## **RAJALAKSHMI ENGINEERING COLLEGE**

## **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

Project Report

On

ADMINISTRATIVE CERTIFICATE REQUEST

PROCESSING SYSTEM

Submitted by

AAADHITHYA RK 220701002

BHARATH KUMAR L 220701043

ADHESH M 220701012

Under the Guidance of

NAME

Course:Cloud Computing(B.E.CSE)

Date Of Submission:

**CERTIFICATE**

This is to certify that the project entitled “ADMINISTRATIVE CERTIFICATE REQUEST PROCESSING SYSTEM” submitted by AADHITHYA RK (220701002),BHARATH KUMAR L (220701043),ADHESH M (220701012) 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:

**ACKNOWLEDEMENT**

We would like to express our sincere gratitude to our faculty guide, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, for their invaluable guidance, support, and encouragement throughout the course of this project. We also extend our heartfelt thanks to the Head of the Department and the management of Rajalakshmi Engineering College for providing us with the opportunity, resources, and a conducive environment to successfully complete this work.

**ABSTRACT**

The Event Management System is a complete-stack internet software designed to streamline the entire occasion lifecycle, from introduction to execution. Built with React 18 on the frontend and Node.Js with Express on the backend, it offers an intuitive consumer interface and a stable, scalable REST API. The platform supports event scheduling, price tag income with QR code generation, person authentication thru JWT, and complete account management. Hosted on Microsoft Azure, it leverages PremiumV2 App Service, Geo-Redundant Blob Storage, and containerization for reliable and scalable deployment. Security is a core cognizance, utilising bcrypt for password hashing, Helmet for HTTP headers, and charge limiting to guard APIs. The device additionally integrates infrastructure as code thru Terraform, making sure automated, constant provisioning. Designed for extensibility, the challenge plans to contain capabilities such as real-time event updates, extra fee gateways, and an occasion analytics dashboard. This answer targets to offer occasion organizers and attendees with a continuing, stable, and efficient platform for dealing with occasions in a cloud-native environment.

**TABLE OF CONTENT**

**CHAPTER NO**   **TITLE**   **PAGE NO**

**ABSTRACT 3**

**LIST OF FIGURES 7**

**LIST OF ABBREVATIONS 8**

**1. INTRODUCTION 9**

1.1 PROBLEM STATEMENT 9

1.2 OBJECTIVE 9

1.3 SCOPE AND BOUNDARIES 10

1.4 STAKEHOLDERS AND END USERS 11

1.5 TECHNOLOGIES USED 12

**2. SYSTEM DESIGN AND ARCHITECTURE 13**

2.1 REQUIREMENT SUMMARY 13

2.2 PROPOSED SOLUTION OVERVIEW 14

2.3 CLOUD DEPLOYMENT STRATEGY 15

2.4 INFRASTRUCTURE REQUIREMENTS 16

2.5 AZURE SERVICES AND MAPPING 17

**3. DEVOPS IMPLEMENTATION 18**

3.1-CI/CD SETUP 18

3.2-TERRAFORM INFRASTRUCTURE

-AS-CODE (IaC) 19

3.3-CONTAINERIZATION STRATEGY 20

3.5-GEN-AI INTEGRATION AND

AZURE AI SERVICE MAPPING 21

**4. CLOUD OPERATIONS AND SECURITY 23**

4.1-DEVSECOPS INTEGRATION 23

4.2-MONITORING AND OBSERVABILITY 24

4.3-ACCESS CONTROL 25

4.4 BLUE–GREEN DEPLOYMENT

& DISASTER RECOVERY PLANNING 25

**5. RESULTS AND DISCUSSIONS 27**

5.1-IMPLEMENTATION 27

5.2-CHALLENGES FACED AND

RESOLUTION 27

5.3-PERFORMANCE OR COST

OBSERVATION 28

5.4-KEY LEARNINGS AND

TEAM CONTRIBUTIONS 29

**6. CONCLUSION AND FUTURE**

**ENHANCEMENT 31**

6.1 CONCLUSION 31

6.2 FUTURE WORK 32

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO.** | **FIGURE NAME** | **PAGE NO**. |
| 1.1 | Terraform code snippet- main.tf | 36 |
| 1.2 | Project File Structure | 37 |
| 1.3 | Build and deploy to Azure web apps | 37 |
| 1.4 | Deployed Workflow | 40 |
| 1.5 | Resource group and resource creation |  |
| 1.6 | Document intelligence deployment | 43 |
| 1.7 | Letter generation | 43 |
| 1.8 | Security scans | 46 |
|  |  |  |

**LIST OF ABBREVATIONS**

|  |  |
| --- | --- |
| **Abbreviation** | **Full Form** |
| AI | Artificial Intelligence |
| OCR | Optical Character Recognition |
| PaaS | Platform as a Service |
| IaC | Infrastructure as Code |
| CI/CD | Continuous Integration / Continuous Deployment |
| ACR | Azure Container Registry |
| API | Application Programming Interface |
| SQL | Structured Query Language |
| VNet | Virtual Network |
| VM | Virtual Machine |
|  |  |
|  |  |
|  |  |
|  |  |

**CHAPTER 1**

**INTRODUCTION**

**1.1 PROBLEM STATEMENT**

Event organizers often face extensive challenges in making plans, dealing with, and executing occasions efficiently. These challenges consist of dealing with more than one duties which includes occasion scheduling, price ticket sales, attendee control, and fee processing, all whilst making sure statistics security and imparting a smooth consumer experience. Traditional event control systems may additionally lack integration, scalability, or actual-time capabilities, leading to inefficiencies, guide errors, and poor attendee engagement. Moreover, with the increasing shift closer to virtual solutions, there is a developing want for a complete platform that supports the whole lot from event introduction to analytics within a stable and scalable cloud surroundings. Event organizers require intuitive gear to streamline workflows, manipulate consumer authentication, and shield sensitive statistics inclusive of payment facts. On the attendee facet, problems get up in buying tickets, verifying access correctly, and acquiring timely occasion updates. Current structures frequently fail to provide seamless cell-pleasant interfaces, QR code integration, or dependable fee techniques, ensuing in a fragmented enjoy. The hassle is exacerbated by way of the dearth of sturdy infrastructure which can accommodate rising person loads, ensure high availability, and defend towards cybersecurity threats. Therefore, there is a critical need for a modern-day, cease-to-cease occasion control solution that combines a effective frontend, a steady backend, and cloud-local infrastructure, permitting organizers and attendees to interact effects, securely, and at scale.

**1.2 OBJECTIVE**

The primary objective of the Event Management System is to develop a robust, user-friendly, and scalable web application that simplifies the entire event lifecycle—from creation and scheduling to ticket sales and post-event analytics. This platform aims to provide event organizers with comprehensive tools to efficiently manage event details, attendee registrations, and payment processing in a secure environment. By integrating features like QR code ticketing, responsive user interfaces, and real-time data handling, the system seeks to enhance the overall experience for both organizers and attendees.

Another key objective is to ensure data security and privacy through strong authentication methods and best practices such as password hashing, JWT-based session management, and API rate limiting. Hosting the application on Microsoft Azure with containerized deployment and managed infrastructure enhances availability, scalability, and fault tolerance, enabling the system to handle varying workloads without compromising performance.

Additionally, the project aims to implement infrastructure-as-code practices using Terraform to automate and standardize environment provisioning, thereby supporting efficient development and deployment workflows. Future plans include adding real-time event updates, expanding payment gateway options, and developing an event analytics dashboard to offer valuable insights. Ultimately, the system strives to be an all-inclusive solution that meets the evolving needs of event management in a modern, cloud-native ecosystem.

**1.3 SCOPE AND BOUNDARIES**

The scope of the Event Management System encompasses the improvement of a complete-stack net software that permits customers to create, control, and execute events efficiently. It includes functionalities together with occasion scheduling, ticket income with QR code generation, user authentication, fee processing, and attendee control. The device will provide a responsive and intuitive user interface, in conjunction with a stable backend API, hosted on Microsoft Azure for scalability and high availability. The platform will support analytics and reporting capabilities to assist organizers song event overall performance and person engagement.

The obstacles of the challenge exclude in-individual event logistics consisting of physical venue setup, catering, or real-time onsite event management offerings. The machine will no longer directly handle advertising or promotion past simple event sharing functionalities. While fee processing is supported, integration is restrained initially to a select payment gateway, with plans for destiny enlargement. The platform does now not include advanced AI-driven pointers or deep integration with 1/3-birthday party calendar apps at this stage. Real-time event update capabilities and recurring occasion help are planned however now not protected inside the initial launch.

**1.4 STAKEHOLDERS AND END USERS**

Stakeholders:

* Event organizers who require an efficient platform to create, manage, and track their events.
* Business owners and promoters looking to streamline ticket sales and payment processing securely.

.

* Cloud infrastructure and DevOps team managing deployment, security, and availability on Azure.

End Users:

* Event Attendees: Individuals who browse events, purchase tickets, and attend events using the platform. They benefit from features like easy ticket purchasing, QR code entry, event details access, and secure payment processing.
* Event Organizers: Users responsible for creating and managing events, setting ticket prices, scheduling, and monitoring attendee participation. They use the system to streamline event operations, track sales and income, and engage with participants effectively.

