GitHub Actions CI/CD Workflows

This document provides comprehensive guidelines for AI agents to create and maintain CI/CD pipelines using GitHub Actions, with emphasis on secure secrets management and environment variable handling.

Overview

GitHub Actions enables automated build, test, and deployment workflows directly within GitHub repositories. This guide focuses on:

- Secure secrets management using GitHub Secrets
- Dynamic creation of `settings.json` and `.env` files
- Multi-environment deployment strategies
- Best practices for different technology stacks

Workflow Structure

Recommended Folder Organization

```
project-root/
├── .github/
│   ├── workflows/
│   │   ├── ci-cd.yml              # Main workflow
│   │   ├── build.yml
│   │   ├── test.yml
│   │   └── deploy.yml
├── src/
├── tests/
└── README.md
```

Basic Workflow Structure

```yaml
name: CI/CD Pipeline

on:
  push:
    branches:
      - main
      - develop
  pull_request:
    branches:
      - main

jobs:
  build:
```

```
    runs-on: ubuntu-latest

  test:
    runs-on: ubuntu-latest
    needs: build

  deploy:
    runs-on: ubuntu-latest
    needs: test
    if: github.ref == 'refs/heads/main'
```

Secrets Management

Step 1: Create Secrets in GitHub

1. Navigate to **Settings > Secrets and variables > Actions**
2. Click **New repository secret** (or **New organization secret**)
3. Add secrets:
   - `DB_CONNECTION_STRING`
   - `API_KEY`
   - `SECRET_KEY`
   - `AZURE_CREDENTIALS`
   - `AWS_ACCESS_KEY_ID`
   - `AWS_SECRET_ACCESS_KEY`
4. Save each secret

Step 2: Create Environment Secrets (Optional)

For environment-specific secrets:

1. Navigate to **Settings > Environments**
2. Create environments: `development`, `staging`, `production`
3. Add environment-specific secrets
4. Configure protection rules (approvals, wait timers, branch restrictions)

Step 3: Reference Secrets in Workflows

```
jobs:
  deploy:
    runs-on: ubuntu-latest
    environment: production  # Access production environment secrets
    steps:
      - name: Use secrets
        env:
          DATABASE_URL: ${{ secrets.DB_CONNECTION_STRING }}
          API_KEY: ${{ secrets.API_KEY }}
        run: |
```

```
          # Secrets are available as environment variables
          ./deploy.sh
```

Complete Workflow Examples

Example 1: Python Application with Database

```yaml
name: Python CI/CD

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

env:
  PYTHON_VERSION: '3.11'

jobs:
  build:
    name: Build Python Application
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: ${{ env.PYTHON_VERSION }}
        cache: 'pip'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Install dev dependencies
      run: |
        pip install pylint pytest coverage

    - name: Run pylint
      run: |
        pylint src/ --exit-zero

    - name: Upload artifacts
      uses: actions/upload-artifact@v4
      with:
```

```
        name: python-app
        path: |
          .
          !.git
          !.github
          !tests

  test:
    name: Run Tests
    runs-on: ubuntu-latest
    needs: build

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: ${{ env.PYTHON_VERSION }}
        cache: 'pip'

    - name: Install dependencies
      run: |
        pip install -r requirements.txt
        pip install pytest pytest-cov

    - name: Run tests with coverage
      run: |
        pytest tests/ --junitxml=test-results.xml --cov=src --cov-report=xml --
cov-report=html

    - name: Upload test results
      uses: actions/upload-artifact@v4
      if: always()
      with:
        name: test-results
        path: test-results.xml

    - name: Upload coverage reports
      uses: actions/upload-artifact@v4
      with:
        name: coverage-report
        path: |
          coverage.xml
          htmlcov/

#   deploy:
#     name: Deploy to Azure
#     runs-on: ubuntu-latest
#     needs: test
#     if: github.ref == 'refs/heads/main' && github.event_name == 'push'
#     environment: production
```

```
#     steps:
#     - name: Download artifacts
#       uses: actions/download-artifact@v4
#       with:
#         name: python-app
#         path: ./app

#     - name: Set up Python
#       uses: actions/setup-python@v5
#       with:
#         python-version: ${{ env.PYTHON_VERSION }}

#     - name: Create .env file
#       working-directory: ./app
#       env:
#         SECRET_KEY: ${{ secrets.SECRET_KEY }}
#         DB_CONNECTION: ${{ secrets.DB_CONNECTION_STRING }}
#         API_KEY: ${{ secrets.API_KEY }}
#       run: |
#         cat > .env << EOF
#         SECRET_KEY=${SECRET_KEY}
#         DB_CONNECTION=${DB_CONNECTION}
#         API_KEY=${API_KEY}
#         ENVIRONMENT=production
#         EOF

#     - name: Deploy to Azure Web App
#       uses: azure/webapps-deploy@v3
#       with:
#         app-name: ${{ secrets.AZURE_APP_NAME }}
#         publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
#         package: ./app

#     - name: Clean up .env file
#       if: always()
#       run: |
#         rm -f ./app/.env
```

