Notes App CI/CD and Security Report

Yassine El-Ghazi

February 15, 2026

Contents

1 Project Overview

The **Notes App** is a Node.js application that allows users to:

- Add notes
- Edit notes
- Delete notes
- List all notes

The project demonstrates the implementation of a complete **CI/CD pipeline** with integrated **DevSecOps practices** using GitHub Actions and Docker.

2 CI/CD Pipeline Implementation

The pipeline includes the following stages:

1. Build
2. Test
3. Security Scan
4. Docker Packaging
5. Deployment

3 Security Stage Configuration

The security stage uses **Gitleaks** to detect exposed secrets.

```
docker run --rm -v ${{ github.workspace }}:/repo \
  zricethezav/gitleaks:latest detect \
  --source=/repo \
  --redact \
  --verbose \
  --exit-code 1
```

The flag `--exit-code 1` ensures that the pipeline fails automatically if a secret is detected.

4 Security Test Performed

To validate the security mechanism, an intentional test was performed by inserting a hardcoded secret into `config.js`.

```
// BAD PRACTICE - Secret exposed
const config = {
  apiKey: "sk-abcdefghijklmnopqrstuvwxyz123456",
  databaseUrl: "mongodb://admin:password123@localhost/todos"
};

module.exports = config;
```

The commit was pushed to trigger the CI security scan.

5 Failed Security Test

The security stage failed as expected. Gitleaks detected exposed credentials.

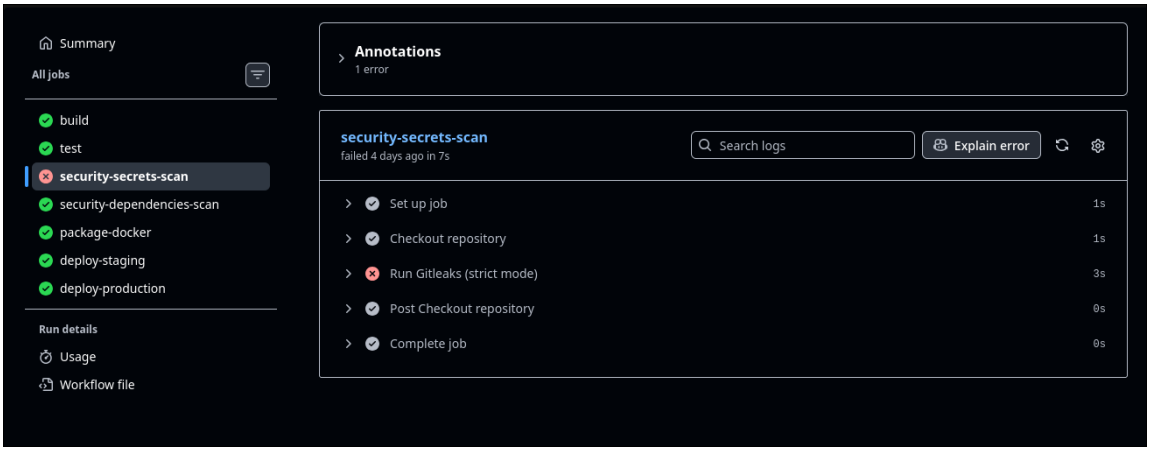

Figure 1: Failed security job showing detected secret and pipeline failure.

The workflow output included:

```
WRN leaks found: 2
Error: Process completed with exit code 1
```

6 Reason for Failure

The pipeline failed because:

- Sensitive information was hardcoded in the source code.
- Gitleaks detected secret patterns.
- The workflow was configured to fail automatically when secrets are found.

This confirms that the security control was correctly implemented and functioning as intended.

7 Resolution

7.1 Removing Hardcoded Secrets

The hardcoded credentials were removed from the application.