**1.5 TECHNOLOGIES USED**

* Microsoft Azure: Provides the cloud infrastructure for deploying and hosting the web application, including Azure App Services, Azure SQL Database, and Azure Monitor for performance tracking.
* Terraform: Used for Infrastructure as Code (IaC) to automate provisioning of Azure resources, ensuring consistency and easy scalability.
* Docker: Containerization tool used to package the application and its dependencies into portable containers for reliable deployment.
* GitHub Actions: Implements Continuous Integration and Continuous Deployment (CI/CD) pipelines to automate the build, test, and deployment processes.
* Azure Container Registry (ACR): Stores and manages Docker images securely before deployment.
* Node.js / Express / React (if used): Backend and frontend technologies used to develop the web portal for both student and administrator modules.
* nginx: Nginx serves as a effective web server and reverse proxy that successfully handles incoming net traffic, dispensing requests to backend servers while enhancing performance and scalability.

**CHAPTER 2**

**SYSTEM DESIGN AND ARCHITECTURE**

**2.1 REQUIREMENT SUMMARY**

**FUNCTIONAL REQUIREMENTS:**

* User Authentication and Authorization: The system shall provide secure user registration, login, and JWT-based authentication with role-based access control.
* User Authentication and Authorization: The system shall provide secure user registration, login, and JWT-based authentication with role-based access control.
* Ticketing System: Users shall be able to purchase tickets, receive QR codes for entry, and manage their bookings within the system.
* Payment Processing: The system shall securely handle payment transactions integrated through supported payment gateways.
* Calendar and Scheduling: The system shall display events in a calendar view with filtering and searching capabilities.
* Notifications: Users shall receive alerts and notifications regarding event updates, cancellations, or reminders.
* Analytics and Reporting: Organizers shall have access to event performance metrics such as ticket sales, attendance, income, and user engagement.

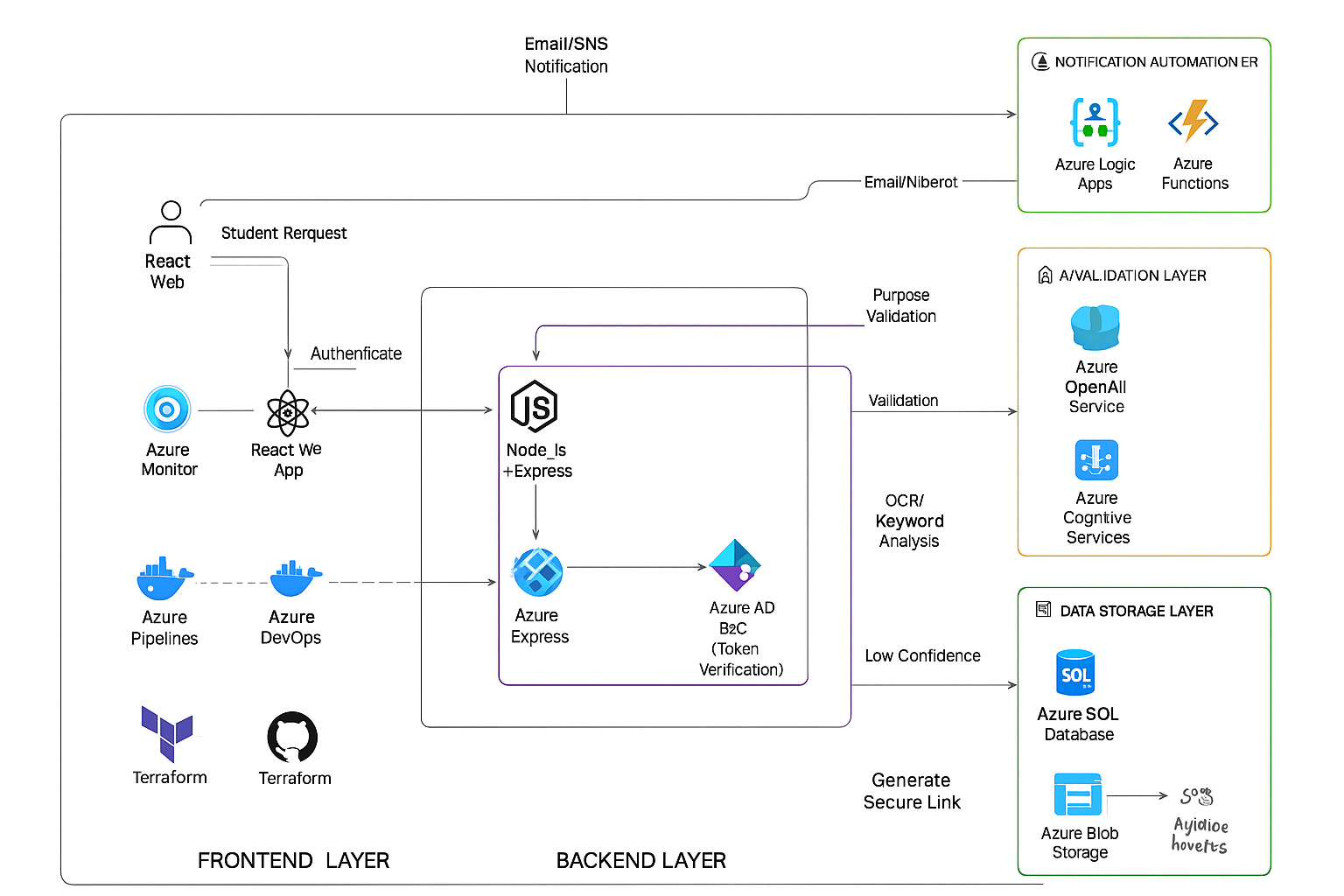
**NON FUNCTIONAL REQUIREMENTS:**

* Scalability:The system should scale dynamically to handle multiple concurrent requests without performance degradation.
* Availability:Ensure 99.9% uptime through Azure cloud hosting and fault-tolerant architecture.
* Security: The system shall implement password hashing, rate limiting, HTTPS, and secure API endpoints to protect user data and prevent attacks.
* Performance: The system shall deliver low latency responses and quick loading times through optimized backend and frontend design.
* Reliability:Use Azure monitoring and backup mechanisms to ensure data integrity and fault recovery.

**2.2 PROPOSED SOLUTION OVERVIEW**

The proposed Event Management System solution is a cloud-based, full-stack web application designed to streamline the entire event lifecycle. It offers event organizers an intuitive front-end built with React 18 and Tailwind CSS, combined with a secure, scalable backend that uses Node.js, Express and MongoDB for data persistence. The system provides comprehensive functions such as event creation, ticket management with QR code generation, user authentication and payment processing. Leveraging Microsoft Azure infrastructure, including Azure App Service and Blob Storage, ensures high availability, scalability and secure data management. The solution also integrates containerization with Docker and infrastructure-as-code using Terraform for automated and efficient deployment. Security is enhanced with JWT authentication, API rate limiting, and encrypted user credentials. The system's modular architecture supports future enhancements such as real-time event updates and advanced analytics, aiming to deliver a seamless, secure and feature-rich platform to enhance event organization and participation experiences in a cloud-based environment.

[**2.3 CLOUD DEPLOYMENT STRATERGY**

The cloud deployment method for the Event Management System focuses on leveraging Microsoft Azure's robust cloud services to make sure scalability, safety, and excessive availability. The utility is deployed the use of Azure App Service in the Premium V2 tier, which supports elastic scaling with more than one times to deal with various workloads efficiently. Containerization with Docker guarantees consistent, transportable deployments throughout environments, with the Azure Container Registry handling Docker photographs. Persistent facts garage makes use of Azure Blob Storage with Geo-Redundant Storage (GRS) for fault tolerance and disaster recovery. Infrastructure provisioning and management are automated thru Terraform as Infrastructure as Code (IaC), permitting standardized, repeatable deployments and easy environment replication. Additionally, NGINX serves as a reverse proxy to optimize site visitors routing and enhance safety. This method supports non-stop integration and non-stop deployment (CI/CD) pipelines as a destiny enhancement, facilitating automatic builds and streamlined updates.

**Architecture diagram**

**2.4 INFRASTRUCTURE REQUIREMENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Azure Resource** | **Configuration / Size** | **Purpose** |
| App Hosting | Azure App Service | B1/S1 Plan | Hosts backend and frontend containers. |
| Database | Azure SQL Database | Basic (2 vCores, 5 GB) | Stores user and certificate data. |
| File Storage | Azure Blob Storage | Hot Tier | Saves uploaded files and certificates. |
| AI Service | Azure Cognitive Services (OCR) | S0 Tier | Extracts and validates text. |
| Monitoring | Application Insights / Azure Monitor | Default | Tracks performance and logs. |
| Container Registry | Azure Container Registry (ACR) | Basic | Stores Docker images securely. |
| Network | Azure Virtual Network (VNet) | Default | Provides internal communication. |
| CI/CD | GitHub Actions | – | Automates build and deployment. |
| Provisioning | Terraform | – | Creates Azure resources via IaC. |

**2.5 – AZURE SERVICES AND MAPPING**

|  |  |  |
| --- | --- | --- |
| **Service** | **Role in System** | **Justification** |
| Azure App Service | Hosts the containerized web app. | Provides scalable and managed hosting. |
| Azure SQL Database | Stores users and requests data. | Reliable, secure, and auto-scalable database. |
| Azure Blob Storage | Saves uploaded files and certificates. | Low-cost and durable object storage. |
| Azure Container Registry (ACR) | Stores Docker images. | Centralized and secure image repository. |
| Azure Cognitive Services (OCR) | Extracts and verifies text from documents. | Simplifies document validation using AI. |
| Application Insights / Azure Monitor | Tracks performance and logs. | Offers real-time monitoring and analytics. |
| GitHub Actions (CI/CD) | Automates build and deployment. | Enables efficient DevOps automation. |
| Terraform | Provisions cloud infrastructure. | Provides repeatable and version-controlled setup. |

**CHAPTER 3**

**DEVOPS IMPLEMENTATION**

**3.1 CONTINUOUS INTEGRATION AND DEPLOYMENT (CI/CD) SETUP**

The CI/CD (Continuous Integration/Continuous Deployment) setup for the Event Management System is designed to automate code integration, checking out, and deployment techniques to make certain fast and reliable software program releases. The improvement workflow begins with builders pushing code modifications to a model manipulate gadget like Git. Automated construct pipelines then cause, jogging unit and integration checks to validate code nice. Docker is used to containerize the application components, creating regular and portable build artifacts. Terraform scripts automate infrastructure provisioning in Azure, making sure environments are reproducible and configured efficiently. Once builds skip all validations, computerized deployment pipelines push the containerized software to Azure App Service and update associated sources in Azure Container Registry and Blob Storage. This CI/CD pipeline complements collaboration, reduces manual mistakes, and facilitates faster characteristic delivery and trojan horse fixes, helping a DevOps lifestyle for non-stop development and operational excellence. Future enhancements may also consist of integration of tracking and rollback mechanisms for enhanced reliability.

