designated logical network, and firewall rules were applied to restrict unauthorised traffic between services. These configurations improved system stability, reduced operational overhead, and prepared the application for future integration with advanced container orchestration platforms.

## 5.7 Operational Monitoring and Resource Management Enhancements

With the deployment workflows optimised, the next critical step was to enhance operational monitoring, system health checks, and resource management within the containerised environment. These improvements were necessary to ensure application availability, identify performance bottlenecks, and maintain infrastructure stability as the system scaled.

As part of this enhancement, **container health checks** were configured within each service’s Dockerfile and Compose configurations. Health checks continuously monitored the operational state of containers, ensuring that only healthy services remained active within the application environment. If a container failed its health check, it was automatically restarted based on predefined restart policies, reducing manual intervention and improving system resilience.

Additionally, **resource limits and constraints** were applied at the container level. Each container was assigned specific CPU and memory allocation limits to prevent any single service from monopolising system resources. This configuration improved infrastructure efficiency by distributing resources evenly and maintaining predictable performance under varying workloads.

Although comprehensive enterprise monitoring tools such as **Prometheus** and **ELK Stack (Elasticsearch, Logstash, Kibana)** were outlined for future integration, preliminary logging and monitoring configurations were established during this phase. Application and system logs were routed to centralised directories within containerised volumes, simplifying access for debugging and performance analysis.

These operational enhancements collectively improved the stability, observability, and fault tolerance of the application, ensuring that the containerised environment could operate reliably in both on-premises and future cloud-native infrastructures.

## 5.8 Securing Container Networking and Service Communication

An essential aspect of modernising the internal application involved securing the communication pathways between containerised services. In the legacy system, services communicated over an open, flat internal network with minimal isolation or restrictions, increasing the risk of lateral movement in the event of a breach. The new containerised architecture required robust network segmentation and access control to align with Zero Trust principles.

To address this, **custom virtual networks** were created within the Docker environment to logically segment services based on their roles and interaction requirements. Application services, databases, and supporting components were assigned to isolated Docker bridge networks, ensuring that containers could only communicate with permitted peers within their designated network.

**Firewall rules and access control lists (ACLs)** were applied to restrict traffic flows between containers. Only explicitly authorised services could communicate over defined ports and protocols. This approach prevented unauthorised access and reduced the attack surface within the infrastructure.

In addition, **encrypted communication channels** were configured for sensitive data transfers between services. Internal APIs handling authentication tokens, user data, and administrative operations were secured using **HTTPS/TLS encryption**, ensuring data confidentiality and integrity within the container network.

These security measures collectively strengthened the application’s internal security posture, safeguarding service-to-service communications, reducing exposure to unauthorised traffic, and enforcing strict access control aligned with the Zero Trust security model.

# 6. Major Findings/ Outcomes/ Output/ Results

The modernization of the internal application at Morgan Stanley resulted in substantial improvements across several critical areas, transforming the infrastructure into a more scalable, secure, and efficient system. The project’s outcomes not only enhanced the operational efficiency of the application but also aligned it with modern technological standards, paving the way for future-proofing the system. Below, we will discuss the major findings, outcomes, and results across various domains, including application performance, scalability, security, compliance, and overall efficiency.

## 6.1 Enhanced Scalability and Deployment Speed

One of the primary goals of modernizing the internal application was to improve its scalability and reduce the deployment cycle. The previous monolithic, server-based application struggled with scaling to accommodate fluctuating demand, particularly during peak times, and deployment often involved manual processes that were prone to errors and delays.

**Key Findings:**

* **Containerization:** By transitioning the application to Docker containers, the application’s components were decoupled and made more modular. This shift allowed for horizontal scaling, where individual services could be scaled independently based on demand, making it easier to manage load fluctuations.
* **Microservices:** With the decomposition of the monolithic application into microservices, each component became independently deployable. This not only enhanced the ability to scale the application on-demand but also sped up the overall deployment cycle.

**Outcomes:**

* **Faster Deployment:** Docker containers and the associated Docker Compose configurations facilitated the automation of the deployment pipeline. New services could be quickly built, tested, and deployed with minimal manual intervention, reducing deployment times significantly.
* **Easier Horizontal Scaling:** The system's scalability improved as containers could be easily spun up or down depending on resource needs. This allowed for more effective management of computational resources, both during normal operations and during spikes in demand.
* **Reduced System Downtime:** The modular nature of the microservices and containerized architecture helped to reduce the impact of failures. If one component failed, it could be restarted independently, minimizing overall system downtime and ensuring higher availability.

## 6.2 Improved Security and Risk Mitigation

A significant portion of the modernization effort was dedicated to improving the security posture of the internal application. The legacy authentication system, based on Kerberos, posed several risks, including the challenge of integration with modern platforms, limited scalability, and insufficient control over user permissions. Additionally, the lack of granular access control mechanisms and the absence of robust internal security practices exposed the system to several security vulnerabilities.