Advanced Patterns

Using Reusable Workflows

.github/workflows/create-env-file.yml

```yaml
name: Create Environment File

on:
  workflow_call:
    inputs:
```

```
      working-directory:
        required: false
        type: string
        default: '.'
    secrets:
      SECRET_KEY:
        required: true
      DB_CONNECTION:
        required: true
      API_KEY:
        required: true

jobs:
  create-env:
    runs-on: ubuntu-latest
    steps:
      - name: Create .env file
        working-directory: ${{ inputs.working-directory }}
        env:
          SECRET_KEY: ${{ secrets.SECRET_KEY }}
          DB_CONNECTION: ${{ secrets.DB_CONNECTION }}
          API_KEY: ${{ secrets.API_KEY }}
        run: |
          cat > .env << EOF
          SECRET_KEY=${SECRET_KEY}
          DB_CONNECTION=${DB_CONNECTION}
          API_KEY=${API_KEY}
          EOF
```

Usage in main workflow:

```
jobs:
  setup-env:
    uses: ./.github/workflows/create-env-file.yml
    with:
      working-directory: './app'
    secrets:
      SECRET_KEY: ${{ secrets.SECRET_KEY }}
      DB_CONNECTION: ${{ secrets.DB_CONNECTION_STRING }}
      API_KEY: ${{ secrets.API_KEY }}
```

Composite Actions for Reusability

`.github/actions/setup-env/action.yml`

```
name: 'Setup Environment Variables'
description: 'Create environment configuration files from secrets'
```

```yaml
inputs:
  secret-key:
    description: 'Application secret key'
    required: true
  db-connection:
    description: 'Database connection string'
    required: true
  api-key:
    description: 'API key'
    required: true
  file-format:
    description: 'Format: env or json'
    required: false
    default: 'env'

runs:
  using: 'composite'
  steps:
    - name: Create .env file
      if: inputs.file-format == 'env'
      shell: bash
      run: |
        cat > .env << EOF
        SECRET_KEY=${{ inputs.secret-key }}
        DB_CONNECTION=${{ inputs.db-connection }}
        API_KEY=${{ inputs.api-key }}
        EOF

    - name: Create config.json file
      if: inputs.file-format == 'json'
      shell: bash
      run: |
        cat > config.json << EOF
        {
          "secretKey": "${{ inputs.secret-key }}",
          "dbConnection": "${{ inputs.db-connection }}",
          "apiKey": "${{ inputs.api-key }}"
        }
        EOF
```

Usage:

```yaml
- name: Setup environment
  uses: ./.github/actions/setup-env
  with:
    secret-key: ${{ secrets.SECRET_KEY }}
    db-connection: ${{ secrets.DB_CONNECTION_STRING }}
    api-key: ${{ secrets.API_KEY }}
    file-format: 'env'
```

