[COMPANY LOGO]

Application Intelligence Report

Comprehensive Analysis and Migration Assessment

Repository: https://github.com/end-of-game/openshift-voting-app

Analysis Date: July 18, 2025

*Generated by Application Intelligence Platform*

Table of Contents

Executive Summary........................................... 3

Application Overview........................................ 4

Component Analysis.......................................... 5

Architecture Assessment..................................... 8

Security Analysis........................................... 10

Git History Analysis........................................ 12

Recommendations............................................. 13

Appendices.................................................. 15

Executive Summary

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Total Components | 3 |
| Programming Languages | python, nodejs, java |
| Containerization Status | 3 containerized |
| Data Sources | 0 |
| Security Findings | 0 |
| Git Commits | 1 |
| Architecture Style | microservices |

Application Overview

This report presents a comprehensive analysis of the application repository. The analysis identified 3 components using 3 different programming languages. The application demonstrates a microservices architecture pattern.

Key Findings

• 📦 3 application components identified

• 🔧 3 programming languages detected: python, nodejs, java

• 🐳 3 components are containerized

• 💾 0 data sources identified

• 🔒 0 security findings require attention

Detailed Analysis

Component Analysis

The analysis identified 3 components across the application:

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Language** | **Type** | **Packaging** |
| vote | python | Unknown | docker |
| result | nodejs | Unknown | docker |
| worker | java | Unknown | docker |

Component: vote

• Language: python

• Runtime: python

• Build Tool: unknown

• Packaging: docker

• Exposed Ports: 8080

• Base Images: python:3.9-slim

Component: result

• Language: nodejs

• Runtime: nodejs

• Build Tool: unknown

• Packaging: docker

• Exposed Ports: 8080

• Base Images: node:10-slim

Component: worker

• Language: java

• Runtime: java

• Build Tool: unknown

• Packaging: docker

• Base Images: maven:3.5-jdk-8-alpine, openjdk:8-jre

**Notes:**

• Alternative C# implementation found at 'worker/src/src/Worker/Program.cs' but does not appear to be the primary build target defined in the Dockerfile or OpenShift manifests. Primary implementation is Java.

• Multiple base images detected: maven:3.5-jdk-8-alpine, openjdk:8-jre. This may indicate multi-stage builds or alternative build strategies.

Architecture Analysis

Architecture Style: microservices (Confidence: ConfidenceLevel.HIGH)

Reasoning: Multiple components with independent deployment characteristics

**Evidence:**

• Found 3 components

• Multiple deployable components detected

• 3 containerized components

• Multiple deployment configurations

Security Analysis

Security analysis identified 2 findings with 3 base image risks.

**Key Security Findings:**

• Unknown: The result component uses node:10-slim base image which is past End-of-Life and contains numerous unpatched vulnerabilities. (Severity: CRITICAL)

• Unknown: A pattern matching hardcoded secrets was detected in the application code. (Severity: HIGH)

Git History Analysis

• Total Commits: 1

• Active Contributors: 0

• Recent Activity: inactive

• Code Stability: high

Recommendations

🔴 High Priority Recommendations

• 🔒 Security: 2 critical/high severity vulnerabilities found. Prioritize security remediation.

🟢 Low Priority Recommendations

• 📊 Development Activity: Low recent activity detected. Consider reviewing development processes and team capacity.

• 🐳 Base Images: 3 base images have known risks. Update to more recent versions.

Appendices

Appendix A: Technical Details

This analysis was generated using the Application Intelligence Platform, which performs comprehensive analysis of application repositories including code structure, infrastructure configuration, and security assessment.

Appendix B: Analysis Methodology

• Component Discovery: Automated scanning of source code and configuration files

• Language Detection: Analysis of file extensions, build configurations, and base images

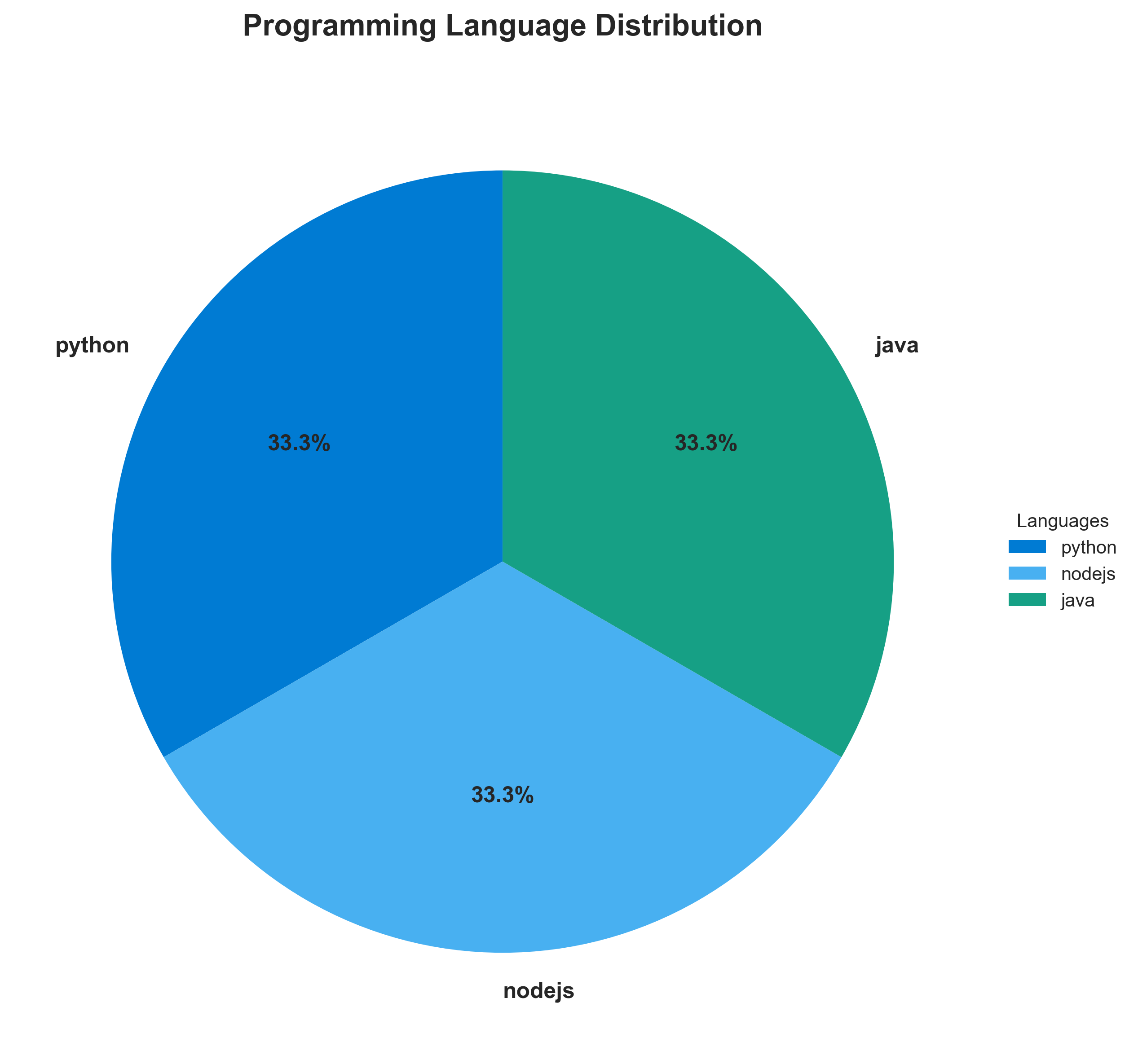
• Architecture Assessment: Evaluation of deployment patterns and component relationships

• Security Analysis: Scanning for common vulnerabilities and configuration issues

• Git History Analysis: Examination of commit patterns and development activity

Charts and Visualizations

Programming Language Distribution



**📊 Context:** This diagram illustrates the programming language distribution across the application's components, specifically focusing on the 'vote', 'result', and 'worker' services. This analysis is crucial for informing strategic decisions regarding application modernization and cloud migration by identifying technology stacks and potential areas for consolidation or refactoring.

