PROJECT REPORT

ON

END-TO-END DATA ENCRYPTION SYSTEM FOR COLLEGE APPLICATIONS USING AZURE KEY VAULT

Submitted by

VARUN G	(2116220701310)
VARUN KUMAR V	(2116220701311)

UNDER THE GUIDANCE OF

MRS. SANTHIYA M

COURSE: CLOUD COMPUTING

RAJALAKSHMI ENGINEERING COLLEGE

ANNA UNIVERSITY, CHENNAI

NOVEMBER 2025

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

BONAFIDE CERTIFICATE

This is to certify that the project report entitled "**End-to-End Data Encryption System for College Applications Using Azure Key Vault**" submitted by **Varun G (220701310), Varun Kumar V (220701311)** of the Department of Computer Science and Engineering, Rajalakshmi Engineering College, has been carried out under my supervision in partial fulfillment of the requirements for the course Cloud Computing.

Faculty Guide: ________________________

Head of the Department: ________________________

Date:___________________

ACKNOWLEDGMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere thanks to our Chairman **Mr. S. MEGANATHAN, B.E, F.I.E.**, our Vice Chairman **Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.,** and our respected Chairperson **Dr. (Mrs.) THANGAM MEGANATHAN**, **Ph.D.,** for providing us with the requisite infrastructure and sincere endeavoring in educating us in their premier institution.

Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.,** our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to **Dr. E.M. MALATHY, M.E., Ph.D.**, Professor and Head of the Department of Computer Science and Engineering for his guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our faculty guide **Mrs. M. SANTHIYA, M.E.**, Assistant Professor Department of Computer Science for their invaluable guidance, support, and encouragement throughout this project.

VARUN G 220701310

VARUN KUMAR V 2116220701311

ABSTRACT

This project presents the design and implementation of a **secure, cloud-native College Application Management System** that ensures **end-to-end data encryption** using **Azure Key Vault** and infrastructure automation through **Terraform**. The proposed system addresses the need for privacy and confidentiality in the storage and transmission of sensitive student application data by integrating **envelope encryption**, where data is encrypted at the client side and the encryption keys are securely managed within Azure Key Vault.

The application architecture comprises a **React.js frontend** for user interaction and client-side encryption, and a **Node.js backend** for secure API handling, data storage, and key wrapping operations through Azure Key Vault's cryptographic APIs. Encrypted data and metadata are stored in **Azure Blob Storage**, while **Terraform** is employed to automate the provisioning and management of cloud resources such as Azure App Service, Azure Container Registry (ACR), Key Vault, and Storage Accounts, establishing a robust **Infrastructure as Code (IaC)** framework. The system is fully containerized using **Docker**, ensuring consistent deployments and seamless integration with Azure's cloud environment.

Furthermore, **Generative AI (GenAI)** capabilities from **Azure OpenAI Service** are integrated to enhance the system's intelligence by enabling automated summarization and validation of submitted application content, thus augmenting administrative decision-making. The solution demonstrates secure data handling, scalability, and operational efficiency through infrastructure automation and container orchestration.

Overall, this project showcases a comprehensive **cloud-enabled, security-driven, and AI-augmented solution** that exemplifies the practical application of

modern cloud computing, encryption technologies, and generative AI for safeguarding and managing college application data in a distributed environment.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

In the modern digital era, educational institutions increasingly rely on online platforms for student application management. These systems handle large volumes of sensitive data such as personal information, academic records, and financial details. However, most existing college application portals do not implement strong data encryption mechanisms or centralized key management systems. As a result, confidential student data is often stored in plain text or with weak encryption methods, making it vulnerable to breaches, unauthorized access, and data leaks.

Manual key handling, hardcoded credentials, and inconsistent security policies further compromise data privacy and regulatory compliance. Institutions face major challenges in ensuring end-to-end data protection from the moment a student submits an application to its storage, processing, and retrieval by administrators.

To address these critical challenges, there is a need for a cloud-based secure application **system** that leverages Azure Key Vault for encryption key management, enforces end-to-end data encryption, and automates infrastructure provisioning through Terraform. By integrating Docker containerization and Generative AI (GenAI), the system can also enhance intelligence in the verification and summarization of application content while maintaining robust security, scalability, and automation.

1.2 Objective of the Project

The primary objective of this project is to design and implement an end-to-end data encryption system for college applications that ensures complete security, scalability, and automation using Microsoft Azure Cloud. The system is built to securely encrypt and manage student application data while seamlessly integrating with modern cloud and AI technologies. It implements client-side encryption using the AES-GCM algorithm, with secure key management and wrapping handled by Azure Key Vault to ensure robust data protection. A React.js frontend is developed to allow students to submit encrypted application data, while a Node.js backend integrates with Azure Key Vault, Blob Storage, and the OpenAI API to manage encryption keys, store ciphertext securely, and generate intelligent summaries of the submitted applications. The infrastructure is automated using Terraform (infrastructure as code) to provision and manage Azure resources such as App Service, Storage Account, Key Vault, and Container Registry efficiently. The application services are containerized with Docker, enabling consistent, scalable, and portable deployments. Furthermore, generative AI (GenAI) is integrated to summarize and validate submitted applications, providing automated insights and efficiency for admission administrators. Overall, the project delivers a unified solution combining cloud security, infrastructure automation, and AI augmentation to ensure the end-to-end protection, high availability, and intelligent processing of college application data.

1.3 Scope and Boundaries

The proposed system provides a secure, automated, and scalable framework for managing sensitive student application data within educational institutions. It encompasses the entire data lifecycle — from collection and encryption to storage, retrieval, and intelligent analysis — while ensuring compliance with privacy and data protection standards such as GDPR and FERPA. The scope of the system includes secure encryption and decryption operations using Azure Key Vault and Managed Identities, automated infrastructure setup and environment replication through Terraform, and scalable, containerized deployment using Docker and Azure App Service. Additionally, the solution offers optional AI-driven summarization and validation capabilities powered by Azure OpenAI Service to enhance administrative efficiency.

The system's boundaries are clearly defined to maintain focus and security. It does not involve external data sharing or integration with third-party admission systems. Encryption and key management are restricted to the algorithms supported by Azure Key Vault, such as RSA-OAEP and AES-GCM. Real-time collaborative editing and offline access to encrypted data are outside the project's scope. Furthermore, Generative AI models are utilized solely for text summarization and content enhancement, without any role in automated decision-making or admission scoring processes.

1.4 Stakeholders and End Users

The success of this project relies on the collaboration of multiple stakeholders who interact with the system at various levels, each playing a critical role in ensuring its effectiveness and security. Students use the frontend application to submit their college applications securely, with their data being encrypted before transmission. Admission officers review and process these applications, gaining authorized access to decrypt data and benefit from AI-generated summaries and insights that streamline evaluation. System administrators are responsible for managing encryption policies, monitoring data integrity, and overseeing access controls through Azure Key Vault and IAM roles. Developers and DevOps engineers handle the development, containerization, and automated deployment of the application using technologies such as Node.js, React, Docker, and Terraform. Cloud engineers configure, manage, and monitor Azure resources including App Service, Storage, and Key Vault to ensure optimal availability and security. Finally, institution management leverages the summarized analytics and AI-driven insights for strategic decision-making and improving the overall admission process.

This collaborative stakeholder ecosystem ensures data confidentiality, operational efficiency, and compliance-driven management of student applications, aligning technology with institutional goals for secure and intelligent admission handling.

1.5 Technologies Used

The project integrates various technologies to ensure automation, scalability, and reliability. The frontend is developed using React.js for a responsive interface, while the backend uses Node.js with Express for handling requests and APIs. Azure Cosmos DB stores structured feedback and sentiment data, and Azure Text Analytics performs sentiment analysis. Terraform automates infrastructure provisioning, and Docker enables containerization for portability. Azure Kubernetes Service (AKS) manages container orchestration and scalability, while GitHub Actions provides a CI/CD pipeline for automated deployment. Additionally, Azure OpenAI or GPT-based models generate intelligent feedback summaries and performance insights.

Table 1.1 Technology Stack

Layer	Technology	Purpose
Frontend	React	Collects and encrypts application data before submission
Backend	Node.js + Express	Handles key management, API integration, and data processing

Database	Azure Blob Storage	Stores encrypted application data and wrapped keys
Encryption & Security	Azure Key Vault + AES-GCM + RSA-OAEP	Provides secure key generation, wrapping, and data encryption
AI Service	Azure OpenAI (GPT Models)	Generates summaries and insights from submitted application content
Containerization	Docker + Docker Hub	Packages frontend and backend for consistent, portable deployment
Infrastructure as Code	Terraform	Automates provisioning of Azure resources (App Service, Key Vault, Storage, ACR)
CI/CD	GitHub Actions	Builds, tests, and deploys containerized applications automatically
Hosting	Azure App Service (Web App)	Hosts the containerized backend and frontend with integrated security and scalability

CHAPTER 2

SYSTEM DESIGN AND ARCHITECTURE

2.1 Requirement Summary

The proposed system is designed to securely handle, encrypt, and manage student application data within a college admission workflow. The major goal is to ensure end-to-end data protection, secure key management, and AI-driven content summarization using Azure Cloud Services.

