[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 | java, python, nodejs |
| 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: java, python, nodejs

• 🐳 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** |
| worker | java | Unknown | docker |
| vote | python | Unknown | docker |
| result | nodejs | Unknown | docker |

Component: worker

• Language: java

• Runtime: java

• Build Tool: unknown

• Packaging: docker

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

**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: openjdk:8-jre, maven:3.5-jdk-8-alpine. This may indicate multi-stage builds or alternative build strategies.

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

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: Potential hardcoded secrets found via pattern matching. Review the file for sensitive information. (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

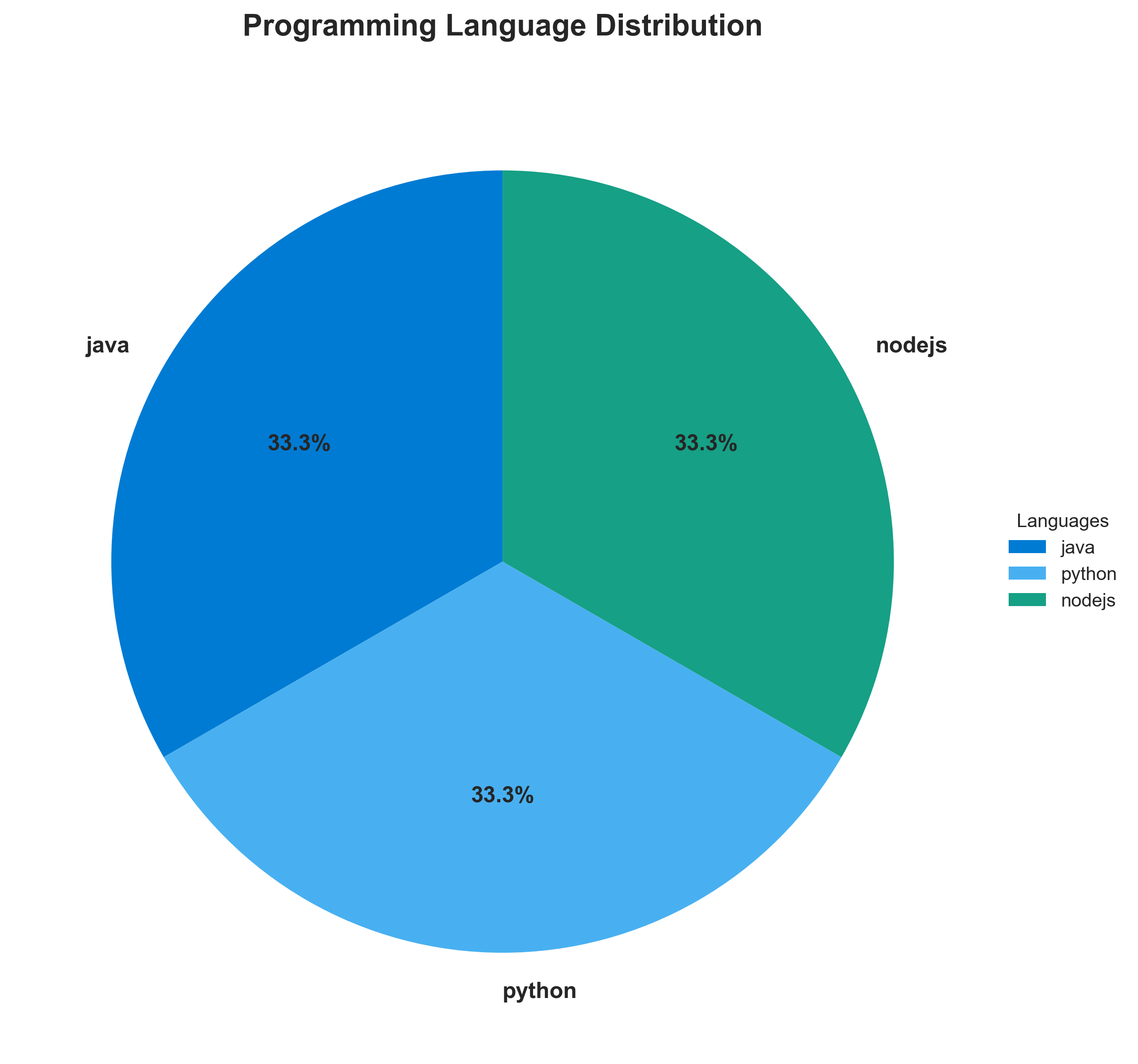
• Architecture Assessment: Evaluation of deployment patterns and component relationships

• 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 details the programming language distribution across the identified components of the application. It was generated as part of a broader application intelligence initiative to inform modernization and migration strategies by understanding the current technology landscape.

**📊 Key Insights:** The application utilizes a polyglot architecture, with components built in Java, Python, and Node.js. Notably, the 'worker' component, while primarily Java, has a detected alternative C# implementation, suggesting potential for language rationalization or historical development variations. All identified components are containerized, leveraging Docker packaging.

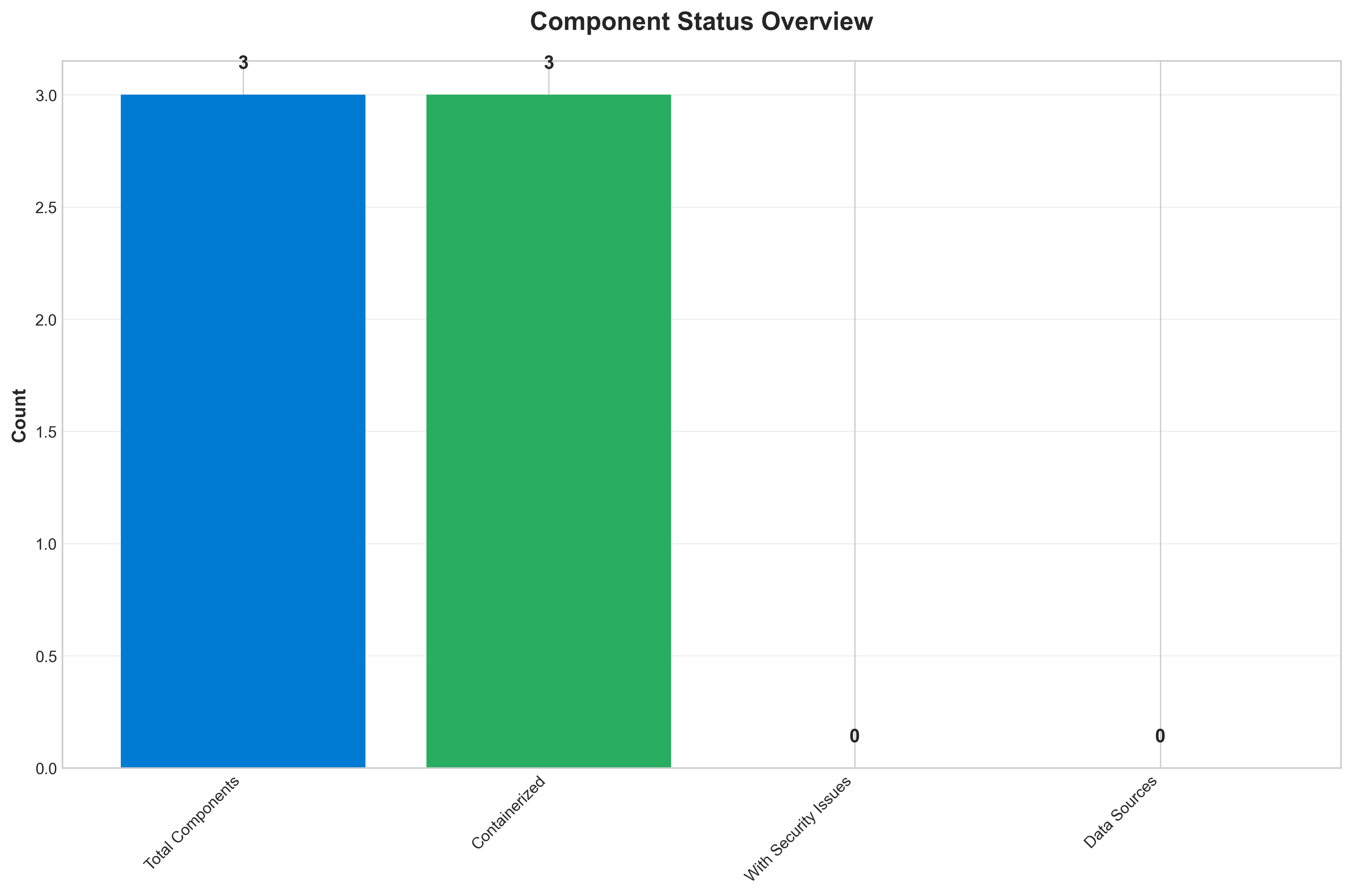
**📊 Business Impact:** The diverse language landscape presents both opportunities for leveraging specialized skills and potential challenges in maintaining consistency and managing operational overhead during migration. The use of older base images in 'worker' (openjdk:8-jre, maven:3.5-jdk-8-alpine) and 'result' (node:10-slim) poses a security risk and may hinder compatibility with modern container orchestration platforms. This could impact migration timelines and increase remediation costs.