**📊 Key Insights:** The application exhibits a polyglot architecture, utilizing Python for the 'vote' component, Node.js for 'result', and Java for 'worker'. All three components are containerized using Docker. Notably, the 'result' and 'worker' components utilize base images identified as having potential vulnerabilities ('node:10-slim', 'maven:3.5-jdk-8-alpine', 'openjdk:8-jre'), posing a security risk.

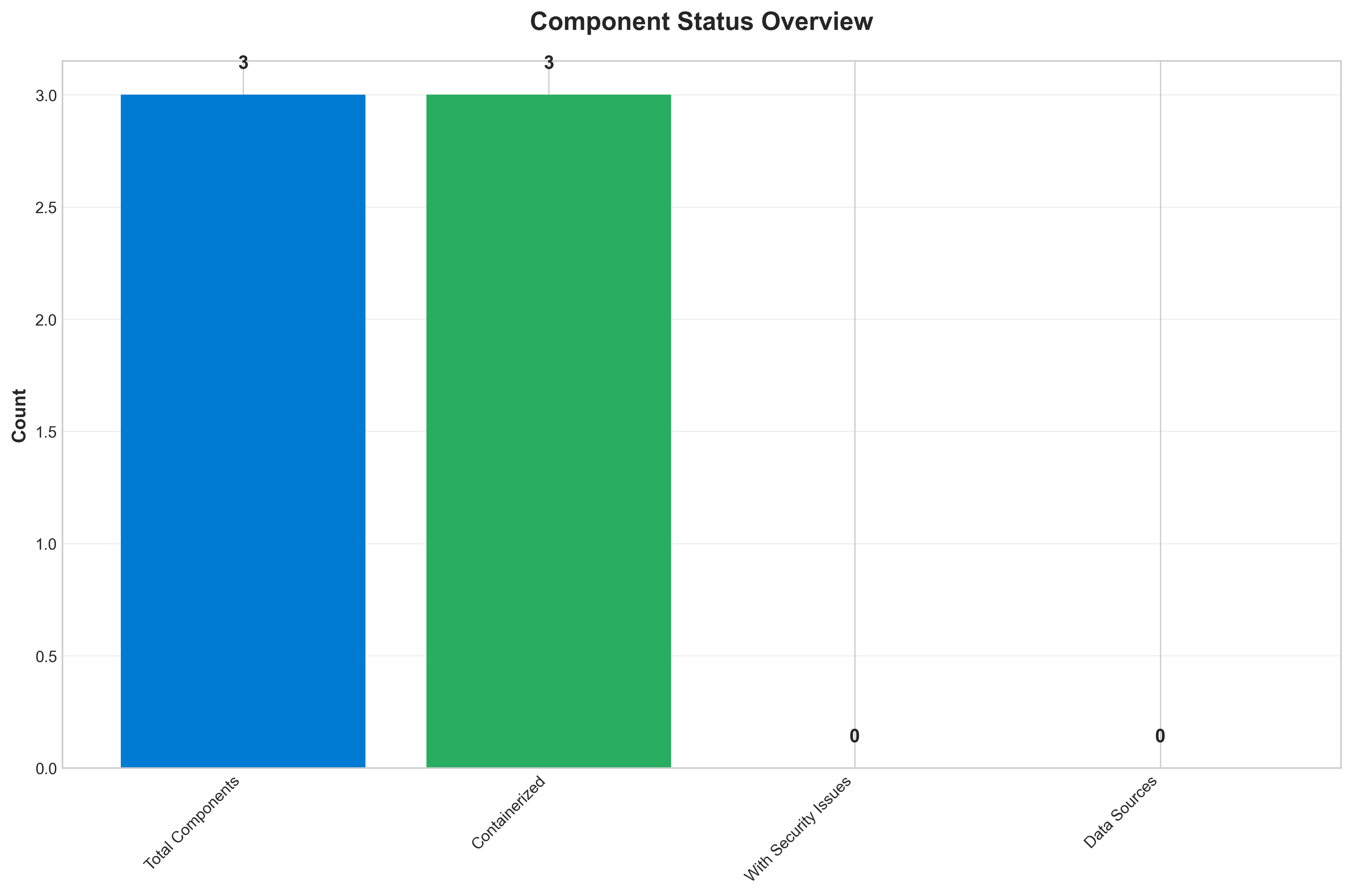
**📊 Business Impact:** The diverse technology stack presents both opportunities and challenges for modernization. While flexibility exists, maintaining and evolving multiple languages can increase operational costs and complexity. The identified vulnerable base images introduce a significant security risk that could impact data integrity and compliance, potentially delaying migration or requiring immediate remediation efforts.

**📊 Recommendations:** Prioritize the remediation of vulnerable base images in the 'result' and 'worker' components by updating them to secure, supported versions. Evaluate the potential for language consolidation across components to reduce maintenance overhead and leverage team expertise, potentially standardizing on a single language or a smaller subset.

**📊 Technical Details:** The 'vote' service is built on Python (confidence 6), 'result' on Node.js (confidence 6), and 'worker' on Java (confidence 8). All components are Dockerized. The 'worker' service's use of multiple base images ('maven:3.5-jdk-8-alpine', 'openjdk:8-jre') suggests a multi-stage build process, which is a common pattern but requires careful management of image versions.

────────────────────────────────────────────────────────────────────────────────

Component Status Overview



**📊 Context:** This Component Status Overview diagram provides a snapshot of the health and status of the application's components, specifically focusing on containerization and security posture. The analysis is crucial for informing our ongoing application modernization and migration planning efforts.

**📊 Key Insights:** All 3 identified application components are containerized. However, a significant security concern is present, with all 3 components exhibiting HIGH severity security findings related to their base images. Specifically, the 'result' component uses an End-of-Life Node.js 10 image, while the 'worker' component has vulnerabilities in both its Maven/JDK 8 and OpenJDK 8 base images.

**📊 Business Impact:** The identified high-severity vulnerabilities present a substantial security risk, potentially exposing the application to exploits and data breaches. Delaying remediation could impact compliance and increase the cost of future security fixes. Addressing these issues is a critical prerequisite for a smooth modernization and migration to the cloud, as outdated and vulnerable components can hinder cloud-native adoption and introduce unforeseen operational challenges.

**📊 Recommendations:** Prioritize immediate remediation of all high-severity security findings by updating the base images for all components. This should be followed by a comprehensive re-scan to validate the effectiveness of the updates. Implementing a continuous security scanning process for base images is crucial to prevent similar issues in future modernization phases.

**📊 Technical Details:** The security findings indicate that 'result' utilizes 'node:10-slim', which is End-of-Life and has numerous vulnerabilities. The 'worker' component is affected by vulnerabilities in its 'maven:3.5-jdk-8-alpine' and 'openjdk:8-jre' base images. All findings were derived from base image analysis, and there are no medium or low severity issues currently identified.

