← AWS for Test Automation

AI in Test Automation →

CI/CD & Test Reporting

Table of Contents

CI/CD Concepts and Workflows

What is CI/CD?

Continuous Integration (CI):

- Automatically build and test code changes
- Integrate code frequently (multiple times per day)
- Catch bugs early

- Automatically deploy to staging/production
- Ensure software is always in deployable state
- Fast and reliable releases

Continuous Deployment:

- Automatically deploy every change to production
- No manual intervention
- Ultimate automation

CI/CD Pipeline

```
┌──────────┐    ┌───────┐    ┌──────┐    ┌────────┐    ┌──────────┐
|   Code   | -> | Build | -> | Test | -> | Report | -> |  Deploy  |
|  Commit  |    |       |    |      |    |        |    |          |
└──────────┘    └───────┘    └──────┘    └────────┘    └──────────┘
```

Benefits of CI/CD for Testing

- **Automated execution**: Tests run on every commit
- **Early detection**: Find bugs immediately
- **Consistent environment**: Same environment every time
- **Fast feedback**: Developers know results quickly
- **Quality gates**: Block bad code from merging
- **Parallel execution**: Run tests faster
- **Historical tracking**: Track test trends over time

CI/CD Tools Comparison

GitHub Actions	Cloud	GitHub repos	Free tier available
Jenkins	Self-hosted	Enterprise	Free (hosting cost)
GitLab CI	Cloud/Self-hosted	GitLab repos	Free tier available
CircleCI	Cloud	Fast builds	Free tier available
Travis CI	Cloud	Open source	Free for open source
Azure Pipelines	Cloud	Microsoft stack	Free tier available

GitHub Actions for Automation Pipelines

What is GitHub Actions?

GitHub Actions is a CI/CD platform integrated directly into GitHub repositories.

Key Features:

- YAML-based workflow definition
- Matrix builds (parallel testing)
- Marketplace with thousands of actions
- Integrated with GitHub
- Free for public repos
- 2000 minutes/month free for private repos

Workflow Basics

Workflow Structure:

```
  job-name:
    runs-on: ubuntu-latest
    steps:
      - name: Step name
        run: command
```

Basic Playwright Workflow

.github/workflows/playwright.yml:

```
name: Playwright Tests

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]
  schedule:
    - cron: '0 2 * * *'  # Daily at 2 AM

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'
          cache: 'npm'

      - name: Install dependencies
        run: npm ci

      - name: Install Playwright browsers
        run: npx playwright install --with-deps

      - name: Run Playwright tests
        run: npx playwright test
        uses: actions/upload-artifact@v3
        with:
          name: playwright-report
          path: playwright-report/
          retention-days: 30
```

Cypress Workflow

.github/workflows/cypress.yml:

```
 name: Cypress Tests

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

jobs:
  cypress-run:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Cypress run
        uses: cypress-io/github-action@v5
        with:
          build: npm run build
          start: npm start
          wait-on: 'http://localhost:3000'
          browser: chrome
          record: true
        env:
          CYPRESS_RECORD_KEY: ${{ secrets.CYPRESS_RECORD_KEY }}

      - name: Upload screenshots
        if: failure()
        uses: actions/upload-artifact@v3
        with:
          name: cypress-screenshots
          path: cypress/screenshots
        uses: actions/upload-artifact@v3
        with:
          name: cypress-videos
          path: cypress/videos
```

Matrix Strategy (Parallel Testing)

Test across multiple configurations:

```
name: Cross-Browser Tests

on: [push]

jobs:
  test:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macos-latest]
        browser: [chromium, firefox, webkit]
        node-version: [16, 18]

    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js ${{ matrix.node-version }}
        uses: actions/setup-node@v3
        with:
          node-version: ${{ matrix.node-version }}

      - name: Install dependencies
        run: npm ci

      - name: Install Playwright
        run: npx playwright install --with-deps ${{ matrix.browser }}

      - name: Run tests on ${{ matrix.browser }}
        run: npx playwright test --project=${{ matrix.browser }}
```

Secrets Management

```
Repository → Settings → Secrets → Actions → New repository secret
```

Use in workflow:

```
steps:
  - name: Run API tests
    env:
      API_KEY: ${{ secrets.API_KEY }}
      API_URL: ${{ secrets.API_URL }}
    run: npm run test:api
```

BrowserStack Integration

Run tests on BrowserStack:

```
 name: BrowserStack Tests

on: [push]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'

      - name: Install dependencies
        run: npm ci

      - name: Run tests on BrowserStack
        env:
          BROWSERSTACK_USERNAME: ${{ secrets.BROWSERSTACK_USERNAME }}
          BROWSERSTACK_ACCESS_KEY: ${{ secrets.BROWSERSTACK_ACCESS_KEY }}
        run: |
          npx playwright test --project=browserstack-chrome
```

