CI/CD Workflow Documentation

Overview

This project implements a complete Continuous Integration and Continuous Deployment (CI/CD) pipeline for the Qt Car Telltales application. The pipeline combines GitHub Actions (cloud-based testing) with Jenkins (local automation) to ensure code quality and automated testing.

Architecture

Developer Push

↓

GitHub Repository (main branch)

↓

GitHub Actions Workflow (build.yml)

├─ Checkout code

├─ Install MSVC compiler

├─ Install Qt 5.15.2

├─ Build with CMake

├─ Deploy Qt DLLs

├─ Run application (60 seconds)

└─ Report status

↓

GitHub Webhook (via ngrok)

↓

Jenkins Pipeline (Local Build)

├─ Checkout code

├─ Build with CMake

├─ Deploy Qt DLLs

├─ Run application (60 seconds)

├─ Auto-close application

└─ Archive artifacts

↓

Branch Protection Validation

├─ GitHub Actions must pass

├─ Jenkins must pass

├─ Pull request required

└─ Only then allow merge to main

GitHub Actions Workflow

Purpose

Automated testing in cloud environment on every push or pull request to main branch.

Location

.github/workflows/Cmake_Build_Win_Local_Run_App.yml

Workflow Stages

Stage: Checkout SCM

- Clones repository from GitHub
- Gets exact commit that was pushed

Stage: Setup MSVC

- Installs Microsoft Visual C++ compiler
- Configures for x64 (64-bit) architecture
- Uses Visual Studio 2022 Enterprise

Stage: Install Qt

- Downloads Qt 5.15.2 framework

- Installs MSVC-compatible version
- Takes approximately 2-3 minutes
- Caches installation for faster subsequent builds

Stage: Set Qt Environment Variable

- Captures Qt installation path
- Stores as QT_DIR environment variable
- Required for CMake configuration

Stage: Setup Repository

- Runs custom Setup_Repo.bat script if present
- Allows custom project initialization
- Handles branch-specific setup

Stage: Clean

- Removes previous build artifacts
- Deletes build folder
- Clears CMake cache
- Ensures fresh build from scratch

Stage: CMake Configure

- Reads CMakeLists.txt configuration
- Generates Visual Studio project files
- Sets CMAKE_PREFIX_PATH to Qt installation
- Prepares build environment

Stage: CMake Build

- Compiles C++ source code
- Uses Release configuration
- Creates QtTelltaleProject.exe executable
- Typical duration: 5-15 seconds

Stage: Deploy Qt DLLs

- Runs windeployqt tool
- Collects all required Qt libraries
- Copies dependencies to Release folder
- Ensures executable can run independently

Stage: Run Executable

- Starts QtTelltaleProject.exe
- Runs for exactly 60 seconds
- Validates application starts without crashes
- Tests basic functionality

Stage: Report Results

- GitHub Actions sends status to GitHub
- Updates pull request with result
- Shows green checkmark if all stages pass
- Shows red X if any stage fails

Triggers

- **Push:** When code is pushed to main branch
- **Pull Request:** When pull request is created for main branch

Environment Variables

BUILD_DIR = build

EXE_NAME = QtTelltaleProject.exe

Success Criteria

All stages must complete successfully:

- Code compiles without errors
- Qt DLLs deploy successfully
- Application starts and runs for 60 seconds

- No crashes during execution

Jenkins Local Integration

Purpose

Local automated building triggered by GitHub webhooks. Provides faster feedback and additional validation beyond cloud testing.

Setup Requirements

Prerequisites:

- Jenkins installed on local development machine
- Port 8080 available on localhost
- Git configured on Jenkins machine
- Qt 5.15.2 installed locally
- MSVC compiler available

How It Works

Webhook Flow:

1. Developer pushes code to GitHub
2. GitHub sends webhook event
3. ngrok tunnel forwards webhook to local machine
4. Jenkins receives webhook on port 8080
5. Jenkins pipeline starts automatically

Local Access

For Local Development Team:

1. **Starting Jenkins:**
    - Launch Jenkins service on your local machine
    - Runs on http://localhost:8080
    - Access only from your network

2. **Remote Access via ngrok:**
    - Jenkins administrator starts ngrok tunnel: ngrok http 8080
    - ngrok generates temporary public URL
    - URL format: https://[random-domain].ngrok-free.dev
    - This URL regenerates each time ngrok restarts
3. **Accessing Remote Jenkins:**
    - Jenkins administrator shares URL privately (email, Slack, Teams)
    - Never commit ngrok URL to GitHub
    - URL expires when ngrok tunnel is restarted
    - New URL must be communicated to team
4. **Login:**
    - Jenkins requires authentication
    - Credentials managed by Jenkins administrator
    - Username and password configured in Jenkins settings
    - Never share credentials via GitHub or public channels

