Automation testing with a C# executable and deploying it to Docker

An honest confession at the very beginning, I love working in the .Net environment and especially in the Console/Windows Forms/WPF “sector”. I know those are so pre-2010 technologies but I guess I started my career with them and have been a fan ever since.

So when I moved to Automation testing sphere, I have always tried and gone extra step to use those. Some of them being:

1. Using Selenium with C#, rather than Java (this doesn’t look like a big thing but the AUT was under Java )
2. I have avoided NUnit almost all of my automation career, just so that I can create an “exe” which “runs” on machines
3. For one of my projects which was/is heavily API services oriented, rather than using a tool like SoapUI which would have made life easier, I decided to create my own “Framework” and then go back and re-invent the wheel.
4. Dissed Java, just because it can run on Linux and MAC in addition to Windows
5. Avoided learning newer technologies like DOCKER cos I thought they were not compatible with C# apps.

But #5 was until the COVID-19 pandemic. With the time available in hand, I decided to see if my beloved “C# exe” can also be ported to Docker. It also helped that I hated the fact that I would have to work on porting my code to Java or Python for it to be Docker-ised

**Spoiler warning:** IT does and guess what it does it gloriously.

Problem Statement

For this document perspective, let’s chalk out our problem statement/ mission

1. Have a simple automation suite written in C#
2. Suite should have an external data file (Excel file in this example)
3. Suite should spit out execution results at the end
4. Have the above suite in a Docker image
5. Run a container with the image created at #4
6. Retrieve results created in the run and store it in our local machine

So what I plan to do in this example, is have a simple automation script based on Selenium C# and get it docker-ised.

What we need?

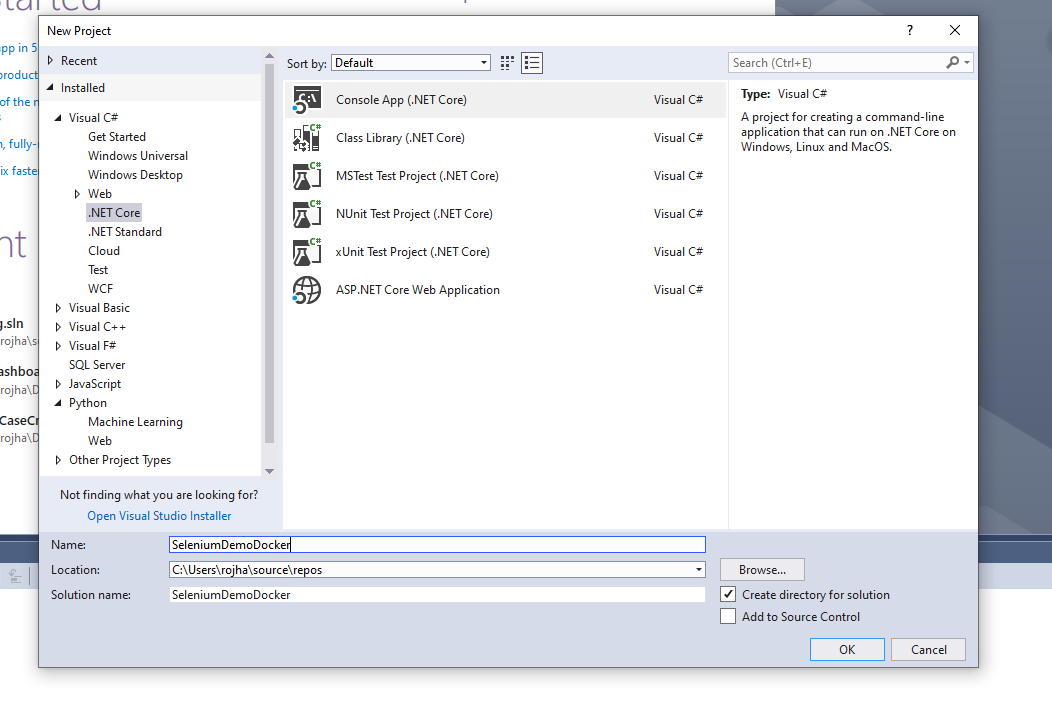
Since it’s a fairly straight-forward thing, we require only the following:

1. Docker for Windows
2. Visual Studio 2017 or later
3. .Net Core SDK
4. Selenium libraries

Getting Started

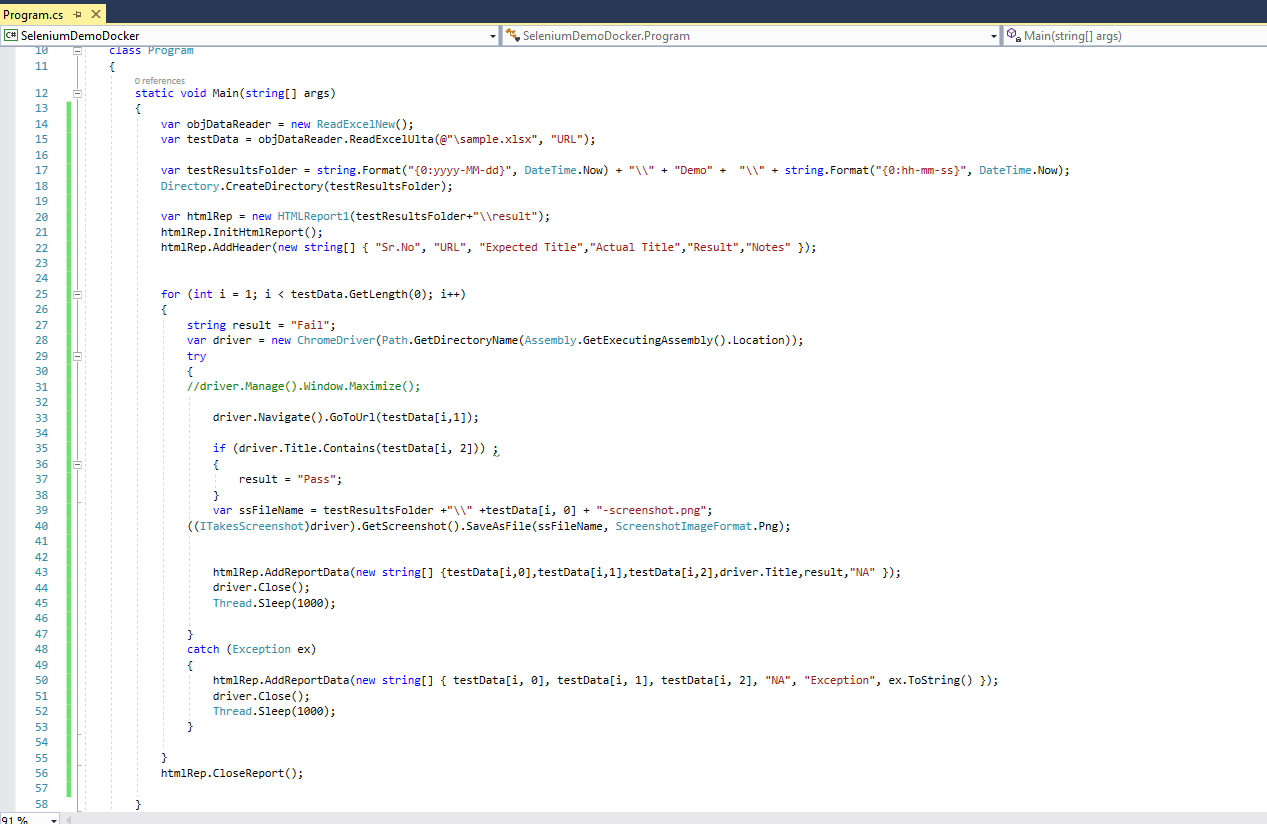
First things first, in my limited knowledge you can’t create Docker images from projects build on .Net Framework 4/4.5.

So make sure whenever you create your project have it created as a .Net Core application



Import your libraries and have your code ready. Make sure it runs on your machines before we even begin to port it to Docker.