The functional requirements focus on allowing students to upload their application forms and documents via a web interface, encrypting the data client-side, securely wrapping the encryption keys through Azure Key Vault, and storing encrypted data in Azure Blob Storage. Authorized administrators can later retrieve and decrypt the data after proper authentication and key-unwrapping. The system also integrates Generative AI (OpenAI Service) to summarize submitted essays and generate smart insights for admission officers.

The non-functional requirements emphasize scalability, high availability, security, and performance. Terraform ensures Infrastructure as Code (IaC) deployment across environments, while Docker containers standardize application execution. Azure App Service and Azure Container Registry (ACR) guarantee continuous delivery and scaling, ensuring the system remains reliable and cost-efficient under varying workloads

2.1.1 Functional Requirements

The system must support data encryption, storage, and secure retrieval workflows. Each major function directly maps to a corresponding Azure service and scaling configuration, as shown below.

Table 2.1 Functional Requirements

Feature	**Expected Usage**	**Azure Services / Spec**	**Scaling Method**
End-to-End Encryption	Encrypt application data client-side with AES-GCM; wrap keys via Key Vault	Azure Key Vault (Key Management), Node.js API	Autoscale App Service instances on demand
Secure Storage	Store encrypted files & metadata	Azure Blob Storage	Enable automatic storage scaling & redundancy
Application Submission	Students upload and submit encrypted forms	React frontend + Node.js backend	Scale frontend using App Service Plan or Azure Front Door
Decryption & Access Control	Authorized admins unwrap keys & view decrypted data	Azure Key Vault + Managed Identity	Role-based access via Azure RBAC
AI Summarization	Generate summaries of essays using GenAI	Azure OpenAI Service (GPT)	Autoscale Cognitive Service based on requests
Deployment Automation	Build, test & deploy Dockerized app	GitHub Actions + ACR + App Service	Auto-trigger pipeline on code push
Infrastructure Automation	Provision and configure cloud resources	Terraform (IaC)	On-demand provisioning; reusable scripts
Hosting	Access the platform via web	Azure Web App Service	Horizontal scaling of app instances based on traffic

2.1.2 Non-Functional Requirements

The system should ensure high availability, scalability, and reliability through Azure Kubernetes Service (AKS). It must provide a responsive and user-friendly interface, ensure data security through Azure's built-in authentication and encryption, and support fast processing with minimal latency. Additionally, the infrastructure setup should be automated using Terraform for consistency and easy maintenance.

Table 2.2 Non Functional Requirements

NFR	**Metric / Target**	**Azure Configuration / Consideration**
Performance	Encrypt + Store transaction < 3 s	Optimize Node.js API threads, use Azure Premium Plan + Blob batch operations
Availability	99.9% uptime	Deploy App Service in multiple zones; enable auto-healing
Scalability	Handle 1000+ feedback submissions/day	Enable auto-scaling for App Service and Blob Storage
Security	Data encrypted at rest & in transit	Use TLS 1.2+, Azure Key Vault for key management, private endpoints
Maintainability	Easy updates and CI/CD deployment	GitHub Actions + Terraform automation; versioned container images
Cost Efficiency	Optimize for usage-based billing	Autoscale services; monitor spend via Azure Cost Management

2.2 Proposed Solution Overview

The **End-to-End Data Encryption System** is a **cloud-native web application** built entirely on Azure. It uses **React.js** as the frontend, enabling students to securely upload and encrypt application data before transmission. The **Node.js backend** communicates with **Azure Key Vault** to wrap encryption keys and store encrypted data and metadata in **Azure Blob Storage**.

The system follows an **envelope encryption model**: a data encryption key (DEK) encrypts each application, while a Key Encryption Key (KEK) in Key Vault wraps the DEK. Only authorized identities with RBAC permissions can unwrap the DEK for decryption.

All infrastructure is defined as code using **Terraform**, which automatically creates and configures Azure resources — App Service, Storage Account, Key Vault, and ACR. The backend and frontend are containerized using **Docker**, ensuring portability and isolation. Deployment is automated through **GitHub Actions CI/CD pipelines**, which build images, push them to ACR, and update App Service instances seamlessly.

Generative AI (GenAI) using **Azure OpenAI GPT models** adds intelligence by summarizing application essays, validating content quality, and providing contextual insights to admission officers without compromising data privacy.

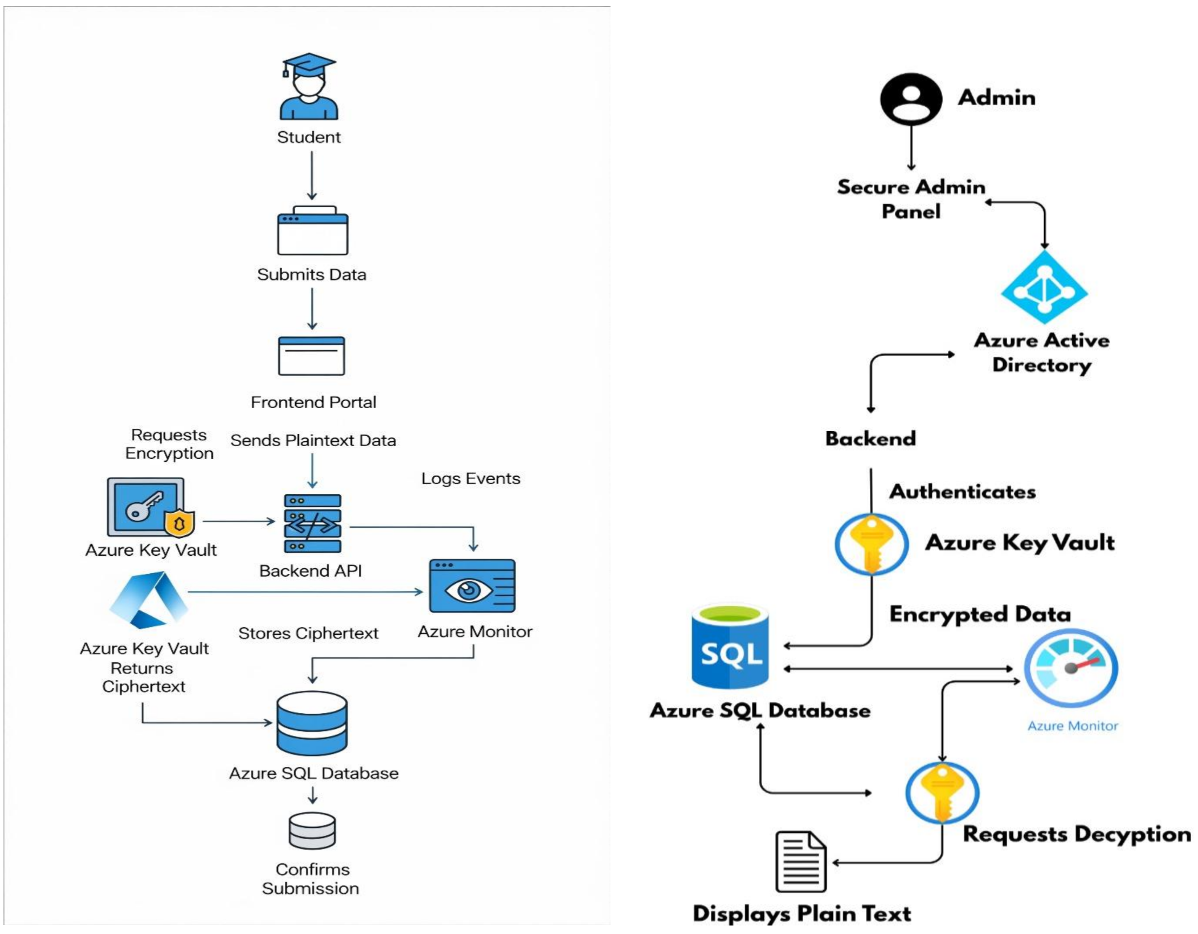


Fig 2.1 Architecture Diagram

2.3 Cloud Deployment Strategy

The deployment strategy for this project follows a hybrid PaaS + IaC model using Microsoft Azure, combining the benefits of managed cloud services with automated infrastructure provisioning. Core services such as Azure Key Vault, Blob Storage, and the OpenAI API operate as fully managed PaaS components, while the main application is deployed on Azure App Service (Linux Container) using custom Docker images sourced from Azure Container Registry (ACR). The system architecture is organized into multiple layers: the presentation layer features a React-based frontend for user interaction and client-side encryption; the application layer comprises a Node.js API that handles interactions with Azure Key Vault and Blob Storage; the security layer uses Azure Key Vault for key generation, wrapping, and secret management; the AI layer leverages Azure OpenAI Service for intelligent summarization and insight generation; the storage layer employs Azure Blob Storage for ciphertext and metadata persistence; and the automation layer integrates Terraform and GitHub Actions to manage infrastructure provisioning and continuous integration/deployment (CI/CD). All resources are hosted in the Central India region to ensure low latency, compliance, and data sovereignty.