Pipeline Stages

Checkout

- Clones repository from GitHub
- Checks out specific commit
- Prepares workspace

Setup

- Creates clean build directory
- Removes previous builds
- Initializes environment

CMake Configure

- Analyzes CMakeLists.txt

- Locates local Qt installation
- Generates Visual Studio project files

Build

- Compiles C++ code
- Creates executable
- Generates object files

Deploy Qt DLLs

- Runs windeployqt
- Collects required libraries
- Prepares for execution

Test Application

- Starts QtTelltaleProject.exe
- Monitors for 60 seconds
- Records any errors or output
- **Automatically closes application after timer expires**

Archive Artifacts

- Saves executable to Jenkins
- Stores build outputs
- Makes available for download

Post-Build Actions

- Reports success/failure to GitHub
- Updates pull request status
- Sends notifications

Automatic Application Closure

After application runs for exactly 60 seconds:

1. Timer completes

2. Jenkins stops the process
3. Application forcefully terminates
4. Resources are released
5. Build completes

This prevents hanging builds and ensures predictable execution.

Required Plugins

- **Git Plugin:** Clone from GitHub repositories
- **Pipeline Plugin:** Execute Jenkinsfile pipelines
- **GitHub Plugin:** Receive webhooks and report status
- **Generic Webhook Trigger:** Listen for GitHub events
- **Pipeline: Stage View:** Visualize pipeline stages
- **Credentials Binding:** Manage secure credentials

---

ngrok Webhook Bridge - Setup Guide

What is ngrok?

ngrok creates a secure public tunnel to your local Jenkins server, allowing GitHub (cloud) to send webhooks to Jenkins (local machine).

Installation

1. **Download ngrok:**
    - Visit https://ngrok.com
    - Download for Windows/Mac/Linux
    - Or use Chocolatey: choco install ngrok
2. **Create Account:**
    - Sign up for free ngrok account
    - Get authentication token from dashboard
3. **Configure Local Machine:**

4. ngrok config add-authtoken YOUR_AUTH_TOKEN

Starting the Tunnel

Command:

ngrok http 8080

Output:

Session Status online

Account your-email@example.com

Version 3.3.0

Region us-central

Web Interface http://127.0.0.1:4040

Forwarding https://[random-domain].ngrok-free.dev -> http://localhost:8080

The public URL: https://[random-domain].ngrok-free.dev

How It Works

GitHub (Cloud)

↓ sends webhook to

ngrok Public URL (https://[domain].ngrok-free.dev)

↓ forwards to

Your Local Machine (localhost:8080)

↓

Jenkins receives webhook

Important Notes About ngrok URLs

URL Regeneration:

- New URL generated each time ngrok starts
- Temporary by default (changes on restart)
- Use paid ngrok plan for reserved/static URLs

Sharing URLs:

- Share ngrok URL privately (email, Slack, Teams)
- Never commit to GitHub
- Update team when URL changes
- URL is temporary access only

Security:

- ngrok provides SSL/TLS encryption
- Jenkins login required for access
- Firewall still active on local machine
- Monitor webhook activity in GitHub

GitHub Webhook Configuration

Steps:

1. Go to GitHub Repository → Settings → Webhooks
2. Click "Add webhook"
3. **Payload URL:** https://[your-ngrok-url]/github-webhook/
4. **Content type:** application/json
5. **Events:** Push events only
6. **Active:** Check the box
7. Click "Add webhook"

Test Webhook:

- After setup, make a test push
- Go to Webhooks page → Recent Deliveries
- Verify delivery was successful

Team Member Access Guide

For Local Development

Team members should:

1. **Install Jenkins Locally:**
    - Download and install Jenkins on personal machine
    - Configure required plugins (see list above)
    - Create Jenkins user account
2. **Receive Jenkins Details Privately:**
    - Jenkins Administrator shares ngrok URL via email/Slack (not GitHub)
    - Administrator provides login credentials securely
    - URL updates communicated to team
3. **Access Jenkins:**
    - Open ngrok URL provided by administrator
    - Enter credentials
    - View build status and logs
4. **Never:**
    - Commit ngrok URLs to GitHub
    - Share credentials publicly
    - Post Jenkins links in public channels
    - Hardcode URLs in documentation

For Remote Access Requests

If new team member needs access:

1. Request from Jenkins Administrator
2. Administrator sends ngrok URL + credentials privately
3. Team member logs in via provided URL
4. Access is temporary and URL-specific

Environment Setup for Each Developer

Step 1: Install Required Tools

Install:

- Git
- CMake (version 3.16+)
- Visual Studio 2022 (with C++ workload)
- Qt 5.15.2 (MSVC build)
- Jenkins (optional for local testing)

Step 2: Clone Repository

git clone https://github.com/Nshravankumar4/QT_Car_Telltales_Cmake_Build.git

cd QT_Car_Telltales_Cmake_Build

Step 3: Build Locally

mkdir build

cd build

cmake -G "Visual Studio 17 2022" -A x64 ..

cmake --build . --config Release

Step 4: Optional - Setup Jenkins Locally

1. Install Jenkins
2. Install recommended plugins
3. Create pipeline job pointing to repository
4. Use Jenkinsfile in repository root

Step 5: Setup ngrok (If Running Local Jenkins)

1. Install ngrok
2. Add authentication token
3. Start tunnel: ngrok http 8080
4. Share URL with team (privately)

Repository Structure

```
QT_Car_Telltales_Cmake_Build/
├── .github/
│   └── workflows/
│       ├── Cmake_Build_Win_Local_Run_App.yml    (GitHub Actions)
│       └── cmake-multi-platform.yml             (Optional check)
├── src/
│   ├── main.cpp
│   └── mainwindow.cpp
├── inc/
│   └── mainwindow.h
├── ui/
│   └── mainwindow.ui
├── Scripts/
│   └── Setup_Repo.bat
├── CMakeLists.txt
├── Jenkinsfile
├── README.md
├── SECURITY.md
└── docs/
    └── CI_CD_WORKFLOW.md
```

Development Workflow

Creating a Feature

1. Create feature branch: git checkout -b feature/name
2. Make code changes

3. Commit: git commit -am "Description"
4. Push: git push origin feature/name
5. Create Pull Request on GitHub

Automated Testing

After push:

- GitHub Actions automatically builds and tests
- Jenkins automatically builds and tests locally (if running)
- Both systems run in parallel
- Results appear in pull request checks within 5 minutes

Merging Code

When all checks pass:

- Green checkmark appears on PR
- "Merge" button becomes available
- Click merge to add to main branch
- Code is now in production

If Build Fails

1. Check error in GitHub Actions or Jenkins logs
2. Common causes:
    - Compilation errors in C++ code
    - Missing include files
    - Qt library issues
    - CMake configuration problems
3. Fix code locally
4. Push changes
5. Automatic rebuild triggers
6. Repeat until passing

Application Execution Details

Pre-Launch

Build system prepares:

- Compiles all source code
- Generates executable
- Collects all required DLL files
- Validates dependencies

Launch

When testing stage begins:

- QtTelltaleProject.exe starts
- Application window opens
- Telltale indicators display
- Application runs normally

Monitoring Phase

Duration: 60 seconds

- Application executes
- No user interaction needed
- System monitors for crashes
- Logs any error messages

Auto-Termination

After exactly 60 seconds:

- Timer expires
- Application process terminated
- No manual intervention
- Resources released

- Build completes

Test Validation

Result marked as:

- **PASS:** If no crashes during 60 seconds
- **FAIL:** If application crashes before timeout

Build Artifacts

GitHub Actions Output

Generated per workflow run:

- Build logs (viewable in Actions tab)
- Build status (shown in commit)
- Execution timeline

Jenkins Output

Stored on local Jenkins machine:

- Executable file (QtTelltaleProject.exe)
- All required DLL files
- Build logs (downloadable)
- Artifact archive

Download Location

Jenkins (local): http://localhost:8080 → Select build → Download artifacts Jenkins (remote): https://[ngrok-url] → Select build → Download artifacts

---

Branch Protection Rules

Configuration

Location: Repository Settings → Rules → Rulesets

Applied To: main branch

Rules Enforced

Require Status Checks to Pass

- GitHub Actions workflow must pass
- Both must complete successfully before merge allowed

Require Pull Request

- Cannot push directly to main
- All changes must go through pull request
- Enables code review

Require Linear History

- Prevents merge commits
- Keeps history clean and readable

Block Force Pushes

- Cannot overwrite history
- Prevents accidental data loss

Require Signed Commits

- Commits must be cryptographically signed
- Proves code came from authorized user

Merge Requirements

All conditions must be met:

- Pull request created
- GitHub Actions passes
- Linear history maintained
- Signed commits used
- No force push attempts

Build Timeline

Minute-by-Minute Execution

Minute 0:00

- Developer pushes code to GitHub

Minute 0:05

- GitHub Actions workflow triggered
- Virtual machine starts spinning up

Minute 0:30

- GitHub sends webhook to ngrok
- ngrok forwards to Jenkins
- Jenkins receives notification