Conditional Steps

Run steps based on conditions:

```
steps:
  - name: Run unit tests
    run: npm run test:unit

  - name: Run integration tests
    if: github.ref == 'refs/heads/main'
    run: npm run test:integration

  - name: Deploy to staging
    if: github.event_name == 'push' && github.ref == 'refs/heads/develop'
    run: npm run deploy:staging

  - name: Deploy to production
    if: github.event_name == 'push' && github.ref == 'refs/heads/main'
    run: npm run deploy:production
```

Running Tests in CI Environments

CI Environment Configuration

Environment variables:

```
env:
  CI: true
  NODE_ENV: test
  HEADLESS: true
  BASE_URL: https://staging.example.com
```

Playwright CI configuration:

```
export default defineConfig({
  use: {
    headless: process.env.CI === 'true',
    video: process.env.CI === 'true' ? 'retain-on-failure' : 'off',
    screenshot: 'only-on-failure',
    trace: 'retain-on-failure',
    baseURL: process.env.BASE_URL || 'http://localhost:3000',
  },

  // CI-specific settings
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 4 : undefined,

  reporter: process.env.CI
    ? [['html'], ['junit', { outputFile: 'test-results/junit.xml' }]]
    : [['html']],
})
```

Parallel Execution in CI

Playwright sharding:

```
 name: Playwright Tests (Sharded)

on: [push]

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        shardIndex: [1, 2, 3, 4]
        shardTotal: [4]

    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      - run: npm ci
      - run: npx playwright install --with-deps

      - name: Run tests (shard ${{ matrix.shardIndex }})
        run: npx playwright test --shard=${{ matrix.shardIndex }}/${{ matr
        uses: actions/upload-artifact@v3
        with:
          name: test-results-${{ matrix.shardIndex }}
          path: test-results/
```

Caching Dependencies

Cache npm dependencies:

```
steps:
  - name: Cache node modules
    uses: actions/cache@v3
    with:
      path: ~/.npm
      key: ${{ runner.os }}-node-${{ hashFiles('**/package-lock.json') }}
      restore-keys: |
        ${{ runner.os }}-node-

  - name: Install dependencies
    run: npm ci
```

Cache Playwright browsers:

```
steps:
  - name: Cache Playwright browsers
    uses: actions/cache@v3
    id: playwright-cache
    with:
      path: ~/.cache/ms-playwright
      key: ${{ runner.os }}-playwright-${{ hashFiles('**/package-lock.json

  - name: Install Playwright browsers
    if: steps.playwright-cache.outputs.cache-hit != 'true'
    run: npx playwright install --with-deps
```

Quality Gates

Fail build on test failures:

```
    run: npx playwright test

  - name: Check test results
    if: failure()
    run: |
      echo "Tests failed! Blocking merge."
      exit 1
```

Minimum coverage requirement:

```
steps:
  - name: Run tests with coverage
    run: npm run test:coverage

  - name: Check coverage threshold
    run: |
      COVERAGE=$(cat coverage/coverage-summary.json | jq '.total.lines.pct
      if (( $(echo "$COVERAGE < 80" | bc -l) )); then
        echo "Coverage $COVERAGE% is below 80% threshold"
        exit 1
      fi
```

Test Reporting with Allure

What is Allure?

Allure is a flexible, lightweight test reporting framework that creates beautiful, detailed test reports.

Features:

- Beautiful HTML reports
- Test history and trends
- Categorization and tagging
- Screenshots and attachments

Installation

For Playwright:

```
npm install --save-dev allure-playwright allure-commandline
```

For Cypress:

```
npm install --save-dev @shelex/cypress-allure-plugin allure-commandline
```

Playwright Configuration

playwright.config.ts:

```
import { defineConfig } from '@playwright/test'

export default defineConfig({
  reporter: [
    ['html'],
    ['allure-playwright', {
      outputFolder: 'allure-results',
      detail: true,
      suiteTitle: true,
    }]
  ],
})
```

Generate report:

```
# Run tests
npx playwright test

# Generate Allure report
npx allure generate allure-results --clean -o allure-report

# Open report
```