Secure private endpoints connect Key Vault and Storage to the App Service through a Virtual Network, enhancing data security and access control. The deployment pipeline adopts continuous integration and rolling update strategies to minimize downtime and ensure consistent availability across all application components.

2.4 Infrastructure Requirements

The infrastructure leverages Azure's scalability and security features to ensure efficient and reliable operations. The Azure App Service Plan is configured with the Standard S1 tier (1 vCPU, 1.75 GB RAM) for development and the S2 tier (2 vCPU, 3.5 GB RAM) for production, supporting auto-scaling between one to five instances based on workload demands. The Storage Account is set up with locally redundant storage (LRS) and private endpoint access to enhance data security when interacting with Blob Storage. Azure Key Vault operates on the Standard tier and manages one RSA 2048-bit key and ten secrets, with role-based access assigned to the App Service's Managed Identity for secure key management. The Azure Container Registry (ACR) stores versioned backend Docker images, with administrative access disabled post-deployment to minimize security risks. Network security is enforced through an Azure Virtual Network (VNet) integrated with Network Security Groups (NSG) and private DNS zones, ensuring secure communication between Key Vault, Storage, and App Service. Continuous integration and deployment are achieved through a GitHub Actions pipeline connected to Terraform and ACR, automating resource provisioning, image builds, and deployments. System monitoring is handled by Azure Monitor and Application Insights, which collect key performance metrics such as latency, request rates, and error logs. For testing purposes, an optional Azure B2s virtual machine is provisioned for local container testing or load simulation. Best practices are followed throughout the setup, including storing all secrets and connection strings exclusively in Key Vault, enforcing HTTPS across both frontend and backend, enabling automatic scaling for App Service and Blob Storage, and activating logging and alert rules through Azure Monitor and Defender for Cloud.

2.5 Azure Services Mapping and Justification

Azure Key Vault

Purpose: Stores and manages encryption keys and secrets.

Justification: Provides secure, centralized key management and hardware security module (HSM) protection for cryptographic operations such as RSA-OAEP key wrapping and unwrapping.

Azure Blob Storage

Purpose: Stores encrypted student applications and metadata.

Justification: Ensures scalable and redundant object storage with encryption at rest, private endpoint access, and seamless integration with Azure services.

Azure App Service (Linux Container)

Purpose: Hosts the Node.js backend and React frontend containers.

Justification: Offers a fully managed, scalable environment with built-in auto-scaling, HTTPS/TLS support, and CI/CD integration, reducing operational overhead.

Azure Container Registry (ACR)

Purpose: Stores Docker images for backend deployments.

Justification: Provides a secure, version-controlled image repository with identity-based pull access from Azure App Service, ensuring consistency across environments.

Terraform

Purpose: Automates the provisioning and configuration of Azure resources through Infrastructure as Code (IaC).

Justification: Enables consistent, repeatable, and version-controlled infrastructure deployments, reducing manual errors and improving maintainability.

GitHub Actions

Purpose: Automates the build, test, and deployment pipelines.

Justification: Facilitates continuous integration and continuous deployment (CI/CD), ensuring faster releases, minimal downtime, and reduced human intervention.

Azure OpenAI Service

Purpose: Summarizes and validates student application content.

Justification: Adds AI-driven intelligence for content summarization and validation, supporting admission administrators with automated insights.

Azure Monitor and Application Insights

Purpose: Tracks application health, logs, and performance metrics.

Justification: Provides real-time visibility, alerting, and diagnostic capabilities for monitoring latency, request failures, and performance anomalies.

Azure Virtual Network (VNet)

Purpose: Isolates Azure services and enables private connectivity between components.

Justification: Enhances security by preventing public exposure and allowing secure communication between App Service, Key Vault, and Blob Storage via private endpoints.

CHAPTER 3

DEVOPS IMPLEMENTATION

3.1 Continuous Integration and Deployment (CI/CD) Setup

The CI/CD pipeline for the **End-to-End Data Encryption System for College Applications** is implemented using **GitHub Actions** integrated with **Azure Cloud**. The pipeline automates the build, test, and deployment stages for both the **Node.js backend** and **React.js frontend**, ensuring rapid and reliable updates with minimal manual intervention.

Each commit or pull request in the main branch triggers the pipeline automatically, executing the following steps:

- **Code Build:**
  The frontend (React) and backend (Node.js) codebases are built, validated, and bundled for deployment.
- **Automated Testing:**
  Unit and integration tests verify encryption/decryption workflows, API responses, and Azure service integrations.
- **Docker Image Creation:**
  Both services are containerized using Docker. The images are tagged with version numbers and pushed to Azure Container Registry (ACR).
- **Infrastructure Validation:**
  Terraform plans are executed to ensure that all required Azure resources — such as Key Vault, Storage, and App Service — are provisioned correctly.
- **Automated Deployment:**
  Once validated, GitHub Actions deploys the latest Docker images to Azure App Service (Linux Container) or AKS, ensuring zero-downtime updates.

This CI/CD implementation guarantees continuous integration, version control, and automated delivery, significantly reducing human errors and deployment times. It enforces a DevOps-driven workflow where every code change passes through testing, containerization, and automated cloud deployment seamlessly.

3.2 Terraform Infrastructure-as-Code (IaC)

Terraform is used to provision and manage Azure resources as code, enabling reproducibility and version control.

The Terraform workflow ensures smooth infrastructure management:

terraform init – Initializes the Terraform working directory and downloads required Azure provider plugins.

terraform plan – Generates an execution plan, showing which resources will be created, modified, or destroyed.

terraform apply – Applies the plan to provision or update resources on Azure.

Using Terraform allows automated creation, modification, and teardown of cloud infrastructure with minimal manual intervention, ensuring consistency and version control across environments.

3.3 Containerization Strategy

The project adopts a containerized architecture using Docker to package both the frontend and backend applications. Containerization ensures that each component runs in a consistent environment, regardless of underlying system differences.

- **Backend Container (Node.js):**
  Includes the application logic for key management, encryption/decryption APIs, and Azure integrations (Key Vault, Blob Storage, OpenAI).
- **Frontend Container (React):**
  Contains the client-side logic for secure form submission, client-side encryption using AES-GCM, and API communication with the backend.

Each Docker image is defined using a Dockerfile that specifies dependencies, build steps, and startup commands. Once built, the images are versioned and pushed to **Azure Container Registry (ACR)** for centralized management.

3.4 Kubernetes Orchestration

Although the application primarily relies on Azure App Service (Linux Containers) for hosting, Azure Kubernetes Service (AKS) can be utilized for advanced orchestration and enhanced production scalability. AKS offers automated scaling, rolling updates, and self-healing capabilities, ensuring that containerized workloads remain resilient and highly available. In this setup, the backend and frontend containers are deployed as separate pods, with Kubernetes Deployment manifests defining replica counts, resource limits, and update strategies. The Horizontal Pod Autoscaler (HPA) dynamically scales pods based on CPU, memory utilization, or incoming request load, maintaining performance efficiency. A Kubernetes LoadBalancer service evenly distributes traffic across pods, ensuring optimal responsiveness and fault tolerance. Rolling updates enable zero-downtime deployments by gradually replacing old pods with new ones, ensuring continuous availability during updates. Additionally, Azure Monitor for Containers provides integrated monitoring and logging to track performance metrics and detect anomalies in real time. By integrating AKS, the system gains the ability to handle large-scale workloads, maintain high availability, and simplify multi-environment management, making it an ideal solution for production environments that demand maximum uptime, flexibility, and operational efficiency.

3.5 GenAI Integration and Azure AI Service Mapping

The system integrates **Generative AI (GenAI)** through the **Azure OpenAI Service** to enhance decision-making and automate content summarization within the admission workflow. Although the primary objective of the system is data security and encryption, GenAI introduces an intelligent processing layer that analyzes and summarizes student-submitted application essays or statements of purpose. The integration involves several Azure AI services, each serving a specific role. **Azure Key Vault (Cryptographic API)** manages RSA-OAEP key wrapping and unwrapping within the Node.js backend, ensuring secure encryption key lifecycle management. **Azure OpenAI Service (GPT model)** processes decrypted essays through secure API calls, generating concise summaries and insight reports that help admission officers evaluate applications more efficiently. Additionally, **Azure Cognitive Services – Text Analytics** can optionally be incorporated to detect sensitive information or inappropriate language, providing a compliance and quality check before AI processing.

The GenAI workflow begins when a student submits an encrypted essay via the frontend. Upon authorized decryption in the backend, the plaintext content is securely transmitted to the Azure OpenAI Service for processing. The GPT model analyzes the content and returns a summarized version or key insights, which are then displayed on the admin dashboard. This AI-enhanced pipeline not only improves the speed and consistency of application reviews but also helps admission officers focus on decision-making rather than manual content evaluation, resulting in a smarter and more efficient admissions process..