**Key Findings:**

* **Zero Trust Security Model:** The adoption of the Zero Trust security model ensured that no user, device, or service could access resources without being verified first. This principle of "never trust, always verify" was applied throughout the architecture, fundamentally changing how security was enforced at every level of the system.
* **OpenID Connect (OIDC) and OAuth 2.0:** By integrating OIDC and OAuth 2.0 for authentication and authorization, the new system utilized token-based security, eliminating the reliance on outdated Kerberos authentication. OIDC provided secure, interoperable authentication, while OAuth 2.0 allowed for more granular control over who could access which resources and under what conditions.
* **Role-Based Access Control (RBAC):** The introduction of RBAC allowed for the enforcement of least-privilege access, ensuring that users only had access to the resources they required for their roles, significantly reducing the potential attack surface.

**Outcomes:**

* **Stronger Authentication and Authorization:** The new token-based authentication system, which leveraged OIDC and OAuth 2.0, enabled secure, distributed identity management. The system’s ability to handle user identity and permissions centrally through JWT tokens reduced the risk of unauthorized access.
* **Continuous Identity Verification:** Zero Trust principles ensured that every access request was authenticated and authorized in real-time, regardless of the user’s location or network. This provided an extra layer of security, reducing internal threats and preventing lateral movement in the event of a breach.
* **Granular Access Control:** The implementation of RBAC ensured that users had only the necessary permissions to perform their job functions, reducing the risk of over-privileged access and improving compliance with internal and external security standards.
* **Reduced Security Risks:** The integration of advanced security models like Zero Trust and RBAC, combined with modern authentication mechanisms, substantially reduced the system's exposure to internal and external threats.

## 

## 6.3 Scalability and Flexibility of the Modernised Architecture

One of the most important goals of the modernisation project was to transition from a monolithic architecture to a more scalable and flexible containerised environment. The legacy system, built around tightly coupled services and traditional server infrastructure, struggled to scale efficiently in response to increased demand or the evolving needs of the business. The modernisation effort aimed to leverage containerisation and cloud-native technologies to improve scalability, facilitate faster development cycles, and enable more flexible deployment options.

**Key Findings:**

* **Containerisation:** The adoption of Docker and the containerisation of both Python and Java-based services enabled the application to be broken down into independent, self-contained units. This modularisation of application components made it easier to scale individual services based on demand.
* **Microservices Architecture:** Transitioning to a microservices architecture allowed for a more agile approach to service development and deployment. Each service could be developed, tested, and deployed independently, reducing bottlenecks and improving overall development velocity.
* **Cloud-Ready Design:** The containerisation of services, along with the integration of modern orchestration tools, laid the groundwork for cloud-native deployment. The ability to deploy and scale the system in a cloud environment, using platforms such as AWS, Azure, or Google Cloud, provided increased flexibility and the potential for dynamic resource allocation.

**Outcomes:**

* **Improved Scalability:** With the transition to containers, individual services could be scaled independently to handle fluctuating workloads. This improved scalability allowed the system to grow in line with the needs of the business without major infrastructure overhauls.
* **Faster Deployment and Updates:** Containerisation simplified deployment pipelines by allowing for automated builds and consistent environments across development, testing, and production. This reduced deployment times, improved CI/CD workflows, and allowed for more frequent updates and faster iteration cycles.
* **Cloud-Native Readiness:** The modernised application was designed to be easily deployable in cloud environments, leveraging container orchestration tools like Kubernetes to manage and scale services. This cloud-native readiness enabled the application to take full advantage of cloud elasticity, reducing the reliance on on-premise infrastructure and allowing for more efficient resource utilization.
* **Easier Integration with New Technologies:** The modular design enabled seamless integration with new technologies and platforms, ensuring that the system could continue to evolve as new requirements or business needs arose. This flexibility positioned the system for future growth, supporting innovations like AI/ML capabilities, big data analytics, and advanced security measures.

## 6.4 Security Improvements Through Zero Trust and RBAC

The transition to a Zero Trust security model, coupled with the integration of Role-Based Access Control (RBAC), was a cornerstone of the modernisation effort. The legacy security model, based on Kerberos authentication and perimeter security, was vulnerable to modern security threats such as internal breaches, privilege escalation, and lateral movement. The modernised application aimed to implement a more robust security posture that would mitigate these risks while ensuring compliance with enterprise security standards.

**Key Findings:**

* **Zero Trust Implementation:** The adoption of the Zero Trust security model fundamentally changed how security was approached. By enforcing the principle of "never trust, always verify," each service-to-service communication and access request had to undergo rigorous identity validation and authorisation checks before access was granted.
* **Decentralised Authentication:** The integration of OpenID Connect (OIDC) and OAuth 2.0 protocols provided decentralised, token-based authentication and authorisation. This reduced the reliance on a centralised authentication system and made it easier to validate user identities and authorisations in real-time across distributed environments.
* **Granular Access Control:** RBAC was implemented to enforce least-privilege access at the role level. This ensured that users only had access to the resources necessary for their job functions, improving both security and compliance. The RBAC system was tightly integrated with the OIDC identity provider to ensure that roles were dynamically assigned based on user identity claims.