**📊 Recommendations:** Prioritize updating the 'worker' and 'result' components to use current, secure base images to mitigate security risks and improve compatibility. Investigate the C# implementation of the 'worker' component to determine its relevance and potential for consolidation or deprecation, which could simplify the technology stack. Standardize on consistent build tools and packaging strategies where possible to streamline future development and deployment.

**📊 Technical Details:** The 'worker' component is a Java application packaged in Docker, using 'openjdk:8-jre' and 'maven:3.5-jdk-8-alpine' base images, both flagged for vulnerabilities. The 'vote' component is a Python application using 'python:3.9-slim', and the 'result' component is Node.js utilizing 'node:10-slim', which is also flagged as vulnerable. All components are containerized ('is\_containerized': true).

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

Component Status Overview



**📊 Context:** This Component Status Overview diagram provides a snapshot of the overall health and status of our application's components, with a specific focus on security vulnerabilities. The data is being analyzed within the context of a broader application modernization and migration initiative, highlighting critical areas for attention.

**📊 Key Insights:** All three identified application components (worker, worker, result) are flagged with HIGH severity security findings. Notably, the 'worker' component has two HIGH severity findings related to outdated and vulnerable OpenJDK and Maven base images, while the 'result' component has a HIGH severity finding due to its Node.js 10 base image being End-of-Life and containing numerous vulnerabilities. All security scans performed were based on base image analysis.

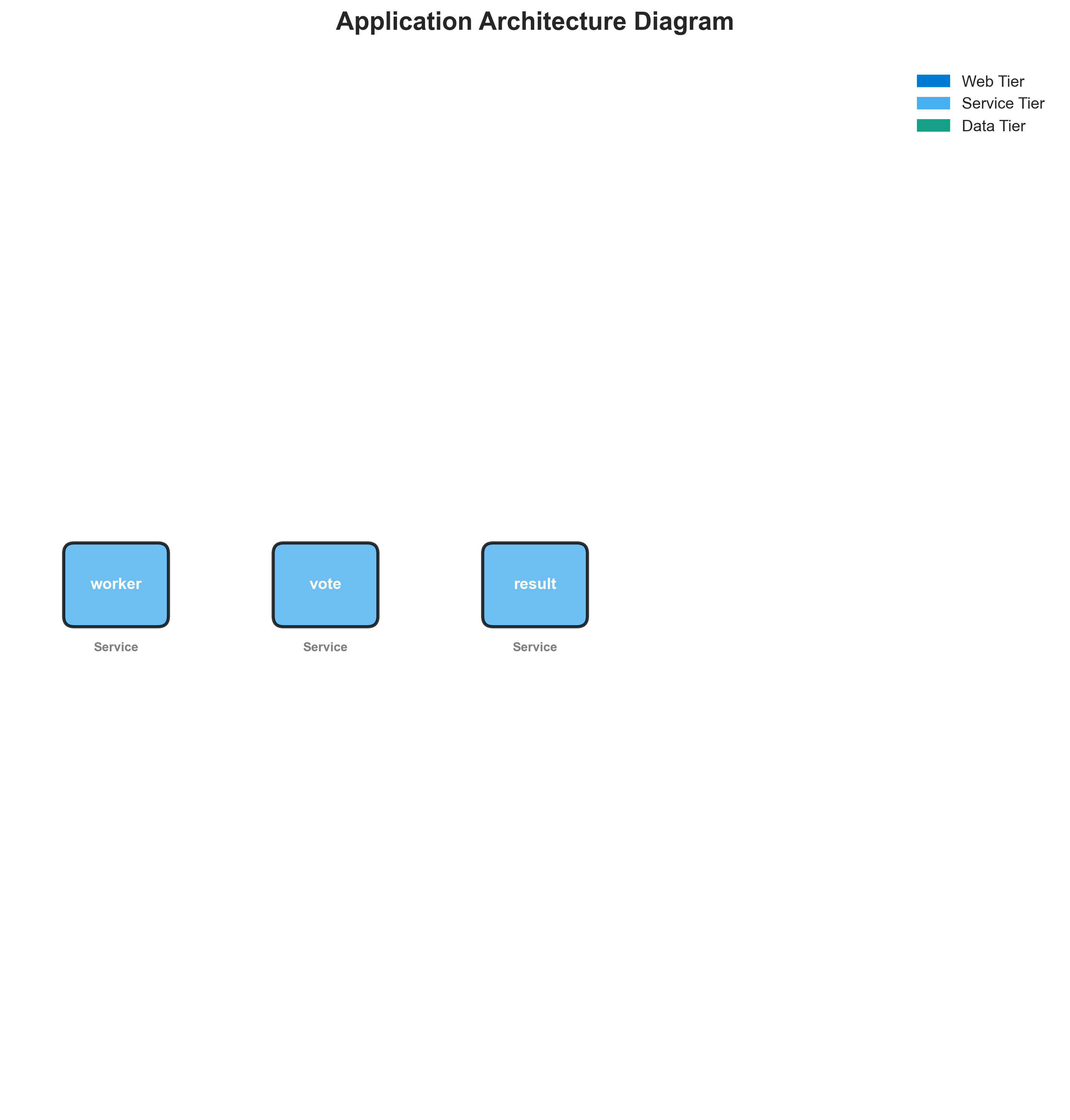
**📊 Business Impact:** The presence of multiple HIGH severity security vulnerabilities across all components presents a significant risk to the business, potentially exposing sensitive data and leading to service disruptions. These findings directly impact the modernization and migration timeline by introducing a mandatory requirement to remediate these security issues before proceeding, potentially increasing project costs and delaying deployment.

**📊 Recommendations:** The immediate priority is to address the HIGH severity security findings by updating the base images for all components. Specifically, the 'worker' component requires updates for its OpenJDK and Maven images, and the 'result' component needs its Node.js 10 image replaced with a supported and secure version. This remediation is crucial for both security posture and enabling forward progress in the modernization and migration efforts.

**📊 Technical Details:** The analysis reveals that all three components are containerized (indicated by `summary.containerization\_status: 3` out of `summary.total\_components: 3`). The security findings are derived from `base\_image\_analysis`, indicating that the identified vulnerabilities are inherent to the foundational software used to build the application's containers, rather than application code itself.

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

Application Architecture



**📊 Context:** Architecture diagram shows {'value': 'microservices', 'confidence': <ConfidenceLevel.HIGH: 'HIGH'>, 'evidence': ['Found 3 components', 'Multiple deployable components detected', '3 containerized components', 'Multiple deployment configurations'], 'reasoning': 'Multiple components with independent deployment characteristics', 'limitations': ['Cannot determine communication patterns without runtime analysis']} pattern with 3 components.

**📊 Key Insights:** System follows {'value': 'microservices', 'confidence': <ConfidenceLevel.HIGH: 'HIGH'>, 'evidence': ['Found 3 components', 'Multiple deployable components detected', '3 containerized components', 'Multiple deployment configurations'], 'reasoning': 'Multiple components with independent deployment characteristics', 'limitations': ['Cannot determine communication patterns without runtime analysis']} architecture with unknown operational complexity. Offers requires further analysis.