Security Best Practices

1. Never Log Secrets

```
# BAD - Don't do this
- name: Debug
  run: echo "Secret is ${{ secrets.SECRET_KEY }}"

# GOOD - Use without displaying
- name: Use secret
  env:
    SECRET_KEY: ${{ secrets.SECRET_KEY }}
  run: ./deploy.sh
```

2. Always Clean Up Sensitive Files

```
- name: Clean up sensitive files
  if: always()
  run: |
    rm -f .env
    rm -f appsettings.Production.json
    rm -f secrets.json
    rm -f config.production.json
```

```
environment:
  name: production
  url: https://myapp.com
```

4. Limit Secret Exposure with Environments

```
# Secrets only available in this job
deploy:
  runs-on: ubuntu-latest
  environment: production  # production secrets only here
  steps:
    - name: Deploy
      env:
        SECRET: ${{ secrets.PRODUCTION_SECRET }}
      run: ./deploy.sh
```

5. Use GITHUB_TOKEN for GitHub API Calls

```
- name: Create release
  env:
    GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}  # Auto-generated
  run: gh release create v1.0.0
```

6. Mask Custom Values in Logs

```
- name: Set and mask custom secret
  run: |
    CUSTOM_SECRET=$(generate-secret)
    echo "::add-mask::$CUSTOM_SECRET"
    echo "CUSTOM_SECRET=$CUSTOM_SECRET" >> $GITHUB_ENV
```

Working with Different Cloud Providers

Azure Deployment

```
- name: Login to Azure
  uses: azure/login@v1
  with:
    creds: ${{ secrets.AZURE_CREDENTIALS }}

- name: Deploy to Azure Web App
  uses: azure/webapps-deploy@v3
  with:
    app-name: ${{ secrets.AZURE_APP_NAME }}
    package: ./app
```

AWS Deployment

```
- name: Configure AWS credentials
  uses: aws-actions/configure-aws-credentials@v4
  with:
    aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
    aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
    aws-region: us-east-1

- name: Deploy to Elastic Beanstalk
  run: |
    eb deploy production-env
```

Google Cloud Platform

```yaml
- name: Authenticate to Google Cloud
  uses: google-github-actions/auth@v2
  with:
    credentials_json: ${{ secrets.GCP_CREDENTIALS }}

- name: Deploy to Cloud Run
  uses: google-github-actions/deploy-cloudrun@v2
  with:
    service: my-service
    image: gcr.io/my-project/my-image:${{ github.sha }}
```

Docker Hub

```yaml
- name: Login to Docker Hub
  uses: docker/login-action@v3
  with:
    username: ${{ secrets.DOCKERHUB_USERNAME }}
    password: ${{ secrets.DOCKERHUB_TOKEN }}

- name: Build and push
  uses: docker/build-push-action@v5
  with:
    push: true
    tags: user/app:latest
```

Advanced Secrets Patterns

Using GitHub CLI to Manage Secrets

```bash
# Set a secret
gh secret set SECRET_NAME --body "secret-value"

# Set from file
gh secret set SECRET_NAME < secret.txt

# List secrets
gh secret list

# Delete a secret
gh secret delete SECRET_NAME

# Set organization secret
gh secret set SECRET_NAME --org myorg --visibility all
```

Environment Variables from Files

```
- name: Load .env file
  run: |
    if [ -f .env.example ]; then
      export $(cat .env.example | grep -v '^#' | xargs)
    fi
    # Override with secrets
    echo "API_KEY=${{ secrets.API_KEY }}" >> $GITHUB_ENV
    echo "DB_URL=${{ secrets.DB_URL }}" >> $GITHUB_ENV
```

Using GitHub App Token for Enhanced Permissions

```
- name: Generate token
  id: generate-token
  uses: actions/create-github-app-token@v1
  with:
    app-id: ${{ secrets.APP_ID }}
    private-key: ${{ secrets.APP_PRIVATE_KEY }}

- name: Use token
  env:
    GH_TOKEN: ${{ steps.generate-token.outputs.token }}
  run: gh api /repos/${{ github.repository }}/issues
```