Minute 1:00

- GitHub Actions: Git checkout complete
- Jenkins: Git checkout complete

Minute 1:30

- GitHub Actions: MSVC installed
- Jenkins: Project configuration started

Minute 2:30

- GitHub Actions: Qt 5.15.2 installed

Minute 3:00

- GitHub Actions: CMake configuration complete
- Build starting
- Jenkins: Build complete

Minute 3:30

- GitHub Actions: Build complete
- DLL deployment starting
- Jenkins: DLL deployment complete

Minute 4:00

- GitHub Actions: Application running
- Jenkins: Application running

Minute 5:00

- GitHub Actions: Application runs 60 seconds
- Jenkins: Application runs 60 seconds
- **Application auto-closes on both**

Minute 5:15

- GitHub Actions: Reports results to GitHub
- Jenkins: Reports results to GitHub
- PR status updated

Minute 5:30

- Branch protection evaluation
- All checks passed
- Merge button available

Common Issues and Solutions

Issue: "Qt not found"

Cause: Qt installation path not detected **Solution:** Verify Qt environment variable is set correctly, check Qt installation location

Issue: "Build failed - compilation error"

Cause: Error in C++ source code **Solution:** Check GitHub Actions logs, fix code, push again

Issue: "Application doesn't run"

Cause: Missing DLL files or broken dependencies **Solution:** Verify windeployqt completed successfully, check all DLLs copied

Issue: "ngrok tunnel disconnected"

Cause: ngrok session ended or network interrupted **Solution:** Restart ngrok with: ngrok http 8080, share new URL with team

Issue: "Webhook not triggering Jenkins"

Cause: Webhook URL misconfigured, Jenkins not running, ngrok tunnel down **Solution:** Verify Jenkins is running, check webhook URL in GitHub matches current ngrok URL, verify GitHub webhook shows successful delivery

Issue: "Cannot access Jenkins remotely"

Cause: ngrok tunnel not running, firewall blocking, incorrect URL **Solution:** Verify ngrok is running, check URL shared with team is current, verify port 8080 accessible

Issue: "Jenkins login failed"

Cause: Incorrect credentials, user account not created **Solution:** Contact Jenkins administrator, verify username and password, ensure account is active

---

Security Best Practices

Protecting Credentials

- Never commit ngrok URLs to GitHub
- Never share Jenkins passwords in messages or emails
- Use private communication channels (Slack, Teams, email)
- Change credentials periodically
- Update team when URLs regenerate

Webhook Security

- Only GitHub sends to webhook URL
- Jenkins login required for access
- SSL/TLS encryption active
- Monitor Recent Deliveries in GitHub
- Disable webhook when not in use

Code Protection

- Repository should be set to private
- Build logs visible only to authorized users
- Artifacts stored locally or securely
- Branch protection enforces code review
- Signed commits verify developer identity

ngrok Security

- Use free tier for development
- Use paid plan for production with reserved URLs
- Regenerate URLs periodically
- Monitor active connections
- Only share URLs with trusted team members

Monitoring and Logging

GitHub Actions Logs

Location: Repository → Actions → Select workflow → Logs **Information:**

- Each stage with timestamps
- Command output
- Error messages
- Build duration

Jenkins Logs

Location: Jenkins dashboard → Select build → Console Output **Information:**

- Pipeline stage execution
- Git operations
- Build commands
- Application output
- Test results

Webhook Activity

GitHub Location: Settings → Webhooks → Select webhook → Recent Deliveries **Shows:**

- Webhook trigger time
- Payload data
- Response status
- Delivery status

Performance Metrics

Build Times

First build (includes Qt download):

- GitHub Actions: 5-7 minutes
- Jenkins: 3-5 minutes

Subsequent builds (Qt cached):

- GitHub Actions: 2-3 minutes
- Jenkins: 1-2 minutes

Resource Usage

GitHub Actions:

- 3 hours per workflow run available
- Automatic cleanup after completion

Jenkins (local):

- Minimal CPU during waiting
- ~500MB disk per build artifacts
- Automatic cleanup configurable

Summary

This CI/CD pipeline provides:

- Automated cloud-based testing (GitHub Actions)
- Automated local testing (Jenkins)
- Secure webhook integration (ngrok)
- Quality gates (branch protection)
- Complete test coverage (build + deploy + run)
- Automatic application closure after testing
- Status reporting to GitHub
- Build artifact archiving

Security First:

- Never expose credentials in public repositories
- Share ngrok URLs privately only
- Rotate URLs and credentials regularly
- Monitor all access and activity
- Use branch protection to enforce code quality

Result: Main branch always contains tested, working code with secure, automated deployment pipeline.