CircleCI Documentation

Last edited on 8/8/22 by Sasha Ronaghi

Overview

This documentation is for the continuous integration of Enspectra Health's codebase using CircleCI. The script (config.yml) is in the .circleci folder of the circleci-project-setup branch in the sasha-pathwand-source Github repository. This is the branch which connects the Github repository and the circleci-project-setup branch. This branch is used so that updates to the config.yml file can be made without an impact on the other branch in this repository, the main branch.

The purpose of this script is to automate the build process of a Visual Studio .sln file and publish a release on Github of the artifact files of the build. The build output is being put into a Github release to be able to keep track of changes made to a branch by different contributors.

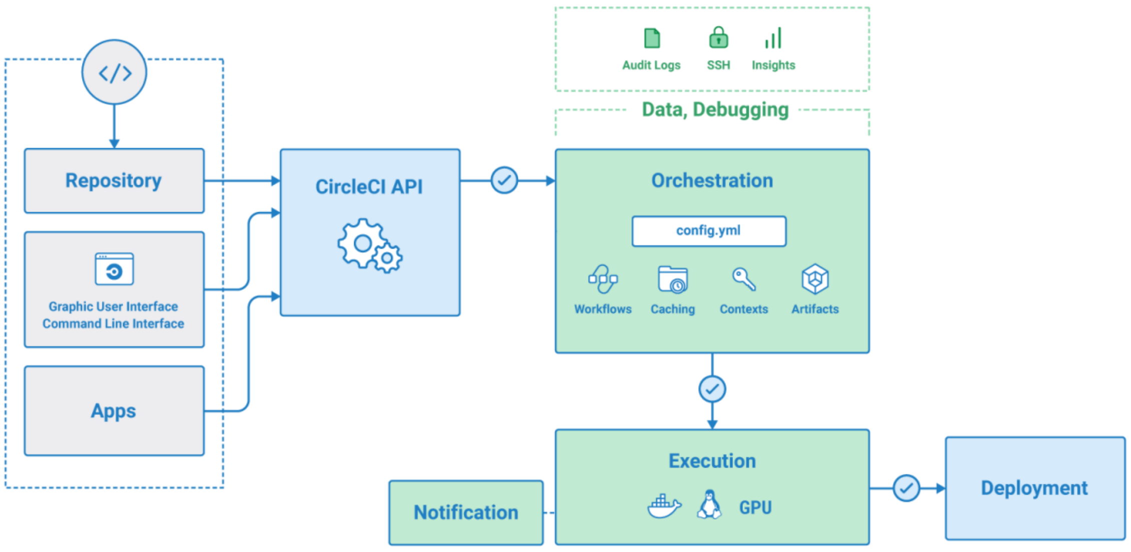

Figure 1. CircleCI Services Architecture

This diagram from CircleCI's operation guide provides a useful way of understanding how CircleCI integrates with Enspectra Health's code base. CircleCI is given access to a specific repository and allows users to run scripts with a config.yaml configuration file that is in the circleci-project branch of that repository. This configuration file utilizes files in the repository and executes upon them. The configuration used here is config.yml in Appendix A.

The configuration file is separated by version, orbs, workflows, and jobs. This script uses version 2.1 of CircleCI, the most up-to-date version as of 8/17/22. An orb is a reusable package of YAML configuration that can be used to integrate with third-party tools. Here, version 4.1 of the windows orb (the most up-to-date version as of 8/17/22) is used to support Visual Studio Community 2022 files and commands. The workflow defines the jobs and run order of the script.

The workflow here is to build the.sln file and publish a github release. This workflow is separated into jobs called “build” and “publish-github-release.” The workflow specifies that the “publish-github-release” job requires the completion of the “build” job because the publishing script needs access to the directory where the artifacts files are built. This directory and the files it contains are only available after a build. Each job is separated into steps with specific scripts that are run in the environments of the job. To write a script in a specific shell, one must first name the script, then write the commands, then specify the shell. Here, the commands use a pipe to allow for each line to be broken up into separate commands on the same shell. It is important to note that the build and publish jobs have different environments.

