🚀 Jenkins Pipeline Documentation

Playwright Test Automation - PlaywrightWithJenkinsDemo

Project Version: 1.0.0
Last Updated: November 26, 2025
Owner: Leela-ditya
Repository: https://github.com/Leela-ditya/PlaywrightWithJenkinsDemo
CI/CD Tool: Jenkins
Test Framework: Playwright with Allure Reporting

📋 Table of Contents

📖 Overview

Purpose

This Jenkins pipeline automates Playwright end-to-end testing with comprehensive reporting capabilities. It provides:

- **Automated Test Execution** with multiple browser support (Chromium, Firefox, WebKit)
- **Intelligent Retry Mechanism** for handling flaky tests
- **Multi-Reporter Support** (HTML Report, Allure Framework)
- **Artifact Management** (Screenshots, Videos, Traces)
- **Email Notifications** with detailed test metrics
- **Report Organization** in a unified structure

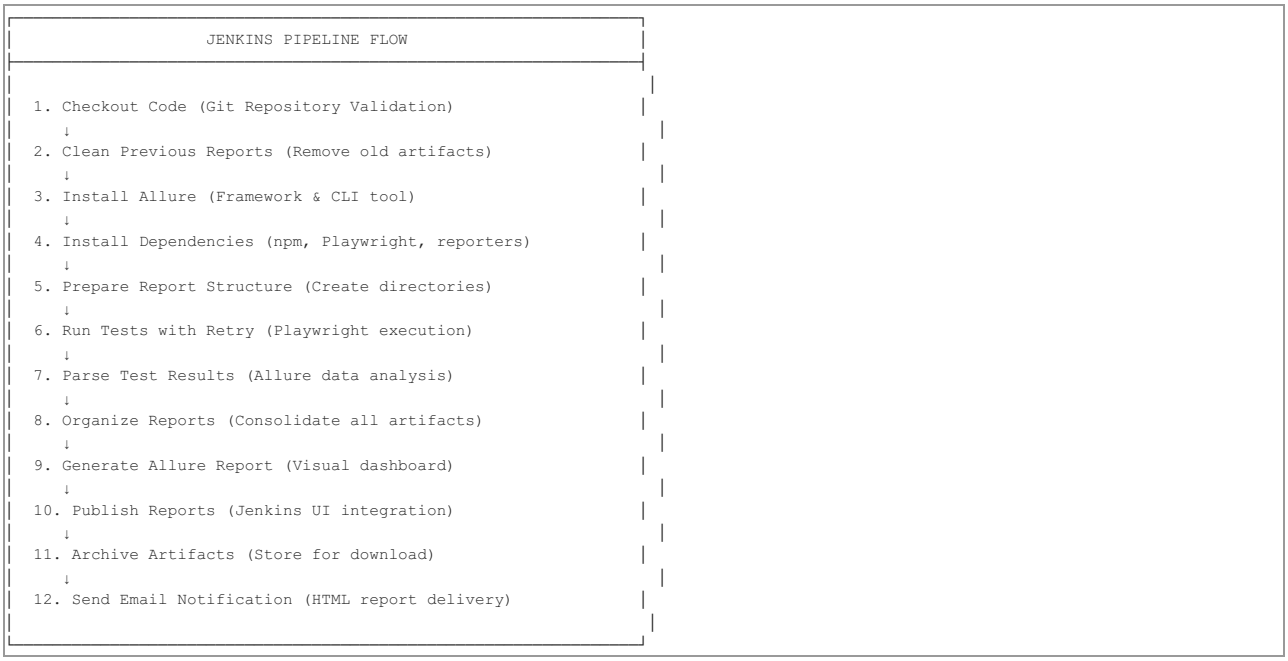
Key Features

☑ **Smart Git Repository Management** - Validates and updates repository automatically
☑ **Cross-Browser Testing** - Support for Chromium, Firefox, WebKit, and all browsers
☑ **Flexible Test Execution** - Run by tag, specific file, or all tests
☑ **Automatic Retry Mechanism** - Configurable retry attempts for failed tests
☑ **Comprehensive Reporting** - HTML and Allure reports with rich artifacts
☑ **Email Notifications** - Detailed HTML email with test metrics and report links
☑ **Artifact Archiving** - Full test results available for download

---

🏗️ Architecture & Design

Pipeline Flow Diagram



Technology Stack

Component	Version	Purpose
Playwright	1.56.1	Web automation framework
Allure	2.34.1	Advanced reporting framework
TypeScript	5.9.3	Test code language
Node.js	Latest	Runtime environment
Jenkins	2.x+	CI/CD orchestration
PowerShell	5.1 (Windows)	Command execution