Cypress Configuration

cypress.config.js:

```
const allureWriter = require('@shelex/cypress-allure-plugin/writer')

module.exports = defineConfig({
  e2e: {
    setupNodeEvents(on, config) {
      allureWriter(on, config)
      return config
    },
  },
})
```

cypress/support/e2e.js:

```
import '@shelex/cypress-allure-plugin'
```

Allure Annotations

Playwright:

```
import { test } from '@playwright/test'

test.describe('Login Tests', () => {
  test('successful login', async ({ page }) => {
    await test.step('Navigate to login page', async () => {
      await page.goto('/login')
    })

    await test.step('Enter credentials', async () => {
      await page.fill('#username', 'testuser')
      await page.fill('#password', 'password123')
    })

    await test.step('Click login button', async () => {
      await page.click('button[type="submit"]')
    })
    })
  })
})
```

Add attachments:

```
import { test } from '@playwright/test'

test('test with screenshot', async ({ page }) => {
  await page.goto('/')

  const screenshot = await page.screenshot()
  await test.info().attach('screenshot', {
    body: screenshot,
    contentType: 'image/png'
  })
})
```

Allure in GitHub Actions

```
name: Tests with Allure

on: [push]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      - run: npm ci
      - run: npx playwright install --with-deps

      - name: Run tests
        run: npx playwright test
        continue-on-error: true

      - name: Generate Allure report
        if: always()
        run: npx allure generate allure-results --clean -o allure-report
        uses: actions/upload-artifact@v3
        with:
          name: allure-report
          path: allure-report/

      - name: Deploy report to GitHub Pages
        if: always()
        uses: peaceiris/actions-gh-pages@v3
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./allure-report
```

Mochawesome Integration

What is Mochawesome?

Mochawesome is a custom reporter for Mocha test framework that generates HTML/JSON reports.

Features:

- Clean, modern UI
- Test filtering
- Quick stats
- Pass/fail visibility
- Screenshot support

Installation

```
npm install --save-dev mochawesome mochawesome-merge mochawesome-report-ge
```

Playwright with Mocha Reporter

```
export default defineConfig({
  reporter: [
    ['html'],
    ['json', { outputFile: 'test-results/results.json' }],
    ['junit', { outputFile: 'test-results/junit.xml' }],
  ],
})
```

Cypress Configuration

cypress.config.js:

```
module.exports = defineConfig({
  reporter: 'mochawesome',
  reporterOptions: {
    reportDir: 'cypress/results',
    overwrite: false,
    html: true,
    json: true,
    timestamp: 'mmddyyyy_HHMMss'
  },

  e2e: {
    setupNodeEvents(on, config) {
      return config
    },
  },
})
```

Merge Multiple Reports

After test execution:

```
# Run tests
npx cypress run

# Merge reports
npx mochawesome-merge "cypress/results/*.json" > cypress/results/merged-re

# Generate HTML
```

```
  --reportPageTitle "Cypress Tests"
```

GitHub Actions Integration

```
name: Cypress with Mochawesome

on: [push]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      - run: npm ci

      - name: Run Cypress tests
        run: npx cypress run
        continue-on-error: true

      - name: Merge test results
        if: always()
        run: |
          npx mochawesome-merge "cypress/results/*.json" > merged.json
          npx marge merged.json -o cypress/report

      - name: Upload report
        if: always()
        uses: actions/upload-artifact@v3
        with:
          name: test-report
          path: cypress/report/
```

ReportPortal for Real-Time Analytics

What is ReportPortal?

Features:

- Real-time test execution tracking
- Historical test data and trends
- AI-powered failure analysis
- Defect management integration
- Team collaboration
- Custom dashboards
- Flaky test detection

ReportPortal Setup

Docker Compose setup:

```
 # docker-compose.yml
version: '3.8'

services:
  postgres:
    image: postgres:12
    environment:
      POSTGRES_USER: rpuser
      POSTGRES_PASSWORD: rppass
      POSTGRES_DB: reportportal
    volumes:
      - postgres_data:/var/lib/postgresql/data

  reportportal:
    image: reportportal/service-api:5.7.0
    depends_on:
      - postgres
    environment:
      RP_DB_HOST: postgres
      RP_DB_USER: rpuser
      RP_DB_PASS: rppass
      RP_DB_NAME: reportportal
    ports:
      - "8080:8080"
      RP_SERVER_PORT: 8080
    ports:
      - "8081:8080"
    depends_on:
      - reportportal

volumes:
  postgres_data:
```

Start ReportPortal:

```
docker-compose up -d
```

Access UI:

```
http://localhost:8081
Default credentials:
  Username: default
  Password: 1q2w3e
```