────────────────────────────────────────────────────────────────────────────────

Application Architecture



**📊 Context:** This Architecture Diagram shows application architecture insights.

**📊 Key Insights:** Analysis of 3 components reveals system characteristics.

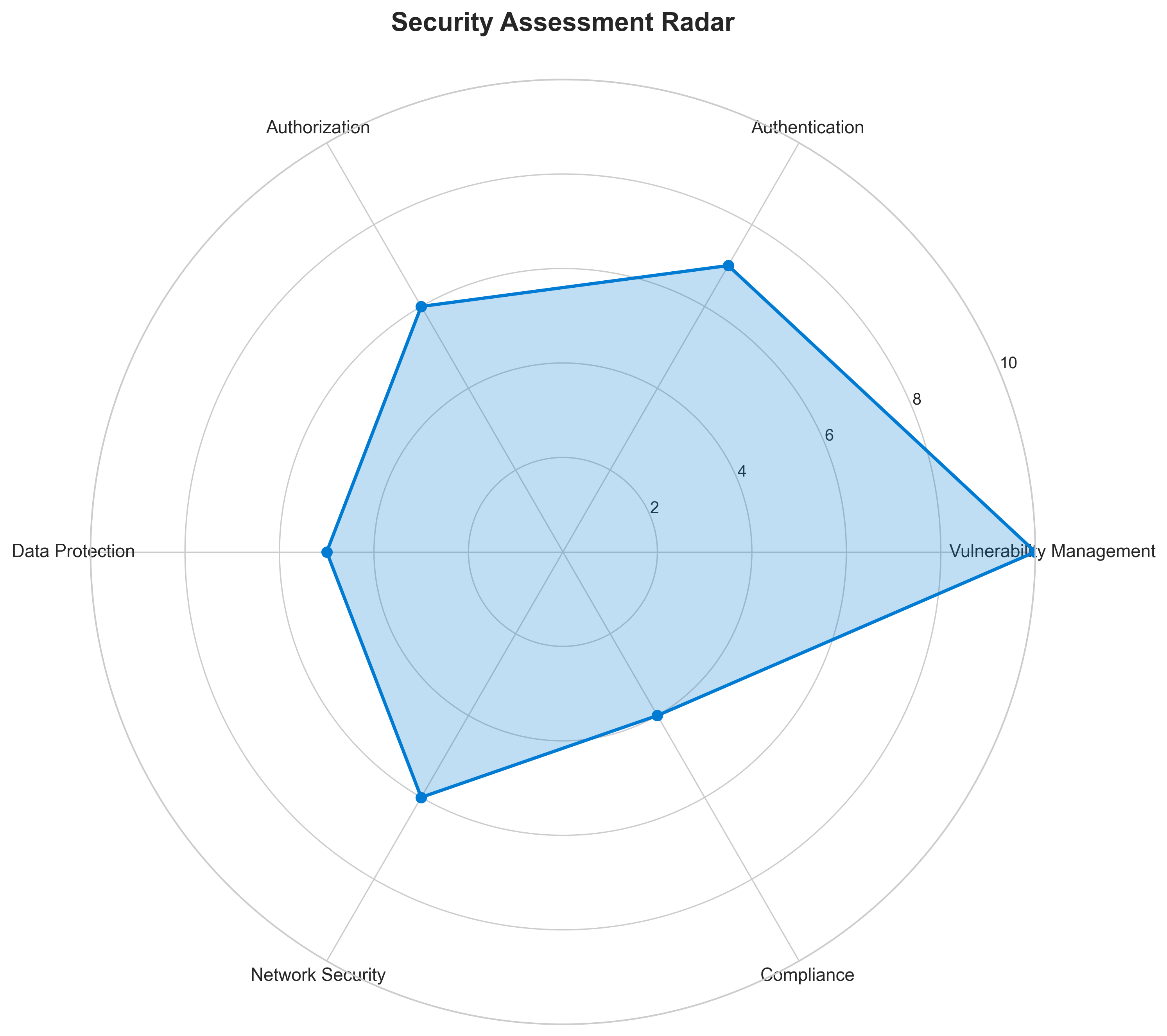
**📊 Business Impact:** Understanding system architecture supports better planning and decision-making.

**📊 Recommendations:** Review analysis results with technical teams for action planning.

**📊 Technical Details:** Diagram type: Architecture Diagram. Based on comprehensive application analysis.

────────────────────────────────────────────────────────────────────────────────

Security Assessment Radar



**📊 Context:** Security analysis identified 3 potential issues.

**📊 Key Insights:** Security posture assessment shows 3 findings requiring attention.

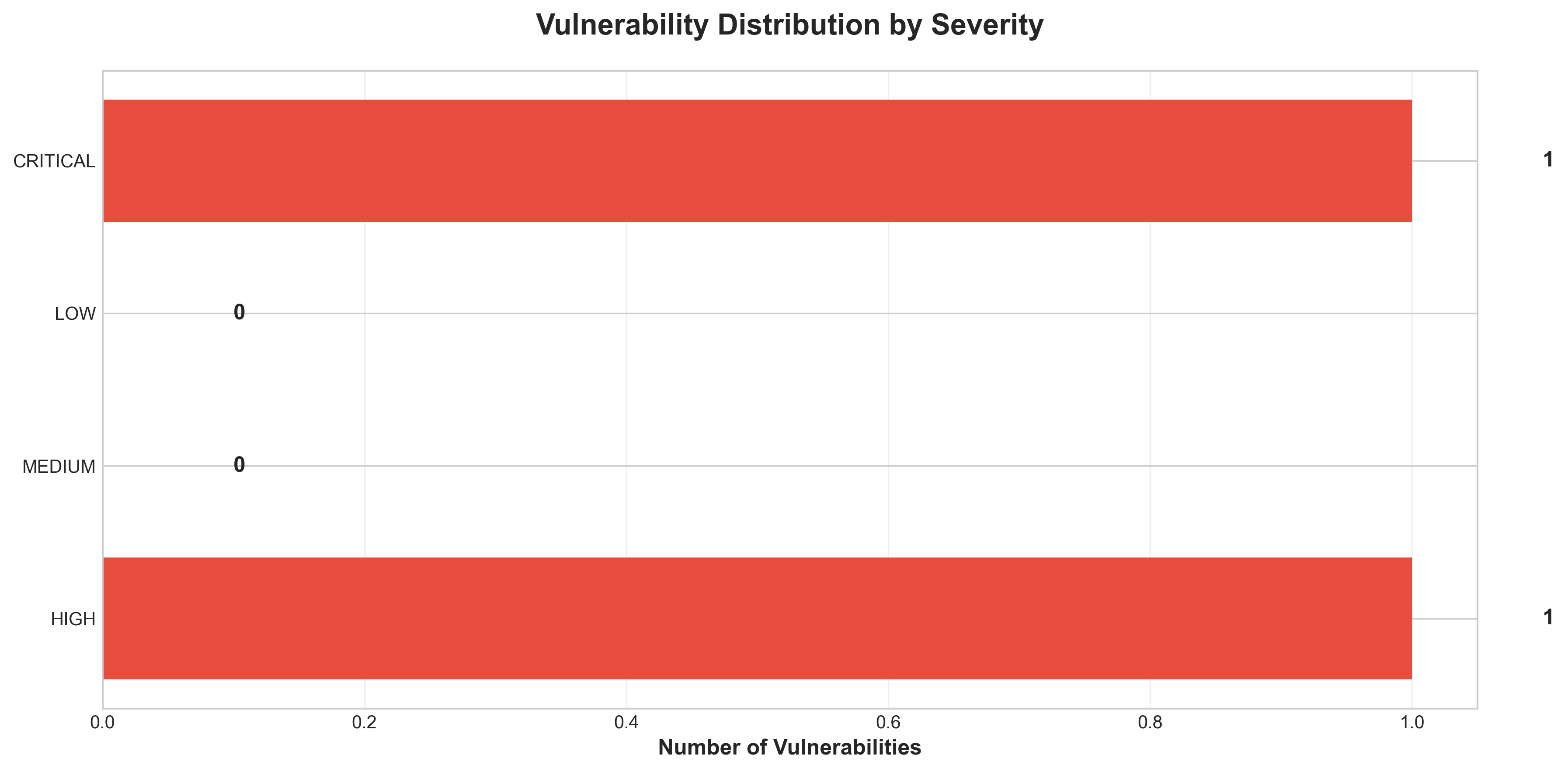
**📊 Business Impact:** Security findings represent potential risks that could impact system reliability and compliance.

**📊 Recommendations:** Address high-severity findings first, then implement security scanning in CI/CD pipeline.

**📊 Technical Details:** Findings: 3. Analysis includes base image vulnerabilities and code patterns.

────────────────────────────────────────────────────────────────────────────────

Vulnerability Analysis



**📊 Context:** This vulnerability timeline analysis highlights security risks identified within the application's components, specifically focusing on the base images used and potential hardcoded secrets. The data is critical for informing modernization and migration planning by quantifying and prioritizing security remediation efforts.

**📊 Key Insights:** The analysis reveals a significant security posture risk stemming from the use of an End-of-Life (EOL) Node.js v10 base image for the 'result' component, which is explicitly flagged as 'CRITICAL'. Furthermore, all three identified security findings are categorized as 'HIGH' severity, indicating widespread use of outdated or vulnerable base images (including Maven and OpenJDK 8 for the 'worker' component) and a potential hardcoded secret in the 'vote' application.