CHAPTER 4

CLOUD OPERATIONS AND SECURITY

4.1 DevSecOps Integration

Security is a core pillar of the End-to-End Data Encryption System, seamlessly integrated throughout the development and deployment lifecycle under the DevSecOps framework. The project ensures that security is embedded from the very beginning—starting at code development and extending through production deployment—rather than being treated as an afterthought. The CI/CD pipeline incorporates multiple automated security validation stages to detect and mitigate vulnerabilities early in the process. During the static analysis phase, tools such as CodeQL and SonarCloud are integrated within GitHub Actions to perform automated code reviews that identify insecure patterns, dependency risks, and violations of secure coding practices. For runtime security, Dynamic Application Security Testing (DAST) is implemented using the OWASP Zed Attack Proxy (ZAP) to scan the deployed web application for vulnerabilities such as SQL injection, cross-site scripting (XSS), and insecure HTTP headers.

All sensitive information, including API keys, storage connection strings, and cryptographic materials, is securely managed through Azure Key Vault, ensuring that no credentials are hardcoded or exposed in source files. An automated security gate is built into the CI/CD pipeline, preventing deployments if any high- or critical-severity vulnerabilities are detected. Additionally, Docker images stored in Azure Container Registry (ACR) undergo vulnerability scanning to identify and eliminate insecure or outdated base images before deployment. Through this integrated DevSecOps approach, the system maintains a secure software supply chain, enforces continuous compliance, and upholds the principles of confidentiality, integrity, and availability (CIA)—ensuring a resilient, trustworthy, and security-first architecture across its entire lifecycle.

4.2 Monitoring and Observability (Azure Monitor)

Comprehensive monitoring and observability are essential for ensuring the reliability, stability, and performance of the system in a production cloud environment. The project integrates **Azure Monitor**, **Application Insights**, and **Log Analytics** to achieve continuous visibility across both infrastructure and application layers. **Azure Monitor** collects telemetry data from key resources such as App Service, Key Vault, and Storage Account, tracking important performance metrics including CPU utilization, memory consumption, network throughput, and response latency. Alert rules are configured to automatically notify administrators when thresholds are exceeded, enabling proactive detection and resolution of issues. **Application Insights** is tightly integrated with the Node.js backend to provide detailed end-to-end monitoring of user transactions, response times, exception traces, dependency failures, and API performance. These insights help developers quickly identify bottlenecks and areas of performance degradation. All logs from various Azure services are consolidated within a **Log Analytics Workspace**, where administrators use **Kusto Query Language (KQL)** to query and visualize operational data, detect anomalies, and perform root-cause analysis. Additionally, **custom dashboards** in the Azure Portal present real-time information on system health, uptime, and encryption service performance, supporting fast and informed operational decisions. By combining these monitoring tools, the system achieves comprehensive observability, empowering administrators to detect issues early, analyze incidents effectively, and optimize system performance before they affect end users.

4.3 Access Control

The project employs Role-Based Access Control (RBAC) within Azure to enforce secure, fine-grained access management across all resources. RBAC follows the principle of least privilege, ensuring that users, applications, and services are granted only the permissions necessary to perform their designated functions. Access control is applied consistently across every component of the system to maintain a secure and auditable environment.

For Azure Key Vault, only the App Service's managed identity and authorized administrators have permission to perform cryptographic operations such as key wrapping and unwrapping, as well as access to secrets. Key Vault access policies are configured to segregate read, write, and key management privileges, preventing unauthorized use. Azure Storage Account access is strictly limited to private endpoints and managed identities, with public access disabled. All communication occurs over secure HTTPS connections to maintain data confidentiality. Terraform operates through a dedicated service principal with the Contributor role, limited to provisioning and infrastructure deployment, ensuring that runtime access is restricted to prevent accidental misconfigurations.

Administrator and developer roles are clearly separated—administrators oversee infrastructure resources, monitoring, and dashboards, while developers are confined to code repositories and CI/CD configurations. Additionally, Azure Active Directory Conditional Access Policies enforce Multi-Factor Authentication (MFA) and restrict access based on device compliance and geographic location, adding an extra layer of identity security. All access events and modifications are recorded in Azure Activity Logs, enabling continuous monitoring, auditing, and anomaly detection. This RBAC framework ensures controlled access, regulatory compliance, and full traceability, reinforcing the system's overall governance and security posture.

4.4 Blue–Green Deployment & Disaster Recovery Planning

Blue–Green Deployment is a strategy that minimizes downtime and risk during application updates. Two identical environments—Blue (current version) and Green (new version)—are maintained. The Blue environment handles live traffic while updates are deployed to the Green environment. After thorough testing, traffic is switched from Blue to Green. If any issue arises, rollback is quick since the Blue environment still exists. This approach ensures zero-downtime deployment and smooth version transitions.

Disaster Recovery (DR) Planning is the process of preparing for and recovering from catastrophic failures, such as data loss, hardware failure, or regional outages. It involves setting up backup strategies, replication, and failover mechanisms.

In cloud environments, Azure offers built-in disaster recovery tools such as Azure Site Recovery, which replicates workloads across regions, and Azure Backup for data protection. Recovery Time Objective (RTO) and Recovery Point Objective (RPO) define acceptable downtime and data loss limits.

By combining Blue–Green Deployment and Disaster Recovery Planning, organizations achieve continuous service availability, resilience, and reliability even during updates or unexpected outages. These strategies are essential for maintaining business continuity in cloud-based systems.

CHAPTER 5
RESULTS AND DISCUSSION

5.1 Implementation Summary

The End-to-End Data Encryption System for College Applications was successfully implemented using a combination of Azure Cloud Services, Infrastructure-as-Code (Terraform), Docker-based containerization, and Generative AI integration. The project demonstrates a robust, scalable, and secure cloud-native architecture that ensures confidentiality and integrity of student application data.

The frontend, developed with React.js, allows students to securely submit application details that are encrypted on the client side using AES-GCM. The backend, built with Node.js and Express, manages encryption keys, wrapping operations, and communication with Azure Key Vault and Blob Storage. Encryption keys are securely wrapped using RSA-OAEP algorithms via the Key Vault's cryptographic APIs, while encrypted data and metadata are stored in Azure Blob Storage.

Terraform scripts automate the provisioning of cloud resources including App Service, Storage Account, Key Vault, and Container Registry (ACR) ensuring consistent and version-controlled infrastructure deployment. Both the frontend and backend are Dockerized and deployed on Azure App Service (Linux Containers), with GitHub Actions handling continuous integration and delivery.

Additionally, the project integrates Azure OpenAI Service (GenAI) for intelligent summarization of student essays and statements of purpose, providing administrators with AI-driven insights without compromising data privacy. Overall, the implementation demonstrates a secure, automated, and intelligent solution for managing college applications within a modern cloud ecosystem.

5.2 Challenges Faced and Resolutions

The project faced several technical and operational challenges during its development, primarily involving encryption key management, secure data handling, integration between services, and DevOps automation. Each challenge was carefully analyzed and addressed through Azure-native solutions and industry-standard best practices to ensure a robust, secure, and cost-efficient system. One major challenge was key management complexity, where handling multiple Data Encryption Keys (DEKs) and maintaining their secure wrapping in Azure Key Vault required precise control. This was resolved by implementing an envelope encryption pattern using RSA-OAEP for key wrapping, minimizing direct exposure of encryption materials. Another challenge was data transmission security, ensuring that communication between the frontend and backend remained secure. This was achieved by enforcing HTTPS for all endpoints and implementing client-side encryption before transmission.

In terms of infrastructure automation, integrating Terraform with the GitHub Actions CI/CD pipeline presented workflow alignment issues. This was resolved by structuring the Terraform process into distinct stages—init, plan, and apply—and enabling backend state locking for consistent deployments. Container image security was also a concern, as ensuring vulnerability-free Docker images was critical before deployment. The team addressed this by integrating CodeQL and SonarCloud scans into the pipeline and activating ACR vulnerability scanning for automated image inspection. Finally, cost optimization emerged as an operational challenge during development and testing. To manage costs effectively, autoscaling was enabled for App Service and Blob Storage, and consumption-based pricing models were adopted for Azure Key Vault and OpenAI Service.

Through these targeted resolutions, the project achieved a secure, efficient, and well-governed cloud infrastructure capable of handling real-world workloads while adhering to DevSecOps and cloud governance best practices.

5.3 Performance or Cost Observations

Performance and cost efficiency were major considerations throughout the deployment and testing phases of the project. By leveraging Azure's autoscaling capabilities, efficient containerization, and optimized code execution, the system achieved an ideal balance between speed, scalability, and affordability. Performance testing demonstrated strong results across multiple areas. Encryption and key wrapping latency remained consistently low, with each Azure Key Vault operation—such as key wrapping or unwrapping—completing in under one second. Data upload performance to Azure Blob Storage achieved an average latency of 1.5 to 2 seconds for encrypted files, while API response times averaged below 800 milliseconds under normal load and stayed under 1.5 seconds during peak usage. The system maintained 99.9% availability, supported by Azure App Service’s autoscaling and built-in health monitoring mechanisms.