🔧 Prerequisites & Installation

System Requirements

Windows Agent

```
OS: Windows 10/11 or Windows Server 2016+
RAM: 4GB minimum (8GB recommended)
Disk Space: 10GB minimum for dependencies and reports
```

Required Software

1. **Node.js & npm**

   ```
   # Download and install from https://nodejs.org/
   node --version   # v18+ recommended
   npm --version    # v9+
   ```

2. **Playwright**

   ```
   npm install -D @playwright/test
   npx playwright install --with-deps
   ```

3. **Allure Command Line**

   ```
   # Via Scoop (automated by pipeline)
   scoop install allure

   # Or manual download
   # https://github.com/allure-framework/allure2/releases
   ```

4. **TypeScript**

   ```
   npm install -D typescript ts-node
   ```

Environment Setup

```
# Clone Repository
git clone https://github.com/Leela-ditya/PlaywrightWithJenkinsDemo.git
cd PlaywrightWithJenkinsDemo

# Install Dependencies
npm install
npx playwright install --with-deps

# Verify Installation
npx playwright --version
allure --version
```

☸ Jenkins Configuration

Jenkins System Configuration

1. Global Tools Configuration

Navigate: **Manage Jenkins → Tools**

NodeJS Configuration

```
Name: NodeJS-18
Install Automatically: ✓ Checked
Version: 18.x
```

Git Configuration

```
Name: Default
Path: C:\Program Files\Git\cmd\git.exe (Windows)
or /usr/bin/git (Linux)
```

2. Global Configuration

Navigate: **Manage Jenkins → Configure System**

Jenkins URL

```
Jenkins URL: http://your-jenkins-url:8080/
```

Email Notification (Extended SMTP)

```
SMTP server: smtp.gmail.com
SMTP port: 587
TLS: ✓ Checked
Use SMTP Authentication: ✓ Checked
Username: your-email@gmail.com
Password: [App Password]
Default From Address: jenkins-notifications@yourdomain.com
Reply-To Address: jenkins-notifications@yourdomain.com
Charset: UTF-8
```

3. Jenkins Location

Navigate: **Manage Jenkins → Configure System → Jenkins Location**

```
Jenkins URL: http://localhost:8080/  (or your Jenkins server URL)
```

Pipeline Stages

Stage 1: Checkout Code

Purpose: Validate and synchronize Git repository with latest code

Process:

1. Check if `.git` directory exists
2. Validate repository origin remote
3. Fetch latest changes from GitHub
4. Compare local vs remote commits
5. Pull latest code if changes detected
6. Clone fresh repository if corrupted

Key Scripts:

```
// Validate repository
git fetch origin
git rev-parse HEAD                    # Local commit
git rev-parse origin/${GIT_BRANCH}    # Remote commit

// Update if different
git reset --hard origin/${GIT_BRANCH}
git pull origin ${GIT_BRANCH}
```

Failure Handling: Automatically clones fresh repository if validation fails

Stage 2: Clean Previous Reports

Purpose: Remove old test artifacts to avoid conflicts

Cleanup Items:

- `test-reports/` - Previous test reports
- `playwright-report/` - Previous HTML reports
- `test-results/` - Old test results
- `allure-results/` - Previous Allure data

Importance: Ensures clean state and prevents disk space issues

---

Stage 3: Install Allure

Purpose: Install and verify Allure Framework

Installation Flow:

```
Is Allure installed?
├─ YES → Skip (display version)
└─ NO → Check Scoop
        ├─ Is Scoop installed?
        │  ├─ YES → Install Allure
        │  └─ NO → Install Scoop first, then Allure
        └─ Verify installation
```

Installation Commands:

```
# Install Scoop (if needed)
powershell -Command "Set-ExecutionPolicy RemoteSigned -Scope CurrentUser -Force;
  [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072;
  iex ((New-Object System.Net.WebClient).DownloadString('https://get.scoop.sh'))"

# Install Allure
scoop install allure

# Verify
allure --version
```

Stage 4: Install Dependencies

Purpose: Install all npm packages and Playwright browsers

Dependencies Installed:

```
{
  "devDependencies": {
    "@playwright/test": "^1.56.1",
    "@types/node": "^24.3.1",
    "allure-playwright": "^3.4.2",
    "ts-node": "^10.9.2",
    "typescript": "^5.9.3"
  },
  "dependencies": {
    "allure": "^3.0.0-beta.18",
    "allure-commandline": "^2.34.1",
    "csv-parse": "^6.1.0",
    "exceljs": "^4.4.0",
    "playwright": "^1.56.1",
    "xlsx": "^0.18.5"
  }
}
```