Playwright Integration

Install agent:

```
npm install --save-dev @reportportal/agent-js-playwright
```

playwright.config.ts:

```
import { defineConfig } from '@playwright/test'

export default defineConfig({
  reporter: [
    ['@reportportal/agent-js-playwright', {
      apiKey: process.env.RP_API_KEY,
      endpoint: 'http://localhost:8080/api/v1',
      project: 'default_personal',
      launch: 'Playwright Tests',
        { key: 'browser', value: 'chromium' }
      ]
    }]
  ],
})
```

Cypress Integration

Install plugin:

```
npm install --save-dev @reportportal/agent-js-cypress
```

reportportal.config.js:

```
module.exports = {
  apiKey: process.env.RP_API_KEY,
  endpoint: 'http://localhost:8080/api/v1',
  project: 'default_personal',
  launch: 'Cypress Tests',
  description: 'E2E tests',
  attributes: [
    { key: 'env', value: 'qa' },
    { key: 'suite', value: 'smoke' }
  ]
}
```

cypress.config.js:

```
const registerReportPortalPlugin = require('@reportportal/agent-js-cypress

module.exports = defineConfig({
  e2e: {
    setupNodeEvents(on, config) {
      return registerReportPortalPlugin(on, config)
    },
  },
})
```

```
 name: Tests with ReportPortal

on: [push]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      - run: npm ci
      - run: npx playwright install --with-deps

      - name: Run tests with ReportPortal
        env:
          RP_API_KEY: ${{ secrets.RP_API_KEY }}
          RP_LAUNCH_NAME: "Build #${{ github.run_number }}"
        run: npx playwright test
```

ReportPortal Features

Defect Types:

- Product Bug (PB)
- Automation Bug (AB)
- System Issue (SI)
- To Investigate (TI)
- No Defect (ND)

Auto-analysis:

- AI suggests defect types
- Identifies similar failures
- Recommends defect linkage

Dashboards:

- Overall statistics

- Pass rate trend
- Flaky tests

Slack and Email Notifications

Slack Notifications

GitHub Actions Slack notification:

```
 name: Tests with Slack Notification

on: [push]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      - run: npm ci
      - run: npx playwright install --with-deps

      - name: Run tests
        id: tests
        run: npx playwright test
        continue-on-error: true

      - name: Send Slack notification (Success)
        if: steps.tests.outcome == 'success'
        uses: slackapi/slack-github-action@v1
        with:
          webhook-url: ${{ secrets.SLACK_WEBHOOK_URL }}
          payload: |
            {
              "text": "✅ Tests passed!",
              "blocks": [
                {
```

```
                "text": "*Tests Passed* ✅ \nBuild: ${{ github.run_numb
              }
            }
          ]
        }

    - name: Send Slack notification (Failure)
      if: steps.tests.outcome == 'failure'
      uses: slackapi/slack-github-action@v1
      with:
        webhook-url: ${{ secrets.SLACK_WEBHOOK_URL }}
        payload: |
          {
            "text": "❌ Tests failed!",
            "blocks": [
              {
                "type": "section",
                "text": {
                  "type": "mrkdwn",
                  "text": "*Tests Failed* ❌ \nBuild: ${{ github.run_numb
                }
              }
            ]
          }
```

Custom Slack notification script:

```
// notify-slack.ts
import axios from 'axios'

async function sendSlackNotification(webhookUrl: string, message: any) {
  try {
    await axios.post(webhookUrl, message)
    console.log('Slack notification sent')
  } catch (error) {
    console.error('Failed to send Slack notification:', error)
  }
}

const testResults = {
  total: 100,
  passed: 95,
  failed: 5,
const message = {
  text: 'Test Results',
  blocks: [
    {
      type: 'header',
      text: {
        type: 'plain_text',
        text: '🧪 Test Execution Complete'
      }
    },
    {
      type: 'section',
      fields: [
        { type: 'mrkdwn', text: `*Total:*\n${testResults.total}` },
        { type: 'mrkdwn', text: `*Passed:*\n✅ ${testResults.passed}` },
        { type: 'mrkdwn', text: `*Failed:*\n❌ ${testResults.failed}` },
        { type: 'mrkdwn', text: `*Duration:*\n⏱ ${testResults.duration}` }
      ]
    }
  ]
}

sendSlackNotification(process.env.SLACK_WEBHOOK_URL!, message)
```

Email Notifications

GitHub Actions email:

```
steps:
  - name: Send email notification
    if: always()
    uses: dawidd6/action-send-mail@v3
    with:
      server_address: smtp.gmail.com
      server_port: 587
      username: ${{ secrets.EMAIL_USERNAME }}
      password: ${{ secrets.EMAIL_PASSWORD }}
      subject: "Test Results - Build #${{ github.run_number }}"
      to: team@example.com
      from: ci@example.com
      body: |
        Test execution completed.
        View details: ${{ github.server_url }}/${{ github.repository }}/ac
      attachments: test-results/report.html
```