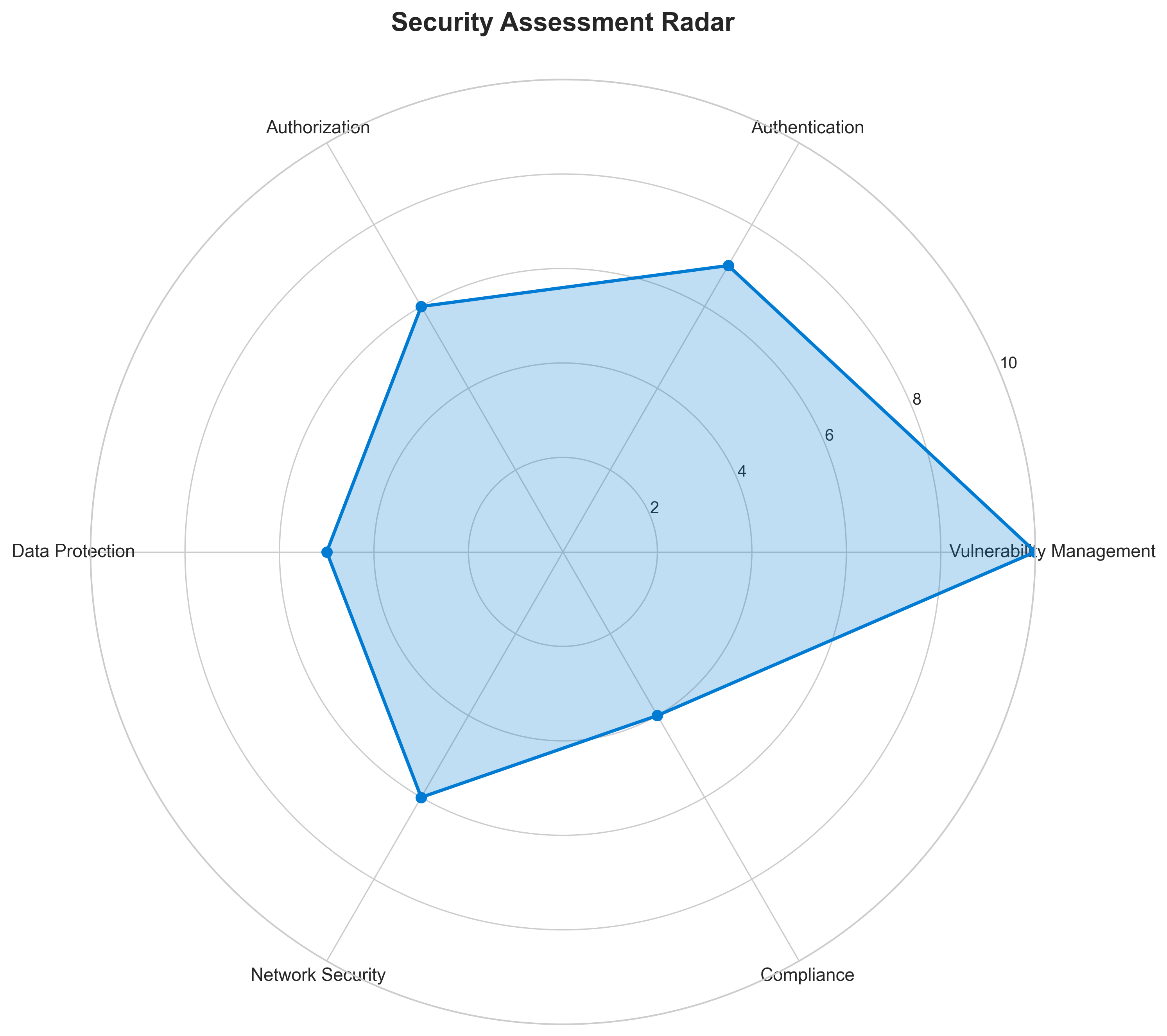
**📊 Business Impact:** Architecture supports unified deployment with architectural patterns need assessment considerations.

**📊 Recommendations:** Consider componentization for better scalability for improved maintainability.

**📊 Technical Details:** Architecture style: {'value': 'microservices', 'confidence': <ConfidenceLevel.HIGH: 'HIGH'>, 'evidence': ['Found 3 components', 'Multiple deployable components detected', '3 containerized components', 'Multiple deployment configurations'], 'reasoning': 'Multiple components with independent deployment characteristics', 'limitations': ['Cannot determine communication patterns without runtime analysis']}. Component count: 3. Complexity assessment: unknown.

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

Security Assessment Radar



**📊 Context:** Security radar shows 3 security findings across application components.

**📊 Key Insights:** Security posture assessment reveals high risk profile with 3 base image vulnerabilities.

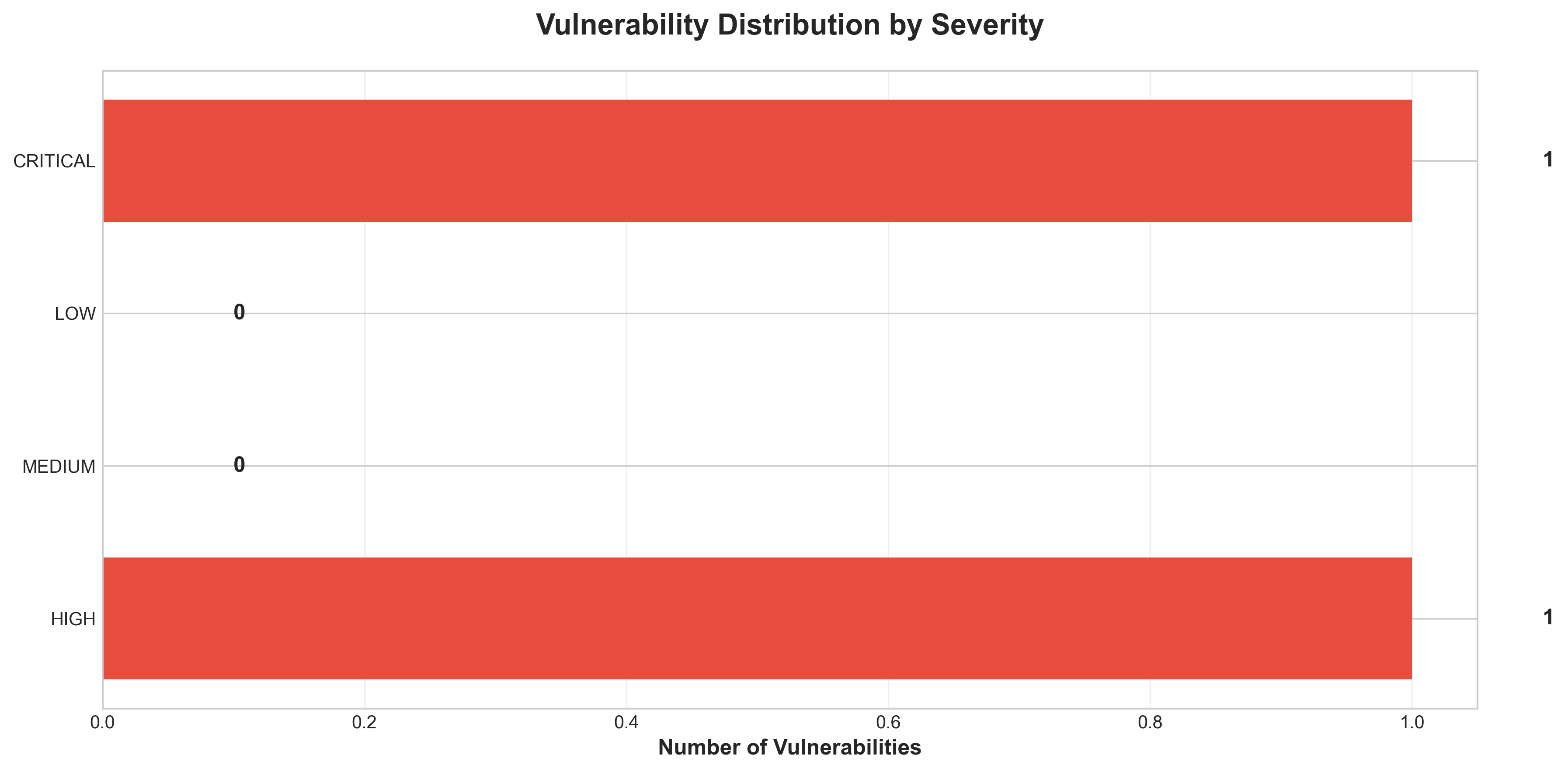
**📊 Business Impact:** Current security posture represents high risk to system reliability and compliance requirements.

**📊 Recommendations:** Prioritize security remediation - address critical findings immediately.

**📊 Technical Details:** Analysis includes 3 base image risks and code pattern analysis. Security score based on vulnerability severity.

