local system setup for cs-15

Matt Russell

February 14, 2022

The Problem

Introduction

Honestly, we were hoping to never need this document. Here's some background. Recently, TA's of the course have been loving using `VS Code` as a development environment. In general, it's friendly to use, and has fewer headaches than `Atom`. Or so we thought.

What Happened

After a few weeks this semester (spring 2022), we noticed that the Halligan server was slowing down. People's connections were dropping, and in general chaos was beginning to foment. People lost work. Students in `11`, `15`, and `40` were all affected. The `IT` staff were losing their minds! What was going wrong? Long story short, `VS Code` was using an inordinate amount of system resources per student; in fact, even when you close your remote connection, the folks over at Microsoft thought it would be a great idea to let resource-intensive processes linger on in the background on the remote system (Halligan servers) for up to 8 hours... And, unfortunately, our servers couldn't take the load.

The Band-Aid

The awesome folks over at `IT` determined the source of the problem, and wrote a 'reaper script' which kills these lingering processes. However, that solution is more of a band-aid than an actual fix. Thus, alas, much to the chagrin of everyone, we are at a standstill with `VS Code`.

What now?

Okay, so we have a few options:

- Use `SFTP` a vscode extension very similar to `Atom`'s `Remote Sync Pro`
- Encourage you to learn `Emacs` or `Vim`, which are text editors you can run from within a terminal.
- Go back to using `Atom` as you previously have.

All of these options are good, in their own way. And they all have problems too. The standard option we are going to propose for the course is to use the `SFTP` extension. Your workflow will then change a bit towards what you did with `Atom`, but things will largely be the same. You'll still have a terminal open inside of `VS Code`, which is manually `ssh'd` to the Halligan server. Here, you will run your programs. And, you'll be able to edit code in the editor - however, the code will be locally edited, and uploaded each time you save.

In general, this is totally fine. However, there is one problem. The unit testing `VS Code` extension that we built requires a terminal which is integrated with `VS Code`. In other words, if you open a terminal in the window, and then manually run `ssh utln@homework.cs.tufts.edu`, you will **not** have access to the `code` command, and, while the workflow above will work fine, the nifty extension will be no longer available.

Options

This brings us to testing options. Fortunately, there is the `unit_test` script on the server. This will still work as normal. However, we imagine that a few of you (probably a very small few, but still) might still want to use the extension! It is pretty nifty, after all :). For those few of you who do, and who are chomping at the bit for a slightly more complex setup, read on! Forge ahead bravely. However, you may feel free to ignore the rest of this document if you don't want to deal with the mess! Again, it was never our intention to do all this. But here we are.

The Local Workflow

Introduction

This brings us to our other option: somehow allowing you work locally, on your own system. This option is possible, but brings up another set of challenges, which, although solvable, are quite daunting!:

- You may not have all the software needed (`clang++`, `valgrind`, etc) to compile and run your code.
- You may have the software, but the versions are different from what we are using, which although unlikely, could cause problems.
- You are probably on a system that doesn't support `valgrind` (OSX and Windows!)
- If you are on your local system, and it catches fire, drowns, etc., all your data is gone and there's nothing you can do about it!
- If you want to work locally, you need some way to get files from the remote system to your system.

Well, fear not! There is a workaround to these issues. **However, be prepared, it's a doozie.** Also, this setup will require 1.5GB of disk space, and at least a few gigabytes of RAM. If your system doesn't meet these criteria, please consult with the course staff.

Docker

So, it turns out that the first three items above can be fixed with something called `Docker`. `Docker` is a tool which allows for the creation/maintenance of what are known as `containers`. `Docker containers` are effectively mini-operating systems that can come pre-packaged with software, etc. Such

containers are commonly used for packaging applications with all the stuff they need to run. Rather than have to build a different version of a given piece of software on multiple systems, using `Docker containers`, you can make one build of your software, and then run it anywhere the containers are supported.

VS Code to the Rescue?

And, it turns out that `VS Code` has an extension, which, just like how `Remote - SSH` allowed you to start and run a terminal 'inside' of a remote system, allows you to start and run a terminal 'inside' of a `Docker container` that is on your local desktop. This means that you can have access to a whole other operating system which comes pre-packaged with all of the necessary software for the course (including `VS Code` extensions), within your local machine. This extension is fittingly called `Remote - containers`.

Local Installation

To get all this running, you have to:

1. Install WSL2 (Only Windows Users need to do this step)
2. Install `Docker`
3. Install the `Remote-Containers` extension
4. Prep and then build the provided cs-15 `Docker Container`
5. Get files to/from the server.

And, after getting everything setup, you'd connect to the remote container to work, just as you would connect to the Halligan server. More details to follow.

WSL 2

First things first – for Windows users: you will need to install WSL and update to WSL 2. Installation for that can be found at this link

Installing Docker

To install `Docker`, go to this link and install the necessary version for your system. Note that because this is for academic use, you don't need to worry about a paid license. Once it's installed, make sure to open the `Docker Desktop`, so it's running in the background. There's no need to run the tutorials, etc, but make sure to accept the license.

Install the Remote - Containers Extension

Okay! Now that you have `Docker`, you'll need to nab the `Remote - Containers` extension from the extension store. If you're on Windows, install the extension **inside your WSL remote**. If you're on Mac, just install it locally. Note! You should **not** be at all connected to the Hallign server. In fact, to be sure, we'd suggest uninstalling the `Remote - SSH` app before continuing.