Installation Steps:

```
# 1. Install npm packages
npm install

# 2. Install Playwright browsers with system dependencies
npx playwright install --with-deps

# 3. Install Allure reporter if missing
npm install --save-dev allure-playwright
```

Stage 5: Prepare Report Structure

Purpose: Create organized directory structure for test artifacts

Directory Structure Created:

```
test-reports/
├── screenshots/        (PNG files from failed tests)
├── videos/             (WebM video recordings)
├── traces/             (Playwright trace files)
├── allure-results/     (JSON test results)
├── allure-report/      (Generated HTML dashboard)
└── html-details/       (Playwright HTML report)
```

Stage 6: Run Tests with Retry

Purpose: Execute Playwright tests with configurable parameters

Test Execution Logic:

```groovy
// Build command based on parameters
def testCmd = "npx playwright test"

// 1. Add retry mechanism
testCmd += " --retries=${params.MAX_RETRIES}"

// 2. Add test selection (TAG or SPEC, not both)
if (params.TAG) {
    testCmd += " --grep \"${params.TAG}\""
} else if (params.SPEC) {
    testCmd += " ${params.SPEC}"
}

// 3. Add browser selection
if (params.BROWSER && params.BROWSER != 'all') {
    testCmd += " --project=${params.BROWSER}"
}

// 4. Add reporters
testCmd += " --reporter=html,allure-playwright"

// 5. Execute with exit code capture
def testResult = bat(script: testCmd, returnStatus: true)
```

Parameters:

- `TAG` - Run tests matching tag (e.g., @StaticTable)
- `SPEC` - Run specific test file
- `BROWSER` - Select browser (chromium, firefox, webkit, all)
- `MAX_RETRIES` - Retry failed tests (default: 2)

Example Executions:

```
# Run all tests with default tag
npx playwright test --retries=2 --grep "@StaticTable" --reporter=html,allure-playwright

# Run specific file on Chrome
npx playwright test tests/playwrightPractice.spec.ts --retries=2 --project=chromium --reporter=html,allure-playwright

# Run all tests on all browsers
npx playwright test --retries=2 --reporter=html,allure-playwright
```

Stage 7: Parse Test Results

Purpose: Analyze Allure results and calculate test statistics

Metrics Extracted:

- Total Tests
- Passed Tests
- Failed Tests (includes broken)
- Skipped Tests
- Pass Rate (%)

Data Source:

- Reads from `allure-results/*-result.json` files
- Calculates statistics from test status

Environment Variables Set:

```
env.TOTAL_TESTS = totalTests
env.PASSED_TESTS = passedTests
env.FAILED_TESTS = failedTests + brokenTests
env.SKIPPED_TESTS = skippedTests
env.PASS_RATE = (passedTests / totalTests) * 100
```

Example Output:

```
Test Results Summary:
  Total Tests: 14
  Passed: 12
  Failed: 2
  Skipped: 0
  Pass Rate: 85.71%
```

Stage 8: Organize Reports

Purpose: Consolidate all test artifacts into unified structure

Operations:

1. **Copy HTML Report**

```
copy /Y playwright-report\index.html test-reports\index.html
xcopy /E /I /Y playwright-report test-reports\html-details
```

2. **Copy Allure Results**

```
xcopy /E /I /Y allure-results test-reports\allure-results
```

3. **Move Screenshots, Videos, Traces**

```
for /r test-results %%f in (*.png) do copy "%%f" test-reports\screenshots\
for /r test-results %%f in (*.webm) do copy "%%f" test-reports\videos\
for /r test-results %%f in (*.zip) do copy "%%f" test-reports\traces\
```

Stage 9: Generate Allure Report

Purpose: Create interactive Allure dashboard from test results

Command:

```
allure generate test-reports\allure-results --clean -o test-reports\allure-report
```

Report Includes:

- Test execution timeline
- Test failure analysis
- Severity breakdown
- Category distribution
- Retry metrics
- Trend analysis

Stage 10: Publish Reports

Purpose: Make reports accessible in Jenkins UI

Jenkins Plugins Used:

1. **HTML Publisher** - Displays Playwright HTML report
2. **Allure** - Publishes Allure dashboard

Configuration:

```
// Publish HTML Report
publishHTML(target: [
    allowMissing: false,
    alwaysLinkToLastBuild: true,
    keepAll: true,
    reportDir: "test-reports",
    reportFiles: 'index.html',
    reportName: 'Playwright Test Report'
])

// Publish Allure Report
allure([
    includeProperties: false,
    jdk: '',
    results: [[path: "test-reports/allure-results"]]
])
```

