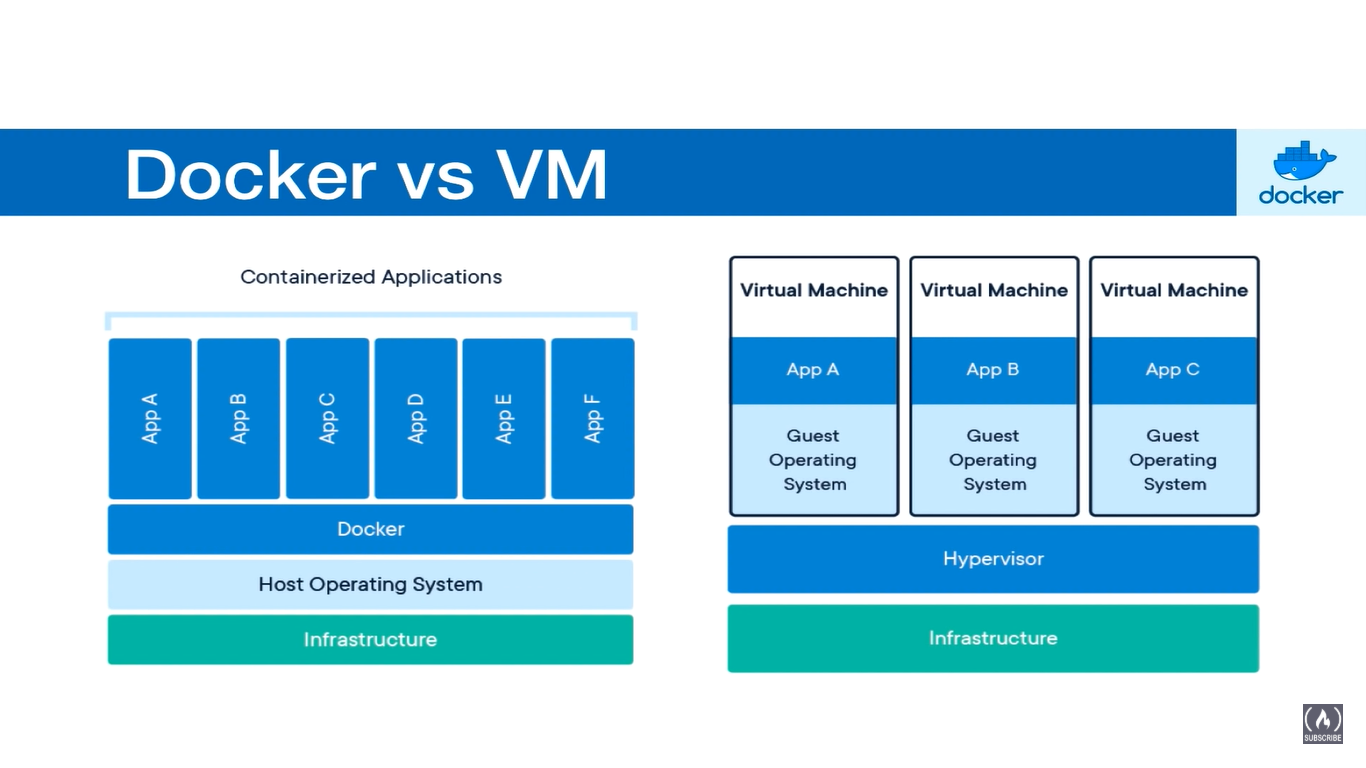
# **Keyboard shortcuts**

**Ctrl + L** - Clears the screen on powershell.



# **Getting Started with docker**

* Download docker desktop (Windows AMD64 in my case).
* Run the installer.
* Verify the installation. *“docker –version”*



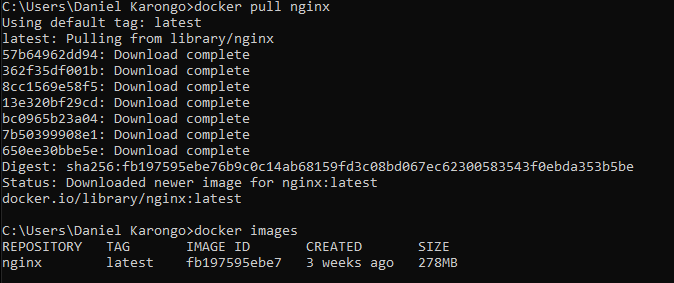
# **Images and Containers**

A docker image is a snapshot/ template of everything you need to run an application. A running instance of this image is what is called a container.