Nodemailer script:

```
 // send-email.ts
import nodemailer from 'nodemailer'
import * as fs from 'fs'

async function sendEmail() {
  const transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: {
      user: process.env.EMAIL_USER,
      pass: process.env.EMAIL_PASS
    }
  })

  const htmlReport = fs.readFileSync('test-report.html', 'utf-8')

  const mailOptions = {
    from: 'ci@example.com',
    to: 'team@example.com',
    subject: 'Test Results - Daily Run',
    html: `
      <h2>Test Execution Report</h2>
      <p><strong>Date:</strong> ${new Date().toLocaleDateString()}</p>
      <p><strong>Status:</strong> Completed</p>
      ${htmlReport}
    `,
    attachments: [
      {
        filename: 'test-report.html',
        path: 'test-report.html'
      }
    ]
  }

  await transporter.sendMail(mailOptions)
  console.log('Email sent successfully')
}
```

Practice Exercises

Exercise 1: Basic GitHub Actions Workflow

Task: Create a complete CI workflow

```
 # Create .github/workflows/ci.yml that:
# Triggers on push and PR
# Runs on multiple Node versions (16, 18)
# Installs dependencies with caching
# Runs linting
# Runs unit tests
# Runs Playwright tests
# Uploads test reports
# Sends Slack notification

# Your implementation
```

Exercise 2: Matrix Testing

Task: Set up cross-browser matrix testing

```
 # Create workflow that:
# Tests on Chrome, Firefox, Safari
# Tests on Windows, Linux, macOS
# Runs tests in parallel
# Collects all results
# Generates combined report

# Your implementation
```

Exercise 3: Allure Reporting

Task: Implement comprehensive Allure reporting

```
// 4. Add custom categories
// 5. Generate and publish report to GitHub Pages

// Your implementation
```

Exercise 4: ReportPortal Integration

Task: Set up ReportPortal with CI/CD

```
 # Deploy ReportPortal with Docker
# Configure Playwright agent
# Add custom attributes and descriptions
# Integrate with GitHub Actions
# Set up auto-analysis

# Your implementation
```

Exercise 5: Complete CI/CD Pipeline

Task: Build end-to-end pipeline

```
 # Create pipeline that:
# Builds application
# Runs unit tests
# Runs integration tests
# Runs E2E tests in parallel
# Generates Allure report
# Sends to ReportPortal
# Deploys to staging on success
# Sends notifications (Slack + Email)

# Your implementation
```

Summary

- Continuous Integration principles
- Continuous Delivery workflows
- Pipeline architecture
- Quality gates

✅ GitHub Actions

- Workflow creation
- Matrix builds
- Secrets management
- Parallel execution
- BrowserStack integration

✅ CI Best Practices

- Environment configuration
- Dependency caching
- Parallel execution
- Quality gates
- Test sharding

✅ Allure Reporting

- Installation and configuration
- Test annotations
- Screenshots and attachments
- Report generation
- GitHub Pages deployment

✅ Mochawesome

- Configuration
- Report generation

✅ ReportPortal

- Docker deployment
- Agent configuration
- Real-time analytics
- AI-powered analysis
- Defect management

✅ Notifications

- Slack integration
- Email notifications
- Custom messages
- Failure alerts

Key Takeaways

- **CI/CD automates** test execution
- **GitHub Actions** provides powerful CI/CD
- **Matrix builds** enable parallel testing
- **Allure creates beautiful** test reports
- **ReportPortal provides** real-time analytics
- **Notifications keep** teams informed
- **Quality gates** maintain code quality

Best Practices

1. Run tests on every commit
2. Use matrix builds for coverage
3. Cache dependencies
4. Implement quality gates
5. Generate comprehensive reports

8. Use test sharding for speed
9. Store artifacts properly
10. Document CI/CD setup

CI/CD Checklist

- ☐ GitHub Actions workflow created
- ☐ Matrix builds configured
- ☐ Secrets stored securely
- ☐ Dependencies cached
- ☐ Tests run in parallel
- ☐ Quality gates implemented
- ☐ Allure reports generated
- ☐ ReportPortal integrated
- ☐ Slack notifications set up
- ☐ Email alerts configured

Next Steps

Only 2 more days!

- **Day 14**: AI in Test Automation
- **Day 15**: Framework Design & Capstone

End of Day 13 Documentation

← AWS for Test Automation

AI in Test Automation →