**Outcomes:**

* **Enhanced Security Posture:** The Zero Trust model significantly reduced the risk of internal threats, as no user or device was implicitly trusted. Identity verification and authorisation checks were applied at every access point, preventing unauthorised access and reducing the attack surface.
* **Mitigated Privilege Escalation Risks:** By enforcing RBAC, the system ensured that users and services only had access to the minimum set of resources necessary for their roles. This reduced the risk of privilege escalation, where an attacker could gain unauthorized access to sensitive resources.
* **Improved Compliance and Auditability:** The integration of RBAC and the centralisation of security policies through the OIDC identity provider improved the traceability of user activities. Each access attempt was logged, capturing detailed information about user identity, roles, and actions. This enhanced auditability, making it easier to comply with regulatory requirements and providing transparency in case of security incidents.
* **Resilience Against Lateral Movement:** The implementation of Zero Trust prevented lateral movement within the infrastructure. Even if a breach occurred, the attacker would face numerous obstacles in attempting to pivot to other parts of the system, reducing the potential impact of the attack.

## 6.5 Operational Efficiency and Automation Gains

The shift from a legacy system relying on manual, error-prone processes to a modernised, automated containerised environment brought significant improvements in operational efficiency. The previous infrastructure had high operational overhead due to manual configuration management, inconsistent deployments, and a lack of integration with modern DevOps practices. In contrast, the modernised application leveraged automated deployment pipelines, containerisation, and orchestration tools to streamline operations.

**Key Findings:**

* **Containerisation Efficiency:** Containerisation was a fundamental driver of operational efficiency. Docker allowed application components to be packaged with all their dependencies, ensuring consistency across different environments and preventing issues caused by "works on my machine" syndrome. The use of Docker Compose further simplified the management of multi-container applications by providing a unified configuration file to manage service dependencies, networks, and volumes.
* **Automated Deployment Pipelines:** By adopting Docker-based deployment pipelines, the application was able to automate the entire build, testing, and deployment process. This reduced human error and accelerated the delivery of new features, patches, and updates. The integration of Gradle with Docker streamlined the process of building Java-based container images, improving build consistency and speed.
* **Scalability and Flexibility:** The use of containers provided a scalable infrastructure capable of handling fluctuating workloads. Docker's lightweight nature allowed for rapid scaling of services, ensuring that the application could handle high traffic volumes without significant performance degradation. The modular nature of containers also allowed individual components to be updated or replaced independently, offering greater flexibility in managing the application.
* **DevOps Integration:** The modernisation effort aligned the application with DevOps principles, facilitating collaboration between development and operations teams. The automated workflows reduced the need for manual intervention and ensured that application updates were deployed seamlessly across development, staging, and production environments.

**Outcomes:**

* **Faster Time-to-Market:** The implementation of automated deployment pipelines significantly reduced the time required to move changes from development to production. This resulted in faster release cycles, allowing new features, bug fixes, and updates to be deployed more quickly and reliably.
* **Consistency Across Environments:** The use of Docker ensured that application environments were consistent from development through production. The standardisation of application dependencies, configurations, and runtime environments eliminated environment drift and minimised the risk of issues arising due to inconsistent setups.
* **Increased Developer Productivity:** The automation of build, test, and deployment processes allowed developers to focus more on coding and less on managing infrastructure. The ability to replicate production environments locally using Docker Compose also improved development workflows, reducing setup time and increasing efficiency.
* **Reduced Operational Overhead:** With the automated deployment pipeline and containerised environments, the operational burden on teams was significantly reduced. The need for manual configuration and troubleshooting was minimised, and the time spent on managing servers and infrastructure was cut down.

## 6.7 Improved Developer Productivity and Collaboration

The modernisation of the internal application at Morgan Stanley led to a significant improvement in developer productivity and collaboration. Prior to the transition, the legacy system was encumbered by manual processes, inefficient workflows, and dependencies that made it difficult for development teams to deliver new features and fix issues in a timely manner. The introduction of a cloud-native, containerised architecture drastically reduced friction, streamlined workflows, and fostered better collaboration across development teams.

**Key Findings:**

* **Consistent Development Environments:** One of the major improvements in the modernised architecture was the use of Docker containers to ensure consistency between development, testing, and production environments. In the legacy system, differences between local development setups and production servers led to issues such as "it works on my machine" and costly deployment delays. By containerising services, every developer could work within the same environment, ensuring that bugs caused by environmental differences were eliminated.
* **Streamlined Collaboration through Containerisation:** The implementation of Docker Compose further enhanced collaboration. Development teams could now easily replicate the entire application stack on their local machines by using a single configuration file, making it easy to run the application and its dependencies in isolated containers. This made it possible for developers to work on separate features or components without worrying about conflicting dependencies or breaking the overall system.
* **Continuous Integration and Continuous Deployment (CI/CD):** A robust CI/CD pipeline, integrated with Docker and Gradle, significantly boosted productivity by automating several stages of the development process. The pipeline facilitated automated testing, build automation, and seamless deployments, allowing teams to quickly push code updates to production. This reduction in manual tasks freed developers from time-consuming deployment processes and minimized human error, leading to quicker and more reliable releases.
* **Isolated Services for Parallel Development:** With the adoption of microservices architecture and Docker containers, development teams could now work on isolated services concurrently without interfering with each other’s work. This encouraged parallel development of features, bug fixes, and updates. Each service could be independently developed, tested, and deployed, which not only sped up development time but also made it easier to manage and maintain the application.