## **Downloading Images and Running Containers**

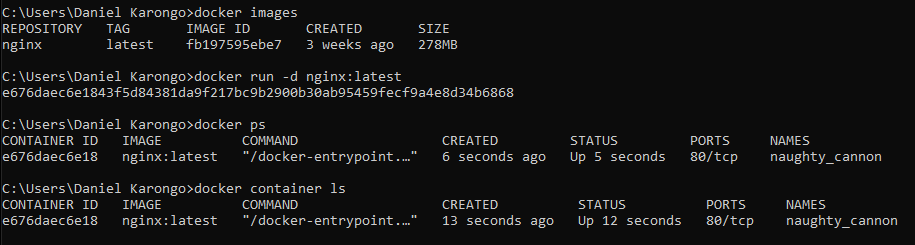
You can download some already-created images from docker hub.

To see all the images you have on your local docker, run *“docker images”*

**

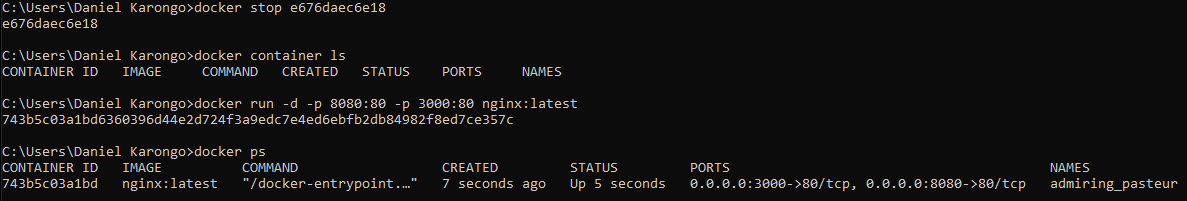
To run a container from an image, run *“docker run -d <imageName>:<imageTage>”.*

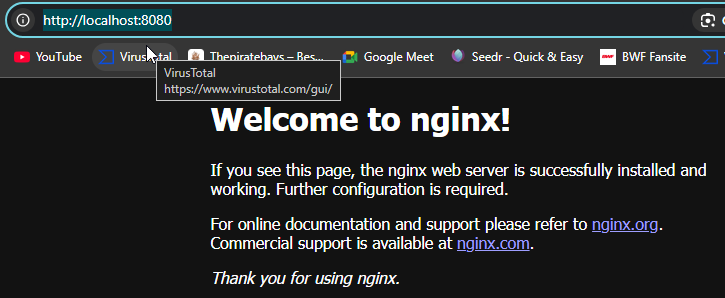
To see the list of all the running containers use either *“docker container ls”* or *“docker ps”.*

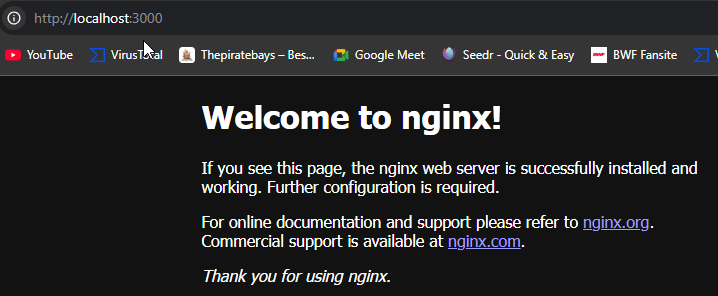


## **Exposing Container Ports**

The containers can be exposed to the host computer using ports such that a host machine port is mapped to the tcp port of the container. This can also be done in a such a manner that multiple host ports are mapped on to the single container port. To export a container port we use the “-p” option on the “docker run” command.