Stage 11: Archive Artifacts

Purpose: Store all test reports for download and history

Artifacts Archived:

- All files in `test-reports/` directory
- Screenshots, videos, traces
- HTML and Allure reports
- Raw test result JSON files

Storage: Jenkins artifacts directory (available for download)

Stage 12: Email Notification (Post Action)

Purpose: Send detailed test report via email

Details Below in Email Configuration Section

Parameters & Environment Variables

Pipeline Parameters

1. TAG

```
Type: String
Default Value: @StaticTable
Description: Enter Playwright tag (example: @smoke, @regression)
Usage: Filters and runs only tests with specified tag
```

2. SPEC

```
Type: String
Default Value: (empty)
Description: Enter spec file to run (example: tests/login.spec.ts)
Usage: Runs specific test file instead of tag-based filtering
```

3. BROWSER

```
Type: Choice
Options: chromium, firefox, webkit, all
Default: chromium
Description: Choose Browser to run tests
Usage: Selects which browser(s) to execute tests on
```

4. EMAIL

```
Type: String
Default Value: leelasonu12@gmail.com
Description: Email to receive test reports
Usage: Recipient for test execution report notification
```

5. MAX_RETRIES

```
Type: String
Default Value: 2
Description: Number of times to retry failed tests
Usage: Configures automatic retry for flaky tests
```

Environment Variables

Pipeline-Level Variables

```
// Git Configuration
REPORTS_DIR = "test-reports"
GIT_REPO_URL = 'https://github.com/Leela-ditya/PlaywrightWithJenkinsDemo.git'
GIT_BRANCH = 'main'
```

Dynamically Set Variables

Variable	Set At Stage	Usage
TEST_EXIT_CODE	Run Tests	Exit code from test execution
TEST_STATUS	Run Tests	PASSED or FAILED
TEST_SUMMARY	Run Tests	Summary message
TOTAL_TESTS	Parse Results	Total test count
PASSED_TESTS	Parse Results	Number of passed tests
FAILED_TESTS	Parse Results	Number of failed tests
SKIPPED_TESTS	Parse Results	Number of skipped tests
PASS_RATE	Parse Results	Pass rate percentage

Plugins Required

Essential Jenkins Plugins

1. Pipeline Plugin (Declarative)

```
Plugin ID: pipeline-model-definition
Version: 2.2+
Purpose: Enables Declarative Pipeline syntax
Installation: Manage Jenkins → Manage Plugins → Available → Search "Pipeline"
```

2. Git Plugin

```
Plugin ID: git
Version: 5.0+
Purpose: Git repository integration
Installation: Manage Jenkins → Manage Plugins → Available → Search "Git"
```

3. HTML Publisher Plugin

```
Plugin ID: htmlpublisher
Version: 1.32+
Purpose: Publish HTML reports in Jenkins UI
Installation: Manage Jenkins → Manage Plugins → Available → Search "HTML Publisher"
```

4. Allure Plugin

```
Plugin ID: allure-jenkins-plugin
Version: 2.31+
Purpose: Integrate Allure reports
Installation: Manage Jenkins → Manage Plugins → Available → Search "Allure"

Configuration:
- Manage Jenkins → Configure System → Allure
- Set Allure Commandline path (usually auto-detected)
```

5. Email Extension Plugin

```
Plugin ID: email-ext
Version: 2.101+
Purpose: Advanced email notifications (HTML, attachments, etc.)
Installation: Manage Jenkins → Manage Plugins → Available → Search "Email Extension"

Configuration: See Email Configuration section below
```

6. Timestamper Plugin (Optional)

```
Plugin ID: timestamper
Version: 1.15+
Purpose: Add timestamps to console output
Installation: Manage Jenkins → Manage Plugins → Available → Search "Timestamper"
```

Installation Steps

```
// Via Jenkins UI:
1. Go to Manage Jenkins → Manage Plugins
2. Click "Available" tab
3. Search for plugin name
4. Check the checkbox
5. Click "Install without restart" or restart Jenkins
```

Complete Installation Command (CLI)

```
# Via Jenkins CLI (if available)
java -jar jenkins-cli.jar -s http://localhost:8080 \
  install-plugin pipeline-model-definition \
  install-plugin git \
  install-plugin htmlpublisher \
  install-plugin allure-jenkins-plugin \
  install-plugin email-ext \
  install-plugin timestamper

# Then restart Jenkins
java -jar jenkins-cli.jar -s http://localhost:8080 safe-restart
```