Job 1 - Build

- **Syntax Information:** For each of the run steps, the name of the run is used for identification in the CircleCI application after execution, without this line the step in the CircleCI application would be the commands in the step. Many of the commands use a pipe, which allows each line to be broken up into separate commands on the same shell. Note that the default shell is powershell.exe. Powershell.exe is used instead of the cmd.exe because the devenv.com command is not recognized in the cmd.exe shell.
- **General (line 14):** The first job is the build job and it utilizes the windows orb. The executor, the environment in which the job is run, uses the server-2022 version of the windows orb which is the most up-to-date as of 8/17//2022 and supports VS Community 2022. The edge version allows for accessing the most recent updates of the Windows image before the stable releases. Without using the edge version, the later command of devenv.com builds the solution file, but hangs so the build job fails because it exceeds 10 minutes without an output.
- **Step 1 - Checkout (line 19):** In checking out the code, the CircleCI environment downloads a copy of the code from the Github repository and uploads it as the working directory. This step is especially important for future debugging with SSH because it adds the required authenticity keys for interacting with Github over SSH. This step also configures Git to skip automatic garbage collection, which is typically used to clean up loose objects created after execution.
- **Step 2 - Run Install Installer Projects extension (line 20):** In this step, InstallerProjects.vsix is being installed using a VSIX Installer. InstallerProjects.vsix is a file for the Microsoft Visual Studio Installer Projects extension which is necessary for building .vdproj files. In this command, an ampersand “&” is the call operator which allows for a command to be stored in a string and run as a command. This is commonly used to execute scripts from their filenames, as used here. The file path to the VSIXInstaller.exe command is in quotation marks next to the ampersand because it has spaces. The “/q” command line option is for quiet, which allows for the .vsix file to be installed without any popups which the command line cannot control. The “/a” command line option is for admin, which allows the script to run the command as an administrator. The “--wait” command line option is for ensuring the extension is installed before the next commands are run. This command line operation works on a local computer, but does not work on the CircleCI environment. As of 8/3/22, this step is not being done properly

as the .vdproj files are not being built in later steps. There is an ongoing discussion with CircleCI support engineers on how to fix this bug. The following commands are commands suggested by the CircleCI support engineers in request #113571.

- **Step 3 - Run Pause for 2 minutes (line 28):** Empirical evidence shows that the VSIXInstaller.exe command does not wait until the installation is finished before running a new build. In this step, the Start-Sleep command is used to pause the script before the next step.
- **Step 4 - Run Solution Build (line 32):**This command is used to build the solution project. In this command, an ampersand “&” is the call operator which allows for a command to be stored in a string and run as a command. This is commonly used to execute scripts from their filenames, as used here. The file path to the devenv.com command is in quotation marks next to the ampersand because it has spaces. devenv.com is the version of the DEVENV tool that shows the output of the command on the command line prompt. The other devenv command, devenv.exe, also builds the .sln file, but does not show the output (i.e. which files are built). The build switch builds the specified command and the Debug solution configuration builds the Debug folder. To build the release folder, simply replace Debug with Release and change the file paths in future commands. Devenv is used instead of MSBuild because MSBuild does not support .vdproj files.
- **Step 5 - Storing artifacts (line 39):** Artifacts are the files that are produced from the build and stored in the Artifacts tab of the build job. For syntax, the path is specified with a “~” character in front which is shorthand for the environment’s working directory. The implementation follows the documentation for artifacts on Circle CI.
- **Step 6 - Persisting to Workspace (line 41):** This step makes the artifact files that are produced in the build job accessible to the next job where a Github release is published. This step only transfers the artifact files to job 2. The way to specify which files are accessible to the next job is by setting the root as the relative path from the working directory and setting the path as the relative path from the root. Here, root is “.”, to show that it is the working directory and path as the path from the working directory to the artifacts. An asterisks, “*” is used after a directory so that all of the files in the directory are also transferred. The implementation follows the documentation for workspaces on CircleCI.