**3.2 TERRAFORM INFRASTRUCTURE-AS-CODE (IAC)**

Terraform Infrastructure-as-Code (IaC) is a powerful tool for defining, provisioning, and managing cloud infrastructure using declarative configuration files. With Terraform, infrastructure resources such as virtual machines, storage accounts, network components, and container services are described in code, enabling automated, repeatable deployments and lifecycle management. This approach eliminates manual configuration errors, ensures consistency across development, test, and production environments, and allows version control of infrastructure changes.

For the event management system deployed on Azure, Terraform manages all necessary resources, including Azure App Service, Blob Storage, Container Registry, and network configurations. The workflow typically involves writing `.tf` files to declare the desired infrastructure, initializing Terraform to configure the environment, running plan commands to preview changes, and applying changes to provisioning or updating the cloud environment. Terraform state files track the deployed resources to detect drifts and enable incremental updates or secure rollbacks. In addition, Terraform scripts automate environment configuration for DevOps, and support CI/CD pipelines and infrastructure scalability. Integrating Terraform ensures that the entire Azure infrastructure is codified, maintainable and easily reproducible for seamless deployment and operation.

**3.3 CONTAINERIZATION STRATEGY (DOCKER)**

The containerization strategy for the Event Management System employs Docker to package the application and its dependencies into lightweight, portable containers. This approach ensures consistent runtime environments across development, testing, and production, eliminating the “works on my machine” problem. Each core component – ​​including the front-end React application, back-end Node.js API, and supporting services – is individually containerized, enabling modular development and deployment.

Docker images are built through automated pipelines, promoting repeatability and versioning of application builds. Containers are deployed to Azure App Service and managed through Azure Container Registry, facilitating seamless upgrades and rollbacks. Containerization improves scalability by allowing rapid instantiation of multiple instances to handle larger loads. It also improves resource utilization and simplifies environment configuration, streamlining the deployment workflow alongside Terraform-managed infrastructure. Overall, Docker-based containerization supports a modern DevOps process, enabling agile, reliable, and scalable application delivery in cloud-native environments.

**3.5-GENAI INTEGRATION AND AZURE AI SERVICE MAPPING**

Generative AI integration in Event Management System can leverage the Azure AI ecosystem to improve user interaction, automate tasks, and improve event data insights. Azure AI services like Azure OpenAI provide access to large language models, enabling conversational agents, intelligent chatbots, and content generation tailored to event contexts. Azure AI Foundry supports the deployment, orchestration, and integration of custom models, enabling the system to incorporate domain-specific knowledge and proprietary data for contextual responses. Retrieval Augmented Generation (RAG) can be used with Azure AI Search to combine generative models with business content, improving the accuracy and relevance of information in user queries. The Azure AI gateway in API Management helps secure and govern AI APIs, ensuring controlled, scalable access to AI-powered resources. Additionally, Azure Cognitive Services contributes capabilities such as speech recognition, language understanding, and image analysis that can enrich the user experience in event logging, support, and analysis. This integration enables the building of intelligent, responsive, and scalable AI-based features into the events platform, offering greater automation and greater user engagement.

**CHAPTER 4**

**CLOUD OPERATIONS AND SECURITY**

**4.1 DEVSECOPS INTEGRATION (CODEQL / SONARCLOUD / ZAP SCANS)**

DevSecOps integration for event management systems includes automated security testing and vulnerability assessment within the CI/CD pipeline to ensure secure software delivery. Tools like CodeQL provide static code analysis to find security flaws and apply coding best practices early in development. SonarCloud complements this by offering continuous code quality inspection, identifying bugs, code smells and security vulnerabilities in multiple languages. OWASP ZAP (Zed Attack Proxy) scans running applications for common security threats such as injection flaws and cross-site scripting in an automated fashion during pre-production or staging phases.

Within Azure DevOps or GitHub actions pipelines, these tools can be integrated to automatically scan source code and deployed artifacts, applying quality gates before merge or deployment. This shift-left approach helps developers identify and fix security issues early, reducing remediation costs and increasing application resilience. In addition, DevSecOps practices extend to infrastructure and container security, including IaC scanning and container vulnerability analysis to protect the entire stack. Overall, this integration fosters a culture of shared responsibility across development, security, and operations teams, ensuring that security is consistently embedded from code commit to product release.

**4.2 MONITORING AND OBSERVABILITY (AZURE MONITOR / APP INSIGHTS)**

Monitoring and observability are enhanced in event management systems using Azure Monitor and Application Insights. Azure Monitor collects, analyzes and acts on telemetry data from applications and cloud infrastructure, providing a comprehensive view of system performance and health. It tracks metrics such as CPU usage, memory, response time, and network traffic, enabling proactive identification of problems before they affect users. Application Insights, a feature of Azure Monitor, focuses on application-level monitoring, capturing detailed diagnostics, request rates, dependency tracking, and exceptions to help developers fine-tune and optimize application code. Together, these tools support real-time alerting, dashboard visualization, and integration with automated incident response workflows. This holistic monitoring strategy ensures high system availability, reliability and performance while providing actionable insights for continuous improvement and operational excellence.

**4.3-ACCESS CONTROL (RBAC OVERVIEW)**

Role-based access control (RBAC) is a critical security mechanism in event management systems that controls user permissions based on their roles in the platform. By defining different roles such as event planners, attendees and administrators, RBAC ensures that users can only access functionality and data relevant to their responsibilities. For example, event organizers have permission to create, update, and manage event details and ticketing, while attendees have restricted access, limited to browsing events, purchasing tickets, and managing their profiles. Administrators oversee platform-wide settings, user management, and security policies.

Implementing RBAC helps protect sensitive information, such as payment details and user data, by applying the principle of least privilege. This reduces the risk of unauthorized actions and potential security breaches. The system's backend uses JWT tokens embedded with user role information, enabling API endpoints to dynamically verify access rights during each request. Combined with secure authentication mechanisms, RBAC strengthens the overall security posture, facilitates compliance with data privacy regulations, and enhances the user experience by providing tailored access to the event management system's cloud-native environment.

**4.4 BLUE–GREEN DEPLOYMENT & DISASTER RECOVERY PLANNING**

Blue-green deployment of the Event Management System involves maintaining two identical production environments — blue and green — to ensure continuous updates with minimal downtime. When a new version of the application is ready, it is deployed to the inactive environment (e.g., green) while the active environment (e.g., blue) continues serving users. Once testing and validation in the green environment is complete, traffic is changed from blue to green using routing mechanisms such as Azure Front Door or NGINX. This approach reduces deployment risk by enabling quick rollback to the previous stable environment if issues arise, ensuring continued availability of event services such as ticketing and scheduling during upgrades.

Disaster recovery planning includes regular backups of critical components such as MongoDB and Azure Blob Storage databases, leveraging Azure geo-redundant storage for data durability across regions. Infrastructure is provisioned and managed with Terraform, enabling rapid environmental restoration in a different region if necessary. Monitoring with Azure Monitor alerts staff to incidents immediately. Together, these strategies ensure high availability, data integrity and resiliency of the Event Management System, minimizing service interruption during failures or outages.

**CHAPTER 5**

**RESULTS AND DISCUSSIONS**

## **5.1 IMPLEMENTATION SUMMARY**

Implementing an event management system involves creating a cloud-native, scalable and secure platform using modern technology and best practices. The frontend was built using React 18 and Tailwind CSS, ensuring a responsive and intuitive user interface for attendees and organizers. The backend was developed with Node.js and Express, which connects to MongoDB for data storage. Security features such as JWT authentication, password hashing, and role-based access control were embedded to protect user data and enforce permissions.

Containerization with Docker enabled consistent deployment across environments, while Azure Kubernetes Service (AKS) provided orchestration, automatic scaling, and high availability. The infrastructure was provisioned and managed by Terraform, enabling repeatable, automated deployments. Continuous integration and continuous deployment (CI/CD) pipelines, integrated code quality and security scans using tools such as CodeQL, SonarCloud and OWASP ZAP, ensuring strong DevSecOps practices. Monitoring and observability was set up using Azure Monitor and Application Insights for proactive issue detection and performance insights.

