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

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

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.

Component: vote

• Language: python

• Runtime: python

• Build Tool: unknown

• Packaging: docker

• Exposed Ports: 8080

• Base Images: python:3.9-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: A pattern matching hardcoded secrets (password = '...') 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.

• 💼 Business Criticality: Assessment is inferred from technical indicators. Validate with business stakeholders.

🟢 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

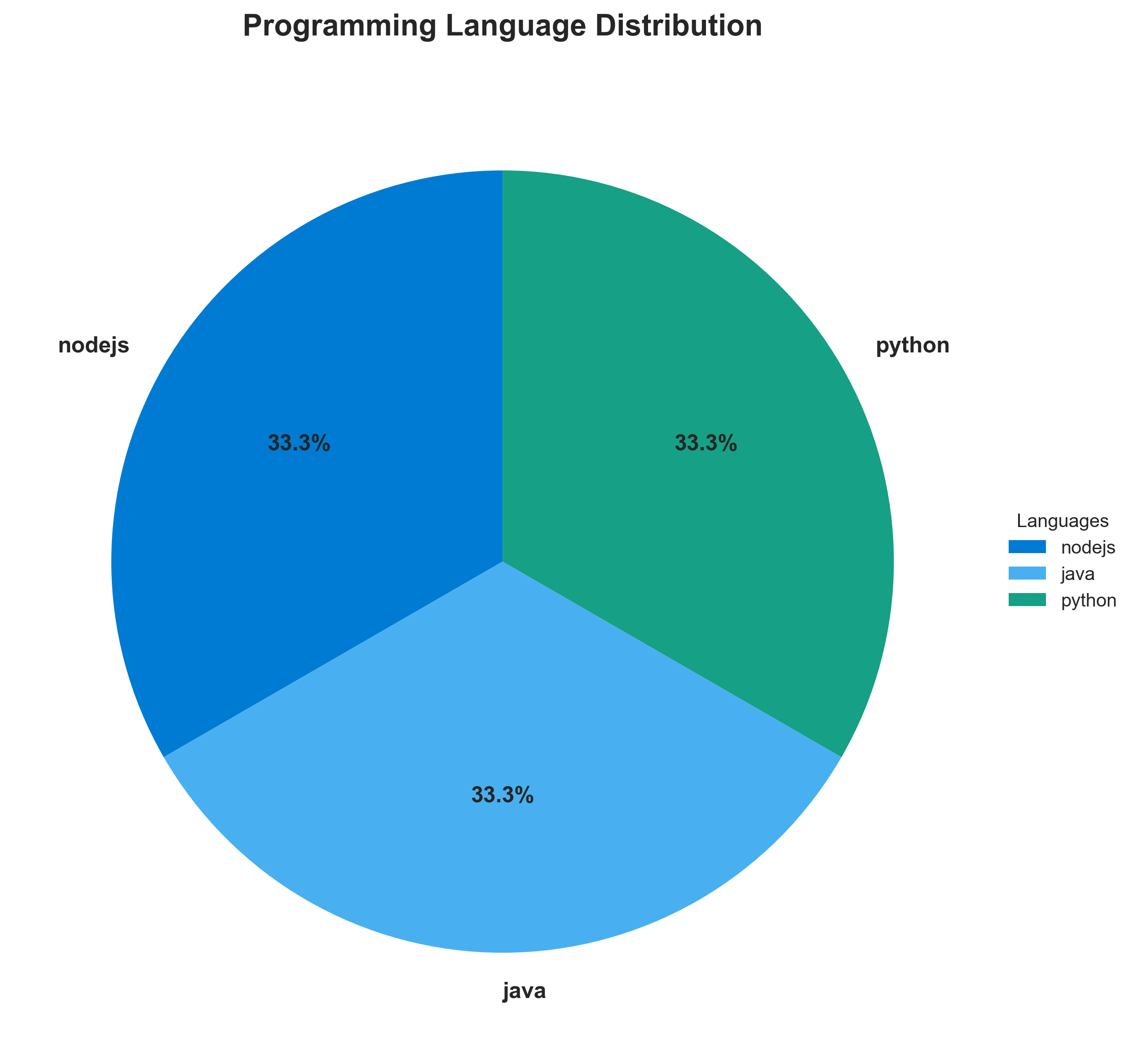
• Architecture Assessment: Evaluation of deployment patterns and component relationships

• 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 three identified components of the application: 'result' (Node.js), 'worker' (Java), and 'vote' (Python). The analysis is conducted within the context of an application intelligence report to inform modernization and migration planning efforts.

**📊 Key Insights:** The application exhibits a polyglot architecture, utilizing Node.js, Java, and Python across its components. Notably, all three components are containerized using Docker, suggesting a degree of modern deployment practices. However, the 'worker' component has identified vulnerabilities in its base images ('maven:3.5-jdk-8-alpine' and 'openjdk:8-jre'), and the 'result' component uses a vulnerable Node.js base image ('node:10-slim'), posing immediate security risks. The 'worker' component also has a note indicating a potential alternative C# implementation, which warrants further investigation.

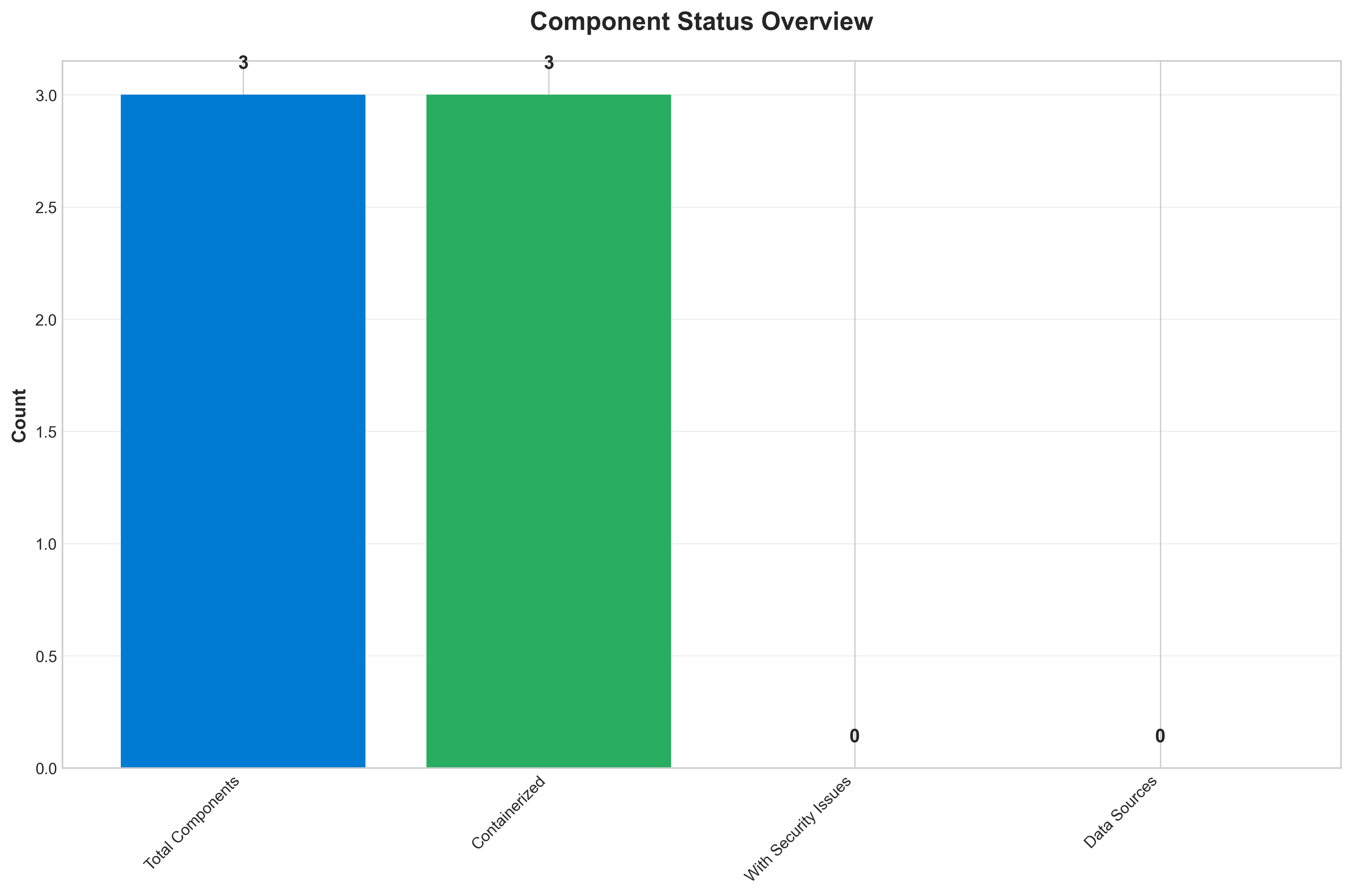
**📊 Business Impact:** The polyglot nature of the application can increase operational complexity and require diverse skill sets for maintenance and development, potentially impacting staffing and training costs. The identified vulnerable base images present significant security risks that could lead to data breaches or service disruptions, necessitating urgent remediation. The presence of potential alternative implementations could represent an opportunity for rationalization or indicate technical debt if not actively managed.