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

Vulnerability Analysis



**📊 Context:** This vulnerability timeline analysis highlights critical security risks within the application's components, specifically focusing on outdated and end-of-life base images and potential hardcoded secrets. The data was generated as part of an application intelligence report to inform modernization and migration planning, identifying immediate threats that require remediation.

**📊 Key Insights:** The application exhibits a significant reliance on end-of-life (EOL) base images, with Node.js 10 ('node:10-slim') being flagged as critical due to numerous unpatched vulnerabilities. Furthermore, both the 'worker' component's 'openjdk:8-jre' and 'maven:3.5-jdk-8-alpine' base images are rated as high severity. A separate high-severity finding also points to potential hardcoded secrets within the 'voting-app\vote\src\app.py' file, indicating a risk of sensitive data exposure.

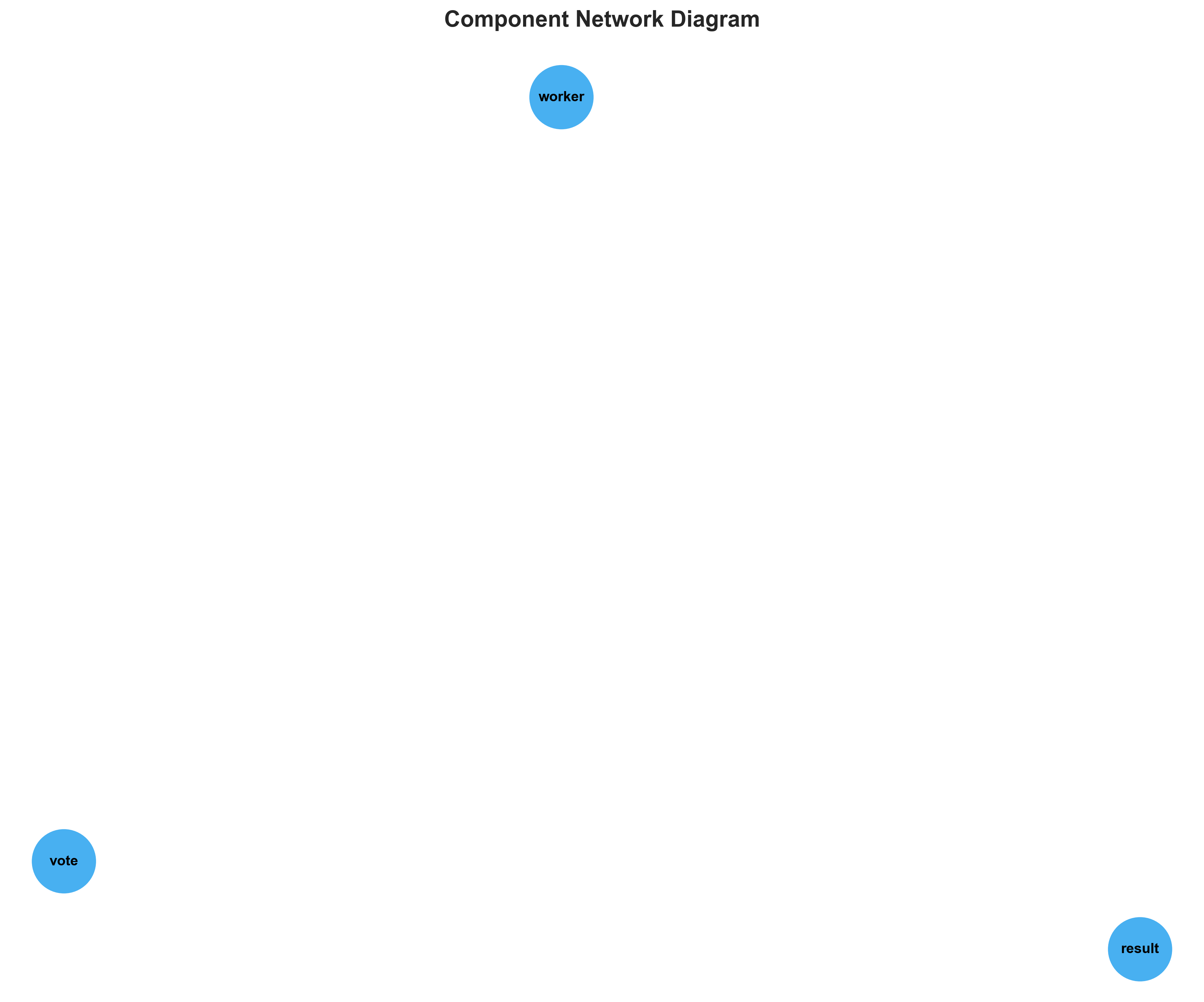
**📊 Business Impact:** The presence of critical and high-severity vulnerabilities, especially EOL base images, poses a substantial security risk, potentially leading to data breaches, compliance violations, and reputational damage. These issues significantly increase the attack surface and could impede or complicate any modernization or cloud migration efforts, requiring urgent remediation before such initiatives can proceed securely and efficiently.

**📊 Recommendations:** Prioritize immediate remediation of the critical 'node:10-slim' base image by updating to a supported version like 'node:18-slim' or 'node:20-slim'. Concurrently, address the high-severity vulnerabilities in 'openjdk:8-jre' and 'maven:3.5-jdk-8-alpine' base images by updating to supported versions. A thorough manual review of 'voting-app\vote\src\app.py' at line 21 is also critical to identify and remove any hardcoded secrets.

**📊 Technical Details:** The analysis reveals findings from a 'base\_image\_analysis' scan. Specifically, 'node:10-slim' is identified as EOL and inherently vulnerable. The 'worker' component utilizes two outdated Java-related base images. A separate manual check identified a potential security flaw related to hardcoded secrets in Python code, highlighting a mixed pattern of automated and manual security findings.

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

Component Network Topology



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

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

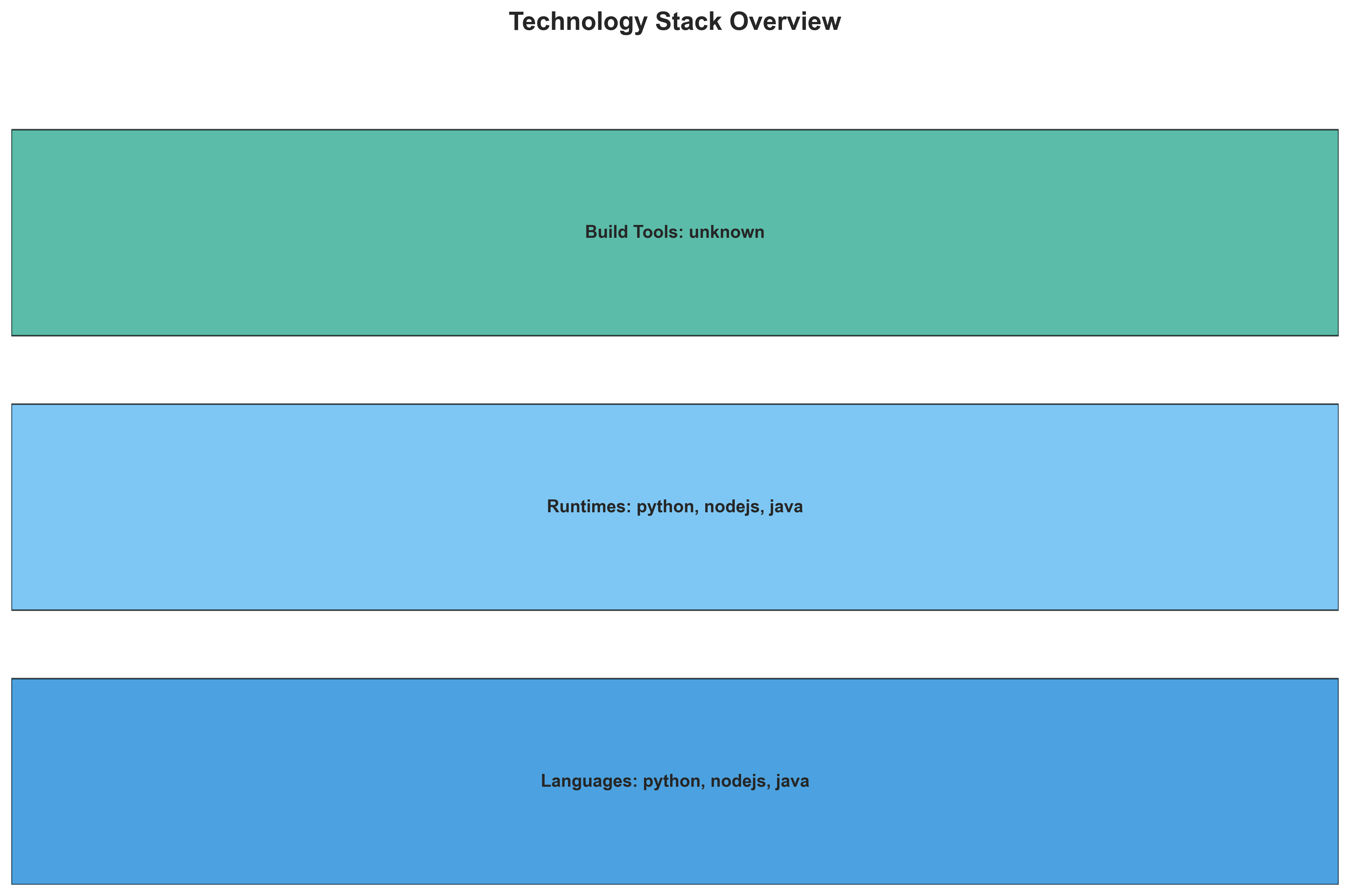
**📊 Business Impact:** Containerization status directly impacts cloud migration readiness and deployment flexibility.