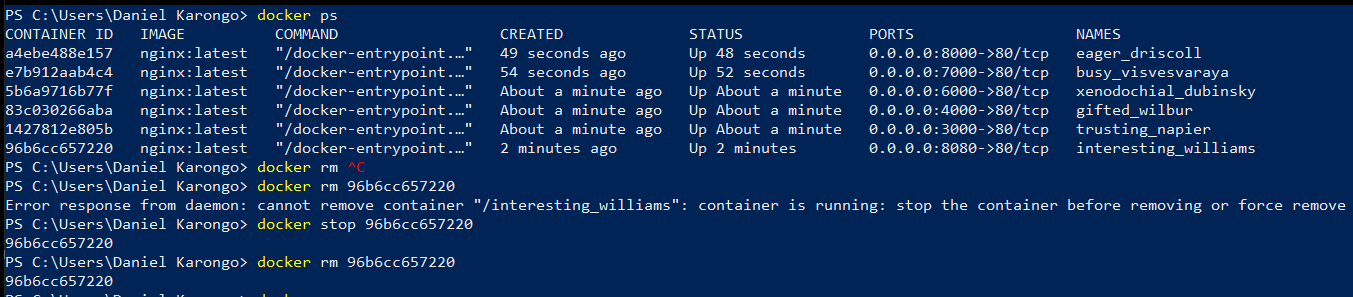


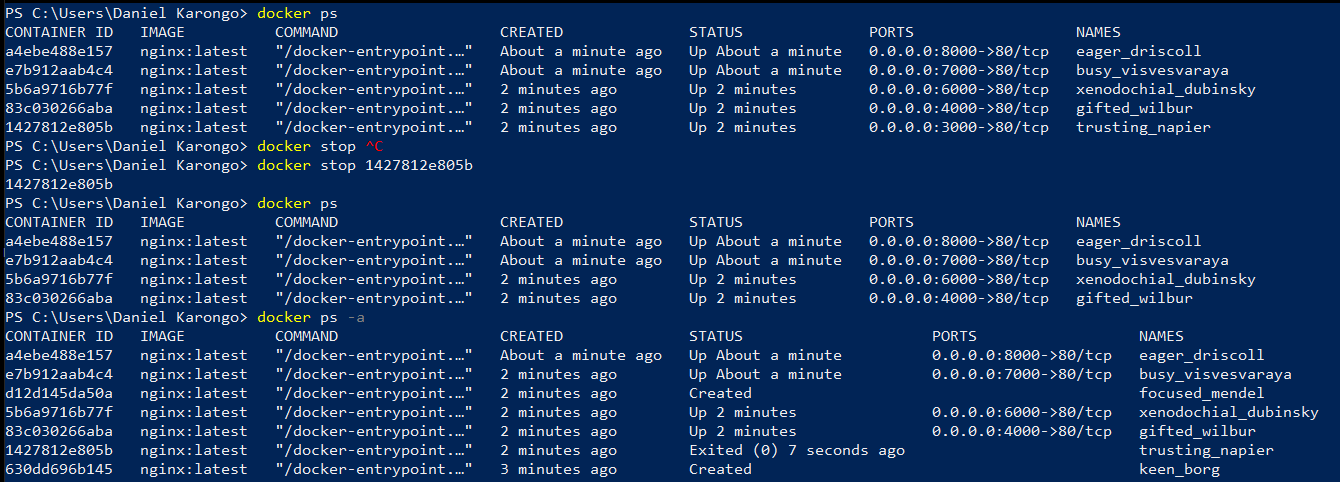


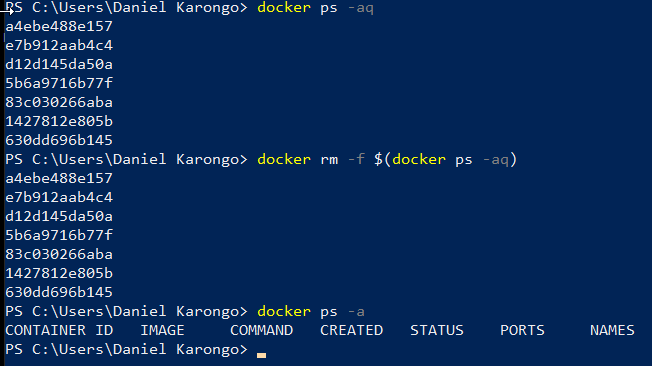
## **Managing Containers**

### **Starting, Stopping and Deleting Containers**

*“Docker stop <containerID>/<containerName>”* does not remove the container from the docker runtime environment. To remove the same we need the *“docker rm”* command.

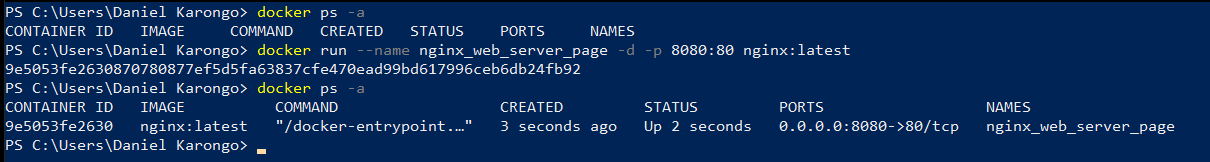






### **Naming Containers**

To name a container, we use the *“--name”* option on the *“docker run”* command.