**📊 Recommendations:** Prioritize immediate remediation of vulnerable base images for 'result' (Node.js) and 'worker' (Java) components by updating to secure, supported versions. Investigate the note regarding the alternative C# implementation in the 'worker' component to understand its purpose and potential impact on future migration strategies. Consider standardizing on a more common language or runtime if feasible during modernization to simplify maintenance and reduce operational overhead.

**📊 Technical Details:** The 'result' component is built on Node.js and exposed on port 8080. The 'worker' component is a Java application, indicated by the `JAVA\_APP\_JAR` environment variable and its use of Maven and OpenJDK base images. The 'vote' component is written in Python and also exposes port 8080. All components are containerized via Docker packaging, with the 'worker' component utilizing multiple base images, potentially indicative of a multi-stage build process.

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

Component Status Overview



**📊 Context:** This Component Status Overview diagram provides a snapshot of the current health and status of the application's components, focusing on containerization and security. It is a critical input for our ongoing application modernization and migration planning efforts, offering visibility into foundational readiness.

**📊 Key Insights:** All 3 identified application components are containerized, indicating a positive step towards modern infrastructure. However, a significant concern arises from the security findings: all 3 critical security vulnerabilities identified are classified as 'HIGH' severity, directly linked to outdated and End-of-Life (EOL) base images used across components. This points to a systemic issue with the current base image management strategy.

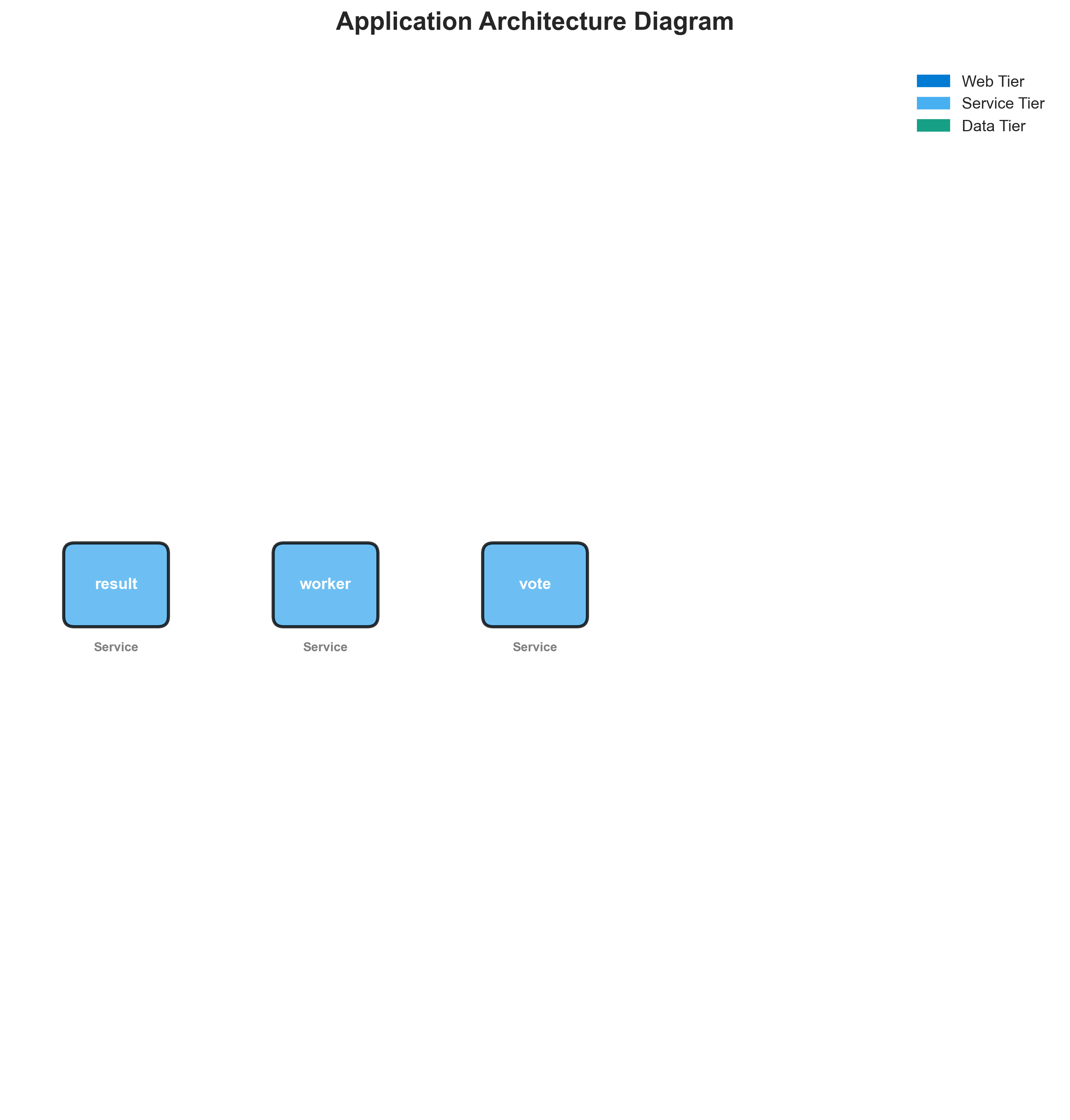
**📊 Business Impact:** The presence of critical vulnerabilities in EOL base images poses a substantial security risk, potentially exposing the application to breaches and compliance failures. This directly impacts the security posture and could delay migration timelines if remediation is not prioritized, increasing the overall cost and risk of the modernization initiative. Failing to address these issues could also lead to operational instability.

**📊 Recommendations:** Immediately prioritize the update of all base images identified with HIGH severity vulnerabilities, starting with 'node:10-slim' due to its EOL status and 'maven:3.5-jdk-8-alpine' and 'openjdk:8-jre' given their known vulnerabilities. Establish a rigorous base image scanning and update policy to prevent recurrence, ensuring a more secure foundation for future modernization and migration phases.

**📊 Technical Details:** The analysis reveals that components are utilizing 'node:10-slim', 'maven:3.5-jdk-8-alpine', and 'openjdk:8-jre' as their base images. Security scans identified specific vulnerabilities within these images, with Node.js 10 being EOL and the Maven/OpenJDK versions containing known exploits. The complete absence of medium or low severity findings suggests a focused effort on critical security issues, but the pervasive nature of HIGH severity findings highlights a significant gap.