📧 Email Configuration

Gmail SMTP Configuration (Most Common)

Step 1: Generate Gmail App Password

1. Go to **Google Account Security**: https://myaccount.google.com/
2. Enable **2-Step Verification** (if not already enabled)
3. Navigate to **App Passwords**
4. Select **Mail** and **Windows Computer**
5. Copy the generated 16-character password

Step 2: Configure Jenkins Email

Navigate: Manage Jenkins → Configure System → Extended E-mail Notification

```
SMTP Server: smtp.gmail.com
SMTP Port: 587
Credentials:
  ├─ Username: your-email@gmail.com
  └─ Password: [16-char app password]

TLS: ✓ Enable SSL/TLS
SMTP Authentication: ✓ Required
Default Charset: UTF-8
Default From Address: jenkins-notifications@gmail.com
Reply-To Address: jenkins-notifications@gmail.com
Default Content Type: HTML
Enable Debug Mode: (Optional for troubleshooting)
```

Step 3: Test Email Configuration

```
// In Jenkins job post section, add test email:
emailext(
    subject: "Test Email from Jenkins",
    body: "If you received this, email configuration is working!",
    to: "your-email@gmail.com"
)
```

Alternative SMTP Configurations

Office 365

```
SMTP Server: smtp.office365.com
SMTP Port: 587
TLS: ✓ Enable
Credentials: Your Office 365 email and password
```

Custom Company SMTP

```
SMTP Server: mail.yourcompany.com
SMTP Port: 25 or 587 (check with IT)
TLS: Depends on company policy
Credentials: Domain\username and password
```

SendGrid

```
SMTP Server: smtp.sendgrid.net
SMTP Port: 587
TLS: ✓ Enable
Username: apikey
Password: [SendGrid API Key]
```

Email Template in Pipeline

Current Implementation (in Jenkinsfile):

```
emailext(
    subject: "Playwright Report - Build #${env.BUILD_NUMBER} (${testStatus})",
    body: emailBody,                    // HTML email body
    mimeType: 'text/html',
    to: "${params.EMAIL}"
)
```

Email Components:

1. **Header Section**
    - Build status (PASSED/FAILED)
    - Build number
    - Color-coded gradient (Green for pass, Red for fail)
2. **Test Summary Section**
    - Pass rate progress bar
    - Test statistics grid (Total, Passed, Failed, Skipped)
    - Build configuration table
3. **Test Artifacts Section**
    - Screenshot count
    - Video count
    - Trace file count
4. **Quick Access Links**
    - HTML Report link
    - Allure Report link
    - Download Artifacts link
    - Console Log link
5. **Report Structure Visualization**
    - ASCII directory tree
    - File counts

6. **Footer**
    - Automated message indicator
    - Copyright year

Email CSS Styling

```
/* Color Scheme */
Header (Pass): Linear gradient #4CAF50 → #45a049 (Green)
Header (Fail): Linear gradient #f44336 → #da190b (Red)
Stat Cards: #f8f9fa background with colored left border
Links: #2196F3 (Blue)

/* Typography */
Body Font: Segoe UI, Tahoma, Geneva, sans-serif
Max Width: 800px
Responsive Layout: 3-column grid for stat cards
```

Testing Email Configuration

```
// Simple test script
@Test
void testEmailSending() {
    // Enable debug logging
    // Manage Jenkins → Configure System → Extended E-mail Notification
    // → Enable Debug Mode

    // Send test email
    emailext(
        subject: "Jenkins Email Test",
        body: "Email configuration successful!",
        to: "your-email@example.com"
    )
}
```

Report Structure

Final Report Organization

```
test-reports/
│
├── index.html                          ← Main Playwright HTML Report
├── playwright-report/                  ← Detailed HTML test results
│   ├── index.html
│   ├── assets/
│   └── data/
│
├── html-details/                       ← Copy of Playwright report
│   ├── index.html
│   ├── assets/
│   └── data/
│
├── allure-results/                     ← Raw Allure test data
│   ├── *-result.json                   ← Individual test results
│   ├── *-container.json                ← Test containers
│   ├── categories.json                 ← Test categories
│   ├── environment.json                ← Environment info
│   └── ...
│
├── allure-report/                      ← Generated Allure Dashboard
│   ├── index.html                      ← Open this in browser
│   ├── css/
│   ├── js/
│   ├── plugins/
│   └── data/
│
├── screenshots/                        ← Screenshot files
│   ├── test-1-failed.png
│   ├── test-2-failed.png
│   └── ...
│
├── videos/                             ← Video recordings
│   ├── test-1.webm
│   ├── test-2.webm
│   └── ...
│
└── traces/                             ← Playwright trace files
    ├── test-1-trace.zip
    ├── test-2-trace.zip
    └── ...
```