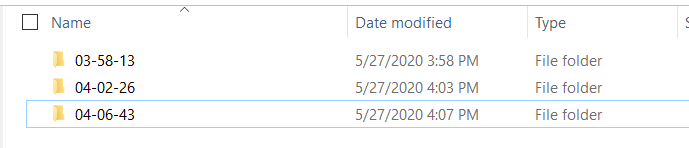
So my code to run it in my local machine looks something like this



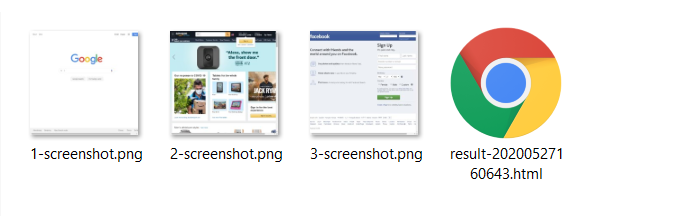
There is an excel sheet which has following data



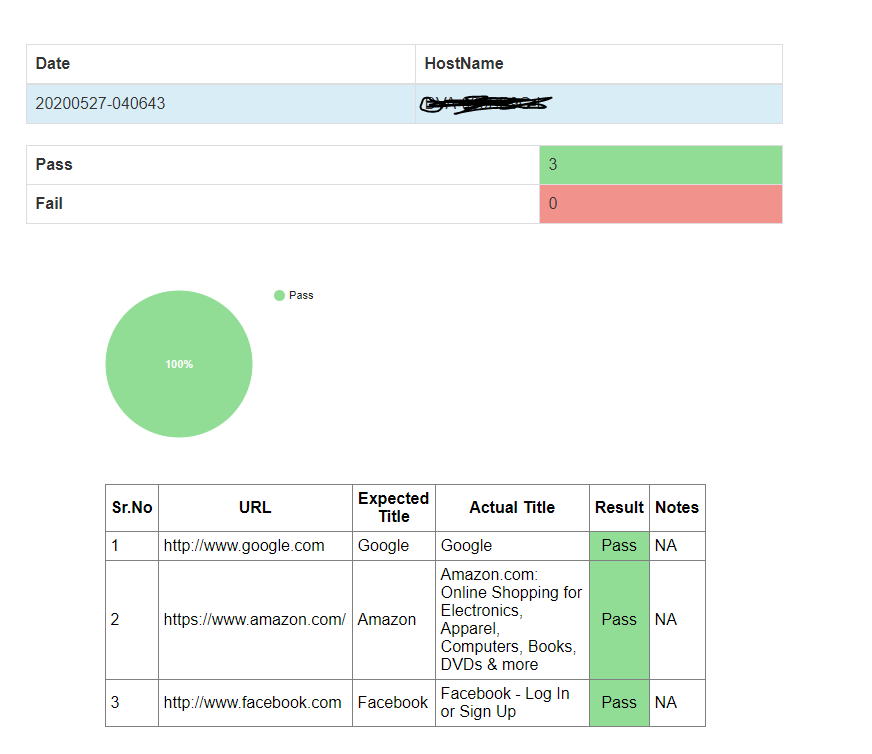
When the script runs it verifies the title from excel to the one in the browser title and marks TC as Pass and Fail… it takes screenshot as well and stores in a timestamped folder



Contents of the folder :



My report file looks like this:



Step 1 - Dockerfile

Now that we have code which runs on local machine, let’s start the process to make it Docker compatible.

First step is to create a Docker file.

Qs: Well what’s a docker file?

Ans: As per the definition from Docker’s official website [here](https://docs.docker.com/engine/reference/builder/) ,

“A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession. This page describes the commands you can use in a Dockerfile”

In a nutshell, Dockerfile is the utility which lets you execute commands you want to run before you actually try to “run/execute the application “. This might include (but not limited to):

1. Downloading the pre-requisites (.Net Core SDK in the docker environment)
2. Copying over your source code to Docker image
3. Copying any additional files (e.g. : certs ,licenses) which might not be part of the build but are required
4. Downloading a file over the internet (e.g. : in our case Test Data sheet)
5. Compiling the project
6. Running the actual executable

Though it might be a wrong analogy, I like to consider Dockerfile to be very similar to a “.bat” file in the Windows environment.