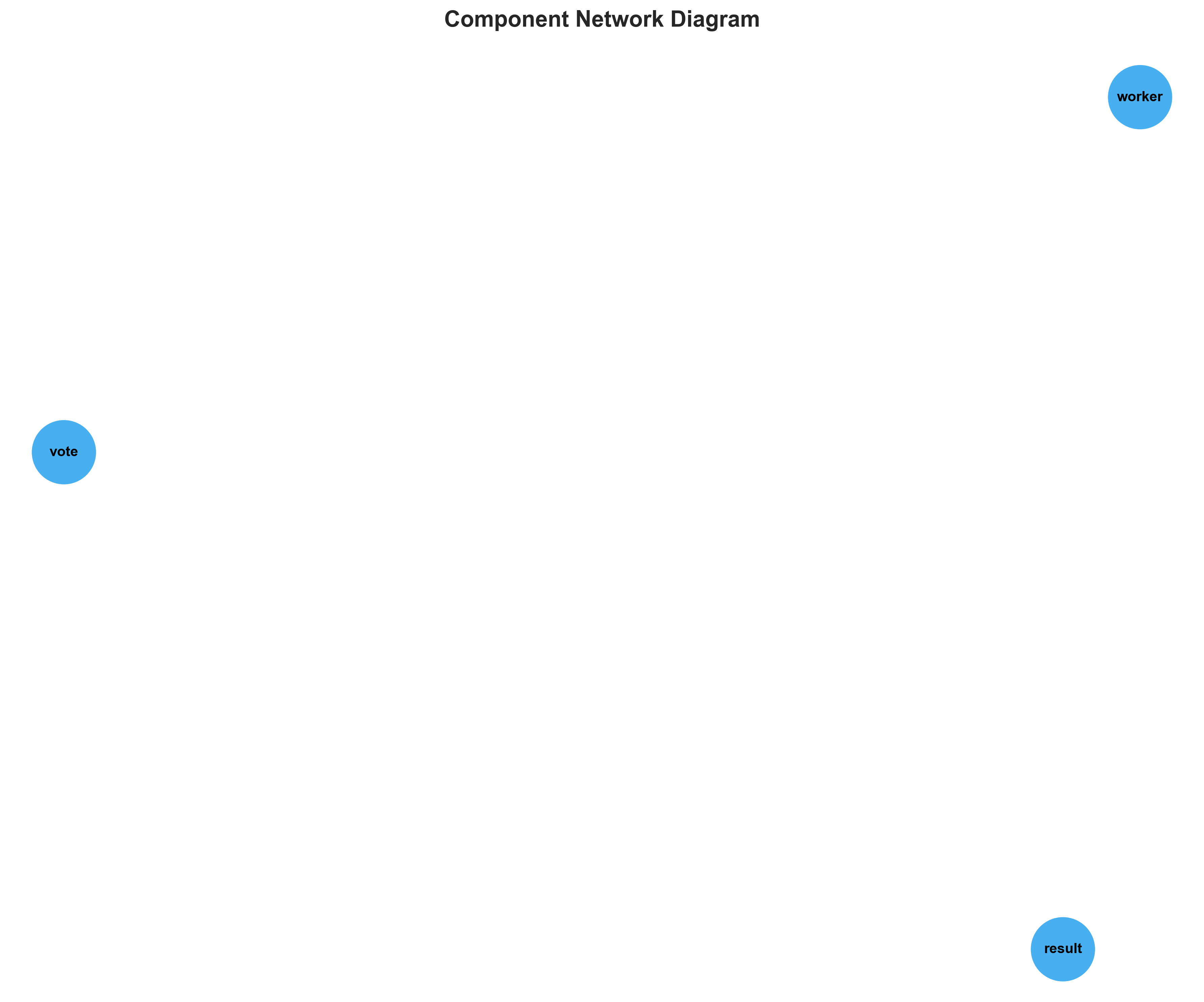
**📊 Business Impact:** The presence of critical and high severity vulnerabilities, particularly the EOL base image, poses a substantial security risk to the business, potentially leading to data breaches, service disruptions, and compliance failures. The reliance on outdated components also increases the technical debt, making modernization and migration to cloud environments more complex, costly, and time-consuming due to the need for significant refactoring and patching.

**📊 Recommendations:** Prioritize immediate remediation of the 'CRITICAL' EOL Node.js v10 base image by updating to a supported version (e.g., Node.js v18 or v20). Concurrently, address the 'HIGH' severity findings by updating Maven and OpenJDK base images for the 'worker' component and thoroughly investigating and removing the potential hardcoded secret in 'voting-app\vote\src\app.py'.

**📊 Technical Details:** The data indicates that the 'result' component's Dockerfile (line 1) uses the 'node:10-slim' base image, which is EOL and unpatched. The 'summary.security\_findings' further confirms this and adds 'maven:3.5-jdk-8-alpine' and 'openjdk:8-jre' as 'HIGH' severity findings for the 'worker' component. A separate finding (MANUAL\_HARDCODED\_SECRETS\_21) points to a potential hardcoded secret in 'voting-app\vote\src\app.py'.

────────────────────────────────────────────────────────────────────────────────

Component Network Topology



**📊 Context:** Shows status of 3 application components.

**📊 Key Insights:** 3 of 3 components are containerized. Fully ready for cloud deployment.

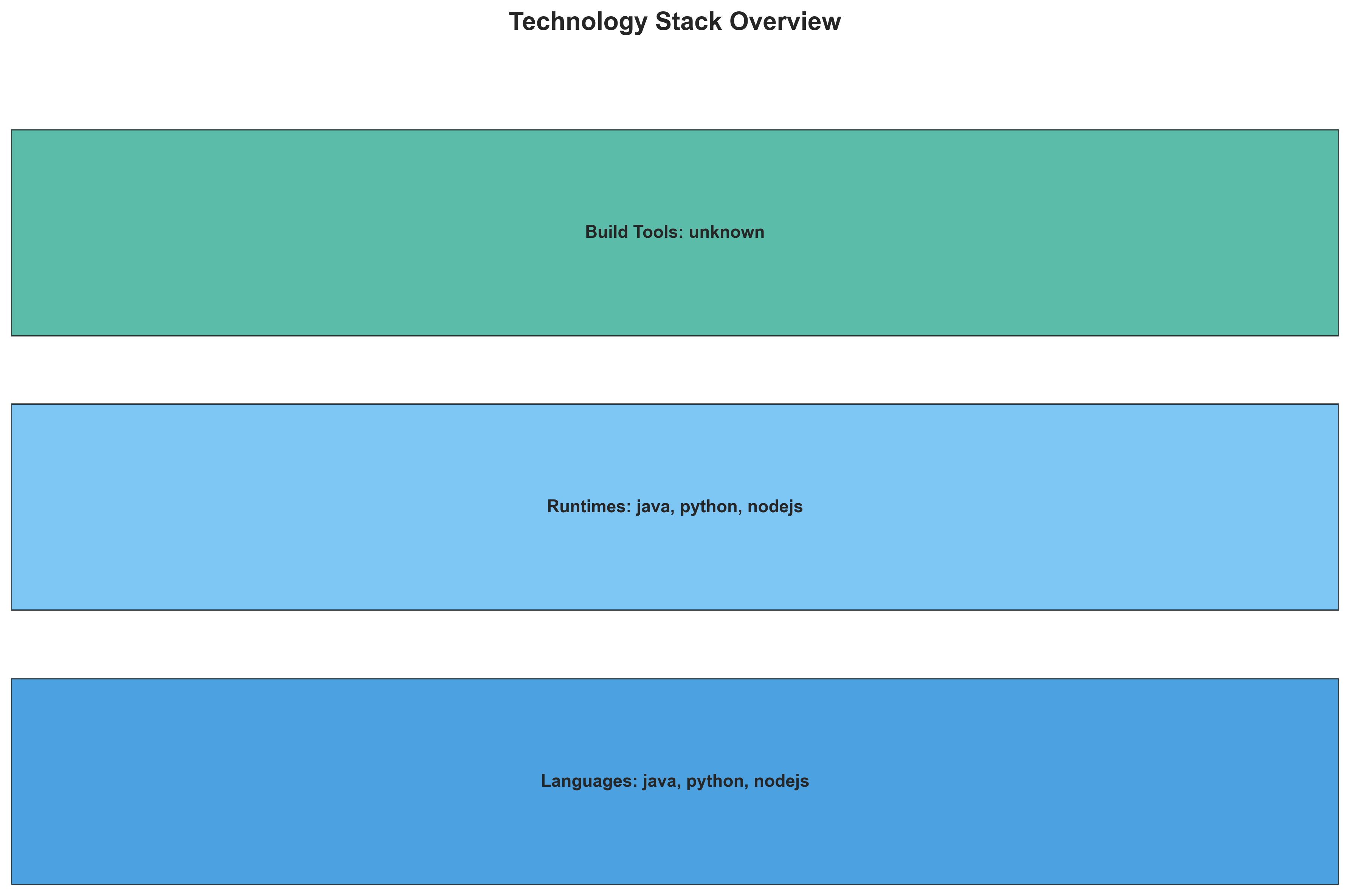
**📊 Business Impact:** Containerization status directly impacts cloud migration readiness and deployment flexibility.