Deployment strategies use blue-green deployments to minimize downtime during updates, and disaster recovery plans leveraging Azure's geo-redundant storage ensure data durability and business continuity. Finally, integration with Azure AI and Generative AI services was planned to enhance user interaction and automate tasks, aligning the system with state-of-the-art cloud-native solutions. This comprehensive implementation empowers efficient event creation, ticket management, secure payments and seamless user experiences in a resilient cloud environment.

**5.2 CHALLENGES FACED AND RESOLUTIONS**

The event management system project faced multiple challenges, including managing user authentication and authorization securely, which were addressed using JWT and RBAC to properly control user access. Scalability and high availability concerns during peak event traffic were addressed by deploying on Azure Kubernetes Service with automatic scaling and geo-redundant data storage. To prevent environment inconsistencies and simplify recovery, Terraform was used for infrastructure automation, ensuring repeatable and reliable deployments. Integrating security tools like CodeQL, SonarCloud and OWASP ZAP into CI/CD pipelines helped identify vulnerabilities early, improving code quality and compliance. Blue-green deployment strategies minimized downtime during updates, while comprehensive monitoring with Azure Monitor and Application Insights ensured prompt detection and resolution of issues. Finally, disaster recovery was planned with regular backups and geo-redundancy, protecting against data loss and facilitating rapid restoration, resulting in a secure, resilient and highly available platform supporting event creation, ticketing and management.

**5.3 PERFORMANCE OR COST OBSERVATIONS**

The Event Management System leveraged Azure Kubernetes Service (AKS) for container orchestration, exploiting its Kubernetes managed control plane to offload cluster management tasks such as API server availability, etcd backups, and automated patching, thereby reducing operational overhead. AKS support for Horizontal Pod Autoscaling and Cluster Autoscaler enabled dynamic scaling of resources based on real-time demand, ensuring optimal CPU and memory utilization across node pools provisioned in VM scale sets. The deployment architecture incorporated multi-node pools to segregate workloads, optimizing scheduling and availability zones for fault tolerance. Azure Container Registry (ACR) has seamlessly integrated with AKS for secure, private storage of Docker images, facilitating CI/CD deployments. The network was configured using Azure CNI for efficient pod communication across Azure Virtual Networks, with network policies that enforce security limits. Persistent storage requirements were met by using Azure disks attached to pods to support stateful components like MongoDB. Monitoring and logging was performed using Azure Monitor and Log Analytics to collect telemetry and detect anomalies, supporting rapid response to incidents. This deep integration of AKS into the architecture provided a scalable, resilient, and secure environment to host the event management workload in a cloud-native, microservices-driven approach.

**5.4 KEY LEARNINGS AND TEAM CONTRIBUTIONS**

Key learnings from the Event Management System project include the critical importance of adopting a cloud-based architecture to achieve scalability and resilience, with containerization and Kubernetes orchestration proving critical to managing complex deployments. The team gained deep expertise in implementing Infrastructure-as-Code using Terraform, which streamlined consistent environment provisioning and disaster recovery. Integrating security early in the development lifecycle via DevSecOps tools such as CodeQL and OWASP ZAP underscored the value of automated vulnerability detection in CI/CD pipelines to maintain code quality and compliance. The use of Azure Active Directory for identity and access management significantly improved system security and simplified user authorization workflows. Moreover, blue-green implementation strategies highlighted hands-on approaches for zero-downtime releases, improving the overall user experience. The project reinforced that comprehensive monitoring and observability using Azure Monitor and Application Insights is indispensable for proactive problem management and performance tuning. Team contributions included collaborative architecture design, development of reusable automation scripts, implementation of secure authentication mechanisms, establishment of CI/CD pipelines, and execution of rigorous testing and security audits. Collectively, these learnings and cross-functional efforts delivered a robust, scalable and secure incident management platform.

**CHAPTER 6**

**CONCLUSION AND FUTURE ENHANCEMENT**

**6.1-CONCLUSION**

The Event Management System project has successfully simplified the event planning and execution process by replacing manual coordination with a user-friendly, web-based platform. It facilitated seamless event creation, attendee registration and automated communications, significantly reducing administrative overhead and improving efficiency. Secure system authentication, role-based access control, and scalable cloud-native architecture ensured a reliable and accessible application that supported diverse user roles and high concurrent usage. Automating deployments using Terraform and orchestrating containers with Kubernetes has improved system availability and manageability. Overall, the project proved valuable in increasing event attendance, saving time, and providing a solid foundation for future improvements in event management technology.

**6.2 FUTURE SCOPE**

The future scope of the Event Management System includes enhancing the platform with advanced AI-driven features such as personalized event recommendations and intelligent virtual assistants powered by Azure’s generative AI capabilities. Expanding multi-cloud deployment options would increase flexibility and resilience, allowing seamless migration or load distribution across cloud providers. Incorporating a service mesh, such as Istio, can improve microservices communication security, observability, and traffic management. Advanced analytics using Azure Synapse Analytics and Power BI can provide deeper insights into event trends, attendee engagement, and operational efficiency. Enhancing mobile and cross-platform support will widen accessibility and user reach. Additionally, automating compliance auditing and further hardening security with zero-trust principles will future-proof the system against evolving cybersecurity threats and regulatory requirements, ensuring sustained scalability and user satisfaction.