Always remember whenever you do “BUILD” in a docker eco-system, it’s always the Dockerfile which is executed

So first let’s list down the things we need in our image to run

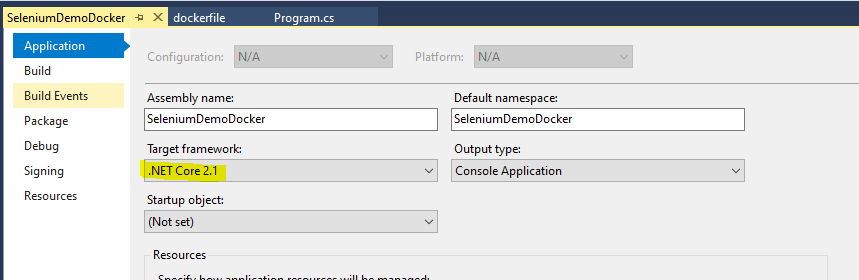
1. .Net Core SDK
2. Chrome
3. Chrome drivers
4. Our code
5. Data file

So our Dockerfile should have commands to get all of the above pre-requisites done

1. First line is to get the Docker image to know that the code is built on which SDK and download it



Since I am using VS 2017, my code was built on

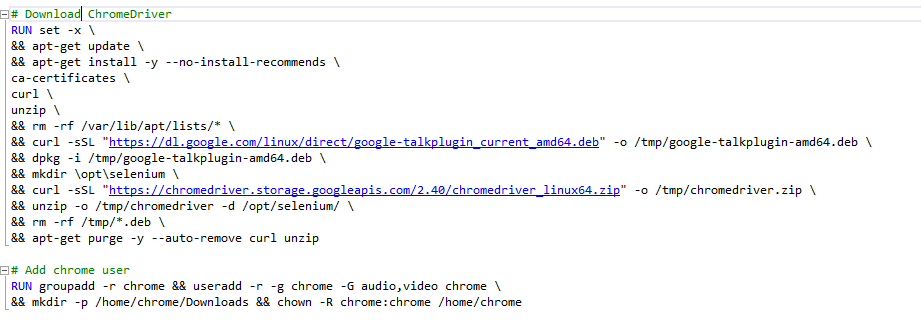


1. Getting chrome installed



The above commands downloads the Google talk plugin and installs Chrome on the image

1. Now let’s add commands to download ChromeDriver to run our execution



Note: For Steps 2 and 3 above, I am really thankful to [Pablo Assis](https://github.com/devpabloassis) for his excellent docker file [here](https://github.com/devpabloassis/seleniumdotnetcore/blob/master/Dockerfile).

Also another point to note is that Step 2 and 3 remain the same, no matter what is the Programming Language you would be using to script your automation test cases.

1. Now that we have the Chrome and ChromeDriver, let’s get our code from local machine to the image.



This copies over all the files from the folder your Dockerfile is located in to the “src” folder in the image.

1. Now let’s download the test data file from the internet over to our image



1. Now let’s compile our code and have an executable created with the Docker command line



DOCKER FILE

FROM mcr.microsoft.com/dotnet/core/sdk:2.1

ADD https://dl.google.com/linux/direct/google-talkplugin\_current\_amd64.deb /src/google-talkplugin\_current\_amd64.deb

# Install Chrome

RUN apt-get update && apt-get install -y \

apt-transport-https \

ca-certificates \

curl \

gnupg \

hicolor-icon-theme \

libcanberra-gtk\* \

libgl1-mesa-dri \

libgl1-mesa-glx \

libpango1.0-0 \

libpulse0 \

libv4l-0 \

fonts-symbola \

--no-install-recommends \

&& curl -sSL https://dl.google.com/linux/linux\_signing\_key.pub | apt-key add - \

&& echo "deb [arch=amd64] https://dl.google.com/linux/chrome/deb/ stable main" > /etc/apt/sources.list.d/google.list \

&& apt-get update && apt-get install -y \

google-chrome-stable \

--no-install-recommends \

&& apt-get purge --auto-remove -y curl \

&& rm -rf /var/lib/apt/lists/\*

# Download ChromeDriver

RUN set -x \

&& apt-get update \

&& apt-get install -y --no-install-recommends \

ca-certificates \

curl \

unzip \

&& rm -rf /var/lib/apt/lists/\* \

&& curl -sSL "https://dl.google.com/linux/direct/google-talkplugin\_current\_amd64.deb" -o /tmp/google-talkplugin-amd64.deb \

&& dpkg -i /tmp/google-talkplugin-amd64.deb \

&& mkdir \opt\selenium \

&& curl -sSL "https://chromedriver.storage.googleapis.com/2.40/chromedriver\_linux64.zip" -o /tmp/chromedriver.zip \

&& unzip -o /tmp/chromedriver -d /opt/selenium/ \

&& rm -rf /tmp/\*.deb \

&& apt-get purge -y --auto-remove curl unzip

# Add chrome user

RUN groupadd -r chrome && useradd -r -g chrome -G audio,video chrome \

&& mkdir -p /home/chrome/Downloads && chown -R chrome:chrome /home/chrome

WORKDIR /src

COPY . .