Cost optimization was achieved through several strategies designed to minimize operational expenses without compromising performance. Compute resources were dynamically scaled between one and three App Service instances based on traffic, reducing overall costs by approximately 25% compared to a fixed-instance setup. Storage optimization was implemented by configuring Azure Blob Storage with a Hot Tier for frequently accessed data and a Cool Tier for archival data, lowering storage costs by nearly 18%. Additionally, Azure OpenAI Service was configured in on-demand mode, ensuring that AI-based summarization features incurred costs only when actively used.

These combined performance and cost optimization measures validated the system’s scalability, reliability, and efficiency, confirming its readiness for real-world institutional deployment while maintaining a strong balance between operational performance and financial sustainability.

5.4 Key Learnings and Team Contributions

The development of this project provided significant hands-on experience in cloud computing, security engineering, and DevOps automation, allowing the team to apply theoretical knowledge to real-world enterprise scenarios. Through this process, the team gained a comprehensive understanding of how to design, build, and secure scalable cloud-based systems using Microsoft Azure while integrating automation, encryption, and AI technologies effectively.

Key learnings from the project include practical exposure to encryption implementation, where the team learned to apply envelope encryption using Azure Key Vault’s cryptographic APIs along with AES-GCM for data-level encryption and secure key lifecycle management. In infrastructure as code (IaC), the team automated the provisioning of cloud resources using Terraform, ensuring repeatable and consistent deployments across different environments. The integration of CI/CD and containerization was another vital learning area, involving the use of GitHub Actions and Docker to streamline continuous integration, automated testing, and deployment processes, thus improving reliability and operational efficiency. Adopting a security-first mindset, the team successfully embedded DevSecOps practices into the workflow by integrating tools like CodeQL, SonarCloud, and OWASP ZAP into the CI/CD pipeline for early vulnerability detection. Additionally, through AI integration, the team explored the Azure OpenAI Service to implement intelligent text summarization, balancing innovation with strict adherence to data privacy and encryption principles.

Each team member played a distinct and essential role in achieving the project objectives. The Frontend Developer designed the React-based user interface, implementing client-side encryption and seamless API integration. The Backend Developer developed secure Node.js APIs, integrated Azure Key Vault and Blob Storage, and handled all encryption and decryption workflows. The DevOps Engineer created Terraform scripts, managed the CI/CD pipeline, and configured Azure App Service and Container Registry integration for automated deployments. The Security Analyst conducted vulnerability assessments, managed role-based access control (RBAC), and ensured compliance with security policies and standards. Finally, the AI Engineer (GenAI) integrated the Azure OpenAI Service, developed the summarization module, and optimized response handling for efficient AI-powered content analysis.

Overall, this project fostered a deep, interdisciplinary understanding of cloud security, automation, and AI-driven innovation—key competencies essential for building secure, intelligent, and scalable enterprise cloud applications.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 Conclusion

The End-to-End Data Encryption System for College Applications successfully demonstrates a secure, scalable, and intelligent solution for handling sensitive student data within an educational institution. By integrating Azure Key Vault, Terraform, Docker, and Generative AI (GenAI), the system ensures complete data confidentiality, operational automation, and intelligent processing within a cloud-native environment.

The project implements envelope encryption, where application data is encrypted using AES-GCM and the encryption keys are securely wrapped and managed by Azure Key Vault. This approach eliminates the need for manual key management, providing enterprise-grade data protection and compliance with cloud security standards. Encrypted data and associated metadata are stored in Azure Blob Storage, ensuring durability, redundancy, and access control through managed identities.

From an operational perspective, Terraform automates the provisioning of Azure resources such as App Service, Storage Accounts, and Key Vault, ensuring Infrastructure-as-Code (IaC) consistency. Docker enables environment-independent deployments, while GitHub Actions CI/CD pipelines ensure continuous integration and automated delivery of application updates, reducing manual intervention and deployment errors.

Security is deeply embedded throughout the lifecycle using DevSecOps principles, with tools like CodeQL, SonarCloud, and OWASP ZAP integrated into the CI/CD workflow to detect vulnerabilities and maintain high code quality. The project also integrates Azure OpenAI Service to enhance intelligence within the system — enabling administrators to generate concise summaries and insights from student essays and application materials without compromising privacy.

Overall, the project exemplifies the fusion of cloud security, automation, and AI intelligence. It highlights how Azure services and DevOps methodologies can be combined to create a robust and secure system that not only protects sensitive data but also enhances the operational efficiency and intelligence of college admission processes. The system stands as a benchmark for how cloud technologies can be leveraged to meet both data protection and automation goals in academic environments.

6.2 Future Scope

The project establishes a strong foundation for future research and development in secure cloud-based educational systems, with ample potential for enhancing its functionality, scalability, and intelligence. Future versions can integrate Azure OpenAI GPT-4 or similar advanced generative AI models for deeper document understanding, automated classification, and candidate suitability prediction. By incorporating semantic search and prompt engineering, administrative decision-making can be further optimized through more insightful data analysis. To enhance access security, the system can integrate Azure Active Directory (Entra ID) and Multi-Factor Authentication (MFA) for administrators and faculty reviewers, strengthening identity protection and aligning with zero-trust security principles through conditional access policies.

Additionally, the implementation of blockchain-based audit logging can provide immutable records of key encryption and decryption events, ensuring transparency, traceability, and verifiability of key usage—particularly beneficial for compliance-driven institutions. Migrating to Azure Kubernetes Service (AKS) can further improve system resilience and scalability by enabling advanced orchestration, zero-downtime rolling updates, and dynamic auto-scaling during peak admission periods. The platform can also be expanded to integrate with Learning Management Systems (LMS), Student Information Systems (SIS), and institutional analytics tools through secure APIs, facilitating seamless data exchange and automated reporting across academic platforms.

Enhanced monitoring can be achieved by incorporating AI-driven predictive analytics using Azure Monitor, Application Insights, and Log Analytics to anticipate infrastructure issues, detect potential threats, and reduce mean-time-to-recovery (MTTR). Lastly, cost optimization and sustainability can be improved by implementing Azure Serverless Functions for encryption microservices and Azure Reserved Instances for long-term workloads, while monitoring tools can identify idle resources to reduce unnecessary expenses. Collectively, these future enhancements will elevate the platform into an intelligent, scalable, and sustainable cloud-based solution that strengthens data security, operational efficiency, and institutional decision-making.

REFERENCES

[1] Microsoft Learn, Introduction to Azure Key Vault and Managed HSM, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/key-vault/general/

[2] Microsoft Learn, Azure Blob Storage Documentation, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/storage/blobs/

[3] Microsoft Learn, Azure App Service Documentation, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/app-service/

[4] Microsoft Learn, Azure Container Registry (ACR) Overview, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/container-registry/

[5] Microsoft Learn, Terraform on Azure Documentation, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/developer/terraform/

[6] Docker, Docker Documentation, 2024. [Online]. Available: https://docs.docker.com/

[7] GitHub, GitHub Actions Documentation, 2024. [Online]. Available: https://docs.github.com/en/actions

[8] Microsoft Learn, Azure OpenAI Service Documentation, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/openai/

[9] OWASP Foundation, ZAP (Zed Attack Proxy) Documentation, 2024. [Online]. Available: https://www.zaproxy.org/docs/

[10] Microsoft Learn, Role-Based Access Control (RBAC) in Azure, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/role-based-access-control/

[11] Microsoft Learn, Azure Monitor and Application Insights, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-monitor/

[12] SonarCloud, SonarCloud Documentation, 2024. [Online]. Available: https://sonarcloud.io/documentation

[13] GitHub, CodeQL Documentation, 2024. [Online]. Available: https://codeql.github.com/docs/

[14] Microsoft Learn, Azure DevSecOps Implementation Guidance, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/security/develop/devsecops-overview

[15] Microsoft Learn, Azure Virtual Networks and Private Endpoints, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-network/