**Outcomes:**

* **Faster Time to Market:** Developers were able to rapidly build, test, and deploy new features, resulting in shorter development cycles. With an integrated CI/CD pipeline and standardised containerised environments, code was tested and deployed faster, leading to quicker delivery of features to end-users.
* **Higher Code Quality and Fewer Errors:** The automated testing and continuous integration features of the new system ensured that potential issues were caught early in the development cycle. With more automated processes in place, there was less opportunity for human error, resulting in cleaner, more reliable code being pushed to production.
* **Improved Collaboration Across Teams:** The uniformity provided by Docker and the adoption of a microservices approach helped break down silos between different teams. Developers, QA engineers, and operations teams could now collaborate more effectively, as they all worked within the same architecture and followed the same workflows. The sharing of configuration files and automated processes made it easier for teams to stay aligned and work together seamlessly.
* **Reduced Downtime and Faster Fixes:** The shift to containerised workflows also meant that the time spent troubleshooting and resolving deployment issues was significantly reduced. With a more predictable and consistent environment, fixes could be implemented and tested faster, reducing overall downtime and improving the responsiveness of the application.

**Overall Impact:**

The modernised architecture transformed the development process by introducing a more structured, automated, and collaborative workflow. Developers were empowered to work more efficiently, with reduced overhead and more streamlined processes. This transformation not only enhanced productivity but also facilitated a culture of continuous improvement and collaboration, ultimately driving faster and more reliable feature delivery while ensuring better quality and stability in the application.

# 7. Social Impact of the Solution

The modernisation of the internal application infrastructure at Morgan Stanley has led to far-reaching social impacts, particularly in the realms of **cybersecurity**, **risk mitigation**, and **customer service**. Beyond the technical and business advantages, the changes have contributed to fostering a more secure, transparent, and customer-centric environment. By implementing cutting-edge solutions like Zero Trust security, OAuth 2.0, and robust authentication protocols, the modernised system has directly enhanced both organisational security and client interactions, creating a ripple effect that extends into the broader financial ecosystem.

**Strengthened Cybersecurity and Protection Against Insider Threats**

A significant social impact of the solution has been the **strengthening of cybersecurity measures**, which directly correlates to increased trust among customers, regulatory bodies, and employees. The adoption of **Zero Trust** principles ensures that every request for access, whether internal or external, is continuously authenticated and authorised, thus drastically reducing the risk of breaches due to compromised credentials or malicious insiders.

Prior to the implementation of these security measures, insider threats posed a considerable risk within financial institutions, especially in environments where legacy systems often lacked adequate monitoring and fine-grained access control. By embracing **Role-Based Access Control (RBAC)** and **advanced authentication protocols** like **OpenID Connect** and **OAuth 2.0**, the modernised system effectively mitigates the potential for insider trading, fraud, or unauthorised access to sensitive data.

In practice, the transition has led to a **reduction in the potential for fraudulent activity** by ensuring that employees and contractors only have access to the specific data and resources necessary for their roles. This minimisation of unnecessary access reduces the likelihood of accidental or deliberate exposure of sensitive financial data. Consequently, the enhanced security protocols not only protect Morgan Stanley’s assets but also help protect the integrity of the broader financial system, which directly impacts the company’s reputation and public trust.

**Building Trust Through Transparency and Compliance**

By implementing robust security measures and complying with modern regulatory frameworks, the modernised application has strengthened Morgan Stanley’s ability to meet regulatory requirements while ensuring a high level of transparency. With the **OAuth 2.0** protocol and **JWT-based tokens**, the system facilitates **auditable interactions** and granular tracking of who accessed what data and when. This auditability is vital for both **regulatory compliance** and addressing stakeholder concerns regarding **data privacy** and **financial transparency**.

As financial institutions increasingly face pressure from regulators to adopt secure and transparent practices, the modernisation has placed Morgan Stanley ahead of the curve. The implementation of a **cloud-native infrastructure** not only ensures resilience against attacks but also enables compliance with global data protection standards, improving the company's standing in the financial community.

**Reducing Risk of Insider Trading and Market Manipulation**

In a financial institution like Morgan Stanley, the risk of **insider trading** is always a significant concern. By modernising the application with **real-time monitoring**, **advanced user tracking**, and **role-based access controls**, the system provides greater visibility into all user actions and interactions with financial data. With better logging and tracking capabilities, it is now easier to identify suspicious activities or patterns that could indicate potential insider trading.

The ability to detect abnormal behaviour, coupled with proactive alerts and automated compliance checks, means that any attempts at insider trading or **market manipulation** can be quickly identified and addressed before they have a chance to escalate. The use of containerisation and microservices further isolates sensitive data, ensuring that no single component has access to all financial information. This **segmentation of duties** and **data silos** is critical in reducing the risk of internal collusion or external attacks targeting sensitive financial transactions.