Setup cs-15 Docker Container

Now that you have both `Docker` and `Remote - Containers`, run the following commands in `VS Code`:

Download Setup Files

1. Open a terminal from within `VS Code`
   Note! For Windows folks, do open a terminal **from within a WSL remote container.** It will make file access, etc. easier for you. For OSX folks, no worries.

2. Move to a local folder where you want to develop your `cs-15` code. If you don't have such a folder yet, make one. We will now copy necessary setup files to create the docker container. First, `cd` to your `cs-15` directory. Then, run the following commands:

   ```
   git clone https://www.github.com/mattrussell2/15-docker
   mv 15-docker/.devcontainer .
   rm -rf 15-docker
   ```

   Make sure to include all of the '.' characters above!

Good! Now, if you run the command `ls -la`, you should see a folder named `.devcontainer`. This folder contains the necessary files to build and load into your docker container. **It will need to remain here.**

Prep the Install

Almost there! Time to register your `utln` with the installer.

1. run: `cd .devcontainer`

2. run: `chmod +x prep_install`

3. run: `./prep_install`
4. at the prompt, enter your `utln`

Build the Container

Okay, we're ready to roll! Now, let's build the `Docker container`.

1. Press `ctrl + shift + p` (`cmd + shift + p` on Mac)
2. Search for and select
   `Remote-containers: Open Folder in Container`
3. Navigate to and select whichever directory you did the above step in (your 'cs-15 root directory', which should contain the `.devcontainer` directory within it).
4. The window should refresh - and the docker container will be created. If you see an error related to the `Docker daemon`, make sure you have `Docker` running in the background. Also, be sure to indicate that you trust the author of the container! Feel free to press the button to view the logs and see what's happening - this step will take about 5-10m. It will only have to happen once. After the container is built, it'll be quick to load.
5. Once the setup is done, the logging window will stop producing output, and you should be able to create a new terminal with the 'plus' sign on the right hand side of the terminal window.
6. Note! In rare cases, your system might hang at the very beginning (on step 2/3). If it's doing this, then run the command
   `rm ~/.docker/config.json` to remove the docker config file, and start over.

Once the installation is complete, you should see `Dev Container: cs-15` in the lower left corner of the `VS Code` window. This is now a new remote you can connect to, just like the old halligan remote via `Remote-SSH`. So, Press the `plus` symbol in the terminal window to open a new terminal. If you press the extensions button on the left-hand panel, you should see that (for the container you're in), the unit testing framework, clang-format, and sftp extensions have all been installed and preconfigured for you.

Register SSH key

At this point, you will want to generate a new ssh key, and register it with the following two commands:

```
# after running this command, press enter at all prompts
ssh-keygen

# after running this command, enter your password
ssh-copy-id utln@homework.cs.tufts.edu
```

Okay! Now, you can run code in this linux container that will be effectively the same exact thing as running on the Halligan server, but it's your own machine! Any time you want to load up, after opening `VS Code`, simply run the `ctrl + shift + p` (`cmd + shift + p` on Mac) - `Remote-containers: Open Folder in Container`. Again, for WSL folks, remember to do this **from within the WSL remote**. Then, you can use `clang++`, `valgrind`, and the `VS Code` unit testing extension!

Dealing with Files

There is a significant distinction in the method of workflow here from the past. You are generally going to be working on your own machine. This means that you need a way to:

1. Get your current halligan files to your local workstation.
2. Have a way to backup files from your local workstation (to halligan).
3. Get future files from halligan to your local workstation.

All of these issues are **vitally important**. We will us the `sftp` extension to manage backing up files, and to pull the current files you have on halligan to your local system. When you installed the container, `sftp` was installed for you automatically. Note that the default path that was provided on the remote server was `/h/utln` - if you'd like to change this, open up the `.vscode/sftp.json`, and update the `"remotePath"` variable to whichever folder holds your `cs-15` code.

Getting your current files

In order to get your current files downloaded to your local workstation, **after you've updated the remote file path**, run `ctrl + shift + p`, and then

search for `SFTP: Sync Remote->Local`. Note the ordering there! This will pull your remote (Halligan) files to your local path. You should only need to do this once.

Backing up your work

IT IS ABSOLUTELY IMPERATIVE THAT YOU BACK YOUR FILES UP ON THE HALLIGAN SERVER. If things were set up properly, whenever you save a file, it should automatically be uploaded to Halligan. Check the output from `SFTP` for info - or, better yet, after saving, ssh in manually to the hw server, and see if your file is updated.

Copying new Files from the server for HWs / Labs

In order to copy new files from the server to your local machine for HWs / Labs, you should:

- First, create a new folder for your files
- Then, cd to that folder
- Lastly, run the command

```
scp utln@homework.cs.tufts.edu:/comp/15/files/HWNAME/* .
```

  Where you will replace `HWNAME` with the name of the hw as on the spec copy command (the short name, e.g. lab1 or hw1). WARNING! This command will overwrite any files with the same names in the current directory, so make sure to only copy to empty directories that you've just made.

Lastly, for projects, or hws that require you to run a `setup` command, things will not work as expected. We are working on a fix for this, but for now, please ssh in manually, run the `setup` command, and then sync the remote files to local. Also, for now, please run the reference implementation on the hw server. Again, we're working on fixes for this.

Last Notes

Thanks for setting this up! Hopefully the workflow will smooth itself out after awhile. If you have any questions please post on the piazza post introducing this document, or message me (Matt Russell) privately on piazza, and we can work it out. Cheers!