Unity Cloud Build and Steam Deployment

December 2021

Background

Unity Cloud Build can automatically create builds when changes are checked into source control. It is easy to set up by following this tutorial:

https://learn.unity.com/tutorial/unity-cloud-build

Although Unity Cloud Build provides a robust solution for automatically creating builds it doesn't provide out of the box support for deployment to Steam.

Unity Cloud Build combined with automatic deployment to Steam provides a low cost continuous integration solution.

Setting Up Node on AWS

The Unity Cloud Build API provides the ability to send REST messages based on build events, such as when a build is successful. A Node application can listen for these events and take the steps to deploy the build to Steam.

Launch EC2 Instance on AWS

There are multiple ways to set up and host a server Node application -- the route I took was to use AWS (Amazon Web Services). You can create an AWS account and set up an EC2 instance for free following the steps in the link below:

https://aws.amazon.com/ec2/getting-started/
(Select "Amazon Linux AMI" as the OS for your EC2 instance)

In addition to the above, you will need to update the security rules to allow for inbound traffic on a port your Node app listens to. In the Description tab for the EC2 instance there is a link to the active security group. Select the security group, and then select the Inbound tab, and then select Edit. From the "Edit Inbound Rules" window select Add Rule and select "Custom TCP Rule" as the type, TCP as the protocol, and 5000 as the Port Range, and Anywhere is the Source.

Install steamcmd on EC2 Instance

steamcmd is used to upload builds to Steam from the EC2 instance.

Install steamcmd on Linux by following the steps shown here:
https://developer.valvesoftware.com/wiki/SteamCMD

steamcmd will invoke Steam Guard the first time it is run. Remember to log into Steam from the ec2 instance once, so the automated uploads are not interrupted by Steam Guard.

Install Node on EC2 Instance

Install Node on the EC2 instance using the steps in the following link:

https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/setting-up-node-on-ec2-instance.html

Install Node App on EC2 Instance

An easy way to get code onto an EC2 instance is to use Github. Use the following command to copy the Node app onto the EC2 instance:

```
git clone https://github.com/alanlawrance/steam-deploy-public.git
```

Install the Node app dependencies by navigating to the app folder (i.e., steam-deploy-public) and type:

```
npm install
```

Node App Configuration

The app needs Steam credentials and some other information to be able to log onto Steam and deploy the build. This information is stored in a configuration file that is specific to each Unity Cloud Build Target. This is done by using the Target Name for the build:

<targetname>.cfg

For example, if the Unity Cloud Build Target Name in win64-master, then the configuration is loaded from win64-master.cfg. The Node app ignores the BuildSuccess event if a matching cfg isn't found.

Create configuration files inside the steam-deploy-public directory.

The contents of this file have to follow the below convention, with 1 line for each entry (no comments or blank lines):

steam_username
steam_password
build/content
build/output
../steamcmd/steamcmd.sh
../steam-deploy-public/build/app_build_XXXXXXX.vdf
build/content/version.txt
steam_appid.txt
build/content/_steam_dll_path

build/content is where the zipped build gets decompressed. **Do not put anything in here as it gets removed and recreated before a build is made.**

version.txt is a text file generated that contains the Unity Cloud Build "build number" and the Git commit hash. You can use this in your app to display along with the Application.version from Unity.

Example From Poly Bridge 2:

<steam username here>
<password here>
build/content
build/output
../steamcmd/steamcmd.sh
../steam-deploy-public/build/app_build_1062160.vdf
build/content/version.txt
steam_appid.txt
build/content/Poly Bridge 2_Data/Plugins/x86_64/steam_api64.dll

The above example assumes a directory structure like this:

steamcmd

steam-deploy-public
steam-deploy-public/build
steam-deploy-public/build/content

Steam VDF Files

Uploading the build to Steam requires an app_build file and one or more depot files.

