Gradescope Autograder Configuration

Matt Russell

May 21, 2022

Introduction

Gradescope is great tool for autograding assignments. However, there is still a substantial amount of infrastructure required to deploy and run an autograder on gradescope. This document provides instructions for both setting up autograders on Gradescope, and for an autograding framework for `C/C++` code. Setup from start to finish is intended to take roughly 30 minutes. If you have any questions, please reach out to me at `mrussell@cs.tufts.edu`. Thanks!

Notes

This setup requires the use of Docker Desktop. In most cases, you will be able to uninstall it after the initial setup. However, it is suggested that you start the Docker installation while reading the document, as it may take a few minutes.

Infrastructure Background

Gradescope's autograders rely on Docker containers which are spun up each time a submission is graded. The default container runs a variant of `Ubuntu 18.04`, coupled with the bare-bones scripts to make the autograding framework function. The usual workflow is to manually upload a `.zip` file containing two scripts: `setup.sh`, which installs dependencies (e.g. `Python`, `clang`, etc.), and a shell script named `run_autograder`, which runs the autograder. The main issue here is that each time you upload the `.zip` file, the Docker container must be built from scratch, which can take quite a bit of time; this can compound quickly during the development of an autograder. This document provides an optional solution to the problem.

Autograding Background

Once the container is built, there is of course the issue of how to run and test student's code. This is no easy task! This document provides documentation on an autograding framework we have developed which makes writing tests for student code as easy as possible.

Infrastructure Setup

The solution for streamlining the infrastructure setup with Gradescope is twofold:

1. Build and upload our own Docker container to Dockerhub, which Gradescope will use.
2. Put the autograding code in a `git` repository which the Docker container can access at autograding time.

Note that these two elements are distinct from one another; if using Docker is something you really don't want to do, that's fine. The container building will just take more time in the aggregegate. In that case, follow the instructions on Gradescope's website regarding setup:
https://gradescope-autograders.readthedocs.io/en/latest/specs/. Note that you can still use the `.git` integration; However, if you wouldn't like to do that either, okay! Just skip ahead to the `Autograding Framework` section below.

Install Docker

Install Docker Desktop: https://www.docker.com/products/docker-desktop/
Note that you don't need to have it start on boot; you can start it before uploading the setup.

Setting up the Autograding Repo

If you don't currently have a repository related to course material, please make one. We suggest using `gitlab` for this: go to https://gitlab.cs.tufts.edu, and login with `LDAP`, using your Tufts eecs `utln` and password. You do not need a `README`. The example below will be for `cs 15`, but please follow the instructions for whichever course you're running. Now, in your terminal:

```
mkdir cs-15-autograding
cd cs-15-autograding
git init
git remote add origin git@gitlab.cs.tufts.edu:your_utln/
    path_to_your_repo.git
git switch -c main
```

We have a sample repo for you to get starter code from. Copy the files as follows:

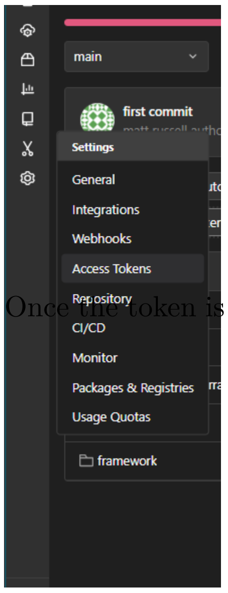
```
git clone git@gitlab.cs.tufts.edu:mrussell/gradescope-
    autograding
rm -rf gradescope-autograding/.git
mv gradescope-autograding/* .
rm -rf gradescope-autograding
```

Configuring the Docker build

Great! Now you have both the grading framework, as well as the elements necessary to build the Docker container for gradescope. We will need to do a few configuration steps to make this work. First, `cd Dockerbuild`. We will need to add three files here (more details for each are below):

- `.repopath` - the remote path of the repository, including an Access Token.
- `.dockertag` - the tag of the Docker container to build
- `.dockercreds` - the credentials to login to Dockerhub.

.repopath



First, go to `gitlab` in your browser, and navigate to the course repository you just created. Next, hover over the settings cog on the lower left, and select ‘Access Tokens’. Create an access token; this will be used by the Gradescope autograder to pull the most recent version of the autograding files for an assignment. We suggest only providing ‘read repository’ access to the token. Feel free to select whatever you’d like for the name, expiration date, and role (Maintainer is fine). Once the token is created, copy the key. Now, open a file named `.repopath` [in the `Dockerbuild` directory]. You will want to format the repository path as follows:

```
https://REPOSITORY-NAME:ACCESS-TOKEN@gitlab.cs.tufts.edu/path/
    to/repository.git
```

For example:

```
https://cs-15-2022uc:glpat-Blah8173Blah8023Blah@gitlab.cs.tufts
    .edu/mrussell/cs-15-2022uc.git
```

Great! Now the autograder will be able to pull the most recent version of the autograding files.