Troubleshooting

Common Issues

Issue: Secrets not resolving

```
# Correct syntax
${{ secrets.SECRET_NAME }}

# Incorrect
${secrets.SECRET_NAME}    # Wrong
$secrets.SECRET_NAME      # Wrong
{secrets.SECRET_NAME}     # Wrong
```

Issue: Secrets in pull requests from forks

- Secrets are NOT available in workflows triggered by pull requests from forks
- Use `pull_request_target` carefully (security risk)

- Consider using labels to trigger workflows with secrets

```
on:
  pull_request_target:  # Use with caution!
    types: [labeled]

jobs:
  deploy-preview:
    if: contains(github.event.pull_request.labels.*.name, 'safe-to-deploy')
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          ref: ${{ github.event.pull_request.head.sha }}
```

Issue: File not found after creation

```
- name: Debug file creation
  run: |
    pwd
    ls -la
    cat > .env << EOF
    KEY=VALUE
    EOF
    ls -la .env
    cat .env
```

Issue: Multiline secrets

```
# For multiline secrets (e.g., SSH keys, certificates)
- name: Use multiline secret
  run: |
    echo "${{ secrets.SSH_PRIVATE_KEY }}" > private_key
    chmod 600 private_key
```

Issue: Secrets with special characters

```
# When setting secrets with special characters via CLI
gh secret set SECRET_NAME --body 'value-with-$pecial-chars'

# Use base64 encoding for complex values
echo "complex$value#here" | base64 | gh secret set ENCODED_SECRET --body -

# In workflow, decode
- name: Decode secret
```

```
    run: |
      echo "${{ secrets.ENCODED_SECRET }}" | base64 -d > secret.txt
```

Workflow Triggers and Conditions

Common Trigger Patterns

```yaml
# Trigger on push to specific branches
on:
  push:
    branches: [ main, develop ]
    paths-ignore:
      - '**.md'
      - 'docs/**'

# Trigger on pull requests
on:
  pull_request:
    types: [opened, synchronize, reopened]
    branches: [ main ]

# Trigger on tags
on:
  push:
    tags:
      - 'v*.*.*'

# Manual trigger with inputs
on:
  workflow_dispatch:
    inputs:
      environment:
        description: 'Environment to deploy to'
        required: true
        type: choice
        options:
          - development
          - staging
          - production

# Scheduled trigger
on:
  schedule:
    - cron: '0 0 * * 0'  # Weekly on Sunday at midnight
```

Conditional Execution

```yaml
jobs:
  deploy:
    runs-on: ubuntu-latest
    # Only deploy on main branch, push events, and when tests pass
    if: |
      github.ref == 'refs/heads/main' &&
      github.event_name == 'push' &&
      success()

  notify-failure:
    runs-on: ubuntu-latest
    if: failure()
    steps:
      - name: Send notification
        run: echo "Build failed!"
```

AI Agent Specific Guidelines

When creating GitHub Actions workflows:

1. **Always use GitHub Secrets** for sensitive data, never hardcode
2. **Create environment-specific configuration files** dynamically in deployment jobs
3. **Clean up sensitive files** using `if: always()` condition
4. **Use environments** with protection rules for production deployments
5. **Implement proper job dependencies** with `needs` keyword
6. **Add appropriate conditions** to prevent accidental production deployments
7. **Use reusable workflows or composite actions** for common patterns
8. **Enable artifact uploading** for build outputs and test results
9. **Implement concurrency controls** to prevent parallel deployments
10. **Add status checks** and notifications for workflow completion

Concurrency Control

```yaml
concurrency:
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true  # Cancel previous runs
```