By improving access control, monitoring user activities, and implementing preventive measures, the application now helps Morgan Stanley to **mitigate risks** associated with the potential misuse of privileged access, further securing the organisation’s operations.

### Conclusion and Future Scope

**7.1 Conclusion**

The transformation of Morgan Stanley’s internal application infrastructure through the adoption of modern technologies and security protocols has brought about a significant leap in the company’s operational efficiency, security, and overall customer experience. By transitioning from legacy infrastructure to a cloud-ready, containerised system, the company has set the stage for a more scalable, resilient, and future-proof platform capable of meeting the demands of modern financial services.

The replacement of outdated authentication mechanisms, such as Kerberos, with **OpenID Connect (OIDC)** and **OAuth 2.0** has fundamentally enhanced the security posture of the organisation. These protocols enable token-based authentication and fine-grained authorisation, reducing the risk of internal and external threats, including insider trading and data breaches. By leveraging the **Zero Trust security model**, the system ensures that no user or device is implicitly trusted, regardless of their location, thereby reinforcing the company's resilience against potential security incidents.

The adoption of **containerisation with Docker** and the transition to a **microservices-based architecture** have facilitated greater flexibility, faster development cycles, and seamless deployment across different environments. These technologies also support the company’s cloud-first strategy, enabling scalability, resource optimisation, and improved performance. By using **Docker Compose** for multi-container orchestration and building robust automation pipelines with tools like **Gradle**, the development team has streamlined their workflows, reduced manual interventions, and improved deployment consistency.

Furthermore, the implementation of **Role-Based Access Control (RBAC)** and **advanced auditing capabilities** has played a key role in enhancing compliance with regulatory standards, ensuring that the application meets both internal and external security requirements. The ability to trace user activities and ensure access controls are strictly enforced has not only increased security but also demonstrated Morgan Stanley’s commitment to maintaining the highest standards of financial integrity and trust.

Overall, the modernisation of the internal application has laid a strong foundation for future growth, empowering the company to better serve its clients, optimise resources, and mitigate security risks. The integration of these advanced tools and practices has enabled Morgan Stanley to remain at the forefront of financial innovation, ensuring that its services are secure, scalable, and adaptable to future demands.

**7.2 Future Scope**

While the transformation of the internal application infrastructure has already yielded impressive results, the journey does not end here. There are several avenues for further enhancement, innovation, and optimisation that will continue to drive Morgan Stanley’s evolution in the coming years. The next steps focus on advancing the capabilities of the system in alignment with emerging technologies and evolving market needs.

One of the key areas of future focus is the **continued evolution of the security framework**. Although the Zero Trust security model and modern authentication protocols have significantly bolstered the security posture of the organisation, there is an ongoing need to stay ahead of emerging cybersecurity threats. As financial services become more digitised, there is an increasing risk of sophisticated attacks targeting sensitive customer data. Future improvements could include the integration of **AI-driven threat detection systems**, which use machine learning to identify abnormal patterns and predict potential breaches in real time. Additionally, leveraging **biometric authentication** and **multi-factor authentication (MFA)** could further enhance the security of user access and data protection.

Another promising area for future development is the **integration of blockchain technology** for transaction verification and smart contract management. The decentralised nature of blockchain could provide an added layer of transparency, security, and efficiency in financial transactions. By combining blockchain with the existing containerised and microservices architecture, Morgan Stanley could offer more secure, transparent, and efficient services to clients while reducing operational costs associated with traditional verification methods.

The **expansion of cloud-native technologies** will also be a key focus in the coming years. With cloud platforms becoming more sophisticated and widespread, Morgan Stanley could explore the integration of **serverless computing** to further optimise resource allocation and reduce operational costs. Serverless architecture can automatically scale based on demand, offering significant flexibility and cost savings, while freeing the development team from having to manage infrastructure.

Lastly, **customer experience improvements** will remain a high priority. The transformation of the internal application has already led to enhanced interaction with clients through improved security, faster response times, and more reliable services. Future innovations could include the incorporation of **AI-driven customer service tools** such as chatbots, predictive analytics, and personalised recommendations based on user data. These tools could help Morgan Stanley provide even more tailored services to clients, further enhancing satisfaction and retention.

#### 7.3.1 Artificial Intelligence and Machine Learning Integration

The increasing importance of **Artificial Intelligence (AI)** and **Machine Learning (ML)** in financial services provides significant opportunities to enhance decision-making, risk management, and customer experience. In the future, Morgan Stanley could expand its use of AI-driven analytics to streamline business processes and provide more precise insights. By integrating advanced AI models into the modernised architecture, the company could enhance fraud detection systems, identify market trends more accurately, and automate complex decision-making processes.

For example, AI algorithms could analyse transaction data in real time to detect suspicious patterns, flagging potential fraud before it occurs. In portfolio management, machine learning models could optimise asset allocation and recommend investment strategies based on market conditions and individual client profiles. Furthermore, the ability to harness AI to process large volumes of unstructured data—such as financial reports, news articles, and social media content—could provide a competitive edge in delivering more accurate, up-to-date market insights.