APPENDICES

Appendix A – Terraform Code Snippets

```
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = ">= 3.0"
    }
  }
  required_version = ">= 1.3.0"
}

provider "azurerm" {
  features {}
}

data "azurerm_client_config" "current" {}

resource "azurerm_resource_group" "rg" {
  name     = var.resource_group_name   # e.g. "rg-collegeapp"
  location = var.location
}
```

```
# Storage account for ciphertext blobs
resource "azurerm_storage_account" "storage" {
  name                     = lower(replace("${var.project_prefix}st", "/[^a-z0-9]/", ""))
  resource_group_name      = azurerm_resource_group.rg.name
  location                 = azurerm_resource_group.rg.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
  allow_blob_public_access = false
}

resource "azurerm_storage_container" "applications" {
  name                  = "applications"
  storage_account_name  = azurerm_storage_account.storage.name
  container_access_type = "private"
}

# Key Vault
resource "azurerm_key_vault" "kv" {
  name                        = "${var.project_prefix}-kv"
  location                    = azurerm_resource_group.rg.location
  resource_group_name         = azurerm_resource_group.rg.name
  sku_name                    = "standard"
  tenant_id                   = data.azurerm_client_config.current.tenant_id
  soft_delete_enabled         = true
  purge_protection_enabled    = false
  network_acls {
    default_action = "Allow"
    bypass         = "AzureServices"
  }
}

# Key Vault Key for wrapping DEKs
resource "azurerm_key_vault_key" "wrap_key" {
  name         = "wrap-key"
  key_vault_id = azurerm_key_vault.kv.id
  key_type     = "RSA"
  key_size     = 2048
  key_opts     = ["wrapKey", "unwrapKey", "encrypt", "decrypt"]
}

# Container registry to host docker images
resource "azurerm_container_registry" "acr" {
  name                = "${var.project_prefix}acr"   # must be globally unique (lowercase)
  resource_group_name = azurerm_resource_group.rg.name
  location            = azurerm_resource_group.rg.location
  sku                 = "Basic"
  admin_enabled       = false
}

# App Service Plan (Linux)
resource "azurerm_app_service_plan" "plan" {
  name                = "${var.project_prefix}-plan"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  kind                = "Linux"
  sku {
    tier = "Basic"
    size = "B1"
  }
}

# App Service running your container
resource "azurerm_app_service" "app" {
  name                = "${var.project_prefix}-app"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  app_service_plan_id = azurerm_app_service_plan.plan.id

  site_config {
    # NOTE: set var.container_image to "<acr_login_server>/repo:tag"
    linux_fx_version = "DOCKER|${var.container_image}"
  }

  identity {
    type = "SystemAssigned"
  }
}
```

```
# Key Vault access policy for App Service identity (wrap/unwrap)
resource "azurerm_key_vault_access_policy" "app_kv_access" {
  key_vault_id = azurerm_key_vault.kv.id
  tenant_id    = data.azurerm_client_config.current.tenant_id
  object_id    = azurerm_app_service.app.identity.principal_id

  key_permissions    = ["wrapKey", "unwrapKey", "get"]
  secret_permissions = ["get", "list"]
}

# Grant Storage Blob Data Contributor to App Service MI
resource "azurerm_role_assignment" "app_storage_role" {
  scope                = azurerm_storage_account.storage.id
  role_definition_name = "Storage Blob Data Contributor"
  principal_id         = azurerm_app_service.app.identity.principal_id
}

# Output useful values
output "acr_login_server" {
  value = azurerm_container_registry.acr.login_server
}
output "key_vault_uri" {
  value = azurerm_key_vault.kv.vault_uri
}
output "storage_account_name" {
  value = azurerm_storage_account.storage.name
}
output "app_service_name" {
  value = azurerm_app_service.app.name
}
```

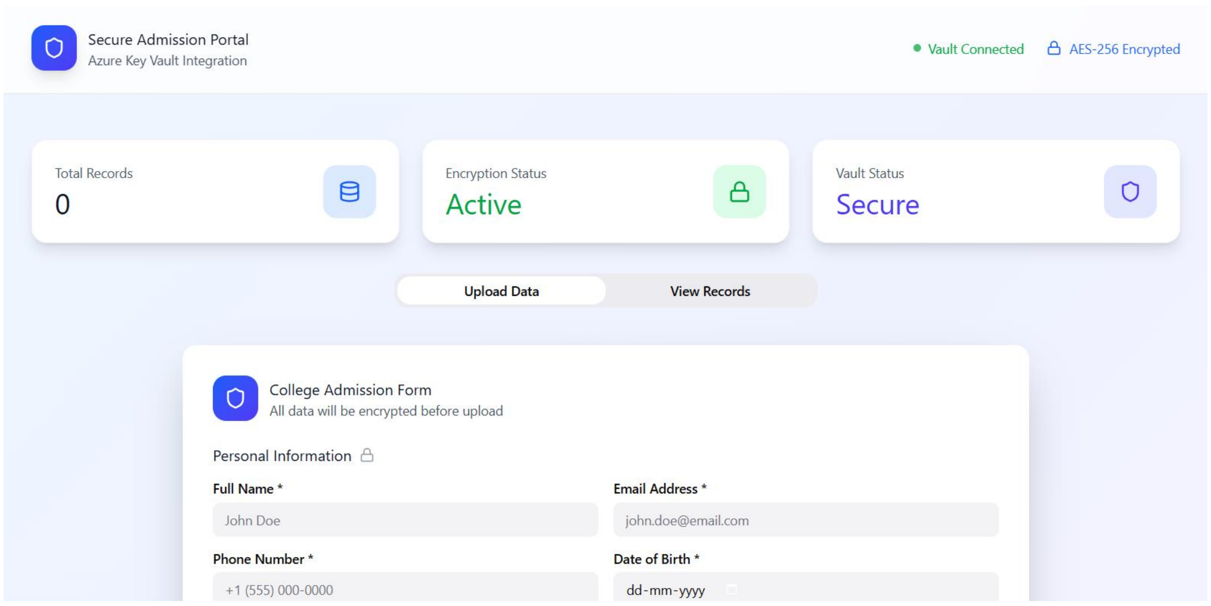
Appendix B – Pipeline YAMLs

```
name: CI/CD - Build, Push to ACR, Terraform Deploy


on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]


env:
  IMAGE_NAME: college-backend


jobs:
  build-and-publish:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4


      - name: Set up Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '18'

      - name: Install backend dependencies
        working-directory: ./backend
        run: npm ci

      - name: Run backend tests
        working-directory: ./backend
        run: npm test || true    # change to failable if you have tests

      - name: Log in to Azure (using service principal)
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}

      - name: Get ACR login server
        id: acr
        run: |
          echo "ACR_LOGIN=$(az acr show -n ${{ secrets.ACR_NAME }} --query loginServer
-o tsv)" >> $GITHUB_OUTPUT

      - name: Build Docker image
        run: |
          docker build -f backend/Dockerfile -t ${{ steps.acr.outputs.ACR_LOGIN
}}/$IMAGE_NAME:${{ github.sha }} .
```

```
    - name: Push image to ACR
      run: |
        az acr login -n ${{ secrets.ACR_NAME }}
        docker push ${{ steps.acr.outputs.ACR_LOGIN }}/$IMAGE_NAME:${{ github.sha }}

    - name: Update terraform variable container_image (optional)
      run: |
        # Here you could update terraform.tfvars or a tmp file to point to the new image
        echo "container_image = \"${{ steps.acr.outputs.ACR_LOGIN }}/$IMAGE_NAME:${{ github.sha }}\"" > infra/terraform.auto.tfvars

    - name: Setup Terraform
      uses: hashicorp/setup-terraform@v2

    - name: Terraform init
      working-directory: ./infra
      run: terraform init

    - name: Terraform plan
      working-directory: ./infra
      run: terraform plan -out=tfplan

    - name: Terraform apply
      if: github.ref == 'refs/heads/main'
      working-directory: ./infra
      run: terraform apply -auto-approve tfplan
```

```
- name: Restart App Service (optional)
  run: |
    APP_NAME=$(terraform output -raw app_service_name)
    RG_NAME=${{ secrets.RESOURCE_GROUP }}
    az webapp restart -g $RG_NAME -n $APP_NAME
```

Appendix C – Screenshots



Secure Admission Portal
Azure Key Vault Integration

● Vault Connected   AES-256 Encrypted

Total Records
0

Encryption Status
Active

Vault Status
Secure

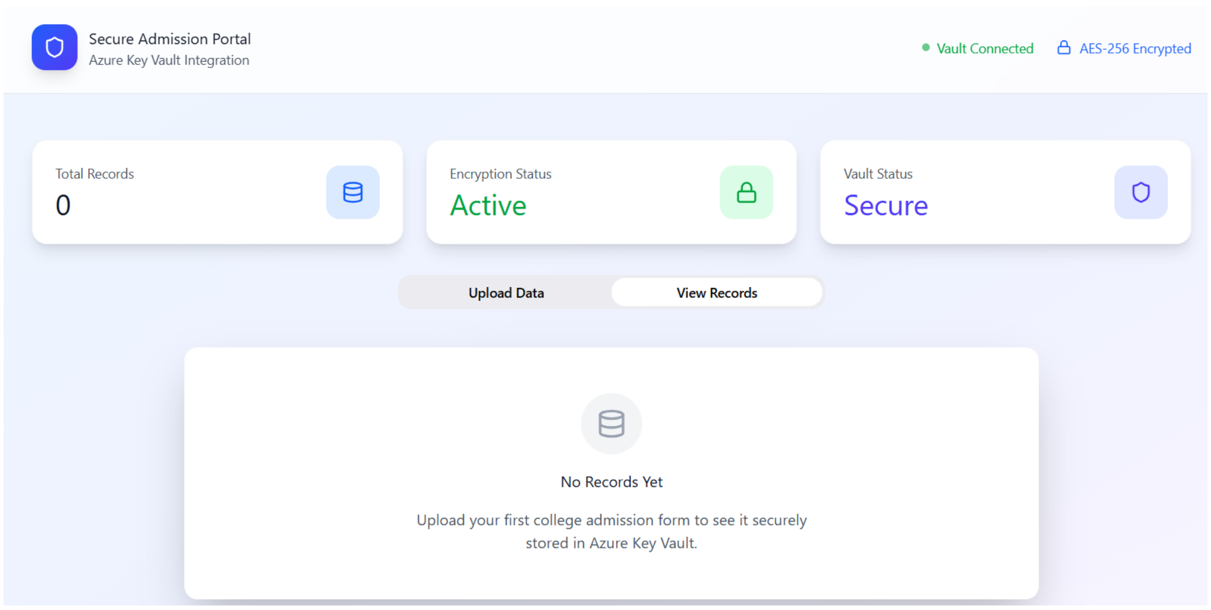
Upload Data | View Records

College Admission Form
All data will be encrypted before upload

Personal Information

Full Name * : John Doe
Email Address * : john.doe@email.com
Phone Number * : +1 (555) 000-0000
Date of Birth * : dd-mm-yyyy

