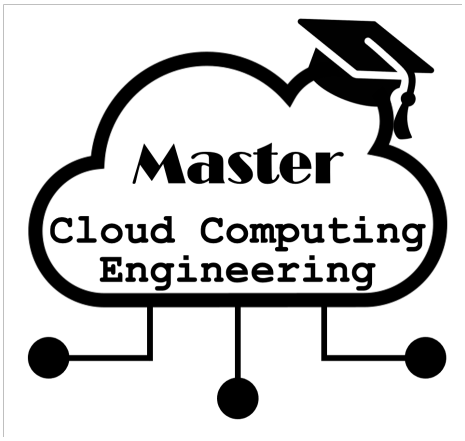


DevOps Projektumsetzung und Dokumentation

CI/CD Pipeline mit SonarQube, Dependabot und Testing

Hochschule Burgenland
Studiengang MCCE
Sommersemester 2025

Harald Beier* Susanne Peer† Patrick Prugger‡
Philipp Palatin§

5. Juni 2025

*2410781028@hochschule-burgenland.at
†2410781002@hochschule-burgenland.at
‡2410781029@hochschule-burgenland.at
§2310781027@hochschule-burgenland.at

Inhaltsverzeichnis

1 Projektumsetzung und Dokumentation

1.1 Projektübersicht

Dieses Dokument beschreibt die High-Level-Umsetzung eines modernen DevOps-Projekts mit Fokus auf automatisierte CI/CD-Pipelines, Qualitätssicherung und kontinuierliche Integration. Das Projekt implementiert eine vollständige Notes-API-Anwendung mit umfassender Testabdeckung und automatisierter Deployment-Pipeline.

1.1.1 Technologie-Stack

- **Backend**: Node.js mit Express.js Framework
- **Testing**: Jest für Unit- und Integrationstests
- **CI/CD**: GitHub Actions
- **Code Quality**: SonarQube für statische Code-Analyse
- **Dependency Management**: Dependabot für automatisierte Updates
- **Containerization**: Docker mit Multi-Stage Builds
- **Registry**: GitHub Container Registry (GHCR)

1.1.2 Repository-Zugang

Das Projekt ist in einem öffentlichen GitHub Repository verfügbar:

- **Repository URL**: `https://github.com/Gruppe1DevOps/DevOpsG1`
- **Hauptbranch**: main
- **Anwendungscode**: `Taucher/PT/Code/`
- **CI/CD Konfiguration**: `.github/workflows/`

1.2 Anwendungsarchitektur

1.2.1 Notes API Implementierung

Die Kern-Anwendung ist eine RESTful API für die Verwaltung von Notizen mit folgenden Endpunkten:

Tabelle 1: API Endpunkte der Notes-Anwendung

HTTP Method	Endpoint	Beschreibung
GET	/api/notes	Alle Notizen abrufen
GET	/api/notes/:id	Spezifische Notiz abrufen
POST	/api/notes	Neue Notiz erstellen
DELETE	/api/notes/:id	Notiz löschen

1.2.2 Datenmodell

Jede Notiz enthält folgende Attribute:

- **id**: Eindeutige Identifikationsnummer (automatisch generiert)
- **content**: Textinhalt der Notiz (erforderlich)
- **important**: Boolean-Wert für Wichtigkeit (optional, Standard: false)
- **date**: Erstellungsdatum (automatisch generiert)

1.3 CI/CD Pipeline Architektur

1.3.1 Pipeline-Übersicht

Die CI/CD-Pipeline implementiert einen mehrstufigen Ansatz mit klarer Trennung der Verantwortlichkeiten:

1.3.2 Pipeline-Stufen

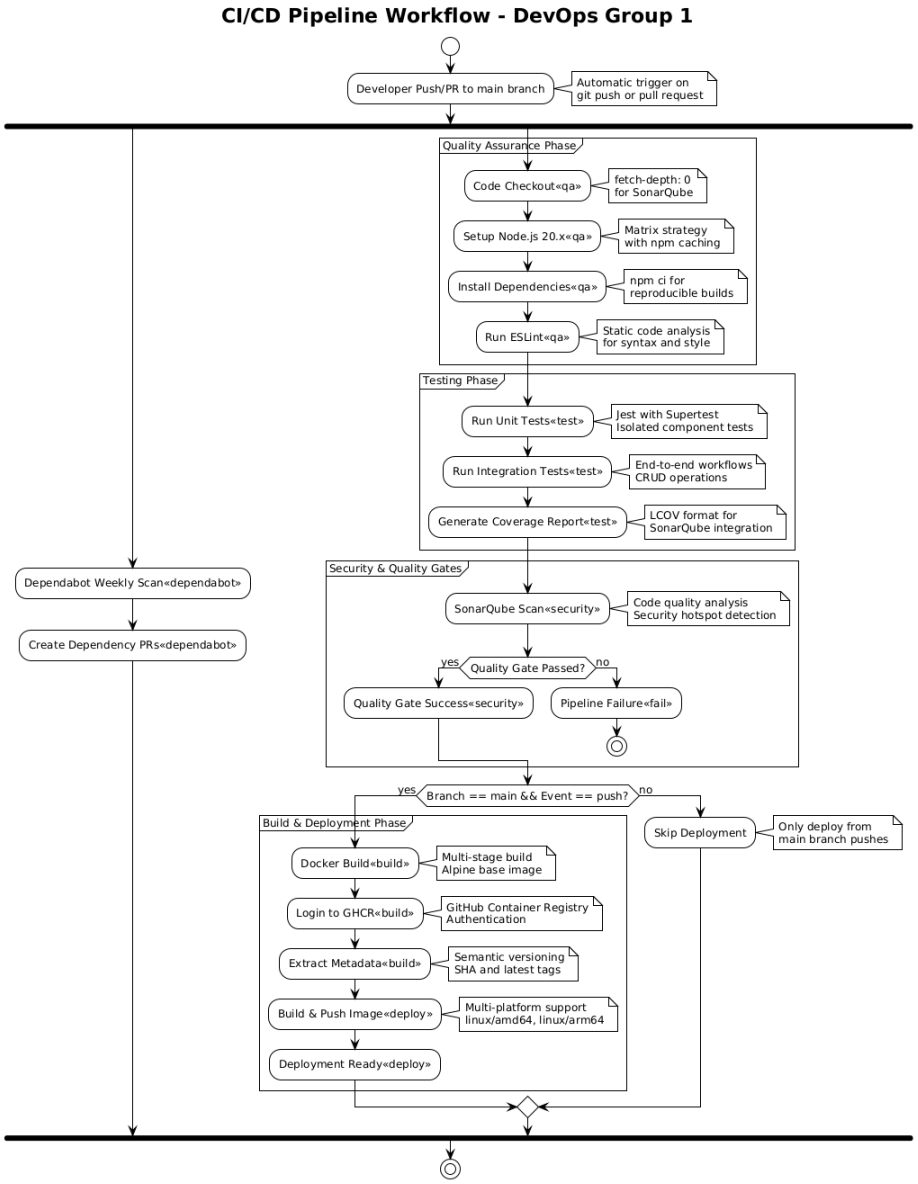
1. Trigger-Phase

- Automatische Auslösung bei Push/Pull Request auf main Branch
- Parallele Ausführung für Pull Requests zur Qualitätssicherung
- Conditional Deployment nur bei main Branch

2. Quality Assurance Phase

- **Code Checkout**: Vollständiger Git-Verlauf für SonarQube
- **Node.js Setup**: Matrix-Strategie für Version 20.x
- **Dependency Installation**: npm ci für reproduzierbare Builds
- **ESLint**: Statische Code-Analyse für Syntax und Style

Abbildung 1: CI/CD Pipeline Workflow



3. Testing Phase

- **Unit Tests**: Isolierte Komponententests
- **Integration Tests**: End-to-End Workflow-Tests

- **Coverage Analysis**: Vollständige Test-Coverage-Berichte

4. Security und Quality Gates

- **SonarQube Scan**: Umfassende Code-Qualitätsanalyse
- **Quality Gate**: Automatische Pipeline-Unterbrechung bei Qualitätsproblemen
- **Coverage Threshold**: Mindestens 80% Test-Coverage erforderlich

5. Build und Deployment Phase

- **Docker Build**: Multi-Stage Containerisierung
- **Registry Push**: Automatischer Upload zu GitHub Container Registry
- **Semantic Versioning**: Automatische Tag-Generierung