**📊 Recommendations:** All components are containerized. Focus on optimizing container configurations and deployment strategies.

**📊 Technical Details:** Containerization rate: 3/3. Analysis includes Docker and orchestration configurations.

────────────────────────────────────────────────────────────────────────────────

Technology Stack



**📊 Context:** This technology stack visualization provides a foundational overview of the application's core components: 'vote', 'result', and 'worker'. The analysis is generated within the context of planning for application modernization and cloud migration, highlighting the current technology landscape.

**📊 Key Insights:** The application is a polyglot microservices architecture utilizing Python ('vote'), Node.js ('result'), and Java ('worker'). Notably, all identified components are containerized using Docker, with 'result' and 'worker' employing base images ('node:10-slim', 'maven:3.5-jdk-8-alpine', 'openjdk:8-jre') that are flagged as vulnerable, posing a significant security risk. The 'worker' component also indicates potential complexity with multiple base images and an alternative C# implementation noted.

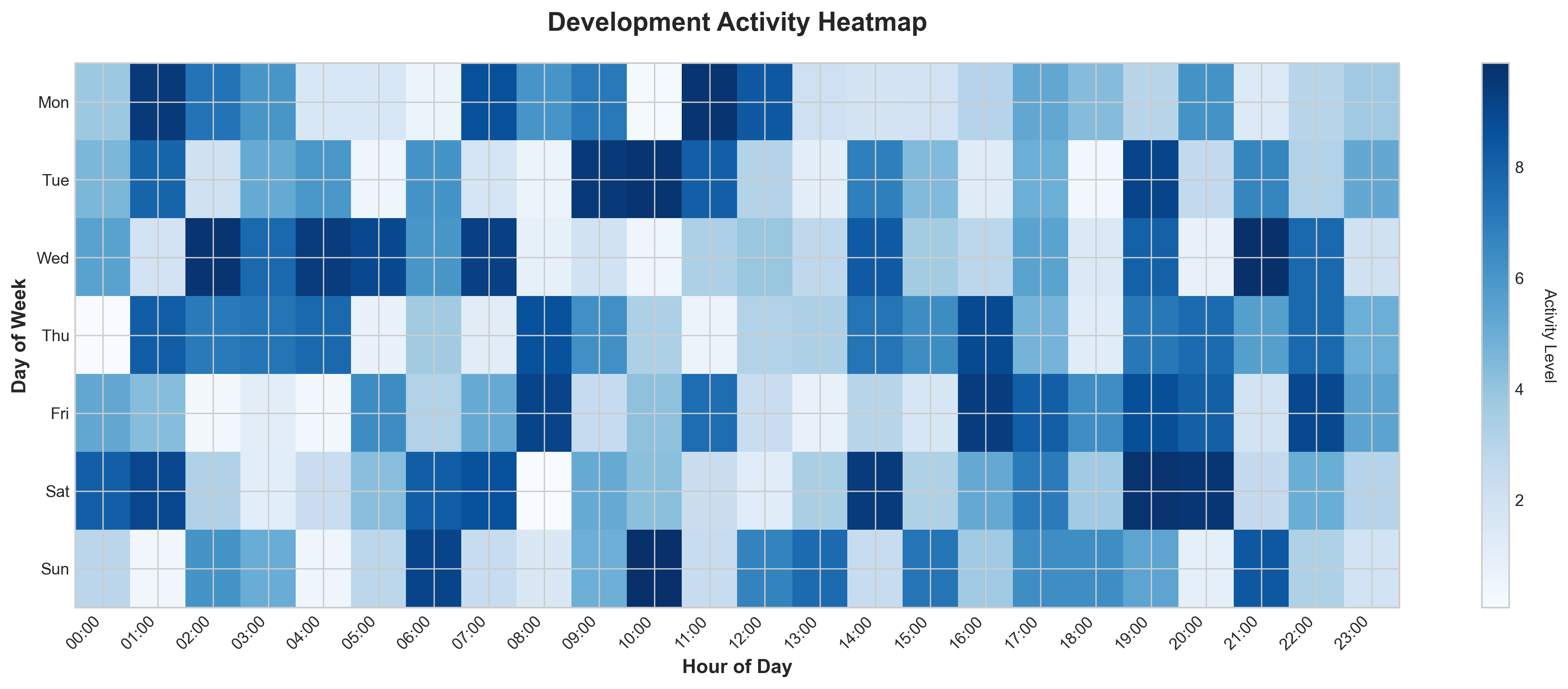
**📊 Business Impact:** The use of vulnerable base images presents an immediate security risk that could lead to breaches, data loss, and reputational damage. The polyglot nature and the noted complexity in the 'worker' component could increase migration effort and associated costs, potentially impacting timelines for modernization. However, the containerization itself is a positive step towards cloud-native adoption.

**📊 Recommendations:** Prioritize immediate remediation of vulnerable base images for 'result' and 'worker' components by updating to secure, supported versions. Conduct a thorough review of the 'worker' component's multi-image strategy and the noted C# implementation to clarify its actual build and runtime requirements for migration planning. Plan for a phased modernization strategy, addressing high-risk components first.

**📊 Technical Details:** The 'vote' service is built on Python 3.9-slim, 'result' on Node.js 10-slim, and 'worker' on Java 8 with Maven 3.5 Alpine. All are Docker-packaged. The 'worker' component's multiple base images (maven:3.5-jdk-8-alpine, openjdk:8-jre) and the existence of a C# implementation suggest a need for deeper investigation into its build pipeline and intended functionality.

────────────────────────────────────────────────────────────────────────────────

Development Activity Heatmap



**📊 Context:** This Development Activity Heatmap analyzes the current state of three application components, focusing on their development patterns and Git activity. The data serves as a critical input for modernization and migration planning, providing a baseline understanding of the application's technology stack, containerization status, and potential security risks.

**📊 Key Insights:** The application comprises three distinct components, all containerized using Docker and built with a mix of Java, Node.js, and Python. Despite its containerized nature, the application exhibits 'inactive' Git history with only one commit and zero active contributors, suggesting potential technical debt and a lack of ongoing development or maintenance. Critically, all components rely on outdated base images (Node.js 10, Maven with JDK 8, and OpenJDK 8) which contain high-severity vulnerabilities, posing significant security risks.