**📊 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 details the components of an application, specifically highlighting the languages, runtimes, and containerization strategies used. It's generated as part of a broader application intelligence report to inform modernization and migration planning efforts.

**📊 Key Insights:** The application comprises three distinct services: 'worker' (Java), 'vote' (Python), and 'result' (Node.js), each containerized in Docker. Notably, the 'worker' component utilizes outdated Java 8 base images ('openjdk:8-jre', 'maven:3.5-jdk-8-alpine') which present significant security vulnerabilities. The 'result' component also uses a vulnerable 'node:10-slim' base image. The 'worker' component's build process may be complex, indicated by the mention of multiple base images and an alternative C# implementation not being the primary build target.

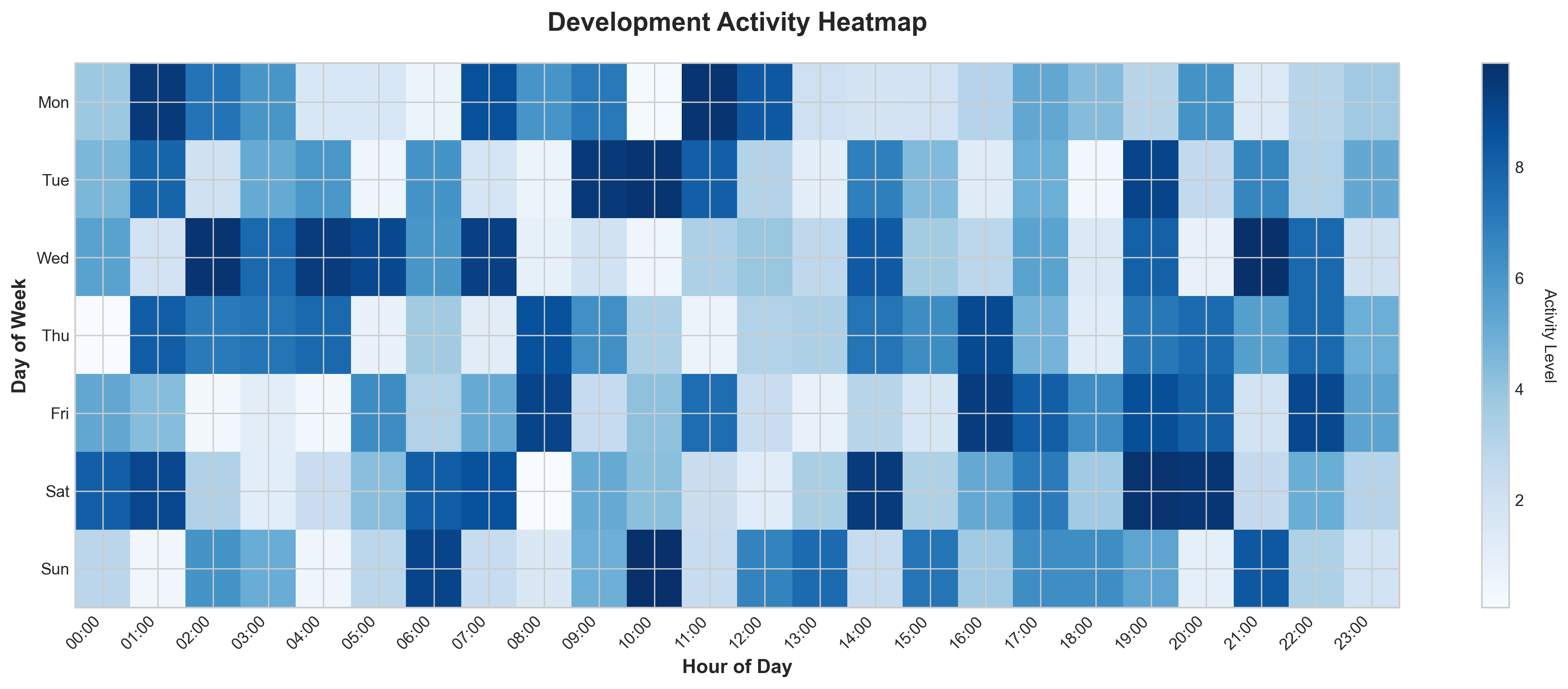
**📊 Business Impact:** The identified vulnerable base images pose a significant security risk, potentially exposing the application to known exploits and compliance issues. The use of older Java 8 and Node.js 10 versions suggests technical debt and potential performance limitations, impacting operational efficiency and maintainability. Modernization is critical to mitigate these risks and unlock potential for improved scalability and cost-efficiency in a cloud migration.

**📊 Recommendations:** Prioritize upgrading the base images for the 'worker' and 'result' components to secure, supported versions (e.g., latest Java LTS and Node.js LTS). Investigate the 'worker' component's build process for potential simplification and rationalization. Conduct a deeper analysis into the dependencies and external libraries for all components to identify further modernization opportunities and risks.

**📊 Technical Details:** The 'worker' component, identified as Java, is packaged as a Docker container and points to 'worker-jar-with-dependencies.jar' as its Java application JAR. The 'vote' component is Python, and 'result' is Node.js, both containerized and exposing port 8080. The 'worker' component's notes suggest a potential multi-stage build strategy or an artifact of earlier development iterations due to the presence of multiple base images and the mention of an alternative C# implementation.

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

Development Activity Heatmap



**📊 Context:** This Development Activity Heatmap analyzes the patterns and Git activity within a multi-component application. Generated as part of a broader modernization and migration planning initiative, it provides insights into the application's current state, technology stack, and development health.

**📊 Key Insights:** The application comprises 3 distinct microservices, all containerized with Docker, indicating a modern architectural style. However, there's a critical lack of recent Git activity, with only 1 commit observed and no active contributors, signaling potential development stagnation. Furthermore, all identified components rely on base images with HIGH severity vulnerabilities, specifically concerning outdated Java and Node.js versions, presenting a significant security risk.