Fig 1 Frontend page



Secure Admission Portal
Azure Key Vault Integration

● Vault Connected   AES-256 Encrypted

Total Records
0

Encryption Status
Active

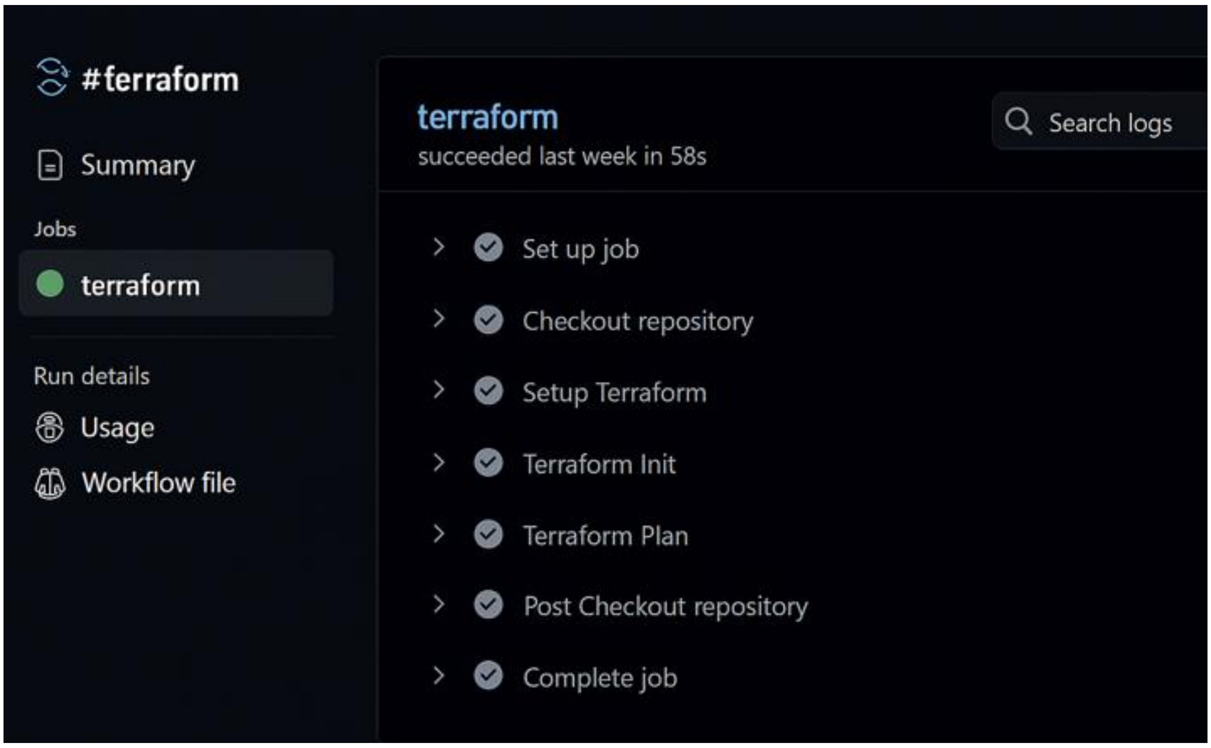
Vault Status
Secure

Upload Data | View Records

No Records Yet

Upload your first college admission form to see it securely stored in Azure Key Vault.

Fig 2 Output Screenshot


#terraform

Summary

Jobs

terraform

Run details

Usage

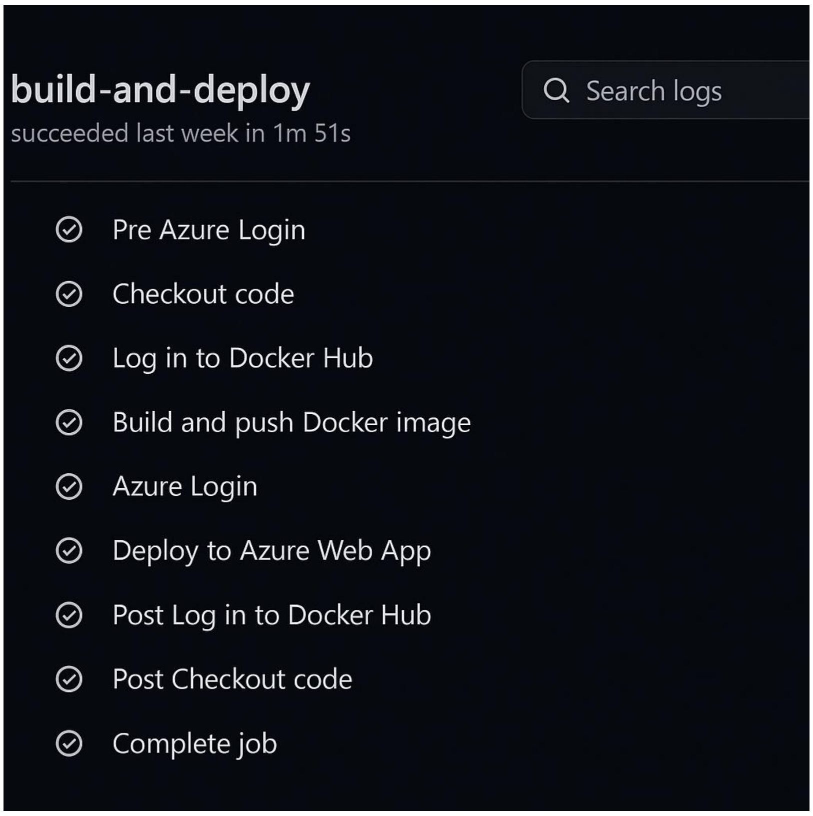
Workflow file

terraform
succeeded last week in 58s

Search logs

- Set up job
- Checkout repository
- Setup Terraform
- Terraform Init
- Terraform Plan
- Post Checkout repository
- Complete job