Report Access Methods

1. Jenkins UI

```
Job Page → Build Details →
  ├── Playwright Test Report (HTML)
  ├── Allure Report (Dashboard)
  └── Artifacts (Download All)
```

2. Email Links

```
Received Email with Direct Links:
  ├── HTML Report URL
  ├── Allure Report URL
  ├── Download Artifacts URL
  └── Console Log URL
```

3. Local File System

```
Jenkins Workspace:
C:\Program Files\Jenkins\workspace\<JobName>\test-reports\
```

Report Details

Playwright HTML Report

- Test execution timeline
- Test details and errors
- Console output
- Screenshots for failed tests
- Retry attempts

Allure Report

- Executive summary
- Test execution graph
- Failures categorization
- Duration analysis
- Trend reports
- Timeline view

Execution Workflow

Manual Execution

From Jenkins UI

1. **Navigate to Job**

   ```
   Jenkins Dashboard → PlaywrightWithJenkinsDemo
   ```

2. **Click "Build with Parameters"**

3. **Set Parameters**

   ```
   TAG: @StaticTable (or your tag)
   SPEC: (leave empty for tag-based run)
   BROWSER: chromium (or firefox, webkit, all)
   EMAIL: your-email@example.com
   MAX_RETRIES: 2
   ```

4. **Click "Build"**

5. **Monitor Execution**

   ```
   Jenkins UI → Console Output → Real-time logs
   ```

Command-Line Execution (Jenkins CLI)

```
# Via Jenkins CLI
java -jar jenkins-cli.jar -s http://localhost:8080 \
  build "PlaywrightWithJenkinsDemo" \
  -p TAG="@StaticTable" \
  -p BROWSER="chromium" \
  -p EMAIL="your-email@example.com" \
  -p MAX_RETRIES="2" \
  -w

# Alternative using curl
curl -X POST http://localhost:8080/job/PlaywrightWithJenkinsDemo/buildWithParameters \
  -d TOKEN=your-token \
  -d TAG=@StaticTable \
  -d BROWSER=chromium \
  -d EMAIL=your-email@example.com
```

Automated Execution (Scheduled)

Setup Periodic Builds

1. **Job Configuration → Build Triggers**

2. **Check "Poll SCM"** or **"Build periodically"**
3. **Set Schedule**

```
Poll SCM:
# Run every 6 hours
H */6 * * *

# Run daily at 2 AM
0 2 * * *

# Run every 30 minutes
H/30 * * * *
```

Setup Webhook (GitHub)

1. **GitHub Repository Settings → Webhooks**
2. **Add Webhook**

```
Payload URL: http://jenkins-url:8080/github-webhook/
Content Type: application/json
Events: Push events
```

3. **Jenkins Configuration**
   - Job → Configure → Build Triggers
   - Check "GitHub hook trigger for GITScm polling"

Test Execution Scenarios

Scenario 1: Run All Tests on All Browsers

```
TAG: (empty)
SPEC: (empty)
BROWSER: all
MAX_RETRIES: 2
Expected: All test files on chromium, firefox, webkit
```

Scenario 2: Run Specific Tag on Chrome

```
TAG: @StaticTable
SPEC: (empty)
BROWSER: chromium
MAX_RETRIES: 2
Expected: Only tests with @StaticTable tag on Chrome
```

Scenario 3: Run Specific File

```
TAG: (empty)
SPEC: tests/playwrightPractice.spec.ts
BROWSER: firefox
MAX_RETRIES: 3
Expected: Only playwrightPractice tests on Firefox with 3 retries
```

🛠 Troubleshooting

Common Issues & Solutions

Issue 1: "Cannot find Allure"

Error Message:

```
⚠ Allure command not found
```

Solutions:

```
# Option 1: Manual Installation
# Install Scoop first
powershell -Command "iex ((New-Object System.Net.WebClient).DownloadString('https://get.scoop.sh'))"

# Install Allure
scoop install allure

# Option 2: Add to PATH
# Add Scoop\shims directory to Windows PATH
# Typically: C:\Users\<username>\scoop\shims
```

Issue 2: Git Repository Corrupted

Error Message:

```
⚠ Error validating repository
```

Automatic Recovery:

- Pipeline automatically detects corruption
- Removes `.git` directory

- Clones fresh repository
- No manual intervention needed

Manual Recovery:

```
# In Jenkins workspace
cd C:\Program Files\Jenkins\workspace\PlaywrightWithJenkinsDemo
rmdir /s /q .git
git clone https://github.com/Leela-ditya/PlaywrightWithJenkinsDemo.git .
```

Issue 3: Email Not Sending

Error Symptoms:

- Email not received after build
- No error in Jenkins logs

Troubleshooting Steps:

```
// Step 1: Enable Debug Logging
// Manage Jenkins → Configure System → Extended E-mail Notification
// → Check "Enable Debug Mode"

// Step 2: Test SMTP Connection
// Use Jenkins Script Console:
def socket = new java.net.Socket()
socket.connect(new java.net.InetSocketAddress("smtp.gmail.com", 587))
println("SMTP connection successful")

// Step 3: Check Credentials
// Verify Gmail App Password (not regular password)
// Verify 2-Factor Authentication enabled

// Step 4: Check Firewall
// Ensure port 587 is open outbound
```

Common Email Issues:

Issue	Cause	Solution
535 Authentication failed	Invalid password	Use 16-char app password, not account password
Connection timeout	Firewall blocked	Check outbound port 587 access
TLS error	SSL/TLS misconfigured	Enable SSL/TLS in Jenkins config
Email not received	Wrong recipient	Check email parameter in build

Issue 4: Tests Timing Out

Error Message:

```
Test execution timeout exceeded
```

Solutions:

```
// Increase Playwright timeout
// In playwright.config.ts:
use: {
    timeout: 30000,              // 30 seconds per action
    navigationTimeout: 30000,    // 30 seconds per navigation
},
timeout: 60000,                  // 60 seconds per test
```

Issue 5: Insufficient Disk Space

Error Message:

```
ENOSPC: no space left on device
```

Solutions:

```
# Clean pipeline automatically does this, but manually:
# 1. Remove old test reports
rmdir /s /q test-reports

# 2. Remove Playwright cache
rmdir /s /q %LOCALAPPDATA%\ms-playwright

# 3. Clean Jenkins workspace
# Jenkins UI → Manage Jenkins → Clean Up

# 4. Increase disk space (if persistent)
# Add storage to Jenkins agent machine
```

Issue 6: Allure Report Not Generating

Error Message:

```
Error generating Allure report
```

Debug Steps:

```
# Check if allure-results directory exists
dir test-reports\allure-results

# Check if result JSON files present
dir test-reports\allure-results\*-result.json

# Manually generate Allure report
allure generate test-reports\allure-results --clean -o test-reports\allure-report

# Check for errors
allure --version
```

Log Analysis

Where to Find Logs

```
Jenkins Logs:
├── Jenkins Root: C:\Program Files\Jenkins\
├── Build Logs: workspace\<JobName>\
└── System Logs: logs\

Critical Log Locations:
├── Jenkins Master: C:\Program Files\Jenkins\logs\jenkins.log
├── Agent Logs: C:\Program Files\Jenkins\agent.log
└── Build Console: Jenkins UI → Build → Console Output
```

How to Debug

1. **Enable Timestamp**

   ```
   Options {
       timestamps()
   }
   ```

2. **Check Console Output**

   ```
   Jenkins UI → Build #X → Console Output
   ```

3. **Enable Script Debug**

   ```
   // Add debug echo statements
   echo "🔍 Debugging: Variable value = ${variableValue}"
   ```

4. **Check Email Debug Log**

   ```
   Manage Jenkins → Configure System → Extended E-mail
   → Check "Enable Debug Mode"
   ```

☑ Best Practices

Code Quality

1. Test Naming Convention

```
// Good
test('should verify login with valid credentials', async ({ page }) => {
    // Test code
});

// Avoid
test('test1', async ({ page }) => {
    // Unclear test purpose
});
```

2. Use Meaningful Tags

```
test('Login functionality', {
    tag: ['@smoke', '@critical', '@regression']
}, async ({ page }) => {
    // Test code
});
```

3. Proper Error Handling

```
test('E-commerce purchase flow', async ({ page }) => {
    try {
        // Test steps
    } catch (error) {
        // Explicit error logging
        console.error(`Test failed: ${error.message}`);
        throw error;
    }
});
```

Jenkins Configuration Best Practices

1. Backup Configuration

```
# Backup Jenkins configs regularly
# Location: C:\Program Files\Jenkins\
# Backup folders:
# - jobs\<JobName>\
# - secrets\
# - config.xml
```