Notification Pattern

```yaml
jobs:
  notify:
    runs-on: ubuntu-latest
    if: always()
    needs: [build, test, deploy]
    steps:
```

```
      - name: Notify success
        if: needs.deploy.result == 'success'
        run: |
          # Send success notification

      - name: Notify failure
        if: needs.deploy.result == 'failure'
        run: |
          # Send failure notification
```

Useful GitHub Actions

Official Actions

- `actions/checkout@v4` - Checkout repository code
- `actions/setup-node@v4` - Setup Node.js environment
- `actions/setup-python@v5` - Setup Python environment
- `actions/setup-dotnet@v4` - Setup .NET environment
- `actions/upload-artifact@v4` - Upload build artifacts
- `actions/download-artifact@v4` - Download artifacts
- `actions/cache@v4` - Cache dependencies

Third-Party Actions

- `azure/login@v1` - Login to Azure
- `azure/webapps-deploy@v3` - Deploy to Azure Web Apps
- `aws-actions/configure-aws-credentials@v4` - Configure AWS credentials
- `google-github-actions/auth@v2` - Authenticate to GCP
- `docker/build-push-action@v5` - Build and push Docker images
- `docker/login-action@v3` - Login to Docker registry

Context Variables

- `${{ github.ref }}` - Git ref (branch or tag)
- `${{ github.sha }}` - Commit SHA
- `${{ github.actor }}` - User who triggered the workflow
- `${{ github.repository }}` - Repository name (owner/repo)
- `${{ github.event_name }}` - Event that triggered workflow
- `${{ runner.os }}` - Operating system of runner
- `${{ job.status }}` - Current status of job