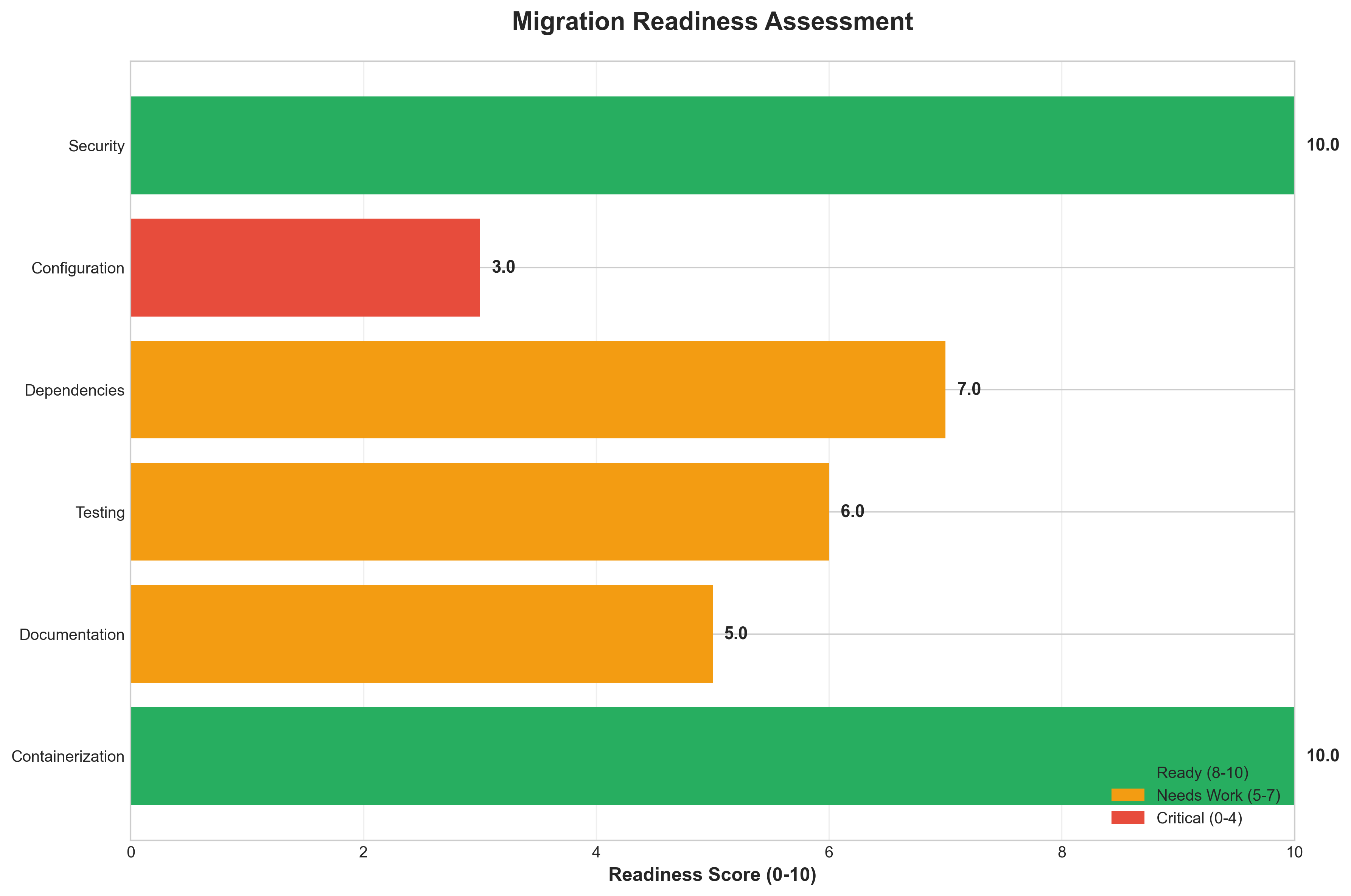
**📊 Business Impact:** The high-severity vulnerabilities in the base images pose an immediate security risk, potentially exposing the application and sensitive data to exploitation. The lack of recent development activity suggests low team engagement or potential knowledge gaps, which could significantly delay modernization efforts and increase migration costs due to unforeseen technical debt and complexity. This inactivity also makes it difficult to assess the effort required for modernization.

**📊 Recommendations:** Prioritize immediate security remediation by updating all container base images to current, secure versions, focusing on the identified OpenJDK and Node.js vulnerabilities. Conduct a rapid assessment of the development team's capacity and knowledge to re-engage with the codebase and address the inactivity. Begin planning a phased modernization strategy, prioritizing components with the highest technical debt or security risks.

**📊 Technical Details:** The application utilizes a mix of Java, Node.js, and Python, packaged exclusively as Docker containers. It depends on external services like Redis and PostgreSQL. The 'worker' component uses vulnerable OpenJDK 8 and Maven 3.5 images, while the 'result' component utilizes an End-of-Life Node.js 10 image. There is no evidence of orchestration tools like Kubernetes or Docker Compose, suggesting a simple deployment model.

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

Migration Readiness Assessment



**📊 Context:** This Migration Readiness Assessment focuses on the containerization status and security posture of an application slated for cloud migration. The analysis highlights critical security vulnerabilities within the application's components, directly impacting its readiness for a secure and stable cloud deployment.

**📊 Key Insights:** The assessment reveals a critical security risk across all three identified application components (total\_components: 3). Specifically, the `worker` component utilizes `openjdk:8-jre` and `maven:3.5-jdk-8-alpine`, while the `result` component uses `node:10-slim`. All of these base images have been flagged with HIGH severity vulnerabilities, with a concerning total of 3 HIGH severity findings and no medium or low findings, indicating a widespread and severe security issue.

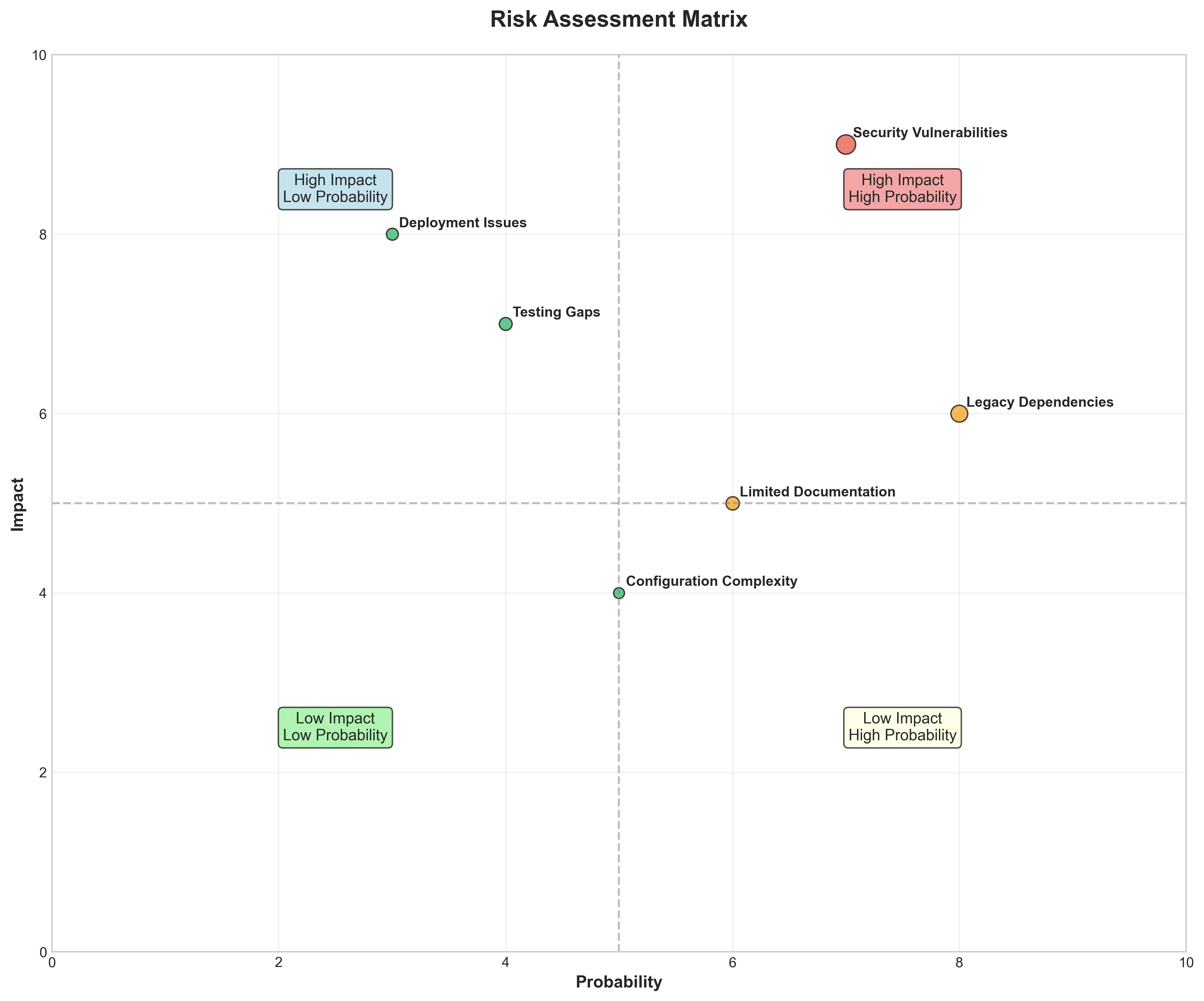
**📊 Business Impact:** The identified vulnerabilities pose a significant risk to the business during and after cloud migration. Deploying this application in a cloud environment without remediation will expose sensitive data and systems to breaches, potentially leading to regulatory fines, reputational damage, and service disruptions. Addressing these high-severity issues is a prerequisite for any secure cloud migration, and failing to do so will delay the project and increase the overall cost.