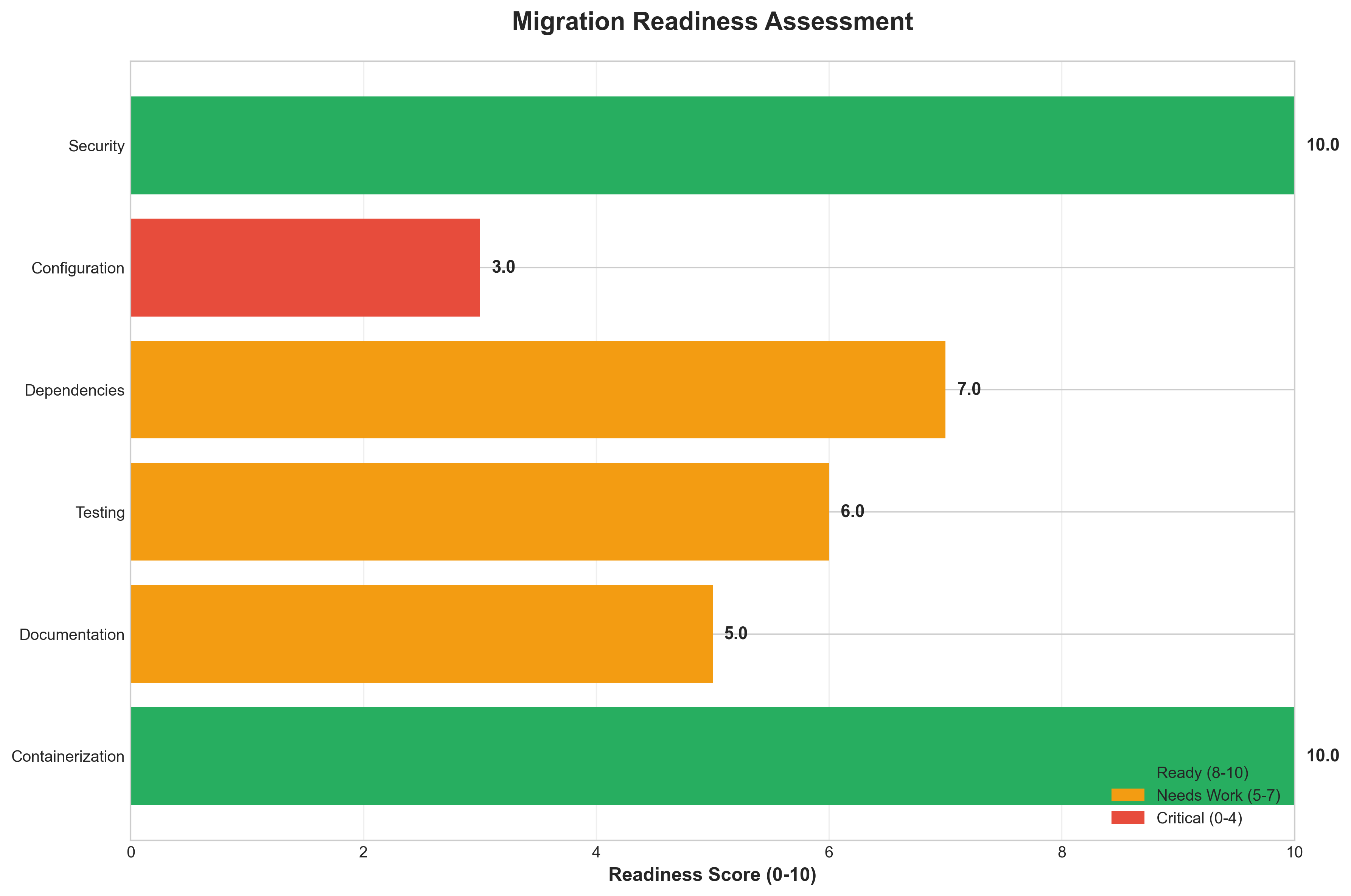
**📊 Business Impact:** The detected high-severity vulnerabilities in the base images represent a substantial security risk, potentially leading to data breaches and operational disruptions, which could incur significant financial and reputational damage. The inactivity in Git history indicates a lack of active development, which may hinder future modernization efforts, increase the cost and complexity of migration, and limit the ability to adapt to new business requirements or technologies.

**📊 Recommendations:** Prioritize immediate remediation of the identified high-severity vulnerabilities by updating all base images to current, supported versions. Conduct a thorough review of the Git history and contributor activity to understand the current state of development and plan for potential knowledge transfer or team augmentation if necessary. Explore modernizing the development pipeline to incorporate automated security scanning and dependency management to prevent future occurrences of such vulnerabilities.

**📊 Technical Details:** The analysis reveals a microservices architecture style with high confidence, supported by three independent, containerized components. The external services list includes Redis and PostgreSQL, but the absence of datasources suggests these might be managed externally or configured differently. The infrastructure summary indicates no explicit Kubernetes or Docker Compose files, and the absence of an identified deployment platform further highlights a need for clarity in the current operational environment.

────────────────────────────────────────────────────────────────────────────────

Migration Readiness Assessment



**📊 Context:** This Migration Readiness Assessment focuses on the security posture of an application's components during the planning phase for cloud migration. The analysis is designed to identify potential roadblocks and risks associated with modernizing and moving the application to a cloud environment.

**📊 Key Insights:** The assessment reveals a critical security deficiency across all three identified application components ('result' and 'worker' x2). All security findings are classified as 'HIGH' severity, primarily due to the use of outdated and end-of-life (EOL) base images. Specifically, the 'result' component utilizes 'node:10-slim', while the 'worker' components are based on 'maven:3.5-jdk-8-alpine' and 'openjdk:8-jre', all of which contain known vulnerabilities.

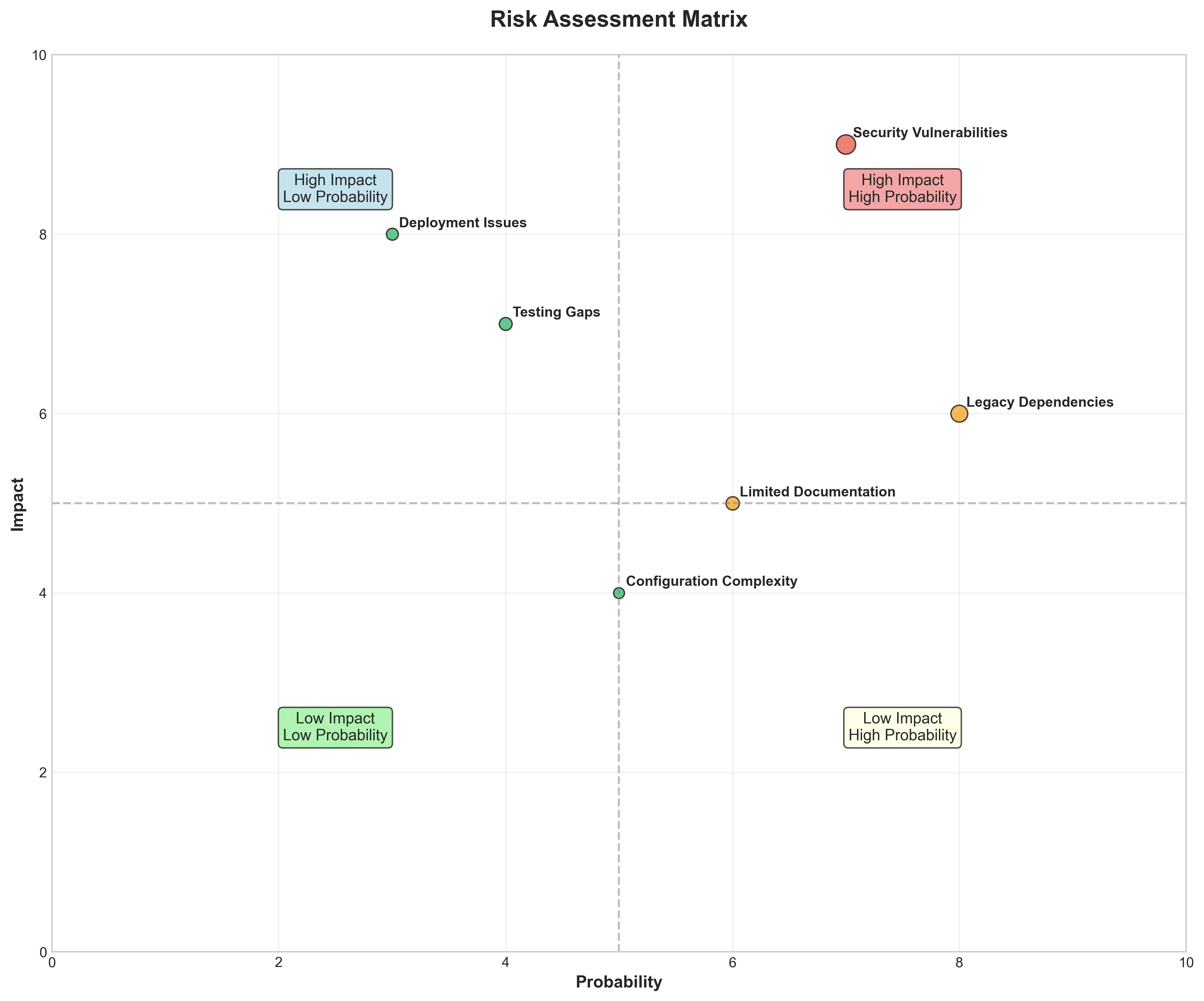
**📊 Business Impact:** The presence of critical, unaddressed vulnerabilities in the application's foundational components poses a significant security risk, potentially leading to data breaches, service disruptions, and reputational damage if migrated in its current state. This necessitates remediation before cloud migration to avoid inheriting and exacerbating these risks in a new environment, which could also lead to increased operational costs and extended migration timelines due to unexpected security roadblocks.