7.2 Using Environment Variables

The configuration file was updated to use environment variables:

```
// GOOD PRACTICE - Secure configuration
const config = {
  apiKey: process.env.API_KEY,
  databaseUrl: process.env.DATABASE_URL
};

module.exports = config;
```

7.3 Storing Secrets Securely in GitHub

Secrets were added in:

Repository → Settings → Secrets and Variables → Actions

The following repository secrets were configured:

- API_KEY
- DATABASE_URL
- DOCKER_USERNAME
- DOCKER_PASSWORD

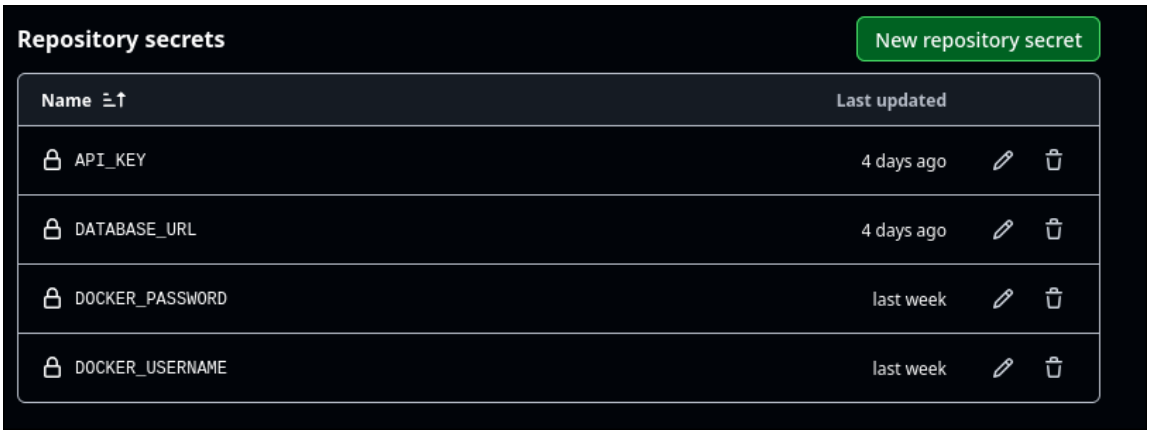

Figure 2: Repository secrets configured securely in GitHub.

8 Final Security Verification

After implementing secure configuration and removing hardcoded secrets, the security scan passed successfully.

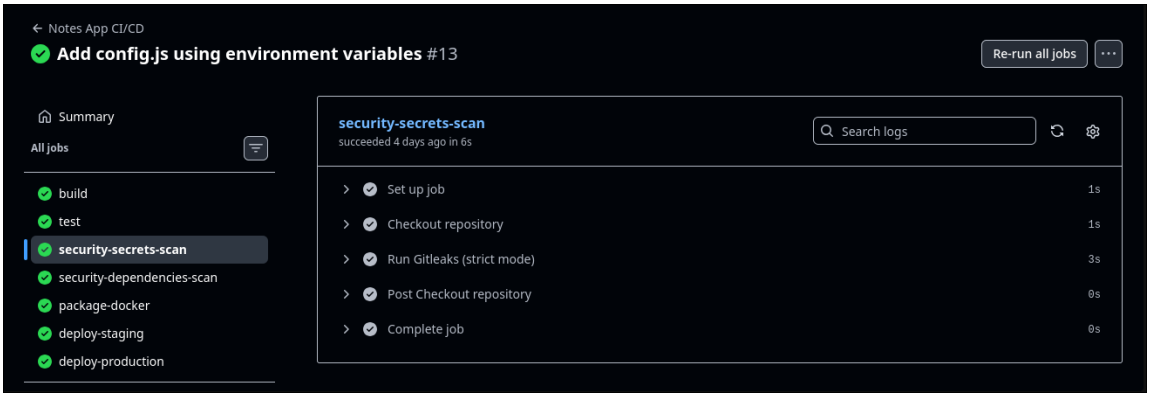

Figure 3: Successful security scan after implementing secure configuration.

9 Conclusion

This project demonstrates:

- Implementation of a full CI/CD pipeline
- Integration of automated security scanning
- Validation of security mechanisms through intentional failure testing
- Secure secret management using environment variables

The intentional failure of the security stage confirmed that the DevSecOps pipeline actively prevents insecure deployments.

The final configuration ensures that:

- No secrets are stored in source code
- Secrets are securely managed via GitHub repository secrets
- Security checks are enforced automatically during CI/CD execution