Job 2 - Publishing Github Release

- **General (line 45):** This job publishes the build to the Github repository. The title of the release is the build number and the build release includes all of the files produced from the build and the source code of the solution file. For this portion of the script, this guide is used. For this job, the cibuilds/github:0.13 docker image enables the use of the ghr command, more detail later.
- **Step 1 - Attaching to Workspace (line 49)**: Here, the files that are made accessible in the persist_to_workspace step above are downloaded onto the file system of publish-github-release job. The workspace is attached at “./” which is syntax for the

directory of the artifacts and the files it contains. This guide is helpful in understanding how to persist and attach workspaces.

- **Step 2 - Run Publish Release on Github (line 52):** In this step, the ghr command is run, which creates a Github release and uploads the artifacts in parallel. The -t parameter for Github API Token can be implemented by creating a private Github token and uploading it to CircleCI as a private environment variable for this project. To create a Github token, use the steps here. Note the expiration date, which is set as 30 days by default. This token was given permission to everything in the repository. A private environment variable in CircleCI called GITHUB_TOKEN was created using this guide. The -u, -r, -c parameters for Github username, repository name, and commit SHA respectively use environment variables already set up by CircleCI and the user does not need to change. The -delete option deletes the release and its git tag in advance if it already exists. The mandatory part of the command is the TAG, which is currently set as the build number. The last part is the PATH which specifies the file or directories uploaded in the release. Here, the PATH is to the artifacts that are built.

Debugging with SSH

One important tool useful for debugging is rerunning a job with SSH, which can be done following this guide. To do this, an SSH key for the Github account was created, shown here. Then, go onto the dashboard and click on one of the jobs in the build-soln workflow, either build or publish-github-release. On the top right corner, there is a Rerun dropdown menu with a Rerun Job with SSH option. Once this option is clicked, click on the step for Enabling SSH and copy the entire command at the top of the section into the command shell. Note that if using an orb which provides access to multiple types of shells, one must specify the shell in the command. From here, one has access to the CircleCI environment with all of the environment variables of the workflow. This is useful for trying different commands or examining the directories in the environment. Note that it may be difficult to test parts of step commands such as checkout or store artifacts as it only provides access to the environment.

Discovered

1. Powershell.exe vs cmd.exe: When writing a script, using each of these shells were considered. Powershell.exe is used instead of the cmd.exe because the devenv.com command is not recognized in the cmd.exe shell. For syntax, powershell.exe requires that the change directory command, cd, is followed by a path in quotes if it has spaces. It also requires that "./" is used to run executables such as a batch file.
2. Using the edge version of the windows orb. Without the edge version of the windows orb, the devenv.com builds the solution file, but hangs so the build job fails because it exceeds 10 minutes without an output. CircleCI has since updated the windows orb to not hang on the devenv.com command, but it has not been released as a stable version. For this reason, the edge version was used.

3. Running vcvarsall.bat: Initially, it was assumed that running vcvarsall.bat was necessary in order to use the Microsoft C++ toolset from the command line. This batch file allows for setting the build environment in the command prompt window and the build architecture which is used in the solution file, x64. This was believed necessary for the environment to have access to the DEVENV command-line project management tool. However, subsequent experimentation showed running the vcvarsall batch file was not necessary for running devenv.com. However, future users of this script may want to run the vcvarsall.bat file or another batch file, so the commands used are commented out so future users could have an example of how to run the file.

```
      - run:
          name: VcVarsall.bat
          command: |
            #cd "C:\Program Files\Microsoft Visual
Studio\2022\Community\VC\Auxiliary\Build"
            #./vcvarsall.bat x64
          shell: powershell.exe
```

The first command is to change directories into a directory that has the vcvarsall.bat batch file. The second command is to run vcvarsall.bat. It is necessary to specify the x64 architecture because the default architecture for this command is x86.