app_build_*.vdf
depot_build_*.vdf

Place these in the build directory, e.g., steam-deploy-public\build

The app_build file has a line for “setlive” that allows for activation on a specific branch. **Note this is not allowed to be the default branch (Steam restriction).**

app_build example:

```
"appbuild"
{
	"Appid"		"Your App Id"
	"Desc"		"Your Game Name"
	"Buildoutput" ".\output\"
	"Contentroot" ""
	"setlive"		""
	"preview"		"0"
	"local"		""

	"depots"
	{
		"1234568" "depot_build_1234568.vdf"
	}
}
```

depot_build example:

```
"DepotBuildConfig"
{
	// Set your assigned depot ID here
	"DepotID" "1234568"

	// Set a root for all content.
	// All relative paths specified below (LocalPath in FileMapping entries, and
FileExclusion paths)
	// will be resolved relative to this root.
	// If you don't define ContentRoot, then it will be assumed to be
	// the location of this script file, which probably isn't what you want
	"ContentRoot" ".\content\"
```

```
      // include all files recursivley
  "FileMapping"
  {
      // This can be a full path, or a path relative to ContentRoot
    "LocalPath" "*"

    // This is a path relative to the install folder of your game
    "DepotPath" "."

    // If LocalPath contains wildcards, setting this means that all
    // matching files within subdirectories of LocalPath will also
    // be included.
    "recursive" "1"
  }

      // but exclude all symbol files
      // This can be a full path, or a path relative to ContentRoot
  "FileExclusion" "*.pdb"
}
```

Launching Node App on EC2 Instance

From within steam-deploy-public launch the Node app using pm2 (a process management tool) so it will restart automatically if there is a crash or abnormal exit: `pm2 start app.js`

You can stop the app using: `pm2 stop app.js`

Show status of pm2 processes: `pm2 list`

If pm2 isn't installed, install with: `npm install pm2 -g`

For additional information on pm2 see: http://pm2.keymetrics.io/docs/usage/quick-start/

To ensure the Node app stays running when you close the session you can use the screen command:

1. Type `screen` then launch the app with pm2
2. Press Ctrl+A then Ctrl+D to detach your screen. You can logout or close the session and the process will continue to run.
3. To resume after logging back in type `screen -r`

PM2 Configuration

It is recommended that you limit the pm2 log size, to ensure it doesn't grow unbounded and possibly cause storage issues.

You can do this by installing:

```
pm2 install pm2-logrotate
```

The default limit is 10MB.

It is also recommended to add timestamps to the pm2 logs by starting pm2 with the –time parameter like:

```
pm2 start –time app.js
```

Updating Node App on EC2 Instance

When changes are checked into the Git repo, you can pull those changes onto the EC2 instance and restart the app. Pull the latest changes using:

```
git pull origin
```

Configuring Unity Cloud Build

Now that we have a Node server set up that can receive messages from Unity Cloud build, the next step is to configure Unity Cloud build to POST a BuildSucess message.

Select Settings > Integrations from the Unity Cloud Dashboard, which will give you an option to start a New Integration. After clicking on New Integration, select Webhook.

On the next screen select the "Cloud Build" service and then "Build Success" event within that service. There are other events that can be sent, but only Build Success is needed to get the information to deploy the build to Steam.

On the next screen you will need to specify the Webhook URL, which is the URL of the EC2 instance. You can find the Public DNS (URL) for the EC2 instance on the Description tab in AWS. Another way to get it is from the EC2 instance command line:

```
host <ip.address>
```

You will see output that looks like
XXX.XXX.XXX.XXX.in-addr.arpa domain name pointer
ec2-XXX-XXX-XXX-XXX.us-east-2.compute.amazonaws.com

The “ec2-XXX-XXX-XXX-XXX.us-east-2.compute.amazonaws.com” part is the URL. You will need to append the port number to the URL, so what you enter will look like:

http://ec2-XXX-XXX-XXX-XXX.us-east-2.compute.amazonaws.com:5000

Target Name

Target Name is what links the build to its configuration on the EC2 instance. The configuration file used is named <targetname>.cfg. For example if the Target Name is win64-master, then the EC2 instance will look for win64-master.cfg.

If the configuration file is not found, the build is not processed and uploaded to Steam. This is necessary since Unity Cloud Build sends a “BuildSuccess” for all build targets, and there may be targets that are built that shouldn’t deploy to Steam.

Change the Target Name via:

Cloud Build > Config > Basic Info > Edit Basic Info > Target Name

Executable Name

Set the executable name generated by Unity Cloud Build so that it matches what is specified to launch on Steam. Set the executable name produced by Unity Cloud Build via:

Cloud Build > Config > Advanced Options > Edit Advanced Options > Executable Name

Questions?

Reach out to alan.lawrance@gmail.com