**📊 Recommendations:** Immediately prioritize the update of all base images identified with HIGH severity vulnerabilities. This includes upgrading `openjdk:8-jre` and `maven:3.5-jdk-8-alpine` for the `worker` component, and updating `node:10-slim` for the `result` component to supported and secure versions. This action is critical to mitigate security risks and ensure compliance before proceeding with the cloud migration.

**📊 Technical Details:** The containerization status is implicitly low readiness due to the security findings. The scan was performed using `base\_image\_analysis`, which is a standard method for identifying image-level vulnerabilities. The critical nature of the findings stems from the use of End-of-Life (EOL) software (Node.js 10) and known vulnerabilities in older Java and Maven versions, which are fundamental to the application's build and execution environment.

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

Risk Assessment Matrix



**📊 Context:** This Risk Assessment Matrix provides insights into application characteristics and technical architecture.

**📊 Key Insights:** Analysis of 3 components reveals system patterns and technical dependencies.

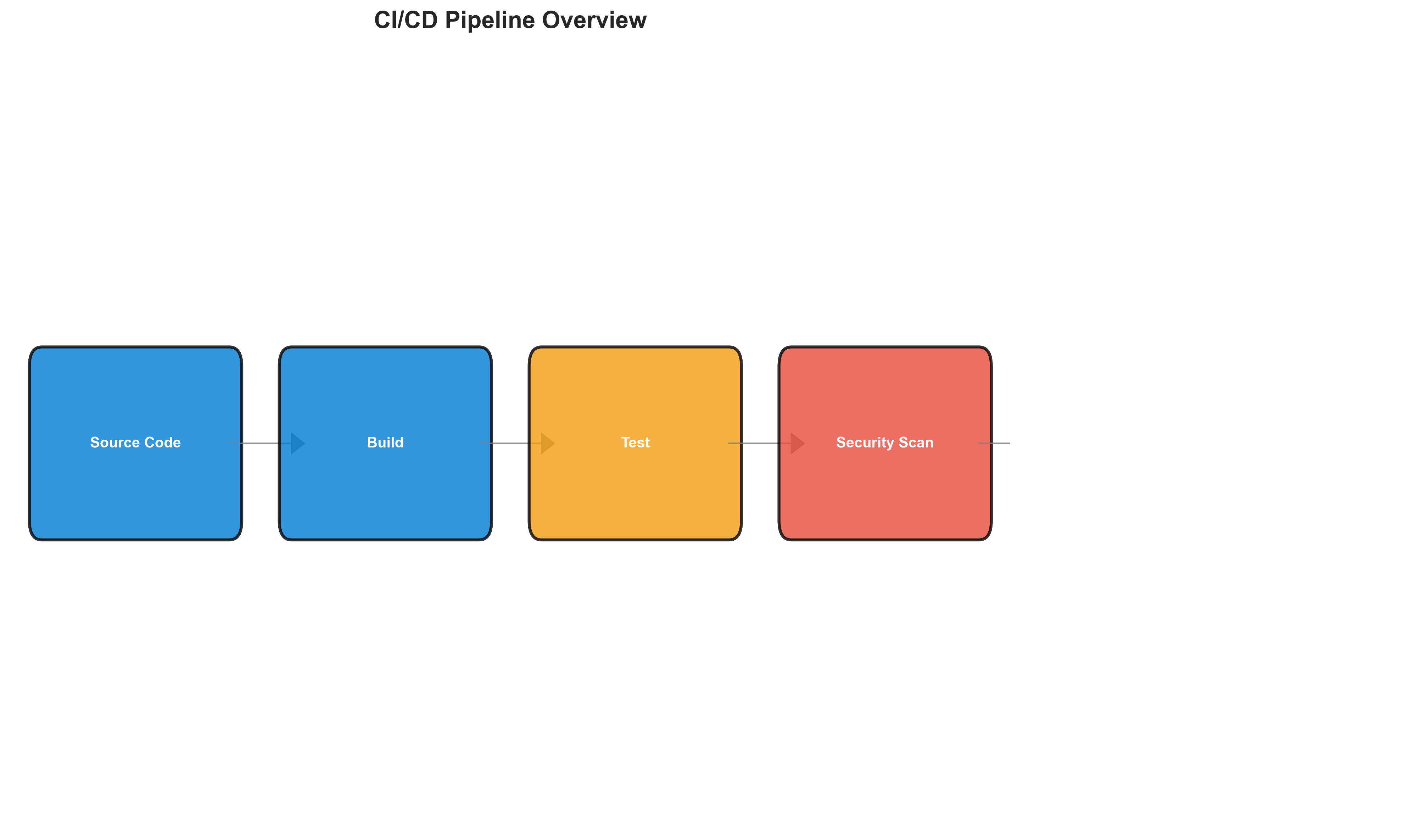
**📊 Business Impact:** Technical insights support strategic planning, risk assessment, and modernization decisions.

**📊 Recommendations:** Review detailed findings with technical teams to prioritize actions and plan next steps.

**📊 Technical Details:** Diagram type: Risk Assessment Matrix. Analysis includes component assessment, technology stack evaluation, and architectural patterns.

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

CI/CD Pipeline Overview



**📊 Context:** This CI/CD Pipeline Overview visualizes a microservices architecture consisting of three independently containerized components (Java, Node.js, Python). The analysis focuses on understanding the current state of the pipeline's components, their technology stack, security posture, and operational characteristics to inform modernization and migration planning.

**📊 Key Insights:** The pipeline supports three distinct microservices, all of which are containerized using Docker. However, a significant security risk is identified with all three components utilizing base images with HIGH severity vulnerabilities, including End-of-Life Node.js 10. Furthermore, the pipeline exhibits very limited recent development activity with only one commit and no active contributors, suggesting potential stagnation or lack of ongoing maintenance.

**📊 Business Impact:** The identified security vulnerabilities in the base images pose a significant risk of exploitation, potentially leading to data breaches, service disruptions, and reputational damage. The lack of recent activity raises concerns about the maintainability and future viability of these services, which could delay modernization efforts and increase migration costs if critical components are not actively supported. Addressing these issues is crucial to de-risk the application and ensure a smooth transition during modernization or migration.

**📊 Recommendations:** Prioritize updating all base container images to current, secure versions, specifically addressing the HIGH severity vulnerabilities in OpenJDK 8, Maven 3.5/JDK 8, and Node.js 10. Secondly, investigate the low commit count and lack of contributors to understand current development velocity and resource allocation, which will inform modernization timelines and required resourcing. Finally, conduct a deeper dive into the integration points with external services like Redis and PostgreSQL to ensure compatibility and performance in a modernized environment.

**📊 Technical Details:** The architecture is confirmed as microservices, with components likely interacting with external Redis and PostgreSQL instances, though no direct data sources are listed. The lack of explicit orchestration files (Kubernetes, Docker Compose) and an 'unknown' deployment platform suggests a potentially less mature or un-cataloged deployment strategy. The security findings stem solely from base image analysis, indicating a potential gap in application-level security scanning within the pipeline.