4. Building a .vdproj file: The first approach to building this file was by using denvenv command line options. This meant using a command such as:

```
& "C:\Program Files\Microsoft Visual
Studio\2022\Community\Common7\IDE\devenv.com" "C:\Users\Enspectra
Lab\source\repos\sasha-pathwand-source\Project\PathWandGUI\PathWandGUI.sln"
/build Debug /project "C:\Users\Enspectra
Lab\source\repos\sasha-pathwand-source\Project\PathWandGUI\GUISetup\GuiSetu
p.vdproj" /projectconfig Debug
```

However, this approach did not work because it was realized that the Microsoft Visual Studio Installer Projects extension is needed for building .vdproj files.This is because the devenv command that is currently used works on a local computer to build the .vdproj file when the extension is installed. This extension, however, is not present in the Visual Studio Community of the windows orb. From communication with CircleCI, they said they will not install this extension which enables the building of .vdproj files.So, steps were taken to install it via the command line. The most common way to install an extension via the command line in Visual Studio is through the code –install-extensions command. However, this command does not work for Visual Studio Community, and is only available for Visual Studio Code. Currently, the best way known to install the extension from the command line is by using the VSIX Installer. However, this has not been successful, likely due to a bug with the Visual Studio command itself. This is likely because when the extension is removed on the local computer and then reinstalled from the GUI in Visual Studio Community, the .vdproj file is built using and then reinstalled using VSIXInstaller.exe from the command line, it does not install and enable the extension. It also does not wait on the command line until it has finished installing. A post about this was made to Microsoft Visual Studio Developer Community, with no response. The current solution has been proposed by the CircleCI engineers who believe the current commands in lines 24-28 would fix this problem. They also suggested pausing the program for two minutes and waiting for the installer to finish because the wait command line option is not working. This is still unresolved and in discussion with CircleCI and Microsoft as of 8/8/22.

Appendix A: Config.yml file

```
version: 2.1

orbs:
  win: circleci/windows@4.1

workflows:
  build-soln:
    jobs:
      - build
      - publish-github-release:
          requires:
            - build
jobs:
  build:
    executor:
      name: win/server-2022
      version: edge
    environment:
      SOLN_CONFIG: Debug
    steps:
      - checkout
      - run:
          name: Installer Projects extension install
          command: |
            & "C:\Program Files\Microsoft Visual
Studio\2022\Community\Common7\IDE\VSIXInstaller.exe" /q /a --wait
"$env:CIRCLE_WORKING_DIRECTORY\Tools\InstallerProjects.vsix"
            $pathtoVsixFile =
"$env:CIRCLE_WORKING_DIRECTORY\Tools\InstallerProjects.vsix"
            $args = '/quiet "{0}InstallerProjects.vsix"' -f
$pathtoVsixFile;
            $result = Start-Process -FilePath "C:\Program Files\Microsoft
Visual Studio\2022\Community\Common7\IDE\VSIXInstaller.exe" -ArgumentList
$args -Wait -PassThru
          shell: powershell.exe

      - run:
          name: Pause for 2 minutes
          command: Start-Sleep -s 120
          shell: powershell.exe
      - run:
          name: Soln build
          command: |
            #cd "C:\Program Files\Microsoft Visual
Studio\2022\Community\VC\Auxiliary\Build"
            #./vcvarsall.bat x64
            & "C:\Program Files\Microsoft Visual
Studio\2022\Community\Common7\IDE\devenv.com"
"$env:CIRCLE_WORKING_DIRECTORY\Project\PathWandGUI\PathWandGUI.sln" /build
Debug

          shell: powershell.exe
      - store_artifacts:
          path: ~\project\Project\PathWandGUI\Debug\net48
      - persist_to_workspace:
          root: .
          paths:
            - Project/PathWandGUI/Debug/net48/*

  publish-github-release:
    docker:
      - image: cibuilds/github:0.13
    steps:
      - attach_workspace:
         at: ./
      - run:
          name: "Publish Release on GitHub"
          command: |
            ghr -t ${GITHUB_TOKEN} -u ${CIRCLE_PROJECT_USERNAME} -r
${CIRCLE_PROJECT_REPONAME} -c ${CIRCLE_SHA1} -delete ${CIRCLE_BUILD_NUM}
./Project/PathWandGUI/Debug/net48/
```