#### 7.3.2 Advanced Automation and Continuous Integration/Continuous Delivery (CI/CD)

With the adoption of a **microservices architecture**, Morgan Stanley can take full advantage of **Continuous Integration (CI)** and **Continuous Delivery (CD)** pipelines to automate testing, deployment, and scaling of applications. By enhancing automation workflows, the company could reduce time-to-market for new features and improve system reliability. In particular, the integration of automated testing frameworks for security and performance can further minimise human error and ensure that new deployments do not introduce vulnerabilities or performance bottlenecks.

Future improvements could include expanding the use of **AI-based testing tools**, which can predict potential code issues before they arise by analysing patterns in previous code commits. This proactive approach to code quality would help to maintain the integrity of the system as it evolves and scales.

Furthermore, **Infrastructure as Code (IaC)** practices could be expanded, allowing for more seamless management of the underlying infrastructure. IaC frameworks such as **Terraform** and **Ansible** could automate the provisioning and configuration of cloud resources, ensuring that the application environment is always aligned with the most up-to-date best practices. This level of automation ensures that both the development and operations teams can focus more on delivering value rather than managing infrastructure components.

#### 7.3.3 Expansion of Data Analytics Capabilities

The growing volume of data generated by both internal and external sources offers significant opportunities to unlock new insights and enhance operational efficiency. Morgan Stanley's modernisation efforts could include the expansion of its **data analytics capabilities** through the integration of **big data platforms** like **Apache Hadoop** or **Apache Spark**. By processing vast amounts of unstructured data in real time, the company could gain deeper insights into market movements, customer behaviour, and internal operations.

As data privacy regulations become more stringent, ensuring compliance while expanding data analytics capabilities will be crucial. With the use of data masking, encryption, and differential privacy techniques, Morgan Stanley can protect sensitive information while still deriving value from large datasets. By leveraging **advanced data visualisation tools** and **interactive dashboards**, business users could gain actionable insights without the need for complex data querying, leading to more informed decision-making at all levels of the organisation.

#### 7.3.4 Further Optimisation of Cloud Infrastructure

As the company continues its cloud migration journey, there are further optimisations to be made in how cloud resources are utilised. One area for improvement is **cost optimisation** in the cloud. By leveraging cloud-native tools such as **AWS Cost Explorer** or **Azure Cost Management**, Morgan Stanley could gain greater visibility into resource utilisation and cost trends, allowing for more informed decisions around resource allocation. This could lead to a more efficient use of cloud infrastructure, reducing operational costs and maximising the return on investment.

Another area of future development is **multi-cloud strategies**. Although the current strategy may focus on a specific cloud provider, future infrastructure may involve multi-cloud environments to ensure redundancy, enhance resilience, and avoid vendor lock-in. A multi-cloud approach would allow Morgan Stanley to select the best-suited services from different cloud providers for each specific use case, ensuring optimal performance, cost-efficiency, and compliance with global regulations.

Additionally, **edge computing** could play a more prominent role in future cloud infrastructure designs, particularly in scenarios that require low-latency processing and real-time data analysis. By processing data closer to the source, such as on IoT devices or at local data centres, edge computing can help reduce delays and improve response times for certain applications, enhancing the customer experience in time-sensitive operations.