**APPENDICES**

**Appendix-A – Terraform Code Snippets**

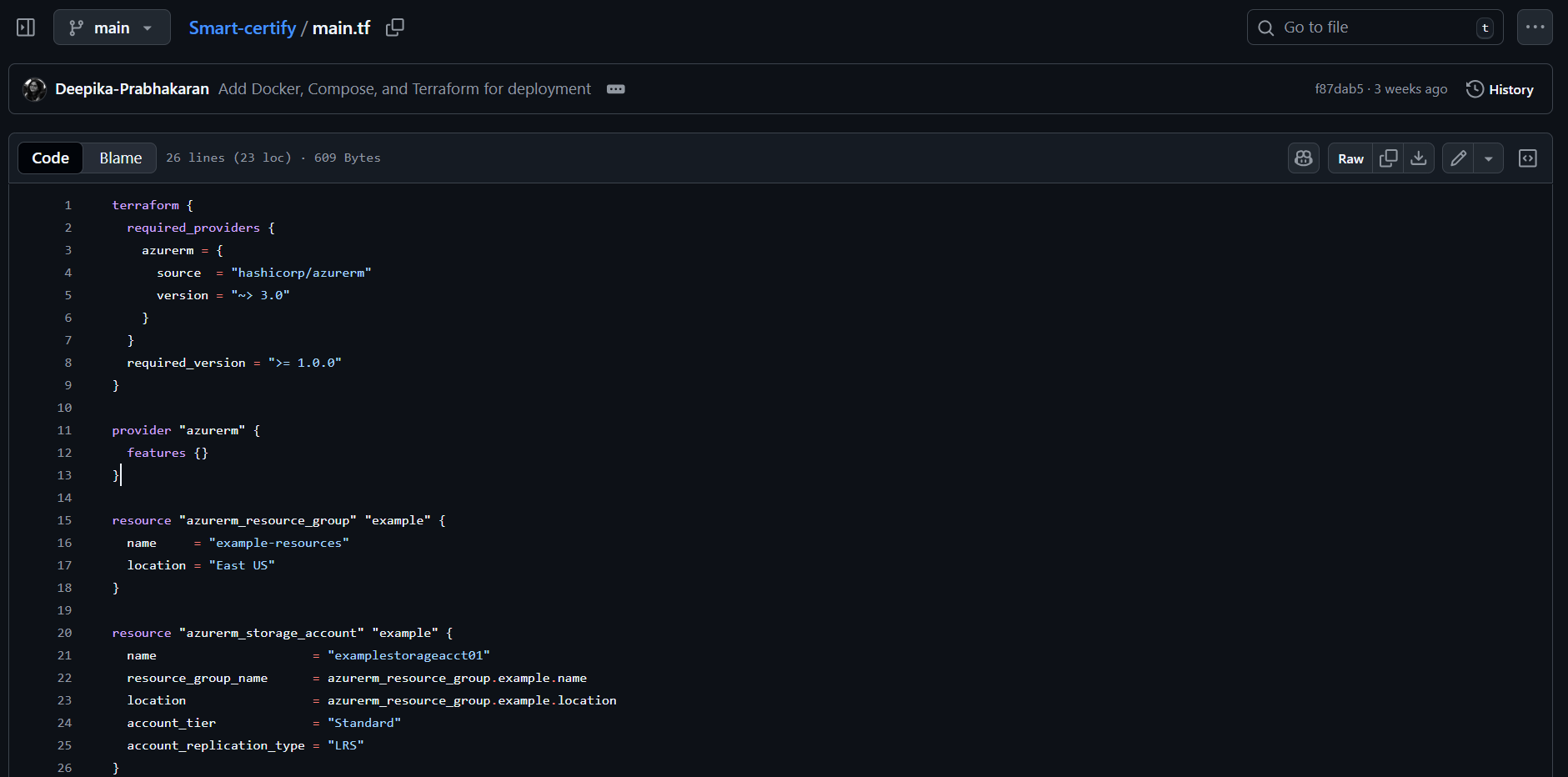


Figure 1.1:Terraform code snippet - main.tf

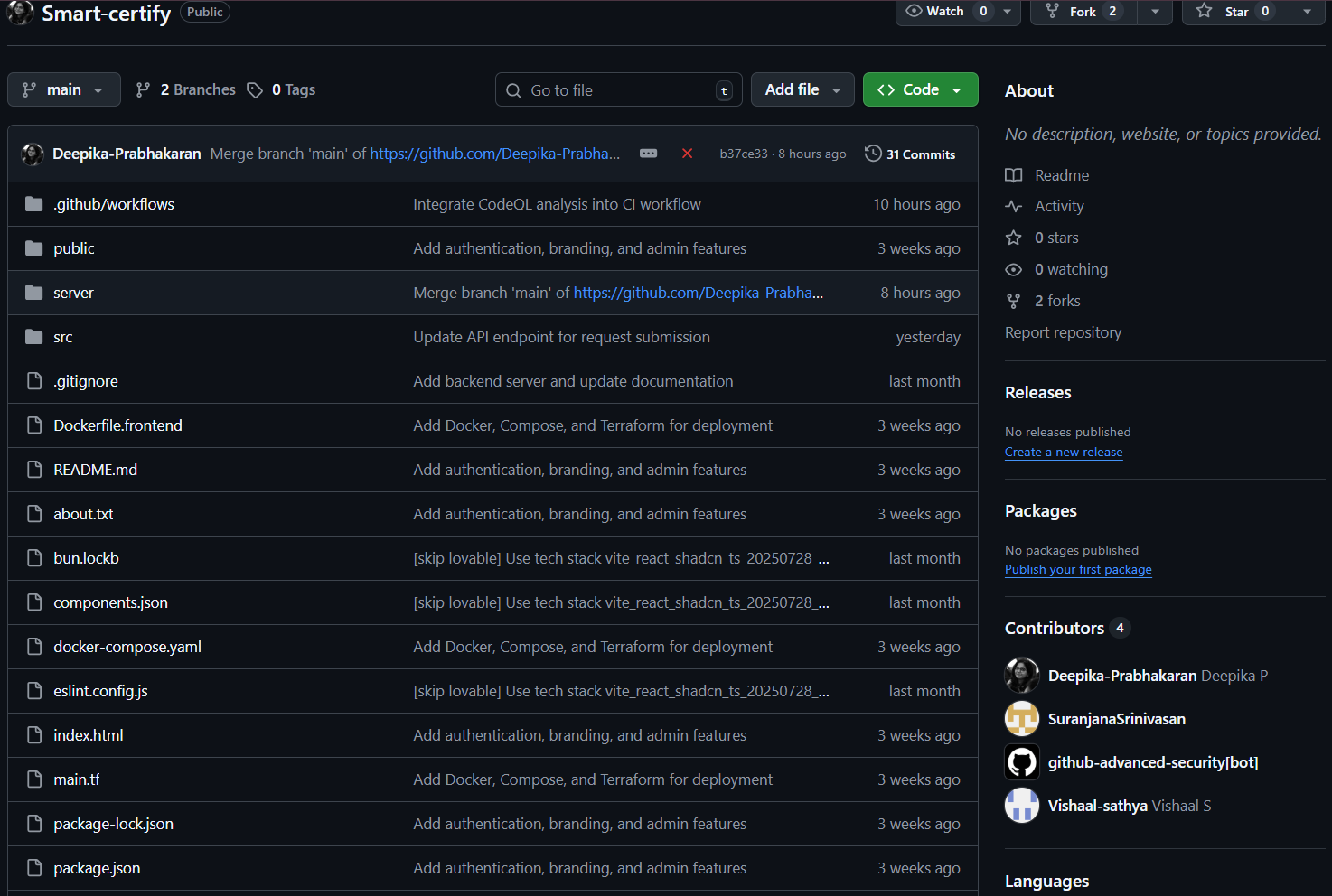


Figure 1.2:Project file structure

**Appendix B – Pipeline YAMLs**



Figure 1.3.1:build and deploy app to Azure web app

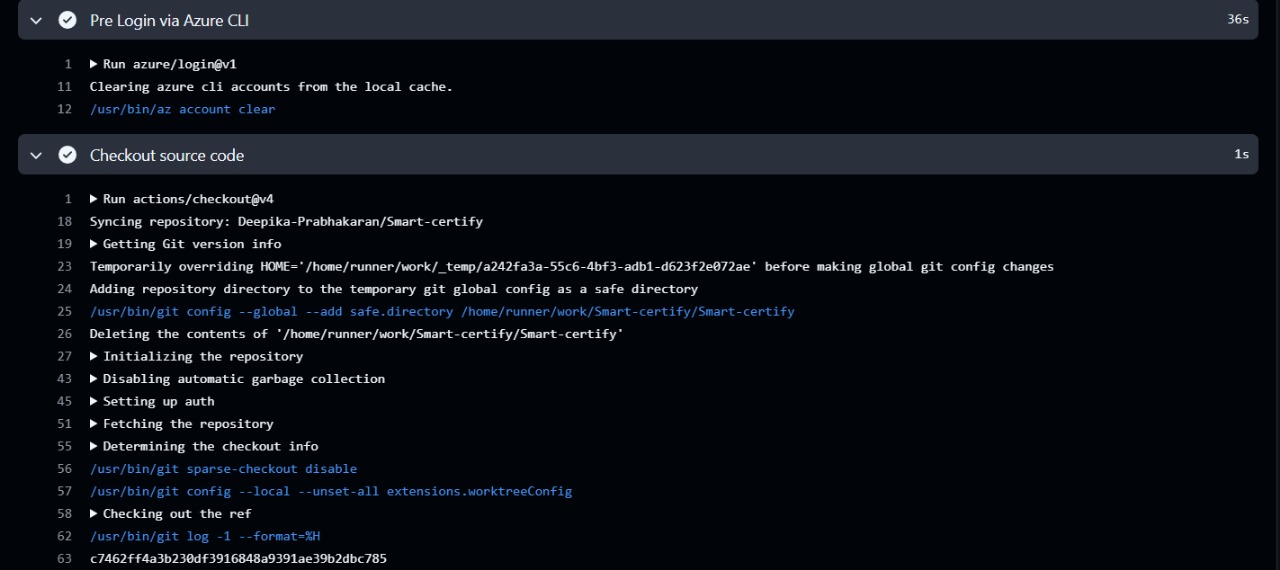


Figure 1.3.2:build and deploy app to Azure web app