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

Component Relationships (Graphviz)



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

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

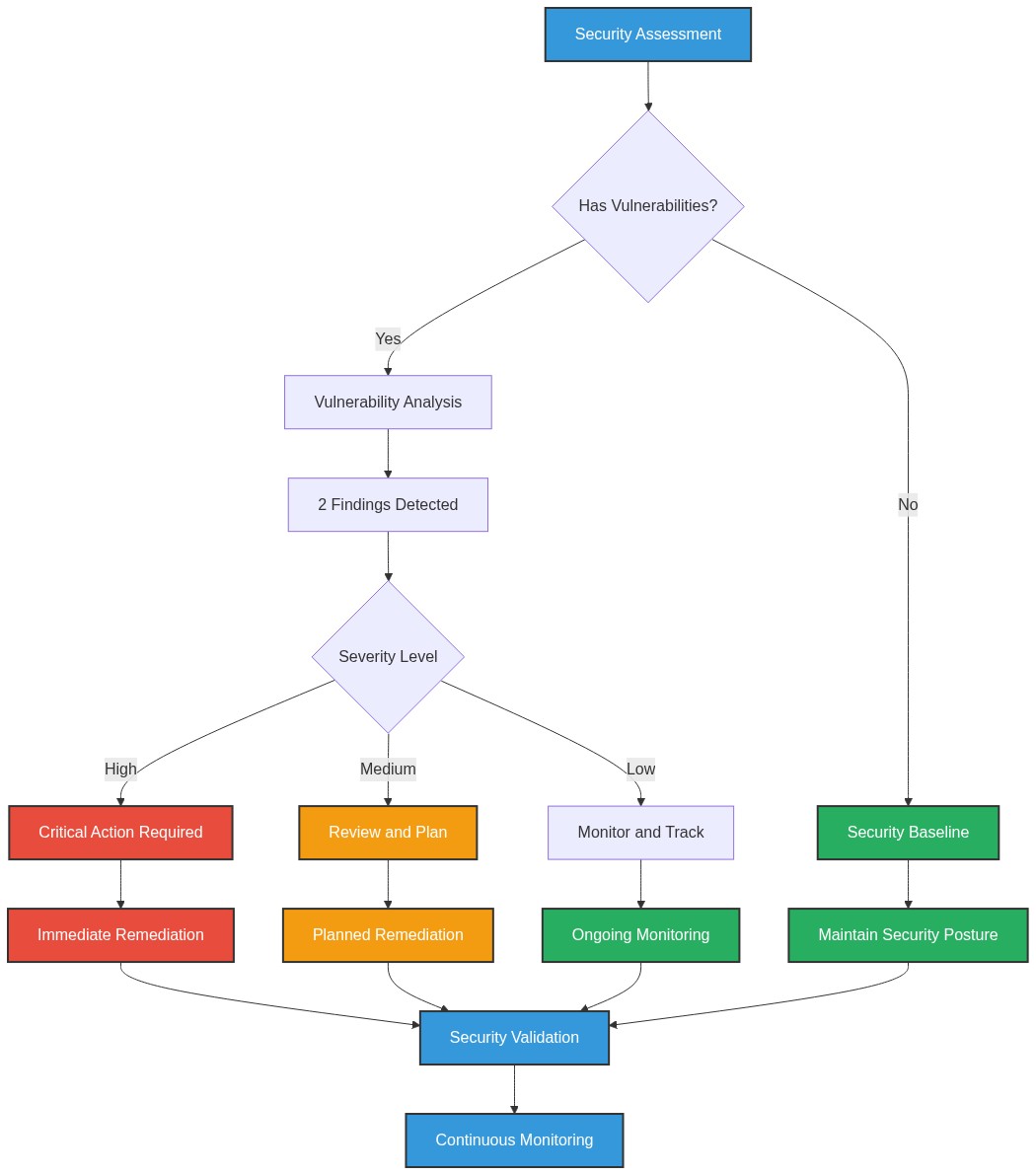
**📊 Business Impact:** Containerization status directly impacts cloud migration readiness and deployment flexibility.

**📊 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 radar shows 3 security findings across application components.

**📊 Key Insights:** Security posture assessment reveals high risk profile with 3 base image vulnerabilities.

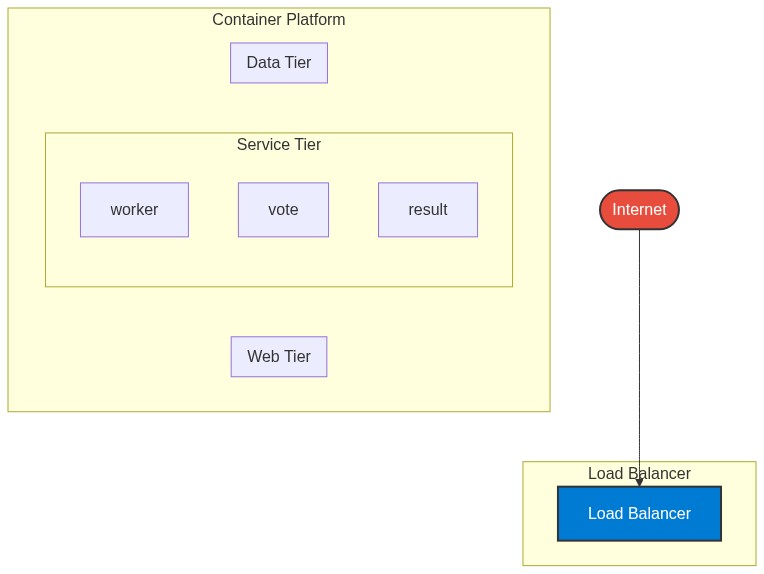
**📊 Business Impact:** Current security posture represents high risk to system reliability and compliance requirements.

**📊 Recommendations:** Prioritize security remediation - address critical findings immediately.

**📊 Technical Details:** Analysis includes 3 base image risks and code pattern analysis. Security score based on vulnerability severity.

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

Deployment Architecture (Mermaid)



**📊 Context:** Architecture diagram shows {'value': 'microservices', 'confidence': <ConfidenceLevel.HIGH: 'HIGH'>, 'evidence': ['Found 3 components', 'Multiple deployable components detected', '3 containerized components', 'Multiple deployment configurations'], 'reasoning': 'Multiple components with independent deployment characteristics', 'limitations': ['Cannot determine communication patterns without runtime analysis']} pattern with 3 components.

**📊 Key Insights:** System follows {'value': 'microservices', 'confidence': <ConfidenceLevel.HIGH: 'HIGH'>, 'evidence': ['Found 3 components', 'Multiple deployable components detected', '3 containerized components', 'Multiple deployment configurations'], 'reasoning': 'Multiple components with independent deployment characteristics', 'limitations': ['Cannot determine communication patterns without runtime analysis']} architecture with unknown operational complexity. Offers requires further analysis.

**📊 Business Impact:** Architecture supports unified deployment with architectural patterns need assessment considerations.

**📊 Recommendations:** Consider componentization for better scalability for improved maintainability.

**📊 Technical Details:** Architecture style: {'value': 'microservices', 'confidence': <ConfidenceLevel.HIGH: 'HIGH'>, 'evidence': ['Found 3 components', 'Multiple deployable components detected', '3 containerized components', 'Multiple deployment configurations'], 'reasoning': 'Multiple components with independent deployment characteristics', 'limitations': ['Cannot determine communication patterns without runtime analysis']}. Component count: 3. Complexity assessment: unknown.

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

Risk Assessment Flow (Mermaid)



**📊 Context:** This Risk Assessment Flow provides insights into application characteristics and technical architecture.

**📊 Key Insights:** Analysis of 3 components reveals system patterns and technical dependencies.

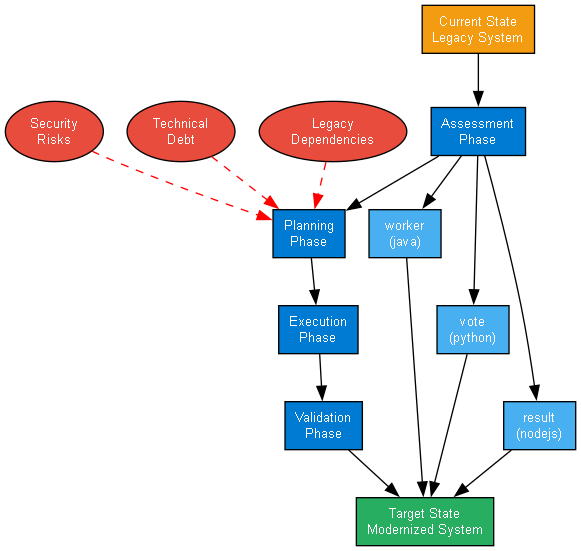
**📊 Business Impact:** Technical insights support strategic planning, risk assessment, and modernization decisions.

**📊 Recommendations:** Review detailed findings with technical teams to prioritize actions and plan next steps.

**📊 Technical Details:** Diagram type: Risk Assessment Flow. Analysis includes component assessment, technology stack evaluation, and architectural patterns.

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

Migration Strategy (Graphviz)



**📊 Context:** Migration strategy analysis for 3 components shows requires preparation cloud readiness.

**📊 Key Insights:** System is 100% containerized with 3 security concerns. Migration complexity: requires preparation.

**📊 Business Impact:** Migration timeline estimated at 6+ months with requires preparation current readiness state.

**📊 Recommendations:** Priority actions: Address containerization gaps and security issues before migration.

**📊 Technical Details:** Readiness factors: 3/3 containerized, 3 security findings. Strategy: lift-and-shift feasible for containerized components.

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