.dockertag

This will be the tag you'd like to use for your Docker container. Open a file named `.dockertag` and write:

```
tuftscs/gradescope-docker:YOURTAGNAMEHERE
```

Feel free to use anything in place of YOURTAGNAMEHERE. Note that the first section is required.

.dockercreds

We are using a single Dockerhub account for all of the autograding courses. The file `.dockercreds` should be available in the course's Tufts Box folder. If not, reach out to me at `mrussell@cs.tufts.edu` from your Tufts email address, and I'll send it to you ASAP. Note that this access token must be kept private; to that end, please keep your course autograding repository private (this is the default on `gitlab`).

Conclusion

Okay, you are ready to begin developing an autograder! Continue to the next section to learn about the autograder, and for a walkthrough to setup an assignment.

Autograding Framework

Introduction

The autograding framework is designed to have you writing and deploying tests as quickly as possible. It supports a variety of options related to test types, etc, however, in general tests will be a set of `.cpp` files. Each one will be compiled and run, and the output of the test will be `diff`'d against a reference implementation that you provide. `Valgrind` can be run on tests, `stderr` can be `diff`'d. The framework depends on a `.toml` file for the configuration.

.toml configutation

The `.toml` file will be configured as follows:

```
[common]
# common test options will go here
# this section is mandatory - it must be named 'common'

[set_of_tests]
# subsequent sections will each contain a group of tests to run
# configuration options placed here will override [common]
# test groups names (e.g. [set_of_tests]) can be anything
# tests in a section must be placed in a list named 'tests'
# tests = [
      {testname="test_0", description="my first test"},
      {testname="test_1", description="my second test"},
      ...,
      {testname="test_n", description="my nth test"},
]
# each test must have testname and description fields
# you may add any other option to a given test
# test-specific options override any 'parent' options
```

Test .toml Configuration Options

option	default	purpose
max_time	30	maximum time (in seconds) for a test
max_ram	-1 (unlimited)	maximum ram (in kb) for a test
valgrind	true	run an additional test with valgrind
diff_stdout	true	test diff of student vs. reference stdout
diff_stderr	true	test diff of student vs. reference stderr
diff_ofiles	true	test diff of student vs. reference output files
ccize_stdout	false	diff canonicalized stdout instead of stdout
ccize_stderr	false	diff canonicalized stderr instead of stderr
ccize_ofiles	false	diff canonicalized ofiles instead of ofiles
ccizer_name	“”	name of a canonicalizer function to use
our_makefile	true	use testset/makefile/Makefile to build tests
pretty_diff	true	use diff-so-pretty for easy-to-ready diffs
max_score	1	maximum points (on gradescope) for this test
visibility	“after-due-date”	gradescope visibility setting
argv	[ ]	argv input to the program

Autograder Files and Directories

Files/Directories Required to Run Autograder

These are all of the possible options, but you may not need many of them depending on your test configuration. [TODO] - ensure that the autograder is ’flexible’ - not sure if missing some directories/files will cause an unexpected crash.

```
.
|---canonicalizers.py [opt. file with canonicalization fn(s)]
|---testrunner.sh     [script that runs this file]
|---submission/       [student submission (provided by gs)]
|---testset/          [everything needed to run tests]
|   |---copy/         [files here will be copied to build/]
|   |---cpp/          [.cpp driver files]
|   |---link          [files here will be symlinked in build/]
|   |---makefile/     [contains custom Makefile]
|   |---ref_output/   [output of reference implementation]
|   |---solution/     [solution code]
|   |---stdin/        [files here are sent as stdin]
|---testst.toml       [testing configuration file]
|-
```

Files/Directories Created After Running Autograder

```
.
|--- results
|    |--- build      [student submission files]
|    |    |---
|    |    |--- test01 [compiled executables]
|    |    |--- ...
|    |    |--- test21
|    |--- logs
|    |    |--- status
|    |    |--- test01.compile.log
|    |    |--- test01.summary
|    |    |--- ...
|    |    |--- test21.summary
|    |--- output
|         |--- test01.ofile
|         |--- test01.ofile.diff
|         |--- test01.ofile.ccized
|         |--- test01.ofile.ccized.diff
|         |--- test01.stderr
|         |--- test01.stderr.diff
|         |--- test01.stderr.ccized
|         |--- test01.stderr.ccized.diff
|         |--- test01.stdout
|         |--- test01.stdout.diff
|         |--- test01.stdout.ccized
|         |--- test01.stdout.ccized.diff
|         |--- test01.valgrind
|         |--- ...
|         |--- test21.valgrind
|-
```

The `.diff`, `.ccized`, and `.valgrind` output files for each test will only be created if your configuation requires them; `.stdout` and `.stderr` files will be created for each test.