ADD "https://srv-file19.gofile.io/download/pWOMw4/sample.xlsx" /src

RUN dotnet build -c Release

ENTRYPOINT ["dotnet", "run"]

Changes to the code

Now since our automation would run on a Docker container, there would be changes required in our code to accommodate that.

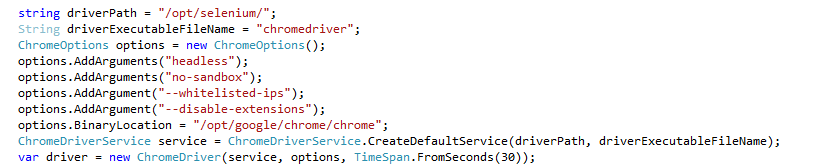
Some of the changes which I would like to point out especially are

1. Chrome would have to be run in “headless” mode
2. We would be using the Docker downloaded binaries related to ChromeDriver

So our code changes from



to something like this



1. Make sure the any path in your code are relative. Also since in the Dockerfile command #4 we had set Working Directory to src, all of the folders created henceforth will be inside the “src” folder



E.g.: The above code creates a folder “Results” in the “src” folder , where we can store our results and screenshots.

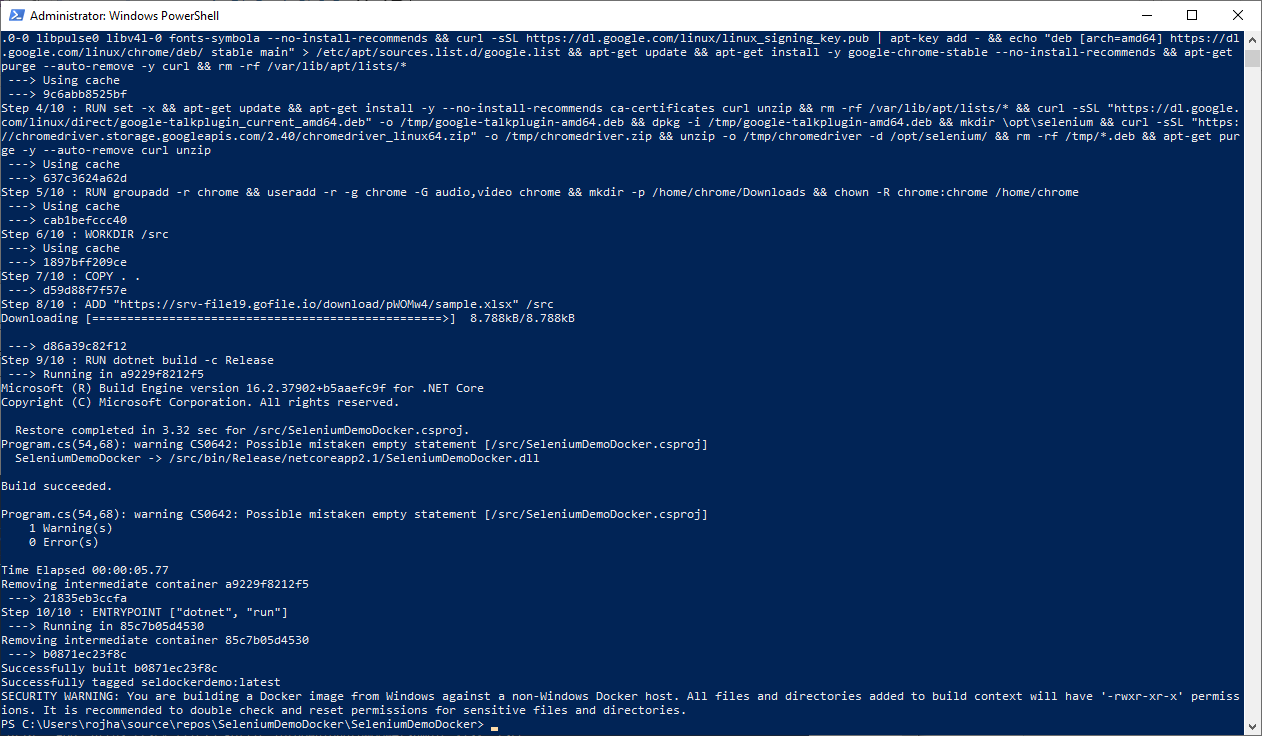
Docker Image creation

Now that we are ready, let’s build the docker image first

1. Open up PowerShell/CMD in admin mode
2. Navigate to the location of the docker file (using cd)
