Setup Continuous Integration Pipeline for an RNW App

Edit

This guide will help you get started on setting up your very first continuous integration pipeline for a React Native for Windows app.

Setting Up a Continuous Integration Pipeline using GitHub Actions

When done developing your app, it's good practice to setup a CI pipeline with automated builds and tests to avoid any future regressions. There are many services available for setting up a CI pipeline. We'll use GitHub Actions as an example here since it doesn't require any extra account setup if you are already hosting your code on GitHub, also the default VM image has all the tools we needed pre-installed.

The VM images supported by GitHub Actions CI/CD can be found here, check the pre-installed tools and compare them with React Native Windows development dependencies, find the image that meets the requirements.

Next you need to create a YAML file for GitHub Actions, the basic steps are:

- Checkout code and setup the environment

```
      uses: actions/setup-node@v1
      with:
        node-version: '14'

    - name: Setup MSBuild
      uses: microsoft/setup-msbuild@v1.0.2
      with:
        vs-version: 16.8

    - name: Install node modules
      run: yarn --frozen-lockfile

    - name: yarn build
      run:  yarn build
```

- Build and run the project

```
    - name: Run Windows x64 release
      run: npx react-native run-windows --arch x64 --release --logging
```

Copy

Check out the full `react-native-webview` example as well as their official example for more info.

Save the YAML file to `.github\workflows\` and then commit. To learn more about YAML syntax, see Workflow syntax for GitHub Actions.

> GitHub Actions should be enabled by default, if it's not enabled for some reason you can go to Settings->Actions tab of the repo to enable it (requires owner access).

Now push your changes and the CI pipeline should be up and running.

Certificates are used to sign RNW apps so that they can be installed locally or published to the Microsoft Store. This data should not be publicly published, so we need to do extra steps if we wish to build/run signed RNW app packages through GitHub Actions.

Storing Certificates Securely

There are a several options where you can securely store your certificate information:

- GitHub Secrets
- Azure Secure Files
- Azure Key Vault Secrets

GitHub Secrets works well, when you are using GitHub Actions to run your pipeline. The latter two work well, if you are using Azure DevOps to run your pipeline.

For Azure Secure Files, you will upload the `.pfx` itself. The remaining two methods expect data in the form of a string. Thus, you must `Base64` encode your `.pfx` and upload the resulting string as your secret by running

```
$fileContentBytes = get-content '<Path-to-Pfx>' -Encoding Byte
[System.Convert]::ToBase64String($fileContentBytes) | Out-File pfx-encoded-bytes.txt
```

in PowerShell. Then upload the contents of `pfx-encoded-bytes.txt` as your secret.

Accessing Certificate Data from Pipeline

For Azure Secure Files, retrieving the secret will download the `.pfx` itself. The remaining two methods will give you the encoded string. Thus, you'll have to decode the secret, and save it to a `.pfx` file before it can be used to sign. In a YAML file the basic steps to generate your certificate from an encoded string stored in GitHub Actions are:

```
    [System.IO.File]::WriteAllBytes("$PfxPath", $PfxBytes)
```

where your encoded string is a GitHub secret named `Base64_Encoded_Pfx` .

In a YAML file the basic steps to use your certificate for a signed RNW app build are:

Copy

```
steps:
- name: run-windows (Release) - CI
  run: yarn windows --no-launch --arch x64 --logging --release --msbuildprops PackageCerti
```

Make sure to delete your `.pfx` from the pipeline once you've finished using it. In a YAML file the basic steps for removing your certificate are:

Copy

```
- name: Remove the pfx
  run: |
    $certificatePath = Join-Path -Path ProjectDirectoryPath -ChildPath GitHubActionsWorkfl
    Write-Host $certificatePath
    Remove-Item -path $certificatePath
```

See the `Xaml-Islands-Samples` repository for an example of a pipeline which uses GitHub Secrets. See the `react-native-gallery` repository for an example of a pipeline which uses Azure DevOps Secure Files. See the react-native-windows repository for an example of a pipeline which uses Azure Key Vault: Setup of Certificate Removal of Certificate Signed RNW App Build

FAQ's

`PackageCertificateKeyFile` MSBuild property. This property expects the file path to the `.pfx` you want to use. See here for an example of this using `VSBuild`. See here for an example of this using the RNW CLI.

I have a pipeline that runs on forks of my repository (i.e. when a PR is being made). Can I access my certificate data from this pipeline?

No. When data is securely stored through GitHub Secrets, Azure DevOps Secure Files, or Azure Key Vault, it can only be run from pipelines that are executing on branches of the original repository. The data cannot be accessed from pipelines running code from repository forks, because if the data was able to be accessed, someone could manipulate the pipeline source code within their fork to retrieve the data, leaving it unsecured.

I want to build a deployable package for my app from a pipeline that doesn't have access to my certificate data. What can I do?

This case may apply to you if you want to do some End To End testing within a PR pipeline to make sure incoming changes don't break your project. You do have a couple of options here to make do without a certificate. You can successfully build and deploy a non-signed RNW app via deploying from layout. The RNW CLI will by default deploys from layout as long as the MSBuild argument `AppxPackageSigningEnabled` or `PackageCertificateFile` is not set in the project file. If this is not the case for your app, you can force deploy from layout by using the `--deploy-from-layout` CLI option.

‹ Upgrading App to Latest Version of React Native Windows

Publishing a React Native Windows App to the Microsoft Store ›