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

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.

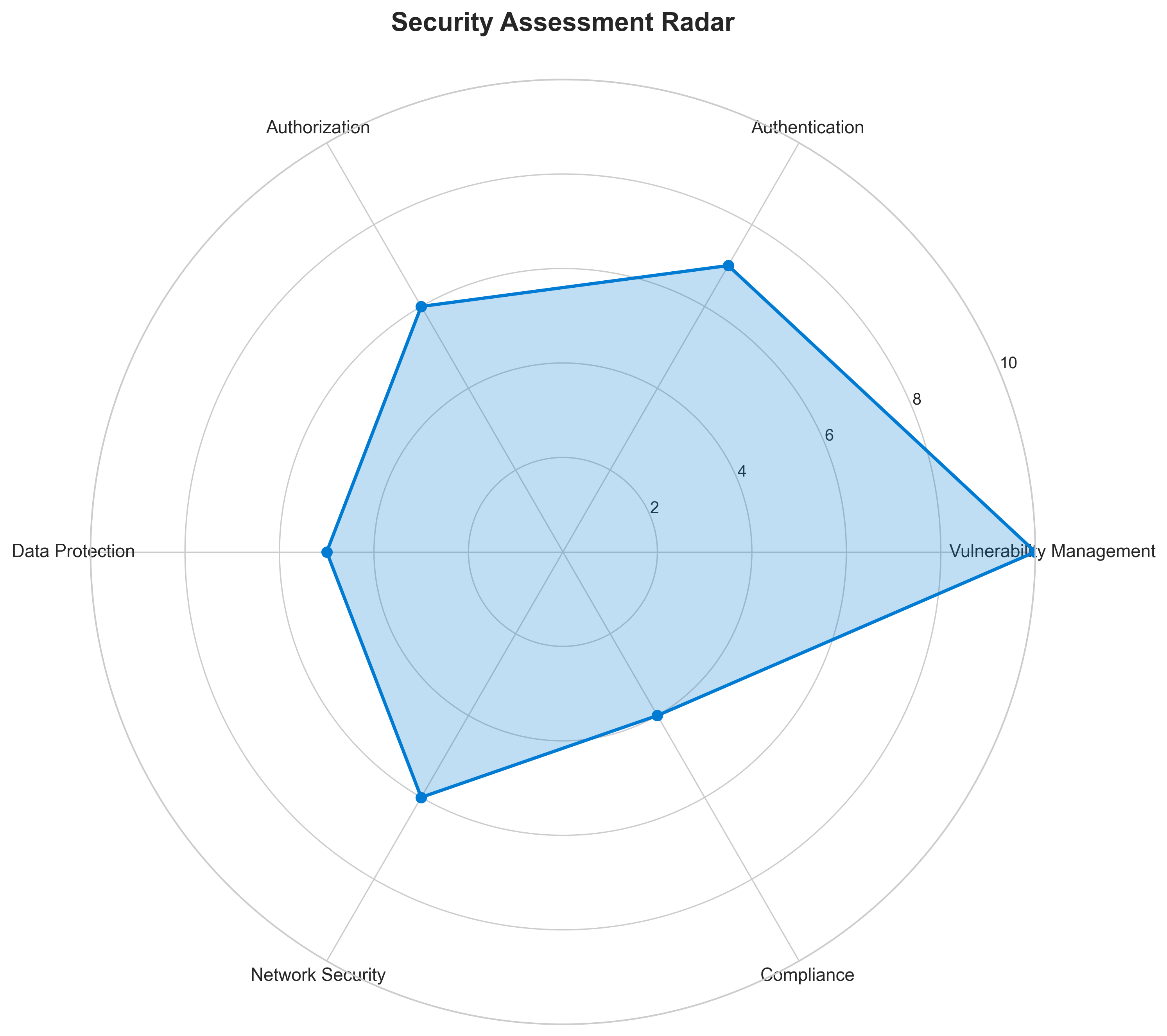
**📊 Business Impact:** Architecture supports unified deployment with architectural patterns need assessment considerations.

**📊 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.

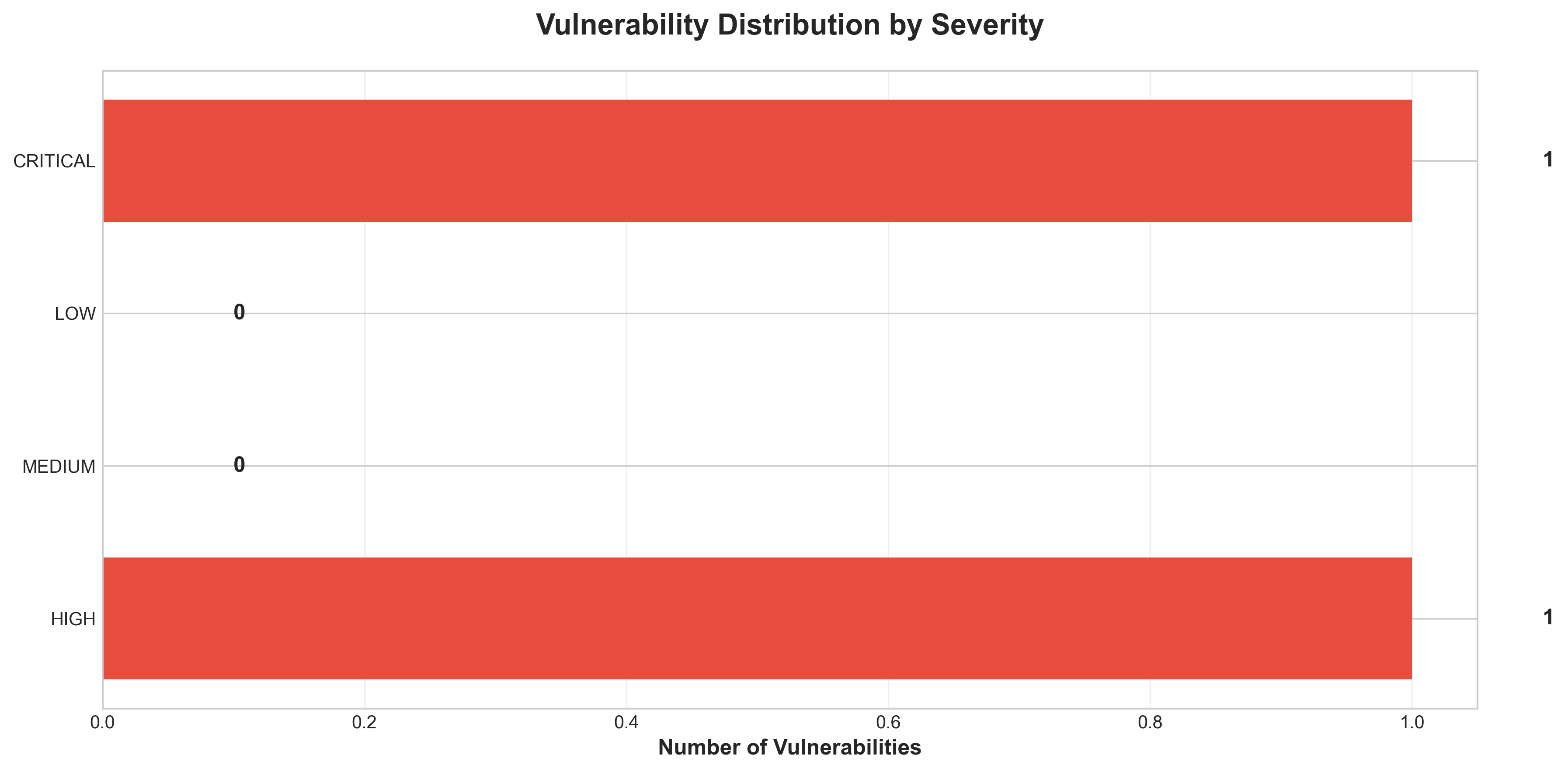
**📊 Business Impact:** Current security posture represents high risk to system reliability and compliance requirements.