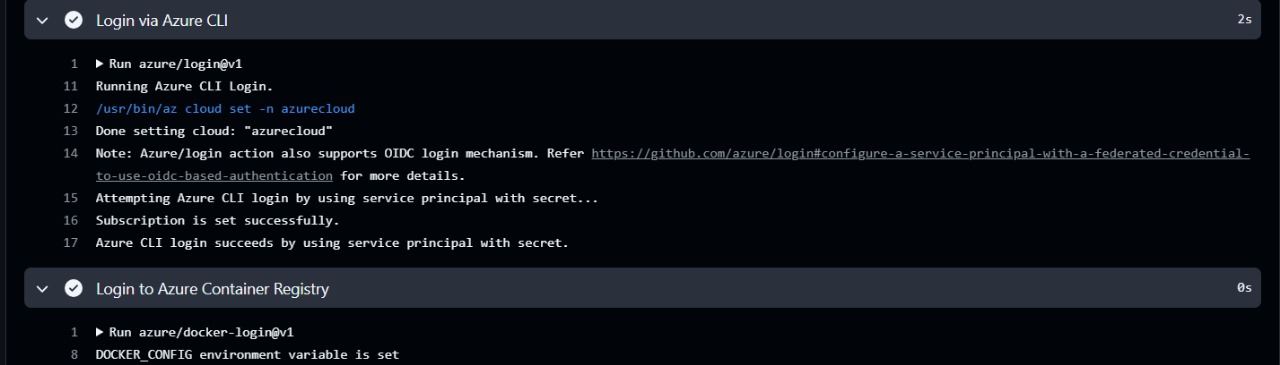


Figure 1.3.3:build and deploy app to Azure web app

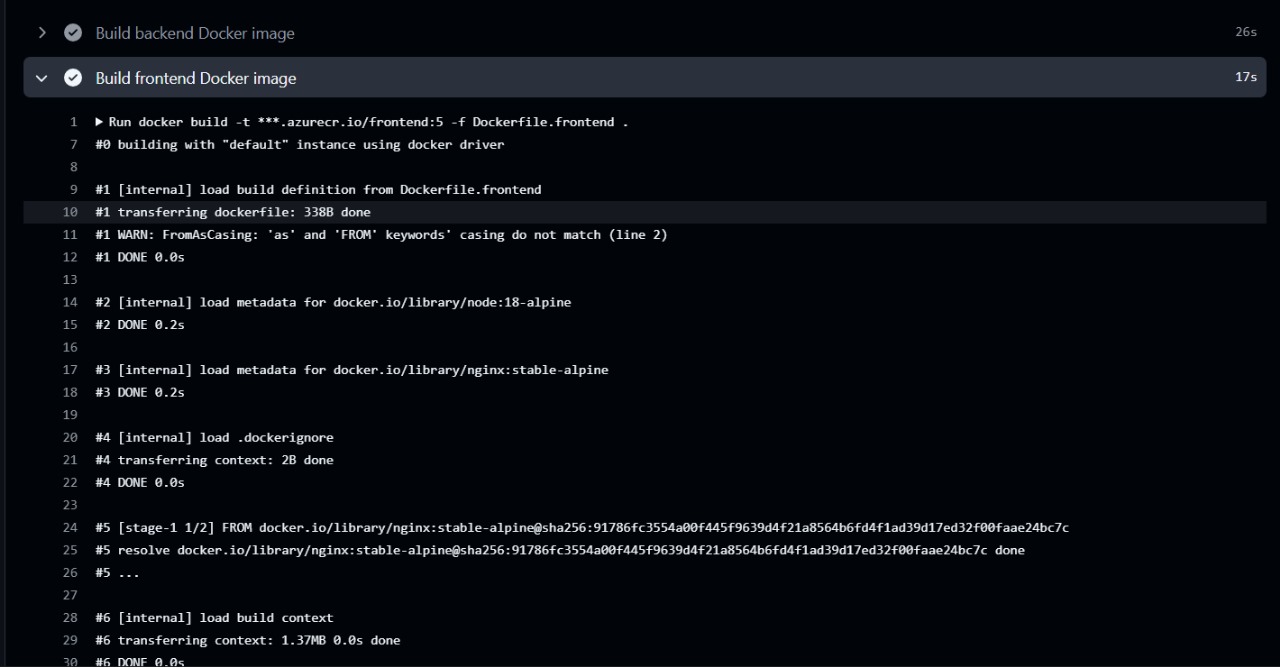


Figure 1.3.4:build and deploy app to Azure web app

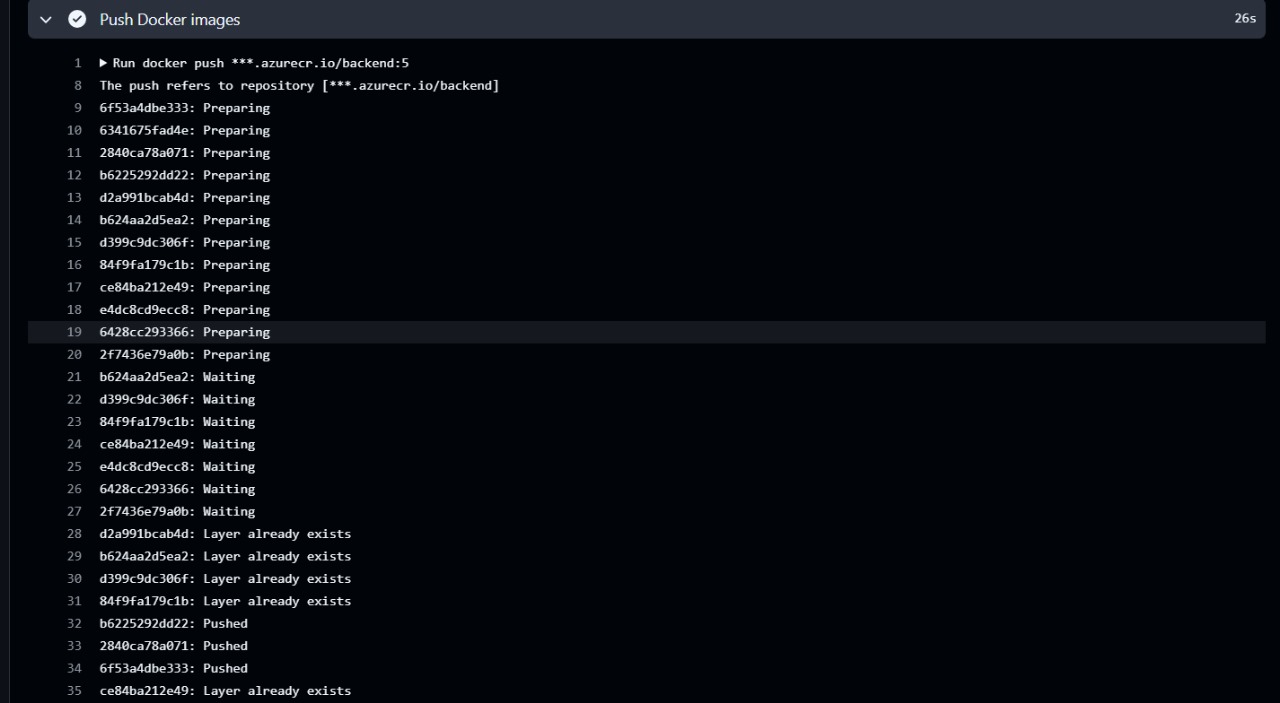


Figure 1.3.5:build and deploy app to Azure web app

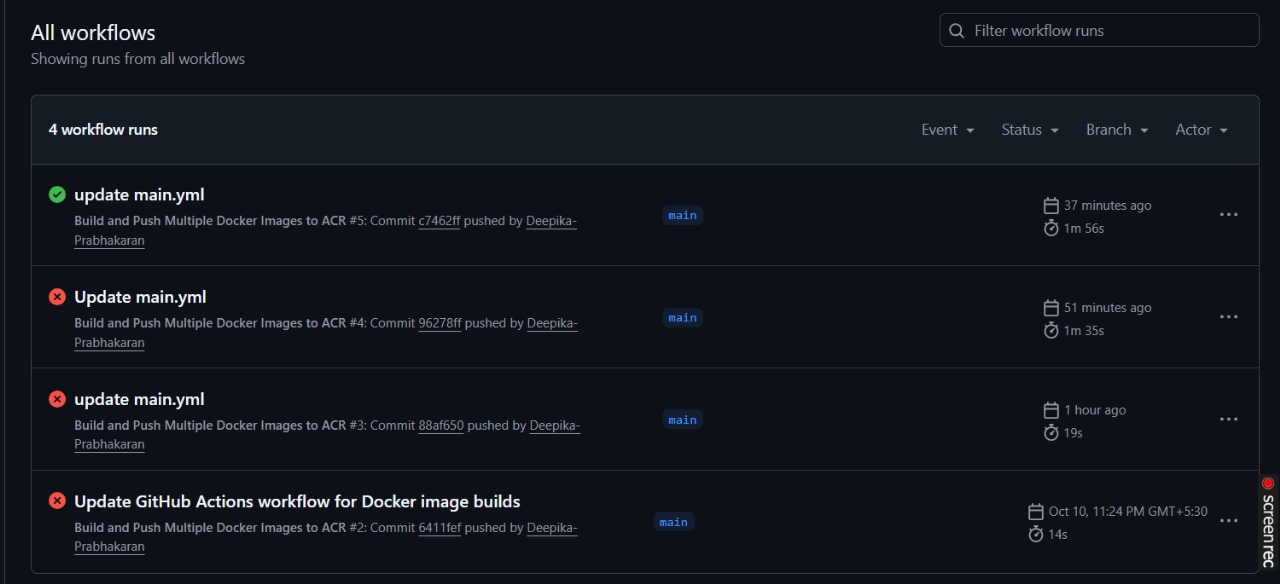


Figure 1.4:Deployed workflow

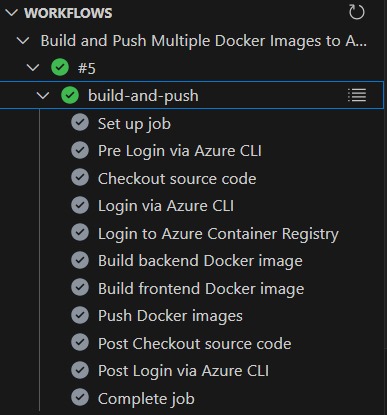
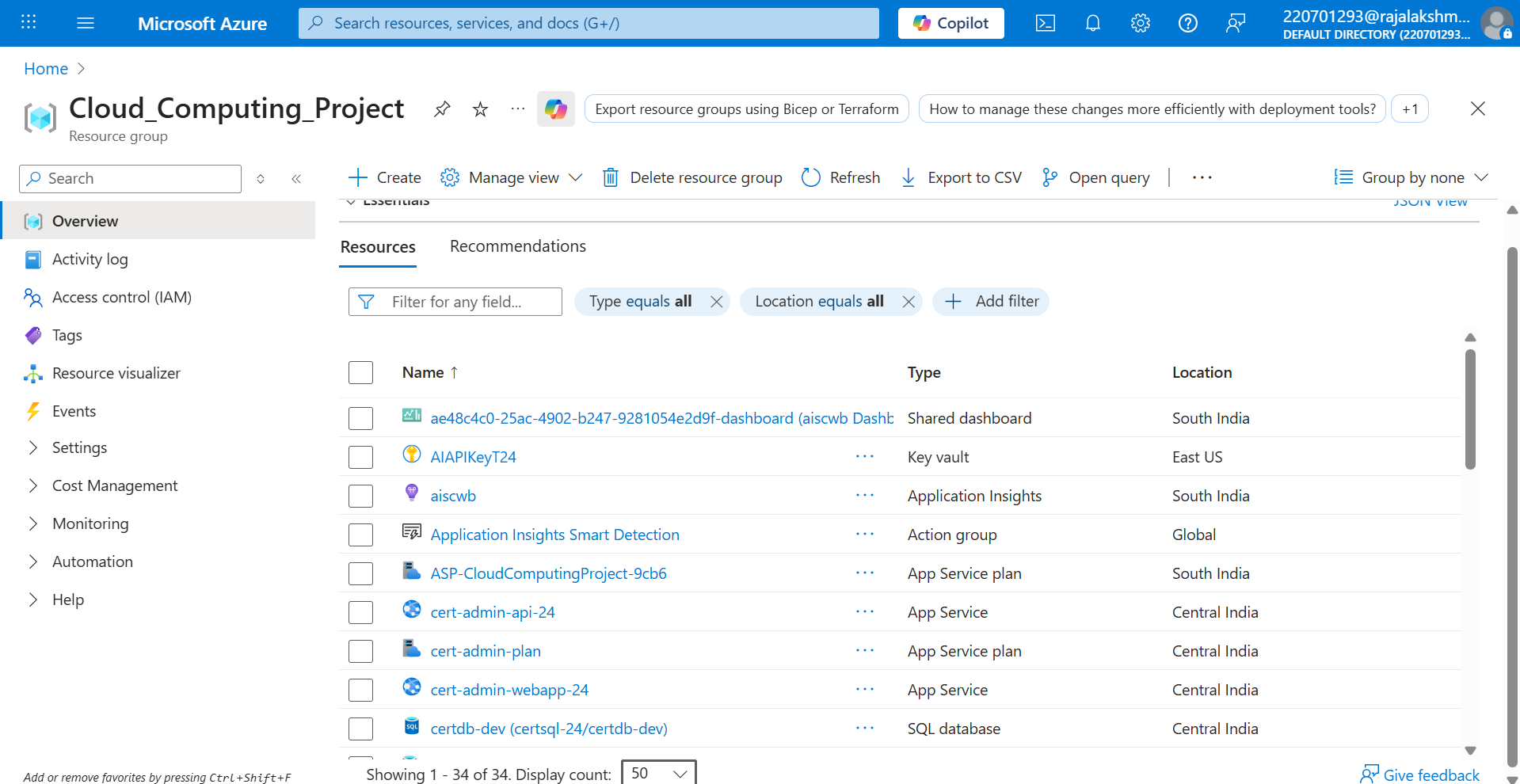