# References

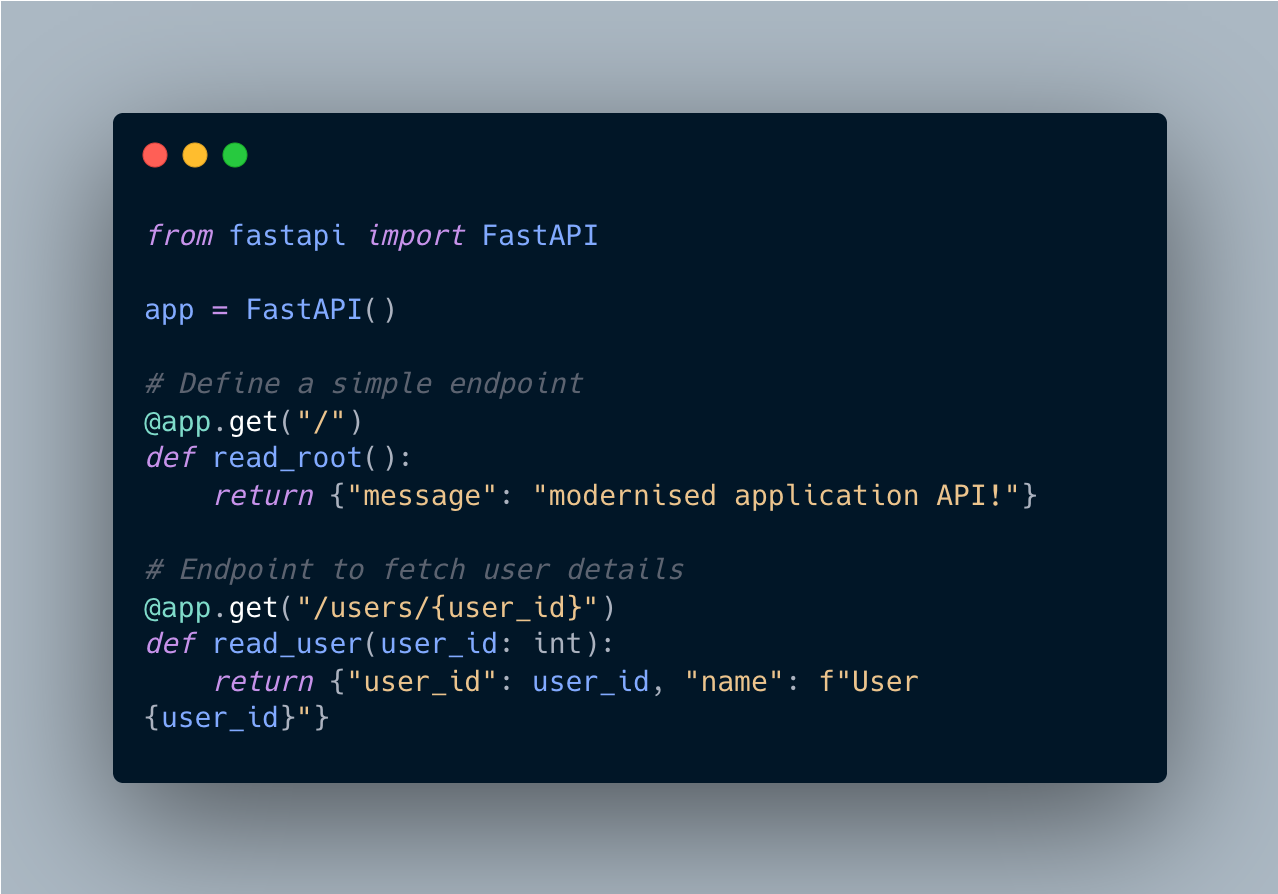
1. **National Institute of Standards and Technology (NIST)**. (2020). *Zero Trust Architecture*. NIST Special Publication 800-207. Retrieved from<https://csrc.nist.gov/publications/detail/sp/800-207/final>
2. **OAuth 2.0**. (2021). *The OAuth 2.0 Authorization Framework*. IETF RFC 6749. Retrieved from https://tools.ietf.org/html/rfc6749
3. **OpenID Connect**. (2021). *OpenID Connect 1.0 - The Authentication Layer for OAuth 2.0*. OpenID Foundation. Retrieved from https://openid.net/connect/
4. **Docker**. (2021). *Docker Documentation*. Retrieved from https://docs.docker.com/
5. **Spring Boot**. (2021). *Spring Boot Documentation*. Pivotal Software. Retrieved from<https://spring.io/projects/spring-boot>
6. **Gradle**. (2021). *Gradle User Manual*. Retrieved from https://docs.gradle.org/current/userguide/userguide.html
7. **Amazon Web Services (AWS)**. (2021). *AWS Well-Architected Framework*. AWS Whitepapers. Retrieved from<https://aws.amazon.com/architecture/well-architected/>
8. **Gartner**. (2020). *The 2020 Magic Quadrant for Cloud Infrastructure and Platform Services*. Gartner, Inc. Retrieved from https://www.gartner.com/document/3985674
9. **The Open Group**. (2020). *The Open Group Zero Trust Architecture (ZTA) White Paper*. Retrieved from<https://www.opengroup.org/>
10. **Microsoft Azure**. (2021). *Azure Security and Compliance Documentation*. Microsoft Corporation. Retrieved from<https://docs.microsoft.com/en-us/azure/security/>
11. **Kubernetes**. (2021). *Kubernetes Documentation*. Kubernetes. Retrieved from https://kubernetes.io/docs/
12. **McKinsey & Company**. (2021). *The Future of Cybersecurity in Financial Services*. McKinsey Insights. Retrieved from<https://www.mckinsey.com/>
13. **Forrester Research**. (2021). *The Future of Cloud Security in Financial Services*. Forrester Research. Retrieved from<https://www.forrester.com/>
14. **PwC**. (2020). *The Role of Cloud in Financial Services Digital Transformation*. PricewaterhouseCoopers. Retrieved from<https://www.pwc.com/>
15. **Cloud Security Alliance (CSA)**. (2020). *Security Guidance for Critical Areas of Focus in Cloud Computing V4.0*. Cloud Security Alliance. Retrieved from<https://cloudsecurityalliance.org/>

# Appendix

## Appendix A

#### A.1 Python Code

This snippet shows how to create a simple API service using FastAPI. FastAPI is a modern web framework for building APIs with Python, designed for high performance.



**Explanation**: This code sets up a basic FastAPI application with two endpoints:

* One that responds to a GET request at the root (/), sending a welcome message.
* Another that retrieves user details based on a user\_id provided in the URL path.

#### 

#### A.2 Simple Java Spring Bean

This snippet demonstrates a simple Spring Bean in Java. Spring Beans are objects that are managed by the Spring IoC (Inversion of Control) container.