**📊 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 and high-severity security flaws within the application's components, specifically focusing on base image vulnerabilities and potential hardcoded secrets. Generated as part of a modernization and migration planning initiative, this data is crucial for understanding the application's current security posture and identifying immediate risks.

**📊 Key Insights:** The analysis reveals a significant reliance on end-of-life (EOL) base images, with 'node:10-slim' being explicitly identified as critical due to unpatched vulnerabilities. Furthermore, three high-severity findings indicate that the 'result' and 'worker' components are using outdated and vulnerable base images ('node:10-slim', 'maven:3.5-jdk-8-alpine', and 'openjdk:8-jre'). A separate high-severity finding points to a potential hardcoded secret in the 'voting-app\vote\src\app.py' file, posing an immediate risk of credential compromise.

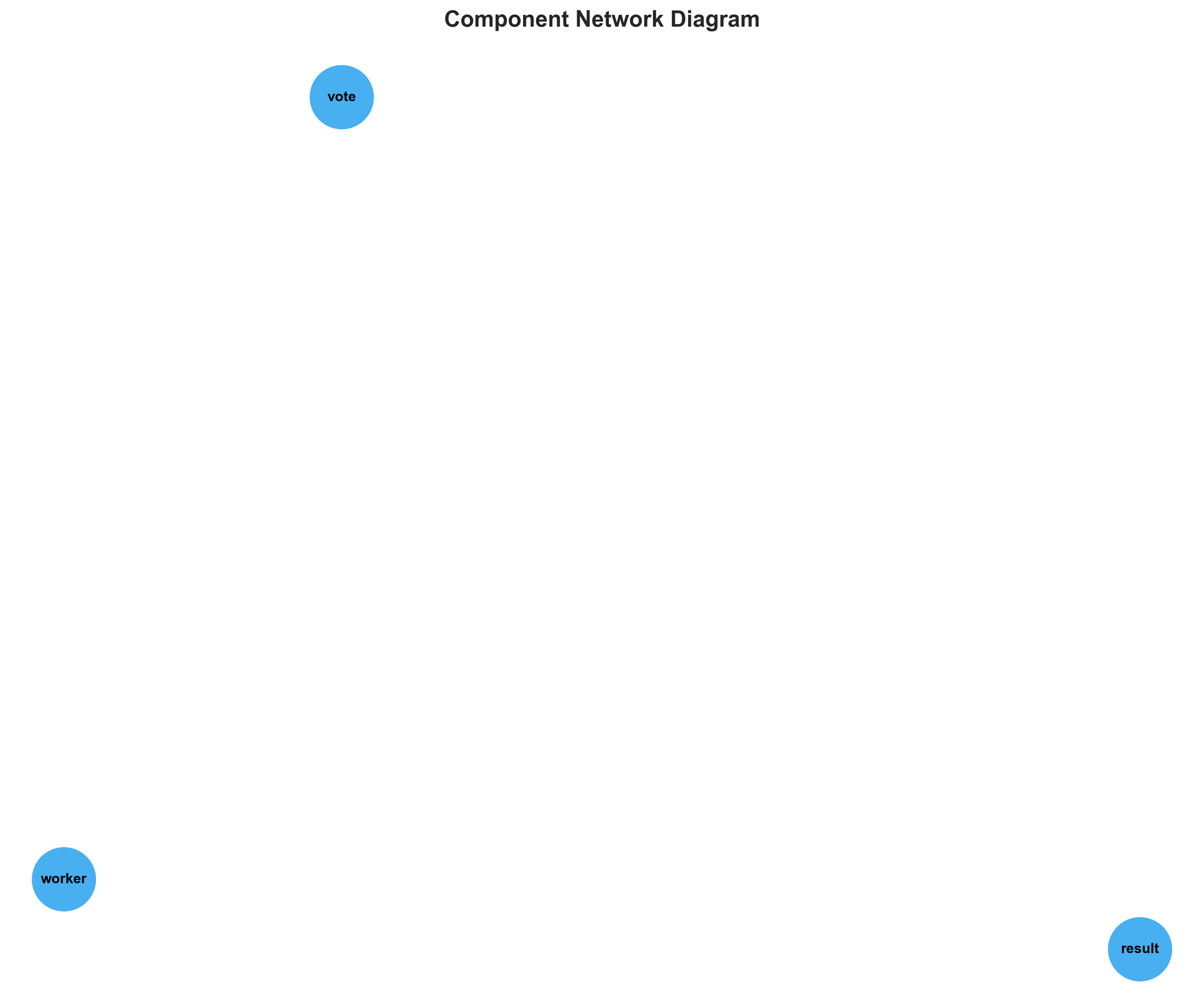
**📊 Business Impact:** The presence of EOL base images and potential hardcoded secrets poses a substantial risk to the business, including data breaches, compliance violations, and operational disruptions. These vulnerabilities can severely hamper modernization and migration efforts, increasing the cost and complexity of moving to a cloud environment. Addressing these issues is paramount to ensure a secure and stable application post-migration.

**📊 Recommendations:** Immediate action is required to address the critical 'End-of-Life Base Image' vulnerability by updating 'node:10-slim' to a supported version. Concurrently, the potential hardcoded secret in 'voting-app\vote\src\app.py' must be investigated and remediated. Prioritizing the update of all identified vulnerable base images for the 'result' and 'worker' components is essential for mitigating immediate security risks.

**📊 Technical Details:** The 'vulnerability\_assessment.findings' detail a specific critical issue with 'node:10-slim' and a high-severity issue related to hardcoded secrets in Python code. The 'summary.security\_findings' aggregates this, confirming three high-severity findings primarily stemming from outdated base images across components like 'result' and 'worker'. The scan method indicates 'base\_image\_analysis' was performed, confirming the root cause of multiple vulnerabilities.