Figure 1.4:Deployed workflow

**Appendix C-Deployment**



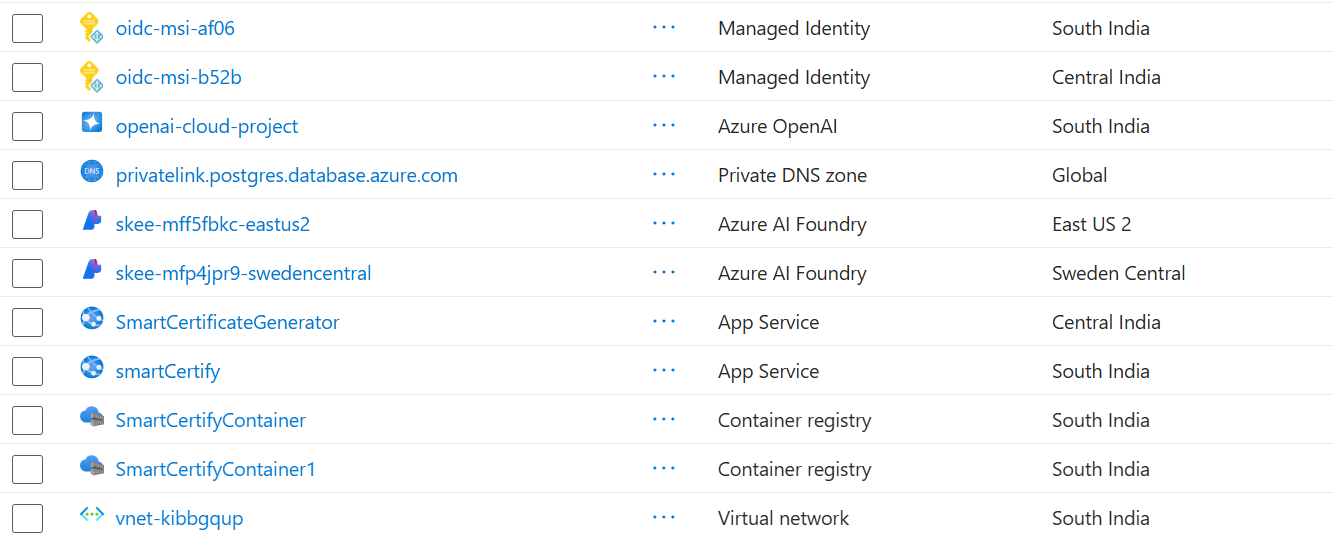
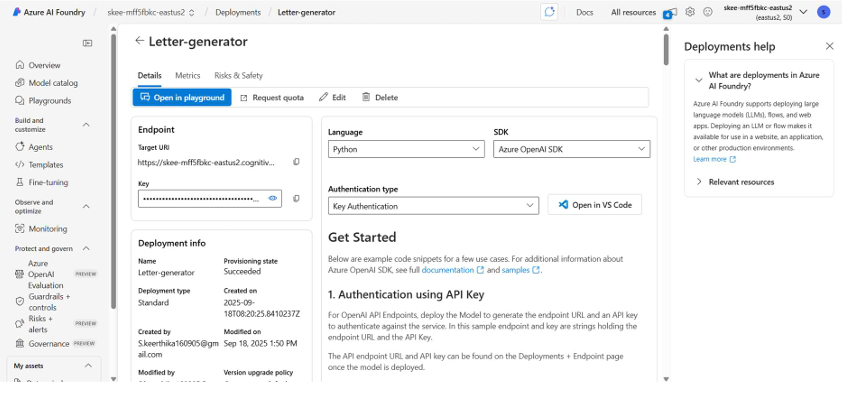