3. Use this command to build your docker image with the name “seldockerdemo”

**PS C:\Users\rojha\source\repos\SeleniumDemoDocker\SeleniumDemoDocker> docker build --tag seldockerdemo .**

1. Now all the files required would be downloaded, code files copied, code compiled and your image would be ready.



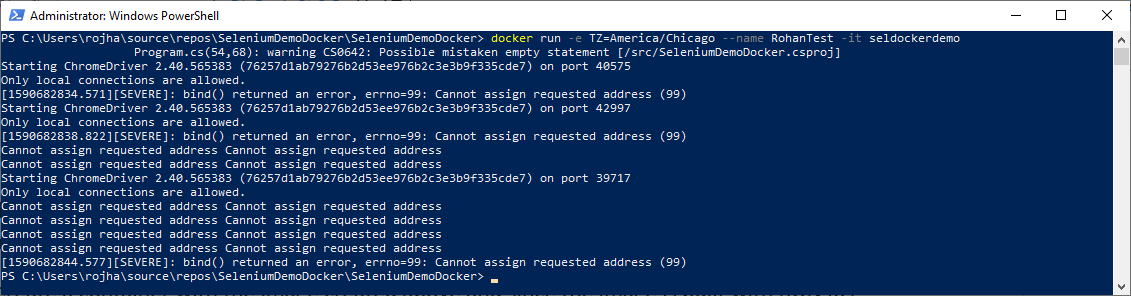
Container and Execution

Now that we have the image ready, all we want to do is run the image in a container and get back the execution results from the Docker container

1. Let’s create a container with the image created above and since the image is built with RUN (as per Dockerfile), it will execute our automation code.

**PS C:\Users\rojha\source\repos\SeleniumDemoDocker\SeleniumDemoDocker> docker run -e TZ=America/Chicago --name RohanTest -it seldockerdemo**

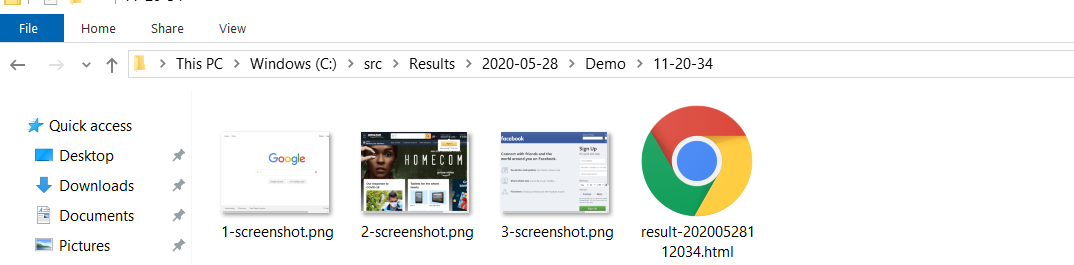
1. Above command creates a container with name RohanTest and executes the image “seldockerdemo”
2. Also it sets the time zone to CST (my local time zone). It’s not important and can be skipped, however I like my results/logs to be in the same time zone as mine (personal preference)