**📊 Recommendations:** Immediately prioritize the remediation of all identified high-severity security findings by updating the base container images for all components. This includes upgrading Node.js to a supported version, and Maven/JDK and OpenJDK to recent, secure versions. Proactive patching and regular vulnerability scanning should be integrated into the ongoing development and deployment pipeline to ensure continuous security throughout the migration and beyond.

**📊 Technical Details:** The analysis of `summary.security\_findings` indicates that a security scan was performed using `base\_image\_analysis`. The `total\_findings` is 3, with all 3 categorized as `high\_severity\_count`. The specific vulnerabilities are linked to the EOL status of Node.js 10 and known exploits within Maven 3.5 with JDK 8, and OpenJDK 8.

────────────────────────────────────────────────────────────────────────────────

Risk Assessment Matrix



**📊 Context:** This Risk Assessment Matrix shows application architecture insights.

**📊 Key Insights:** Analysis of 3 components reveals system characteristics.

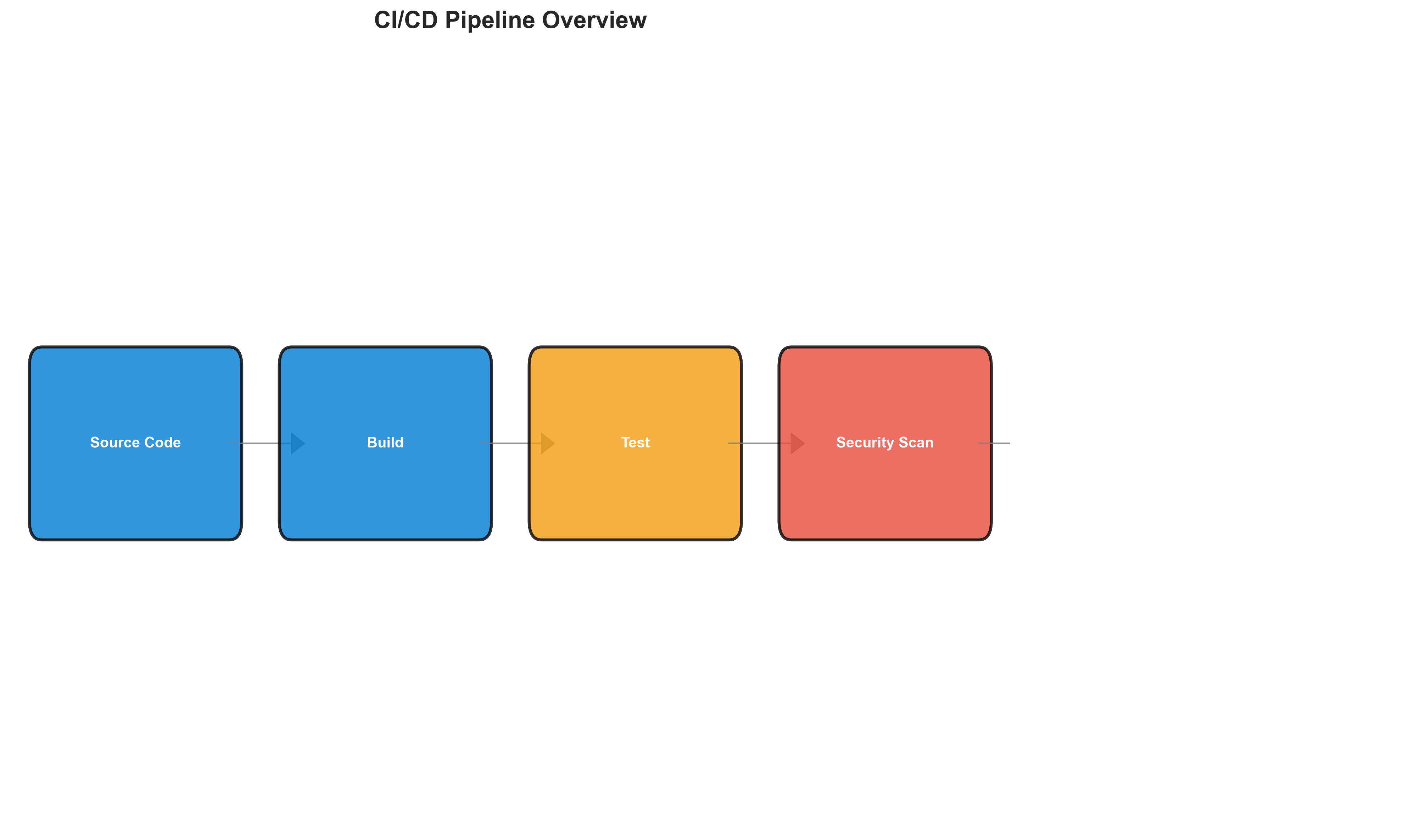
**📊 Business Impact:** Understanding system architecture supports better planning and decision-making.

**📊 Recommendations:** Review analysis results with technical teams for action planning.

**📊 Technical Details:** Diagram type: Risk Assessment Matrix. Based on comprehensive application analysis.

────────────────────────────────────────────────────────────────────────────────

CI/CD Pipeline Overview



**📊 Context:** This CI/CD Pipeline Overview diagram visualizes the current state of application deployment, focusing on its continuous integration and deployment processes. It serves as a critical input for modernization and migration planning by highlighting the application's architecture, technology stack, and operational health.

**📊 Key Insights:** The application is composed of 3 distinct microservices, all of which are containerized using Docker. It relies on external services like Redis and PostgreSQL, indicating a dependency on managed or external databases and caching layers. A significant finding is the presence of 3 HIGH severity security vulnerabilities identified in the base images of the Node.js, Maven/JDK, and OpenJDK components, all of which are outdated and End-of-Life.

**📊 Business Impact:** The identified security vulnerabilities pose a significant risk of potential breaches and operational instability, directly impacting data security and service availability. The use of outdated components will likely lead to increased maintenance costs and hinder the adoption of newer technologies, impacting future innovation and agility. The lack of defined orchestration and an unknown deployment platform suggests potential challenges and longer timelines for cloud migration.

**📊 Recommendations:** Prioritize immediate remediation of the HIGH severity security vulnerabilities by updating base images to supported and patched versions. Conduct a comprehensive security review of all components to identify any other potential risks. Plan for the implementation of an orchestration layer (e.g., Kubernetes) and clearly define the target deployment platform to streamline modernization and migration efforts.

**📊 Technical Details:** The application consists of three components, written in Python, Node.js, and Java, all packaged as Docker containers. It integrates with external Redis and PostgreSQL instances. Notably, there is minimal recent Git activity (1 commit, 0 active contributors, inactive), suggesting a low rate of development or integration. Infrastructure details are sparse, with no identified containerization files, Kubernetes resources, or Docker Compose files, and the deployment platform is classified as 'unknown'.