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

Component Network Topology



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

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

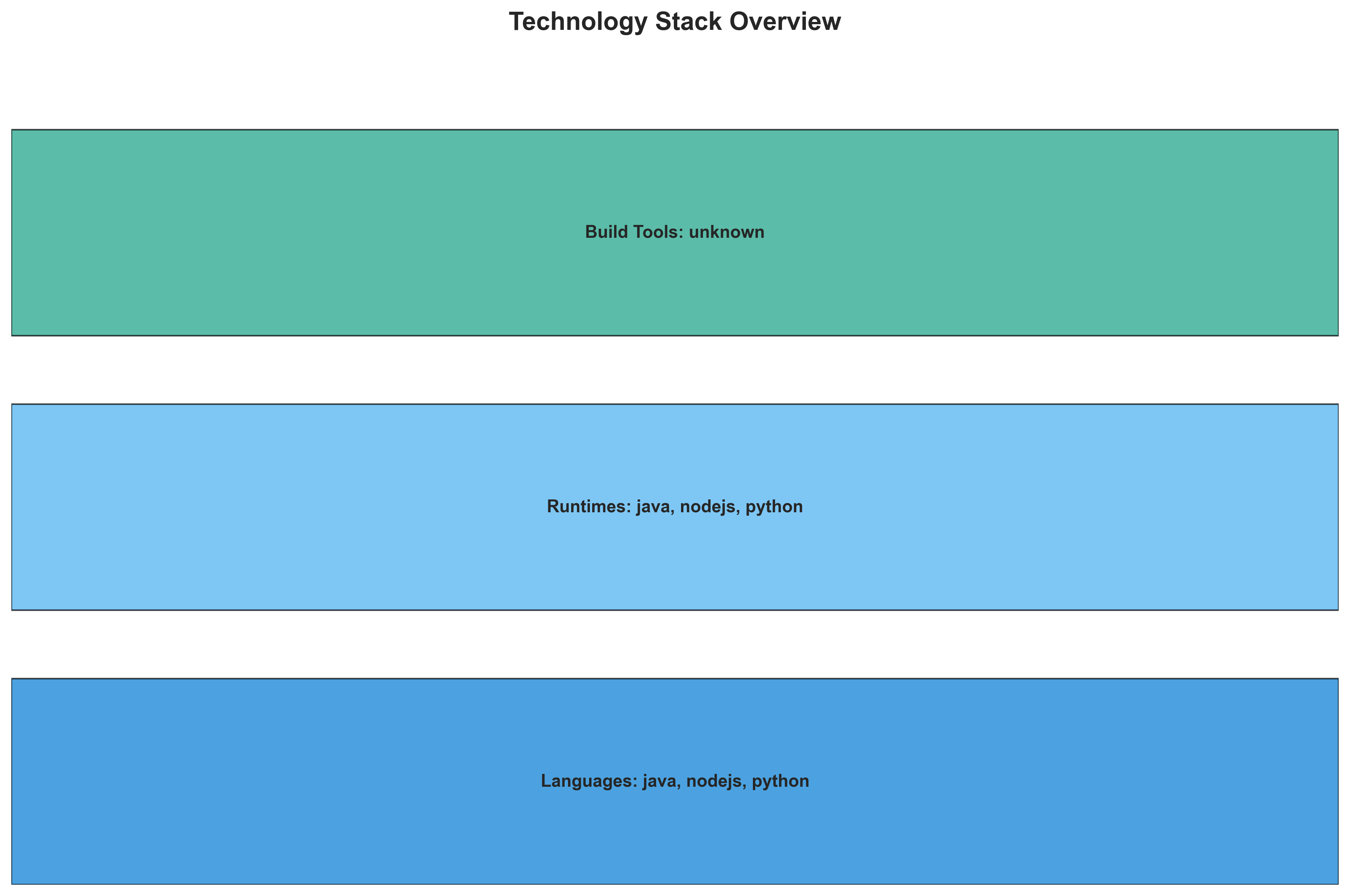
**📊 Business Impact:** Containerization status directly impacts cloud migration readiness and deployment flexibility.

**📊 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 a multi-language application, consisting of 'result', 'worker', and 'vote' services. The analysis is framed within a broader application intelligence report focused on planning for modernization and cloud migration.

**📊 Key Insights:** The application utilizes a polyglot architecture, with Node.js, Java, and Python components identified. All identified components are containerized using Docker, exposing port 8080 for 'result' and 'vote'. A significant concern is the use of 'Vulnerable base image: node:10-slim' for the 'result' service and 'Vulnerable base image: maven:3.5-jdk-8-alpine' and 'Vulnerable base image: openjdk:8-jre' for the 'worker' service, indicating potential security risks and out-of-date runtime environments.

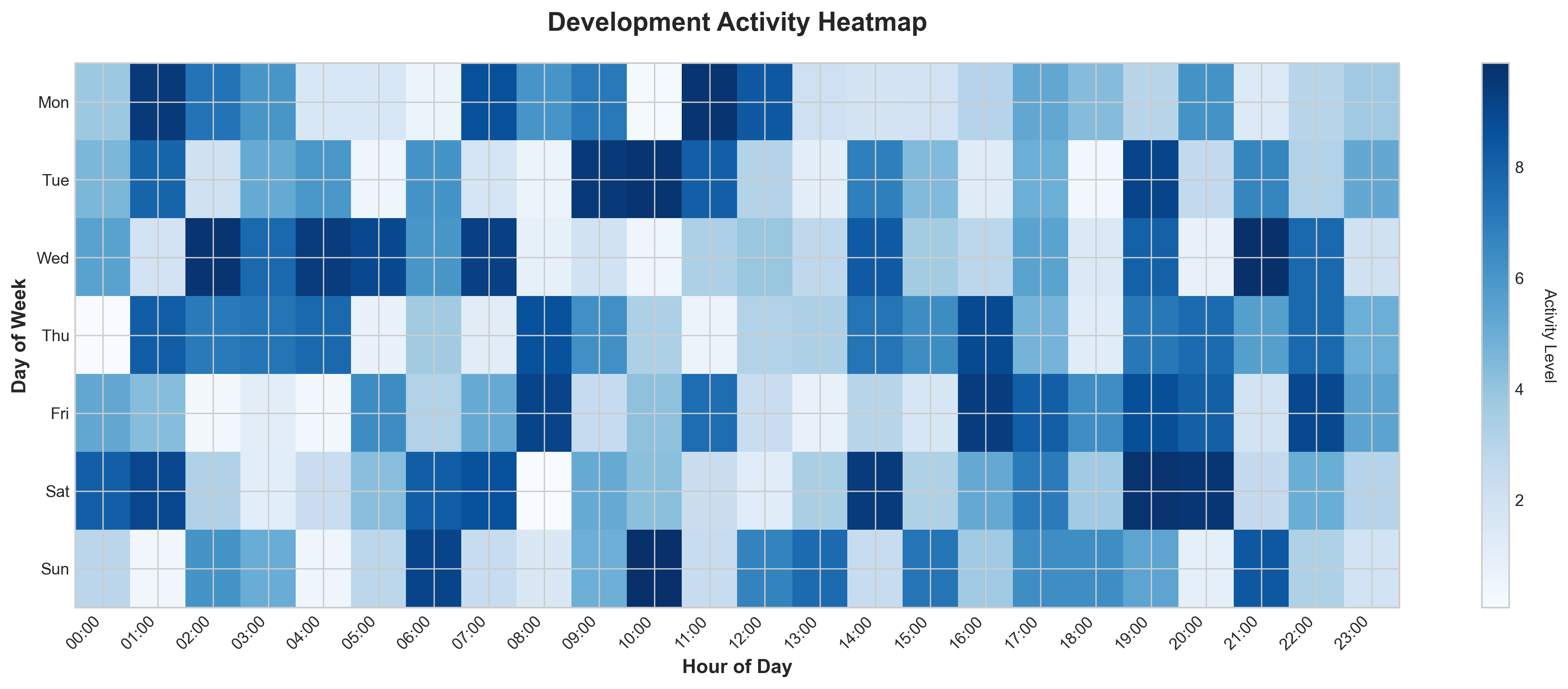
**📊 Business Impact:** The identified vulnerabilities in base images pose a significant security risk, potentially exposing the application to exploits and data breaches. The use of older Node.js and Java versions may also limit access to newer features, performance optimizations, and vendor support, impacting future development velocity and operational costs during migration. The polyglot nature, while offering flexibility, could also introduce complexity in management and maintenance.

**📊 Recommendations:** Prioritize immediate remediation of vulnerable base images by upgrading to supported and secure versions for all components. Investigate the 'worker' service's multi-base image strategy to standardize on a single, secure, and efficient build process. Plan for refactoring or updating dependencies to leverage modern language features and frameworks during the modernization effort.

**📊 Technical Details:** The 'result' and 'vote' services are Node.js and Python respectively, both running in Docker containers and exposing port 8080. The 'worker' service is Java, also containerized, with 'JAVA\_APP\_JAR' environment variable indicating a JAR-based executable. The 'worker' service's use of multiple base images suggests potential multi-stage Docker builds, requiring further investigation into their build pipeline.

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

Development Activity Heatmap



**📊 Context:** This Development Activity Heatmap provides an overview of the application's current state, focusing on its components, technologies, and development patterns. It was generated to inform modernization and migration planning by highlighting key technical characteristics and potential risks.