Explanation: This UserService class is annotated with @Component, making it a Spring Bean. The class contains methods to get and set the username, which can be injected into other parts of the Spring application.

#### 

#### A.3 Gradle Dependency Example

This is an example of a Gradle build file that includes some common dependencies for a Java-based Spring Boot project.



Explanation: The build.gradle file configures the project to use Spring Boot for building a web application. It includes dependencies for Spring Web, Spring Data JPA, and Spring Security, which are necessary for the services in your modernised application.

#### A.4 Dockerfile Example

A simple Dockerfile for building a container image for a Python-based API service.

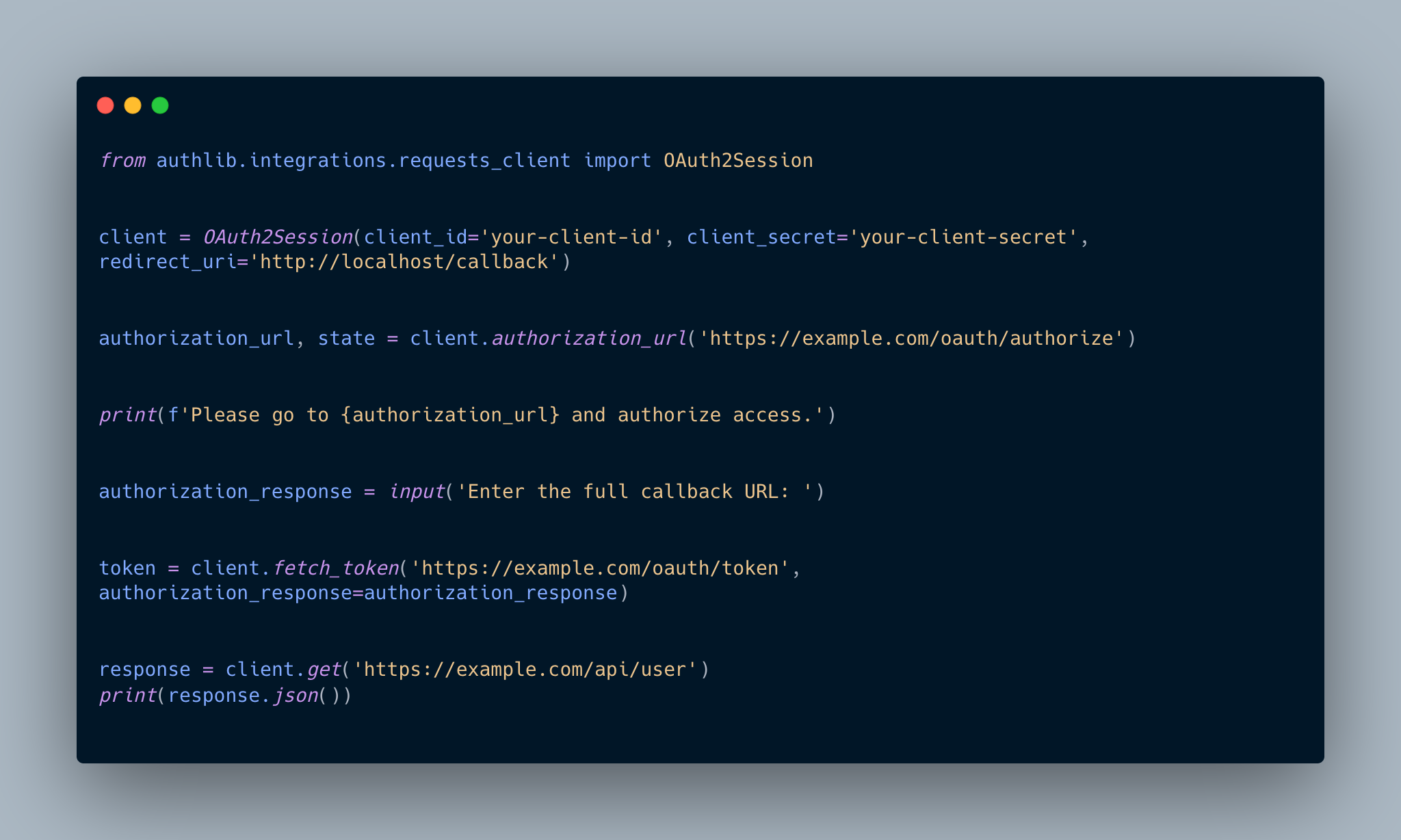


Explanation: This Dockerfile:

* Starts with a Python base image.
* Copies the requirements.txt file and installs dependencies.
* Copies the application code and exposes port 8000.
* Specifies the command to run the FastAPI application using Uvicorn.

#### A.5 OpenID Connect (OIDC) Implementation in Python

This code snippet shows how to implement OpenID Connect (OIDC) in Python using the Authlib library for handling OAuth2 and OIDC flows.



Explanation: This code handles the OAuth2 flow with OpenID Connect. It:

* Initializes an OAuth2Session with the OIDC configuration.
* Directs the user to the authorization URL for login.
* Fetches the access token after the user authorizes the request.
* Makes a request to a protected resource using the token.