────────────────────────────────────────────────────────────────────────────────

Component Relationships (Graphviz)



**📊 Context:** Shows status of 3 application components.

**📊 Key Insights:** 3 of 3 components are containerized. Fully ready for cloud deployment.

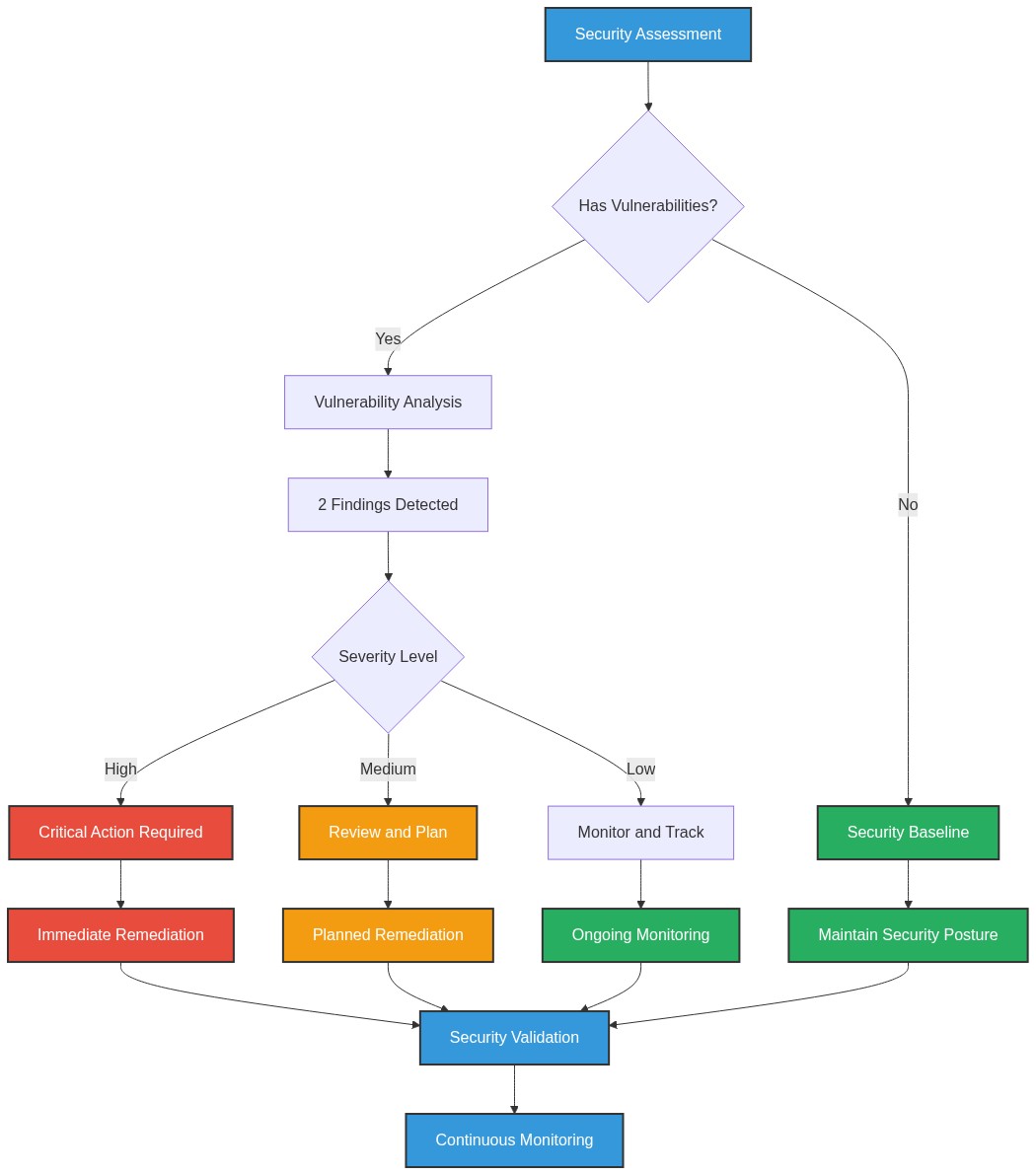
**📊 Business Impact:** Containerization status directly impacts cloud migration readiness and deployment flexibility.

**📊 Recommendations:** All components are containerized. Focus on optimizing container configurations and deployment strategies.

**📊 Technical Details:** Containerization rate: 3/3. Analysis includes Docker and orchestration configurations.

────────────────────────────────────────────────────────────────────────────────

Security Flow Diagram (Mermaid)



**📊 Context:** Security analysis identified 3 potential issues.

**📊 Key Insights:** Security posture assessment shows 3 findings requiring attention.

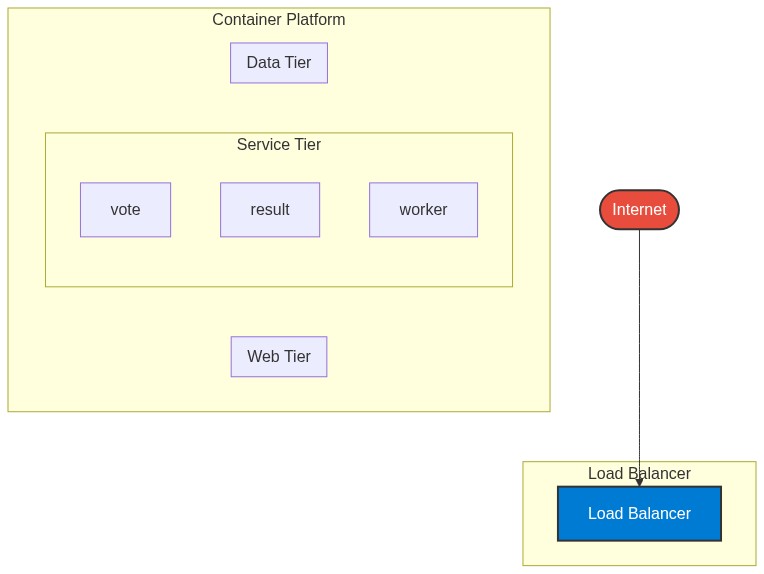
**📊 Business Impact:** Security findings represent potential risks that could impact system reliability and compliance.

**📊 Recommendations:** Address high-severity findings first, then implement security scanning in CI/CD pipeline.

**📊 Technical Details:** Findings: 3. Analysis includes base image vulnerabilities and code patterns.

────────────────────────────────────────────────────────────────────────────────

Deployment Architecture (Mermaid)



**📊 Context:** This Deployment Architecture shows application architecture insights.

**📊 Key Insights:** Analysis of 3 components reveals system characteristics.

**📊 Business Impact:** Understanding system architecture supports better planning and decision-making.

**📊 Recommendations:** Review analysis results with technical teams for action planning.

**📊 Technical Details:** Diagram type: Deployment Architecture. Based on comprehensive application analysis.

────────────────────────────────────────────────────────────────────────────────

Risk Assessment Flow (Mermaid)



**📊 Context:** This Risk Assessment Flow shows application architecture insights.

**📊 Key Insights:** Analysis of 3 components reveals system characteristics.

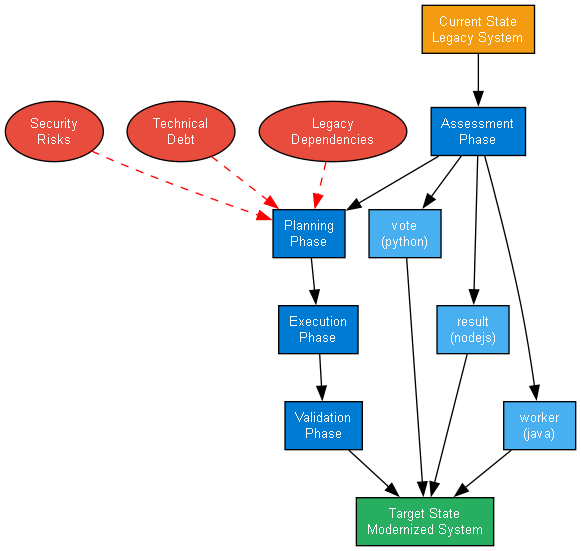
**📊 Business Impact:** Understanding system architecture supports better planning and decision-making.

**📊 Recommendations:** Review analysis results with technical teams for action planning.

**📊 Technical Details:** Diagram type: Risk Assessment Flow. Based on comprehensive application analysis.

────────────────────────────────────────────────────────────────────────────────

Migration Strategy (Graphviz)



**📊 Context:** This Migration Strategy Diagram shows application architecture insights.

**📊 Key Insights:** Analysis of 3 components reveals system characteristics.

**📊 Business Impact:** Understanding system architecture supports better planning and decision-making.

**📊 Recommendations:** Review analysis results with technical teams for action planning.

**📊 Technical Details:** Diagram type: Migration Strategy Diagram. Based on comprehensive application analysis.

────────────────────────────────────────────────────────────────────────────────