Example: Complete Production-Ready Workflow

```
name: Production CI/CD Pipeline
```

```yaml
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
  workflow_dispatch:
    inputs:
      environment:
        description: 'Deployment environment'
        required: true
        type: choice
        options: [staging, production]

env:
  NODE_VERSION: '18.x'
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

concurrency:
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true

jobs:
  build:
    name: Build Application
    runs-on: ubuntu-latest
    outputs:
      version: ${{ steps.version.outputs.version }}

    steps:
    - name: Checkout code
      uses: actions/checkout@v4
      with:
        fetch-depth: 0

    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: ${{ env.NODE_VERSION }}
        cache: 'npm'

    - name: Get version
      id: version
      run: echo "version=$(node -p "require('./package.json').version")" >>
$GITHUB_OUTPUT

    - name: Install dependencies
      run: npm ci

    - name: Run linter
      run: npm run lint

    - name: Build
```

```
      run: npm run build

    - name: Upload build artifacts
      uses: actions/upload-artifact@v4
      with:
        name: build-${{ steps.version.outputs.version }}
        path: |
          dist/
          package.json
          package-lock.json
        retention-days: 30

  test:
    name: Run Tests
    runs-on: ubuntu-latest
    needs: build

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: ${{ env.NODE_VERSION }}
        cache: 'npm'

    - name: Install dependencies
      run: npm ci

    - name: Run unit tests
      run: npm run test:unit -- --coverage

    - name: Run integration tests
      run: npm run test:integration

    - name: Upload coverage
      uses: actions/upload-artifact@v4
      with:
        name: coverage-report
        path: coverage/

  security:
    name: Security Scanning
    runs-on: ubuntu-latest
    needs: build

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Run npm audit
      run: npm audit --audit-level=moderate
```

```yaml
        continue-on-error: true

      - name: Run Snyk security scan
        uses: snyk/actions/node@master
        continue-on-error: true
        env:
          SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}

    deploy-staging:
      name: Deploy to Staging
      runs-on: ubuntu-latest
      needs: [test, security]
      if: github.event_name == 'push' && github.ref == 'refs/heads/main'
      environment:
        name: staging
        url: https://staging.myapp.com

      steps:
      - name: Download artifacts
        uses: actions/download-artifact@v4
        with:
          name: build-${{ needs.build.outputs.version }}
          path: ./app

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}

      - name: Install production dependencies
        working-directory: ./app
        run: npm ci --only=production

      - name: Create environment file
        working-directory: ./app
        env:
          API_KEY: ${{ secrets.STAGING_API_KEY }}
          DATABASE_URL: ${{ secrets.STAGING_DATABASE_URL }}
          SECRET_KEY: ${{ secrets.STAGING_SECRET_KEY }}
        run: |
          cat > .env.production << EOF
          NODE_ENV=production
          API_KEY=${API_KEY}
          DATABASE_URL=${DATABASE_URL}
          SECRET_KEY=${SECRET_KEY}
          EOF

      - name: Deploy to Azure Staging
        uses: azure/webapps-deploy@v3
        with:
          app-name: ${{ secrets.AZURE_STAGING_APP_NAME }}
          publish-profile: ${{ secrets.AZURE_STAGING_PUBLISH_PROFILE }}
          package: ./app
```

```yaml
    - name: Run smoke tests
      run: |
        curl -f https://staging.myapp.com/health || exit 1

    - name: Clean up
      if: always()
      run: rm -f ./app/.env.production

  deploy-production:
    name: Deploy to Production
    runs-on: ubuntu-latest
    needs: [deploy-staging]
    if: github.event_name == 'push' && github.ref == 'refs/heads/main'
    environment:
      name: production
      url: https://myapp.com

    steps:
    - name: Download artifacts
      uses: actions/download-artifact@v4
      with:
        name: build-${{ needs.build.outputs.version }}
        path: ./app

    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: ${{ env.NODE_VERSION }}

    - name: Install production dependencies
      working-directory: ./app
      run: npm ci --only=production

    - name: Create environment file
      working-directory: ./app
      env:
        API_KEY: ${{ secrets.PROD_API_KEY }}
        DATABASE_URL: ${{ secrets.PROD_DATABASE_URL }}
        SECRET_KEY: ${{ secrets.PROD_SECRET_KEY }}
      run: |
        cat > .env.production << EOF
        NODE_ENV=production
        API_KEY=${API_KEY}
        DATABASE_URL=${DATABASE_URL}
        SECRET_KEY=${SECRET_KEY}
        EOF

    - name: Deploy to Azure Production
      uses: azure/webapps-deploy@v3
      with:
        app-name: ${{ secrets.AZURE_PROD_APP_NAME }}
        publish-profile: ${{ secrets.AZURE_PROD_PUBLISH_PROFILE }}
```

```
          package: ./app

      - name: Run smoke tests
        run: |
          curl -f https://myapp.com/health || exit 1

      - name: Create release
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        run: |
          gh release create "v${{ needs.build.outputs.version }}" \
            --title "Release v${{ needs.build.outputs.version }}" \
            --notes "Automated release from commit ${{ github.sha }}"

      - name: Clean up
        if: always()
        run: rm -f ./app/.env.production

    notify:
      name: Notify Results
      runs-on: ubuntu-latest
      needs: [deploy-production]
      if: always()

      steps:
      - name: Notify success
        if: needs.deploy-production.result == 'success'
        run: |
          echo "Deployment successful!"
          # Add your notification logic (Slack, email, etc.)

      - name: Notify failure
        if: needs.deploy-production.result == 'failure'
        run: |
          echo "Deployment failed!"
          # Add your notification logic
```

Additional Resources

Documentation Links

- GitHub Actions Documentation
- Workflow syntax
- Encrypted secrets
- Environments
- Reusable workflows

Best Practices Summary

1. Use secrets for all sensitive data

2. Implement environment protection rules
3. Use `if: always()` for cleanup steps
4. Cache dependencies to speed up workflows
5. Use matrix strategies for testing multiple versions
6. Implement proper error handling
7. Add concurrency controls
8. Use job outputs to pass data between jobs
9. Implement proper logging without exposing secrets
10. Regularly rotate secrets and credentials

Version: 1.0
Last Updated: 2025-12-18
Maintainer: James Stanbridge