1.4 Testing-Strategie

1.4.1 Multi-Layer Testing Approach

Das Projekt implementiert eine umfassende Testing-Strategie mit verschiedenen Test-Ebenen:

Unit Tests

- **Zweck**: Isolierte Komponententests für einzelne Funktionen
- **Framework**: Jest mit Supertest für HTTP-Testing
- **Coverage**: Fokus auf hohe Code-Coverage einzelner Module
- **Ausführungszeit**: Schnelle Ausführung für sofortiges Feedback

Tabelle 2: Unit Test Abdeckung

Test Case	**Beschreibung**
GET /api/notes	Abruf aller Notizen mit korrekter Struktur
POST /api/notes	Erstellung neuer Notizen mit gültigen Daten
POST /api/notes (invalid)	Fehlerbehandlung bei ungültigen Daten
GET /api/notes/:id	Abruf spezifischer Notizen
GET /api/notes/:id (404)	404-Response für nicht existierende Notizen

Integration Tests

- **Zweck**: End-to-End Workflow-Tests für komplette User Journeys
- **Scope**: Vollständige CRUD-Operationen und Concurrent Request Handling
- **Datenintegrität**: Verifikation der Datenkonsistenz über mehrere Operationen
- **Performance**: Concurrent Request Testing für Skalierbarkeit

Tabelle 3: Integration Test Szenarien

Test Szenario	**Beschreibung**
Complete CRUD Workflow	Vollständiger Lebenszyklus einer Notiz
Concurrent Requests	5 simultane Notiz-Erstellungen
Data Consistency	Verifikation der Datenintegrität
Unique ID Generation	Eindeutige ID-Generierung unter Last

1.5 Code Quality Management

1.5.1 SonarQube Integration

SonarQube wird für kontinuierliche Code-Qualitätsanalyse eingesetzt:

Konfiguration

- **Projekt**: DevOps Group 1 - Note App
- **Organisation**: gruppe1devops
- **Platform**: SonarCloud
- **Quality Gate**: Automatische Pipeline-Unterbrechung bei Fehlern

Qualitätsmetriken Die folgenden Metriken werden von SonarQube überwacht und analysiert:

Tabelle 4: SonarQube Qualitätsmetriken

Metrik	**Threshold**	**Beschreibung**
Code Coverage	$> 80\%$	Mindest-Testabdeckung
Code Smells	0	Wartbarkeits-Issues
Security Hotspots	0	Potenzielle Sicherheitslücken
Duplicated Code	$< 3\%$	Code-Duplikation
Cyclomatic Complexity	< 10	Komplexitätsanalyse

1.5.2 ESLint Konfiguration

Statische Code-Analyse für JavaScript/Node.js:

- **Standard**: ESLint 8.x mit modernen JavaScript-Standards
- **Rules**: Konsistente Code-Formatierung und Best Practices
- **Integration**: Automatische Ausführung in CI/CD Pipeline
- **Fail-Fast**: Pipeline-Unterbrechung bei Linting-Fehlern

1.6 Dependency Management

1.6.1 Dependabot Konfiguration

Automatisierte Dependency-Updates für Sicherheit und Aktualität:

NPM Dependencies

- **Schedule**: Wöchentliche Updates jeden Montag um 09:00
- **PR Limit**: Maximal 10 gleichzeitige Pull Requests
- **Reviewer**: Automatische Zuweisung an t-stefan
- **Commit Format**: Conventional Commits mit "chore"Prefix

GitHub Actions Dependencies

- **Schedule**: Wöchentliche Updates für Action-Versionen
- **PR Limit**: Maximal 5 gleichzeitige Pull Requests
- **Scope**: Alle GitHub Actions im Repository

1.6.2 Security Benefits

- **Vulnerability Patching**: Automatische Sicherheitsupdates
- **Compliance Tracking**: Aktuelle Dependency-Inventarisierung
- **Risk Mitigation**: Reduzierung bekannter Sicherheitslücken
- **Automated Testing**: Jedes Update wird durch CI/CD validiert

1.7 Containerization und Deployment

1.7.1 Docker Implementation

Multi-Stage Docker Build für optimierte Container:

Build Strategy

- **Base Image**: node:20-alpine für minimale Größe
- **Multi-Stage**: Separate Build- und Runtime-Stages
- **Security**: Non-root User für Container-Ausführung
- **Optimization**: Layer-Caching für schnellere Builds

Container Registry

- **Registry**: GitHub Container Registry (ghcr.io)
- **Tagging Strategy**: Semantic Versioning mit SHA und Latest Tags
- **Multi-Platform**: Support für linux/amd64 und linux/arm64
- **Caching**: GitHub Actions Cache für Build-Optimierung

1.7.2 Deployment Pipeline

Tabelle 5: Deployment Tag-Strategie

Tag Type	**Format**
Branch Reference	main
SHA Reference	main-{sha}
Latest	latest (nur main branch)
Version	v1.0.{run_number}

1.8 Monitoring und Wartung

1.8.1 Pipeline Health Monitoring

Kontinuierliche Überwachung der Pipeline-Gesundheit:

Key Performance Indicators

- **Build Success Rate**: Ziel ¿ 95%
- **Average Build Time**: Monitoring für Performance-Regression
- **Test Coverage**: Aufrechterhaltung ¿ 80% Line Coverage
- **Security Vulnerabilities**: Null High-Severity Issues
- **Dependency Freshness**: ¡ 30 Tage hinter aktueller Version

1.8.2 Wartungsplan

Wöchentliche Aufgaben

- Review von Dependabot Pull Requests
- Überprüfung der SonarQube Quality Trends
- Monitoring der Build-Performance-Metriken
- Review der Security Scan Ergebnisse

Monatliche Aufgaben

- Update der GitHub Actions Versionen
- Review und Update der Quality Gates
- Audit der Secret-Nutzung und Rotation
- Performance-Optimierung Review

1.9 Best Practices und Lessons Learned

1.9.1 Implementierte Best Practices

- **Fail-Fast Strategy**: Optimierte Job-Reihenfolge für schnelles Feedback
- **Security-First**: Secrets Management und Least-Privilege Access
- **Caching Strategy**: Dependency und Build-Caching für Performance
- **Parallel Execution**: Matrix-Builds für Effizienz
- **Quality Gates**: Automatische Pipeline-Unterbrechung bei Qualitätsproblemen

1.9.2 Herausforderungen und Lösungen

SonarQube Integration

- **Problem**: Quality Gate Timeouts bei großen Projekten
- **Lösung**: Timeout-Konfiguration und selective Scanning

Docker Build Optimization

- **Problem**: Lange Build-Zeiten
- **Lösung**: Multi-Stage Builds und Layer-Caching

Test Environment Consistency

- **Problem**: Unterschiedliche Verhalten zwischen lokaler und CI-Umgebung
- **Lösung**: Containerisierte Test-Umgebungen und Matrix-Builds

1.10 Fazit und Ausblick

1.10.1 Projektergebnisse

Das implementierte DevOps-Projekt demonstriert erfolgreich:

- **Vollautomatisierte CI/CD Pipeline** mit Quality Gates
- **Umfassende Test-Strategie** mit hoher Coverage
- **Kontinuierliche Code-Qualitätssicherung** durch SonarQube
- **Automatisierte Dependency-Verwaltung** für Sicherheit
- **Containerisierte Deployment-Strategie** für Skalierbarkeit

Literaturverzeichnis

Literaturverzeichnis

[1] GitHub Inc. *GitHub Actions Documentation*. `https://docs.github.com/en/actions`, 2024.

[2] SonarSource SA. *SonarQube Documentation*. `https://docs.sonarqube.org/`, 2024.

[3] Docker Inc. *Docker Documentation*. `https://docs.docker.com/`, 2024.

[4] Meta Platforms Inc. *Jest Testing Framework*. `https://jestjs.io/docs/getting-started`, 2024.

[5] GitHub Inc. *Dependabot Documentation*. `https://docs.github.com/en/code-security/dependabot`, 2024.

[6] Gene Kim, Jez Humble, Patrick Debois, John Willis. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.

[7] Jez Humble, David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.

[8] Node.js Foundation. *Node.js Documentation*. `https://nodejs.org/en/docs/`, 2024.

[9] Express.js Team. *Express.js Documentation*. `https://expressjs.com/`, 2024.