**📊 Key Insights:** The application comprises 3 independent, containerized microservices written in Java, Node.js, and Python, leveraging Docker. Notably, all discovered base images (Node.js 10, Maven 3.5 with JDK 8, and OpenJDK 8) contain high-severity vulnerabilities and are outdated. Development activity appears to be minimal, with only 1 commit and no active contributors recorded in the Git history, suggesting a lack of recent engagement or maintenance.

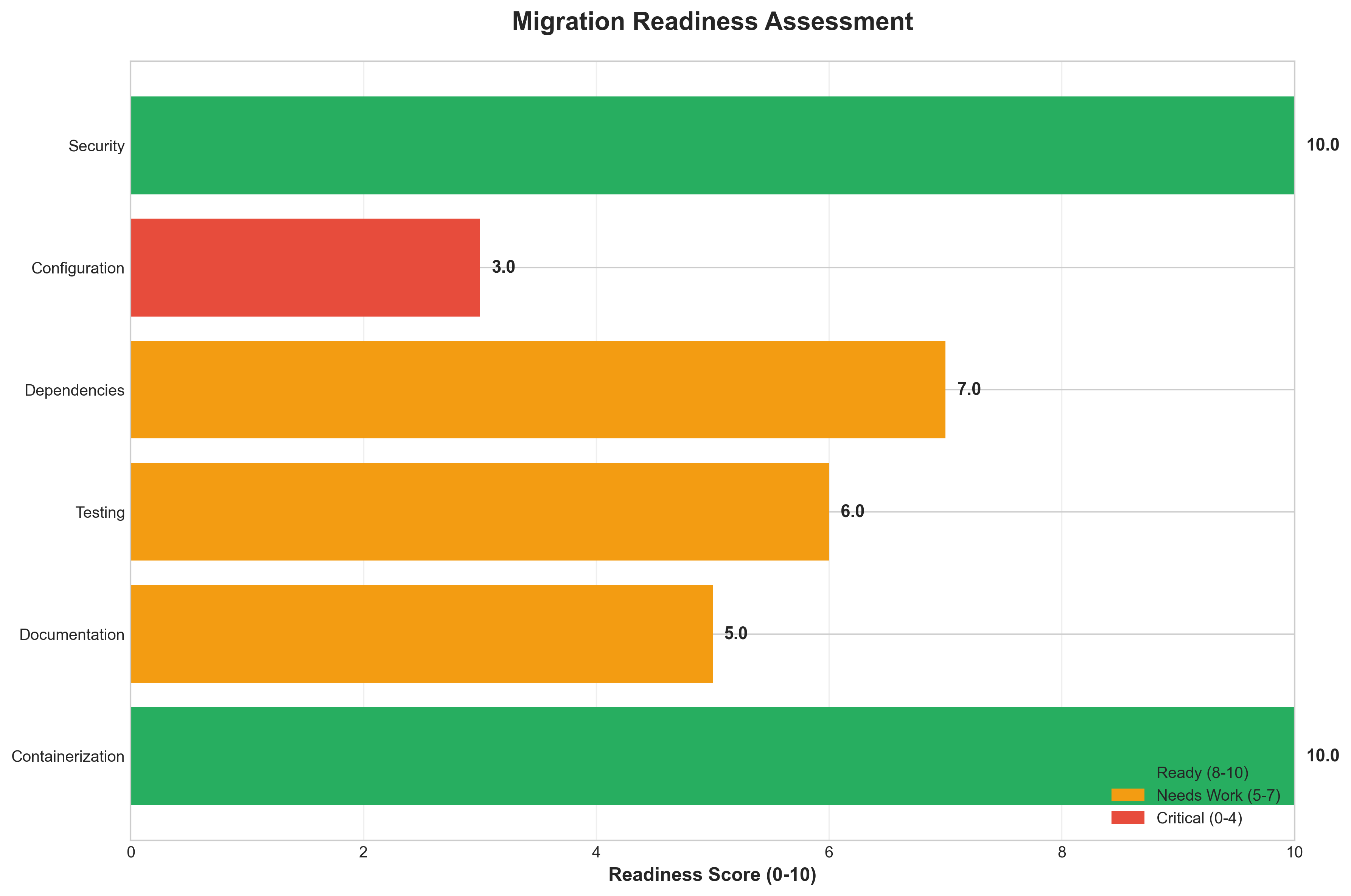
**📊 Business Impact:** The presence of critical vulnerabilities in the base images poses a significant security risk, potentially exposing the application and its data to breaches. The outdated components also present a technical debt, which could increase the complexity and cost of future modernization or migration efforts. The low development activity suggests a risk of knowledge loss and difficulty in addressing issues or driving future enhancements.

**📊 Recommendations:** Prioritize updating all identified base images to secure, supported versions to mitigate immediate security risks. Conduct a thorough security review of all components and dependencies to identify any further vulnerabilities. Initiate a plan to re-engage development teams or re-architect components to foster ongoing maintenance and modernization efforts.

**📊 Technical Details:** The application exhibits a clear microservices architecture, with each of the 3 components containerized using Docker. External services like Redis and PostgreSQL are utilized. The absence of Kubernetes resources, Docker Compose files, or explicit containerization files (beyond the implied Docker images) indicates an unknown deployment platform and a lack of defined orchestration or infrastructure-as-code for these services.

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

Migration Readiness Assessment



**📊 Context:** This Migration Readiness Assessment (MRA) diagram provides a focused view on the security posture of the application's components, specifically concerning container base images. It's generated as part of a broader application intelligence report to inform modernization and cloud migration strategies by highlighting critical security risks.

**📊 Key Insights:** A significant security risk is present across all three identified components, as evidenced by three high-severity findings. Specifically, the container images used for 'result' and 'worker' components rely on outdated and vulnerable base images ('node:10-slim' and 'maven:3.5-jdk-8-alpine' respectively), alongside the 'worker' component also using 'openjdk:8-jre', which also carries known vulnerabilities. This indicates a systemic issue with the base image strategy across the application.