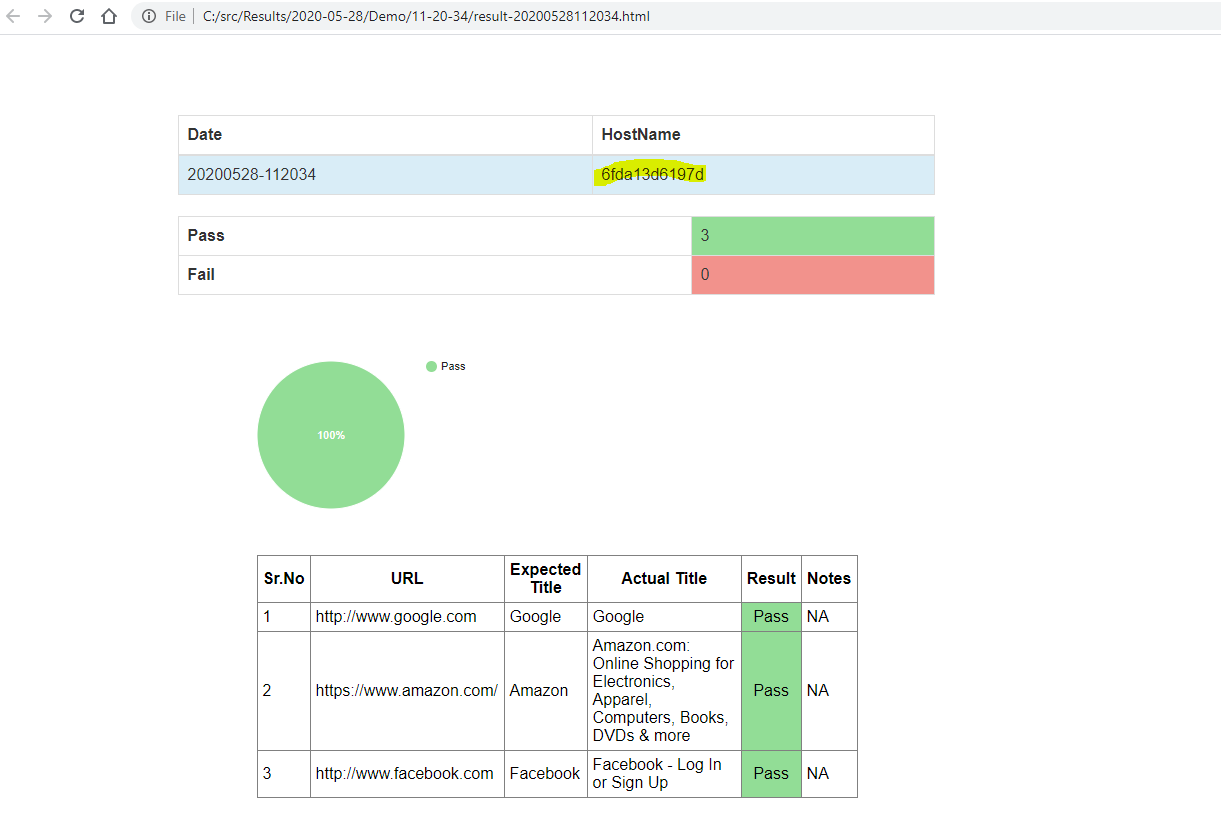


1. You will see execution logs on the command line
2. Now let’s see how the execution went and copy over the results to our local machine.

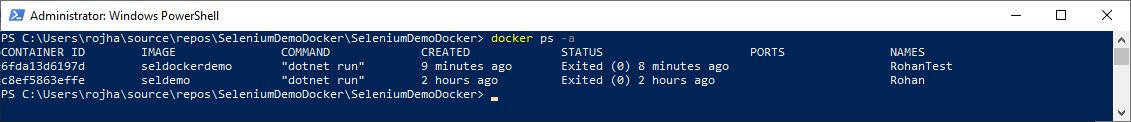
**PS C:\Users\rojha\source\repos\SeleniumDemoDocker\SeleniumDemoDocker> docker cp RohanTest:/src C:\src**

The above command would copy over the “src” folder from the container over to my local machine’s “C:\src” folder (I copy the whole folder, to cross-check the code file version)





Notice the HostName in the result file, it’s the same as Container name



So hurray, we have run out .Net automation code for Selenium on Docker.

I know it was a very simple example, but the only intention of this was to show that it can be DONE.

**Noob Tip:** Create a batch file (.bat) and have all the commands one after the other, to run the image in container…copy the results... and then finally delete the container.

I will put this whole project in GitHub for anybody who might be interested.