2. Use Parameterized Builds

```
// ✓ Good - Flexible and reusable
parameters {
    string(name: 'TAG', defaultValue: '@smoke')
    choice(name: 'BROWSER', choices: ['chrome', 'firefox'])
}

// ✗ Avoid - Hardcoded values
def tag = "@smoke"
def browser = "chrome"
```

3. Implement Build Discarding

```
options {
    buildDiscarder(
        logRotator(
            daysToKeepStr: '30',    // Keep 30 days
            numToKeepStr: '50',     // Keep last 50 builds
            artifactDaysToKeepStr: '7',
            artifactNumToKeepStr: '10'
        )
    )
}
```

4. Error Handling

```
// Always include try-catch for critical operations
try {
    bat 'npm install'
} catch (Exception e) {
    echo "❌ Installation failed: ${e.message}"
    // Graceful degradation
    unstable("Dependencies installation failed")
}
```

Security Best Practices

1. Secure Credentials

```
// ✓ Good - Use Jenkins Credentials Store
withCredentials([string(credentialsId: 'github-token', variable: 'GIT_TOKEN')]) {
    // Use ${GIT_TOKEN}
}

// ✗ Avoid - Hardcoding passwords
bat "git clone https://user:password@github.com/repo.git"
```

2. Sensitive Data in Emails

```
// ✓ Good - Don't include sensitive data
emailext(
    body: "Test results available at: ${buildUrl}",
    to: "${params.EMAIL}"
)

// ✗ Avoid - Including passwords or tokens
emailext(
    body: "API Key: ${API_KEY} Test results...",
    to: "${params.EMAIL}"
)
```

3. Workspace Cleanup

```
post {
    always {
        // Clean sensitive files
        deleteDir()  // Delete entire workspace

        // Or selective cleanup
        sh 'rm -rf .credentials'
    }
}
```

Performance Optimization

1. Parallel Execution

```
// Run tests on multiple browsers in parallel
parallel(
    'chromium': {
        // Run on chromium
    },
    'firefox': {
        // Run on firefox
    }
)
```

2. Smart Retry Logic

```
// Only retry flaky tests
def flakytests = ['@flaky', '@intermittent']
testCmd += " --grep \"${flakytests.join('|')}\""
testCmd += " --retries=3"

// Non-flaky tests
testCmd += " --retries=0"
```

3. Artifact Management

```
// Limit artifact storage
archiveArtifacts(
    artifacts: 'test-reports/**/*',
    fingerprint: true,
    allowEmptyArchive: true,
    excludes: '**/node_modules/**'
)
```

Maintenance

1. Regular Updates

```
# Update npm packages monthly
npm update

# Update Playwright
npm update @playwright/test

# Update Allure
scoop update allure
```

2. Log Rotation

```
Jenkins Configuration:
- Daily logs rotation
- 30-day retention
- Monthly archive
```

3. Plugin Management

```
Jenkins UI → Manage Plugins:
- Check for updates weekly
- Test updates on staging first
- Document plugin versions
```

Support & Resources

Documentation Links

- **Playwright Official:** https://playwright.dev/
- **Allure Framework:** https://docs.qameta.io/allure/
- **Jenkins Documentation:** https://www.jenkins.io/doc/
- **Groovy Syntax:** https://groovy-lang.org/documentation.html

Repository

- **GitHub:** https://github.com/Leela-ditya/PlaywrightWithJenkinsDemo
- **Issues:** Report via GitHub Issues

Contact

- **Project Owner:** Leela-ditya
- **Email Support:** leelasonu12@gmail.com

Version History

Version	Date	Changes
1.0.0	Nov 26, 2025	Initial implementation with Playwright, Allure, and Email integration

---

Last Updated: November 26, 2025
Status: ☑ Active & Maintained
License: ISC

Quick Reference Guide

Common Commands

```
# Test Execution
npx playwright test                              # Run all tests
npx playwright test --grep "@StaticTable"       # Run specific tag
npx playwright test tests/login.spec.ts         # Run specific file
npx playwright test --project=chromium          # Run on Chrome

# Report Generation
allure generate allure-results --clean -o report  # Generate Allure
npx playwright show-report                        # Show HTML report

# Installation
npm install                                       # Install dependencies
npx playwright install --with-deps               # Install browsers

# Cleanup
rmdir /s /q test-reports                         # Remove reports
rmdir /s /q allure-results                       # Remove Allure data
```

Job Execution Quick Start

```
1. Open Jenkins UI
2. Navigate to PlaywrightWithJenkinsDemo job
3. Click "Build with Parameters"
4. Set desired parameters
5. Click "Build"
6. Wait for completion
7. View reports in Jenkins UI or receive via email
```

END OF DOCUMENTATION