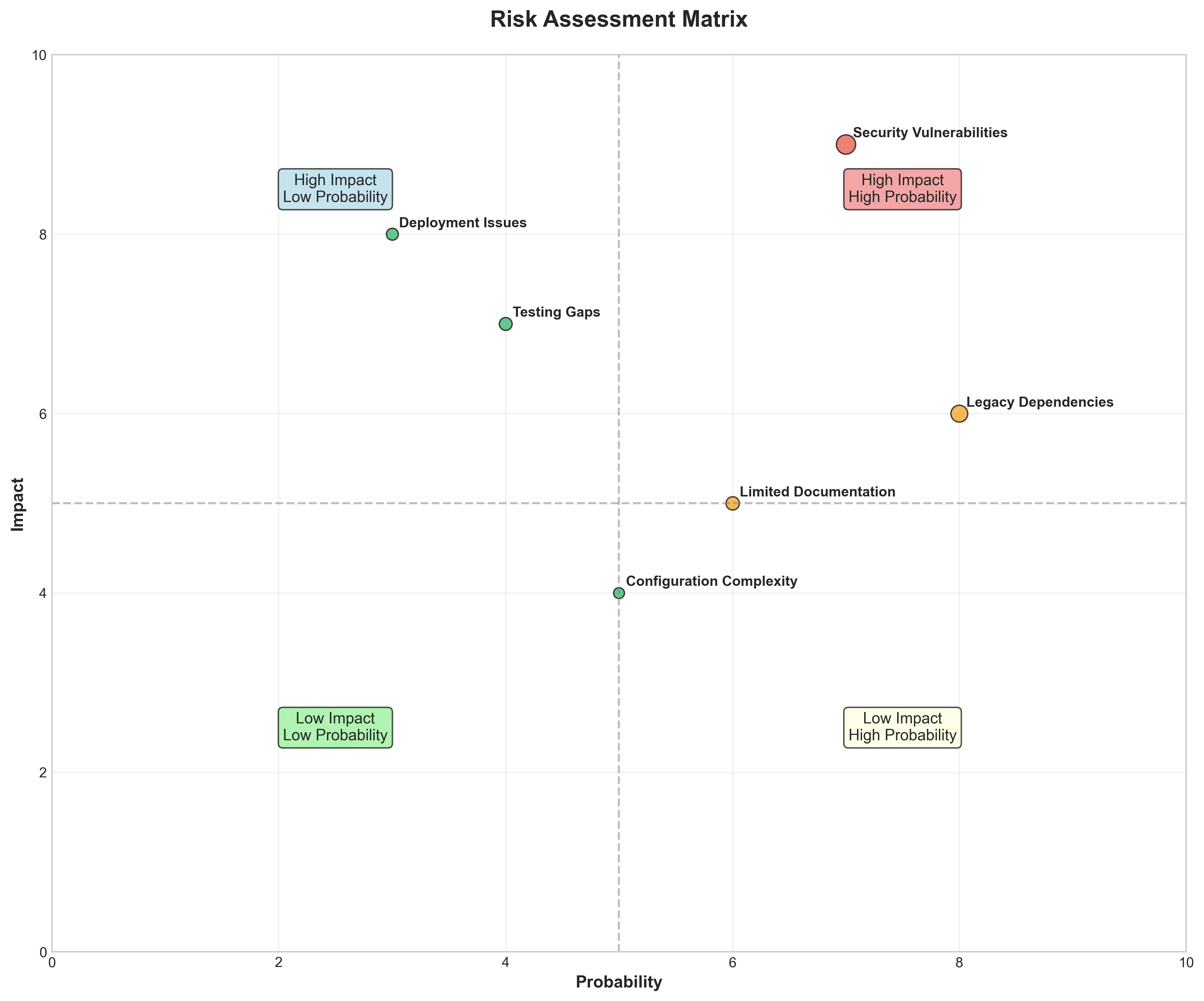
**📊 Business Impact:** The identified high-severity vulnerabilities in the container base images pose a substantial security risk to the business, potentially leading to data breaches, service disruptions, and reputational damage. The immediate need to update these critical components will likely impact migration timelines and require dedicated engineering resources, potentially increasing project costs. Failure to address these risks could severely hinder the ability to migrate to a secure cloud environment.

**📊 Recommendations:** The highest priority recommendation is to immediately remediate all identified high-severity security findings by updating the base images for all components to supported and patched versions. Following this, a thorough review of the containerization strategy should be undertaken to establish a policy for using up-to-date and secure base images for all future development and deployments. Exploring automated image scanning within the CI/CD pipeline should also be considered to prevent future recurrences.

**📊 Technical Details:** The security scan was performed using 'base\_image\_analysis' and identified vulnerabilities in 'node:10-slim', 'maven:3.5-jdk-8-alpine', and 'openjdk:8-jre'. All three findings are rated 'HIGH' severity, indicating critical security flaws. The 'summary.containerization\_status' is '3', which likely represents a partial or in-progress containerization effort, and the 'summary.total\_components' is '3', aligning with the components flagged for vulnerabilities.

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

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.

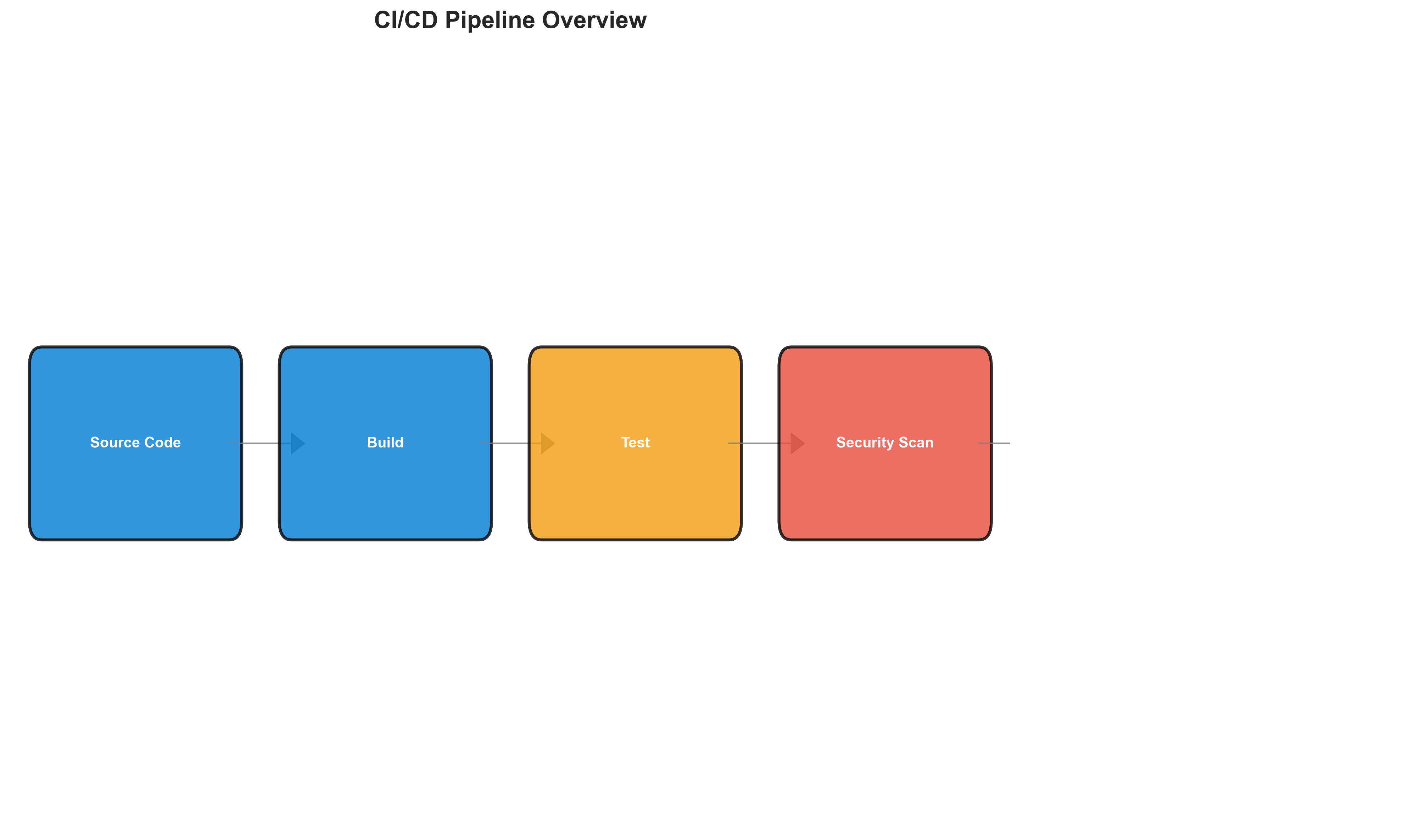
**📊 Business Impact:** Technical insights support strategic planning, risk assessment, and modernization decisions.

**📊 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 comprised of three containerized components built with Node.js, Java, and Python. Generated as part of a modernization and migration planning initiative, it highlights the current state of the application's build, deployment, and foundational security posture.

**📊 Key Insights:** The pipeline demonstrates a mature approach to containerization, with all three components (1 Node.js, 1 Java, 1 Python) packaged as Docker images. However, significant security vulnerabilities have been identified in the base images used across all components, particularly related to end-of-life Node.js and outdated Java versions, posing a critical risk. Limited recent Git activity suggests potential stagnation in development or a need for improved developer engagement.