Figure 1.5:Resource Creation

AI Integration



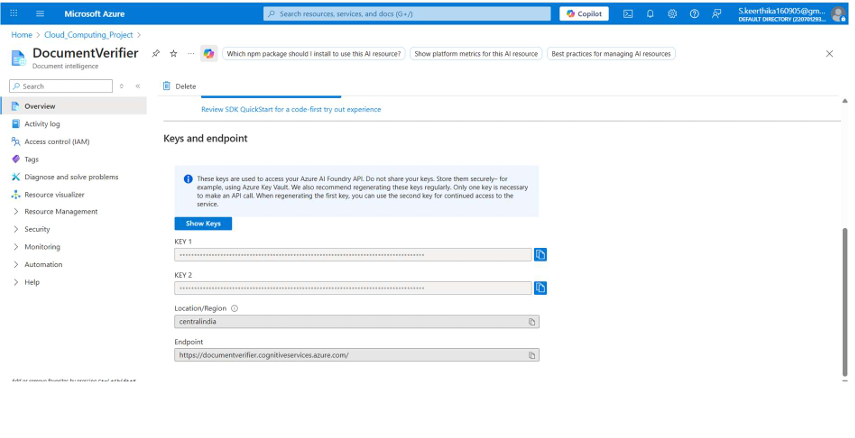


Figure 1.6:Document intelligence deployment

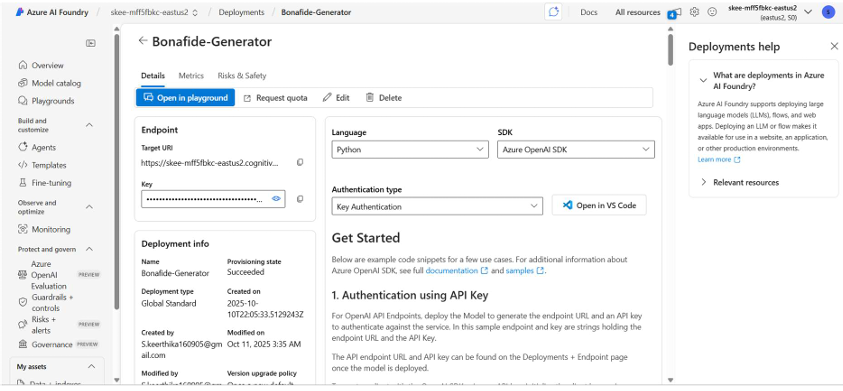


Figure 1.7:Letter generation

Security Scans

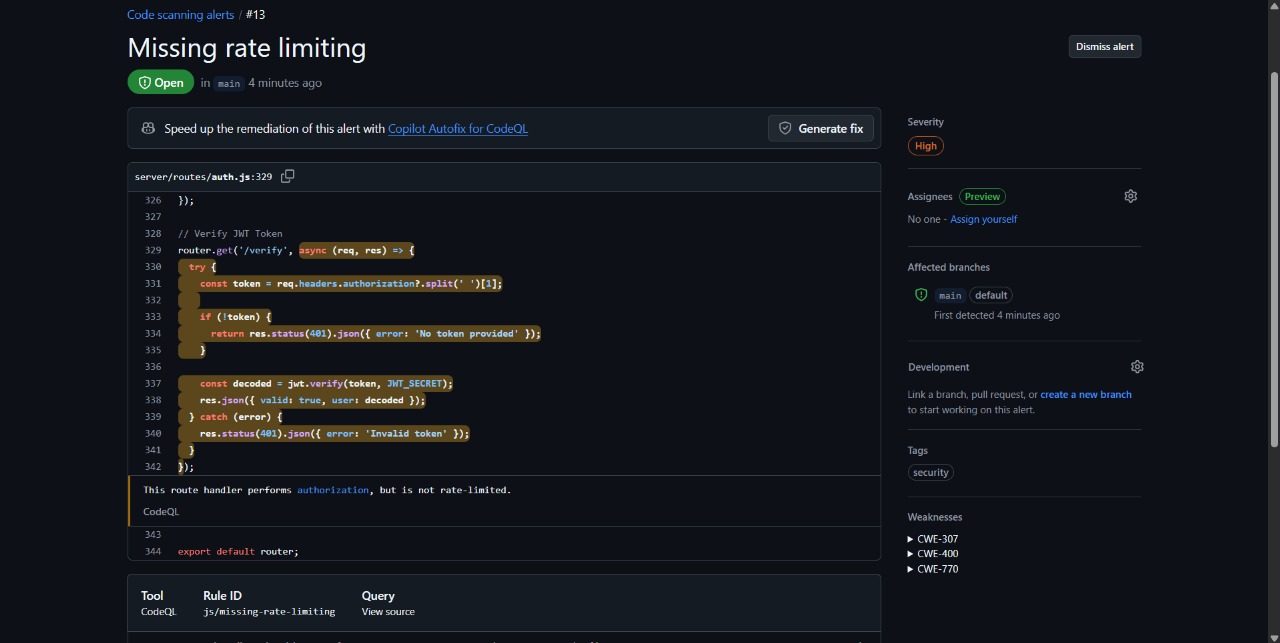


Figure 1.8.1: Security Scan



Figure 1.8.2: Security Scan

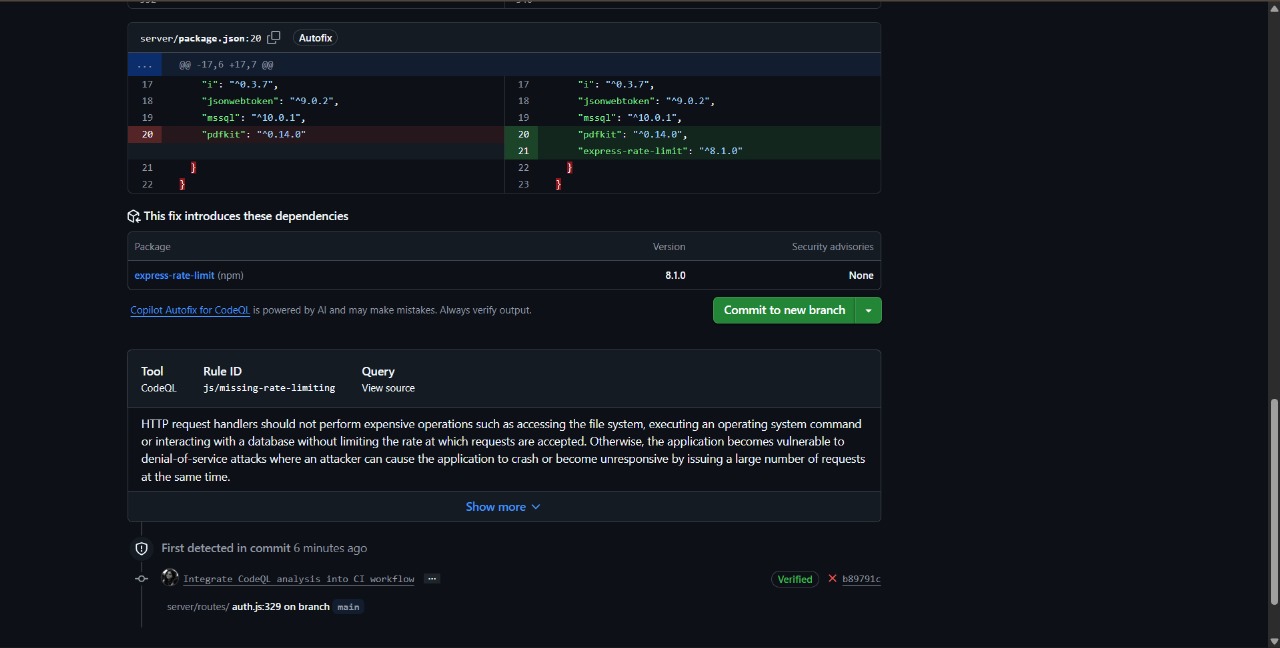


Figure 1.8.3: Security Scan

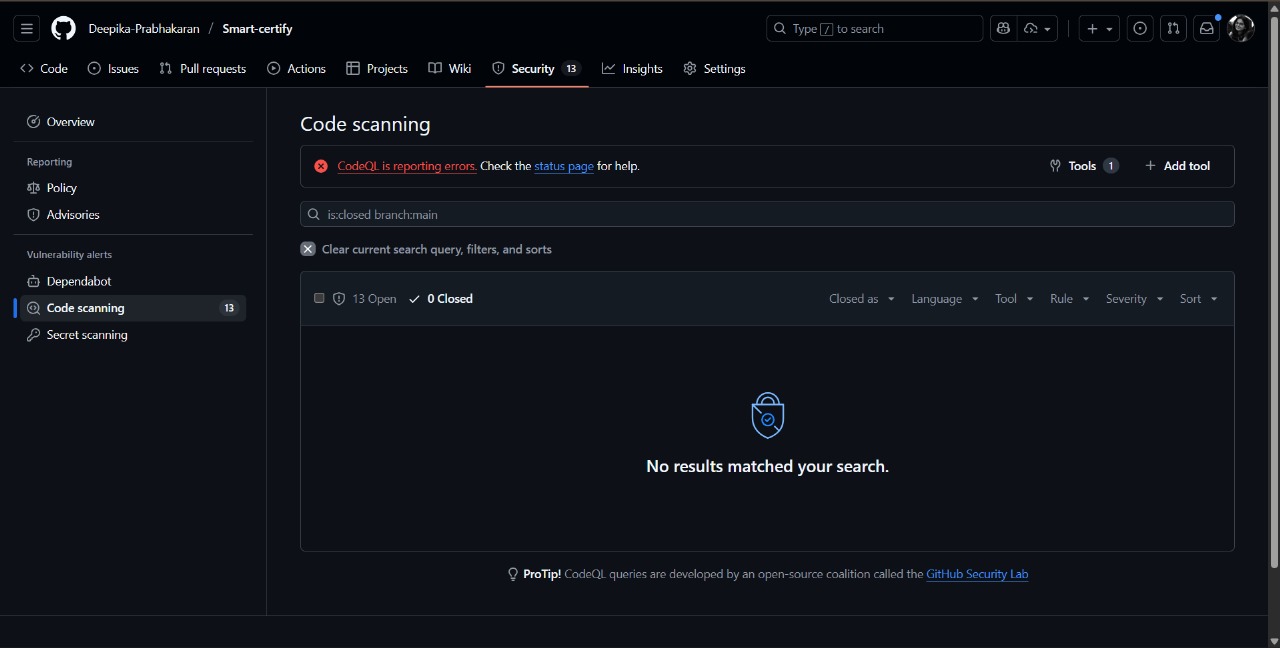


Figure 1.8.4: Security Scan

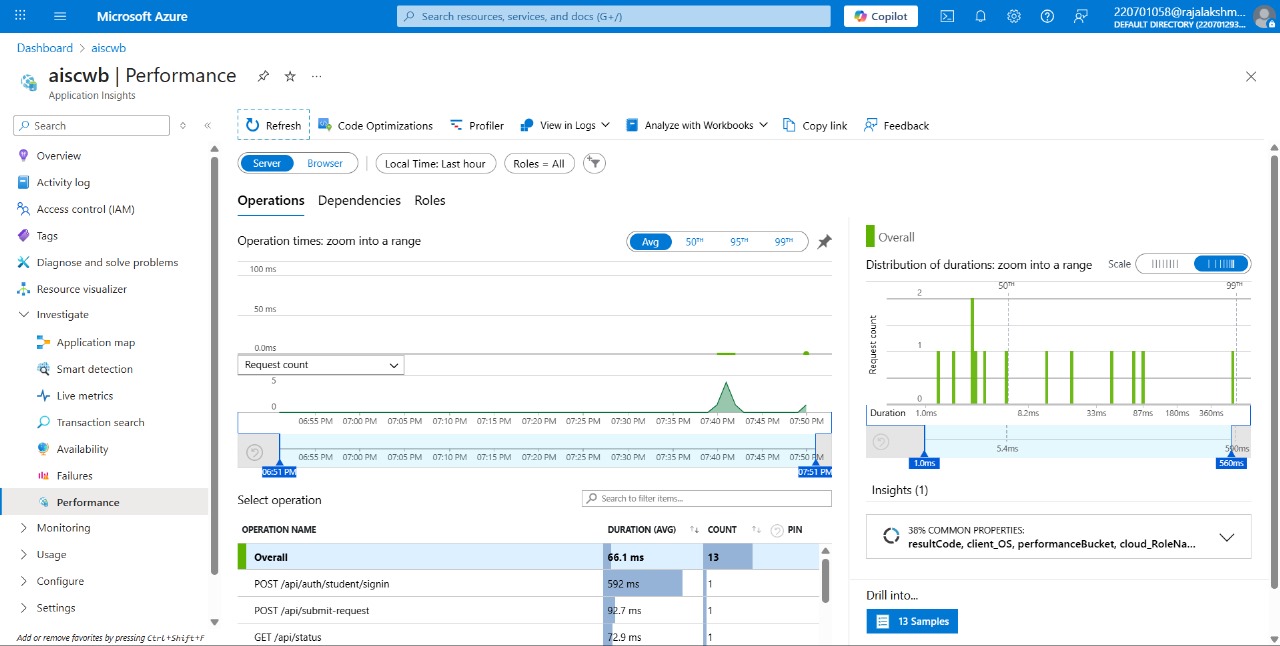


Figure 1.9: Azure monitor