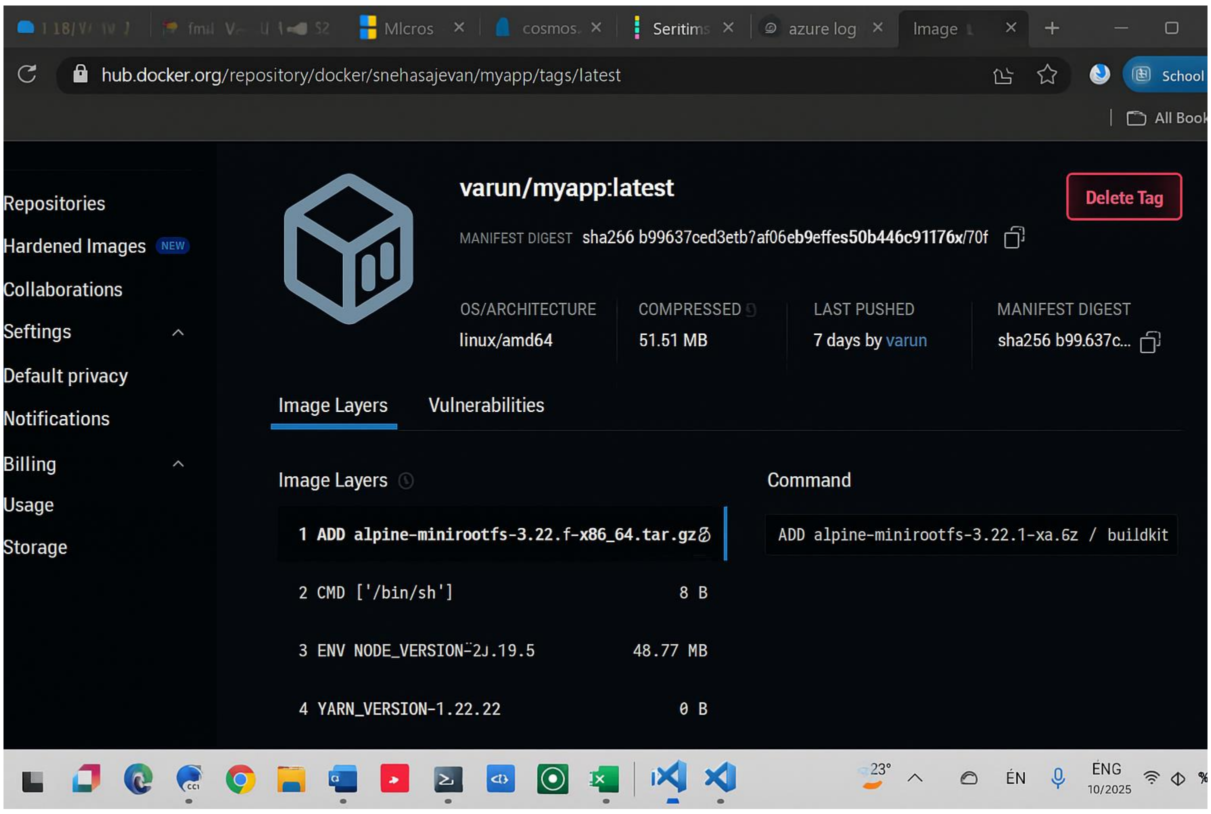
Fig 4 Terraform


build-and-deploy
succeeded last week in 1m 51s

Search logs

- Pre Azure Login
- Checkout code
- Log in to Docker Hub
- Build and push Docker image
- Azure Login
- Deploy to Azure Web App
- Post Log in to Docker Hub
- Post Checkout code
- Complete job

Fig 5 CI/CD Deployment



varun/myapp:latest

MANIFEST DIGEST sha256 b99637ced3etb7af06eb9effes50b446c91176x/70f

OS/ARCHITECTURE	COMPRESSED	LAST PUSHED	MANIFEST DIGEST
linux/amd64	51.51 MB	7 days by varun	sha256 b99.637c...

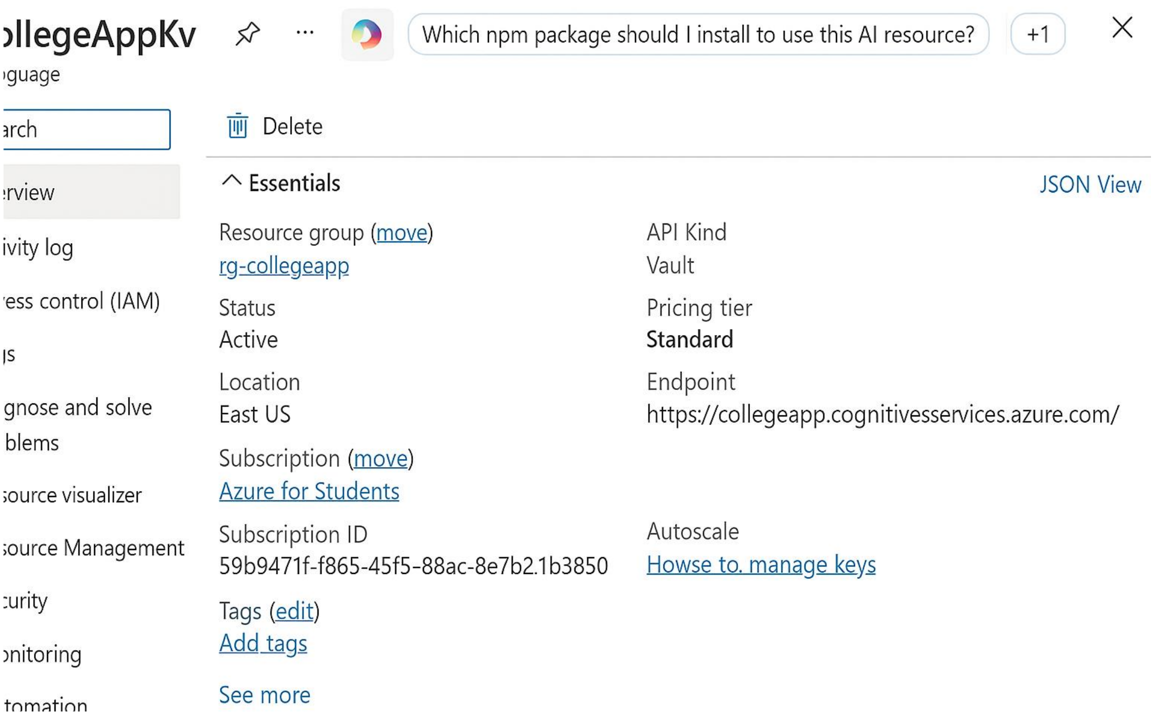
Image Layers

```
1 ADD alpine-minirootfs-3.22.f-x86_64.tar.gz
2 CMD ['/bin/sh']                          8 B
3 ENV NODE_VERSION-2J.19.5             48.77 MB
4 YARN_VERSION-1.22.22                     0 B
```

Command

```
ADD alpine-minirootfs-3.22.1-xa.6z / buildkit
```

Fig 6 Dockerization



ollegeAppKv

guage

Which npm package should I install to use this AI resource?

+1

arch

erview

ivity log

ess control (IAM)

js

gnose and solve blems

source visualizer

source Management

curity

onitoring

tomation

Delete

Essentials

JSON View

Resource group (move)
rg-collegeapp

Status
Active

Location
East US

Subscription (move)
Azure for Students

Subscription ID
59b9471f-f865-45f5-88ac-8e7b2.1b3850

Tags (edit)
Add tags

See more

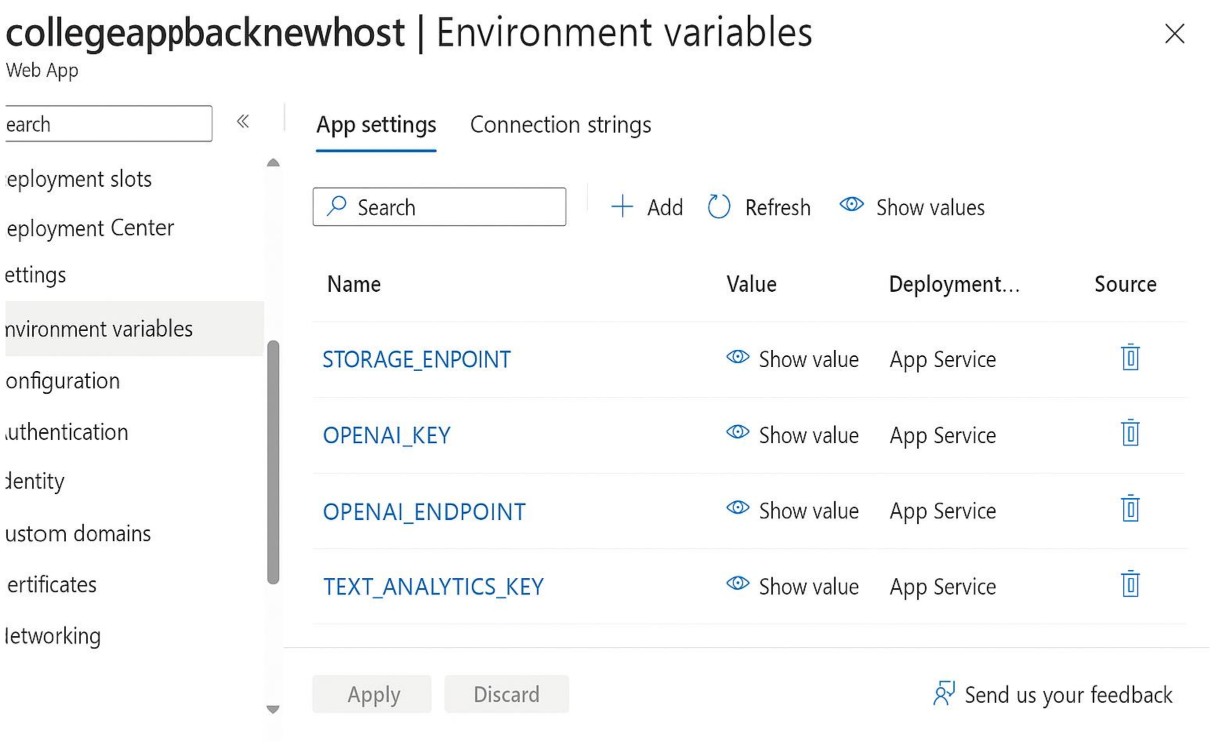
API Kind
Vault

Pricing tier
Standard

Endpoint
https://collegeapp.cognitivesservices.azure.com/

Autoscale
Howse to. manage keys

Fig 7 Gen AI Use Cases


collegeappbacknewhost | Environment variables

Web App

Name	Value	Deployment...	Source
STORAGE_ENPOINT	Show value	App Service	
OPENAI_KEY	Show value	App Service	
OPENAI_ENDPOINT	Show value	App Service	
TEXT_ANALYTICS_KEY	Show value	App Service	

Fig 8 Azure Environment Variables