**📊 Business Impact:** The identified high-severity vulnerabilities present a substantial security risk, potentially leading to breaches and reputational damage if not addressed promptly. The reliance on outdated software also impedes innovation and future compatibility, creating technical debt that will increase migration costs. The inactive Git history indicates a potential lack of active development, which could impact the application's evolution and competitiveness.

**📊 Recommendations:** Prioritize immediate remediation of the identified base image vulnerabilities by updating to supported and secure versions (e.g., newer Node.js and JDK versions). Conduct a review of recent development activity and consider strategies to re-engage development teams to foster continuous improvement and address the inactive Git history. Plan for an updated security scanning strategy that goes beyond base image analysis to cover application-level vulnerabilities.

**📊 Technical Details:** The analysis reveals three distinct microservices, all confirmed to be containerized (Docker). The pipeline utilizes external services, specifically Redis and PostgreSQL, but does not appear to integrate directly with any traditional databases within the application's direct control. The absence of orchestration files (Kubernetes, Docker Compose) and a declared deployment platform suggests a potentially unmanaged or legacy deployment strategy.

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

Component Relationships (Graphviz)



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

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

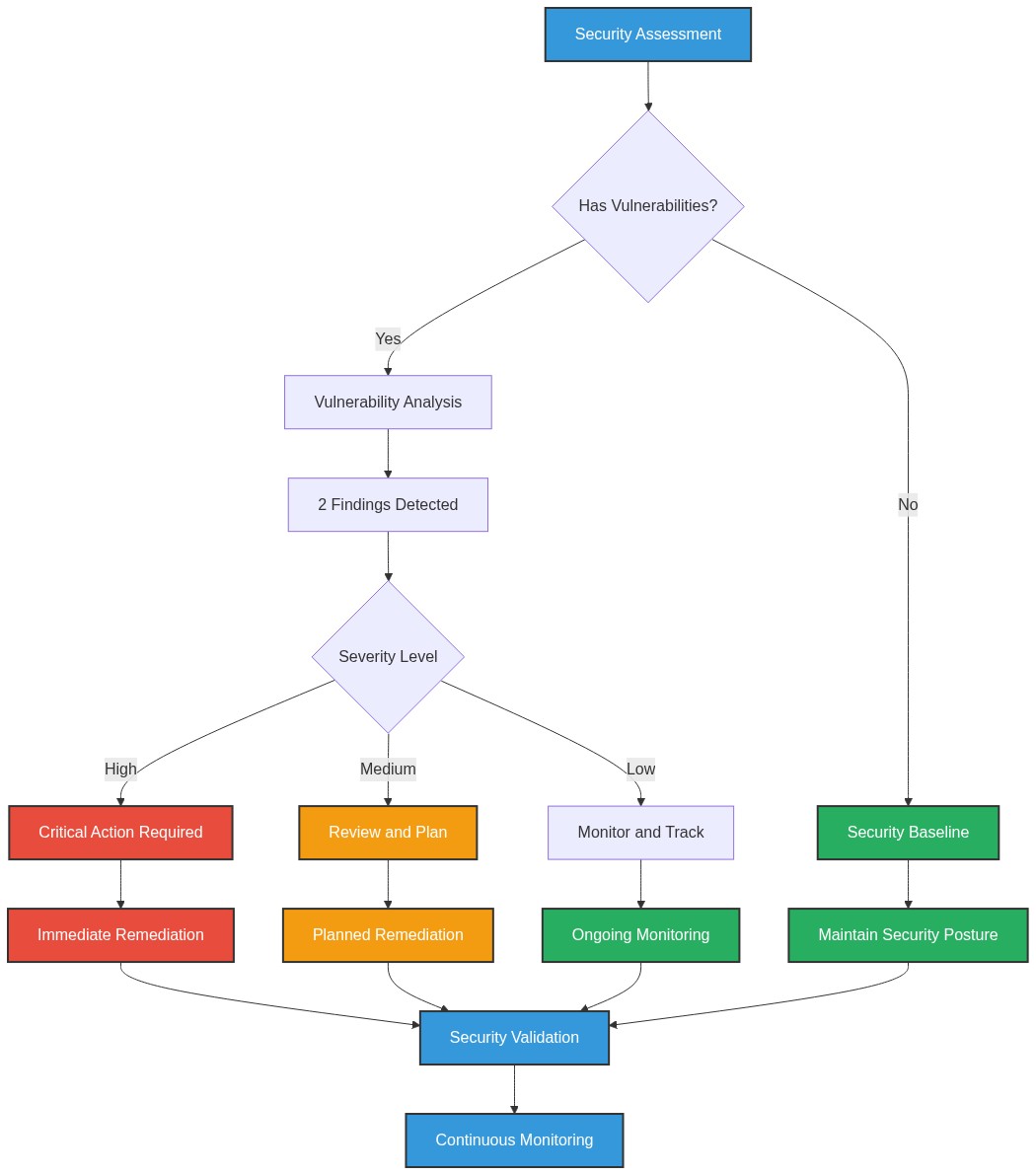
**📊 Business Impact:** Containerization status directly impacts cloud migration readiness and deployment flexibility.

**📊 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.

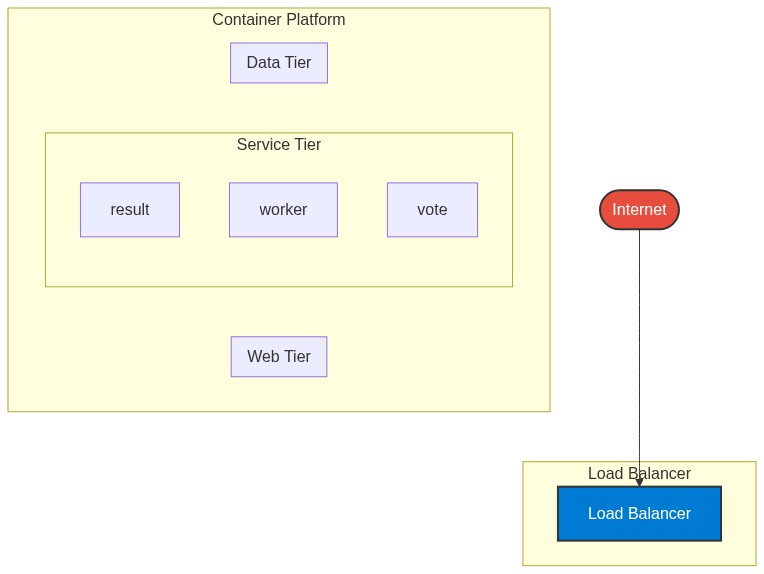
**📊 Business Impact:** Current security posture represents high risk to system reliability and compliance requirements.

**📊 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.

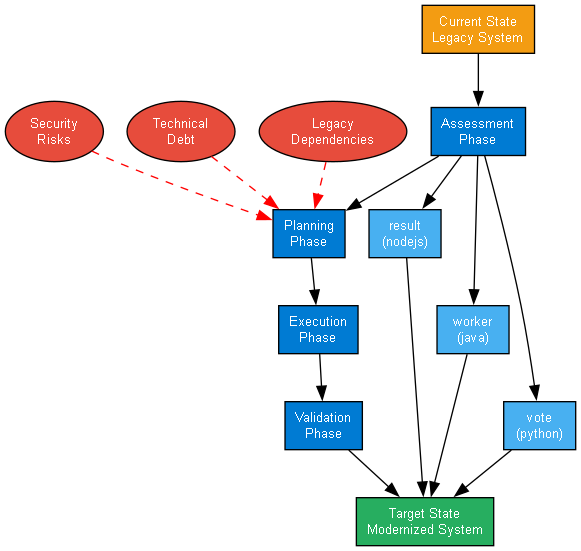
**📊 Business Impact:** Technical insights support strategic planning, risk assessment, and modernization decisions.

**📊 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.

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