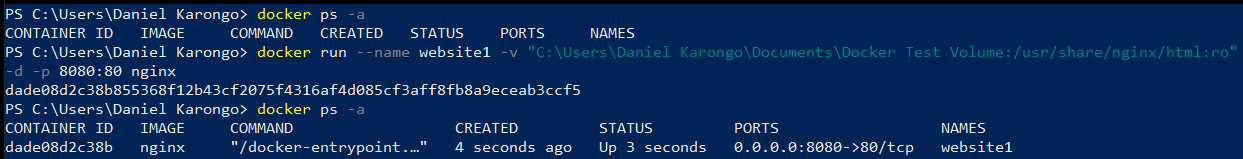


# **Volumes**

These are the means by which the containers and the host computer, or different containers on the same host communicate. The volumes can store folders and files, and these are accessible to the container(s) as well as the host.

## **Creating a volume**

We use the *“–v”* option on the “docker run” command and pass the pass to the directory that will be shared between the container and the host.



***Note:***

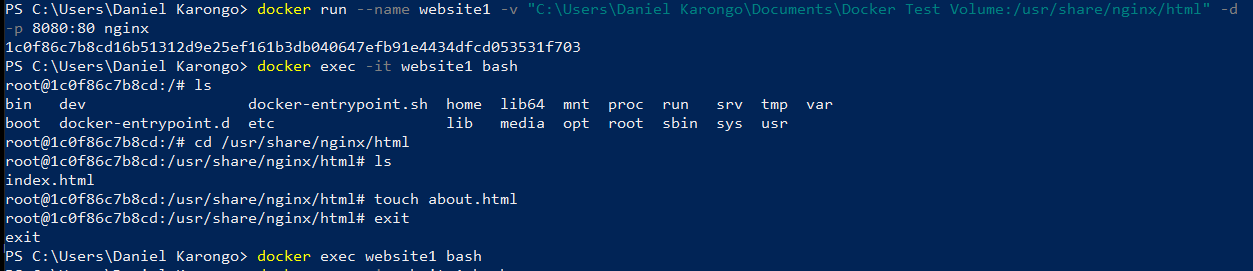
*The “C:\Users\Daniel Karongo\Documents\Docker Test Volume” refers to the volume’s directory.*

*The “:/usr/share/nginx/html” refers to the corresponding directory in the container.*

*The “ro” at the end of the path means “read only”.*

## **Adding files from to the volume from the container**

This requires that you omit the “ro” - read only from the –v option when creating the container.



***Note***

*The “docker exec” e*xecutes a command inside a running container.

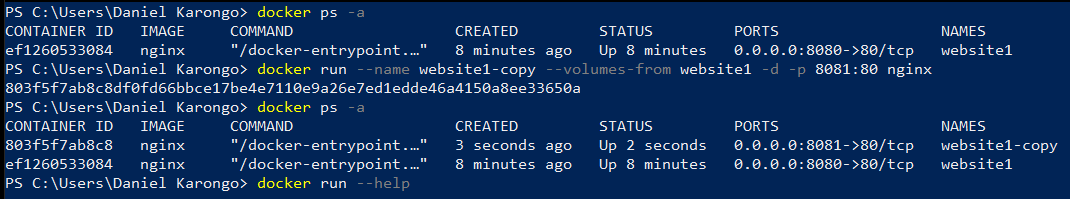
*The “*-it*” signals interactive mode with a pseudo terminal interface so that you can navigate through the containers file system as though it were a machine in itself.*

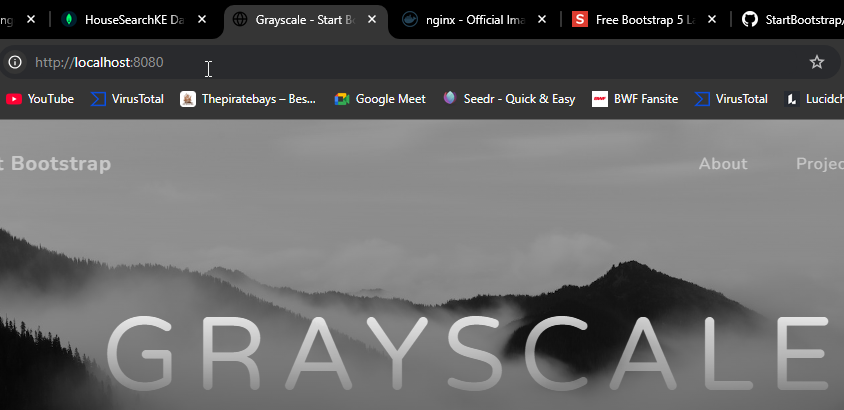
*The “website1” is the name of the running container where the command will be executed.*

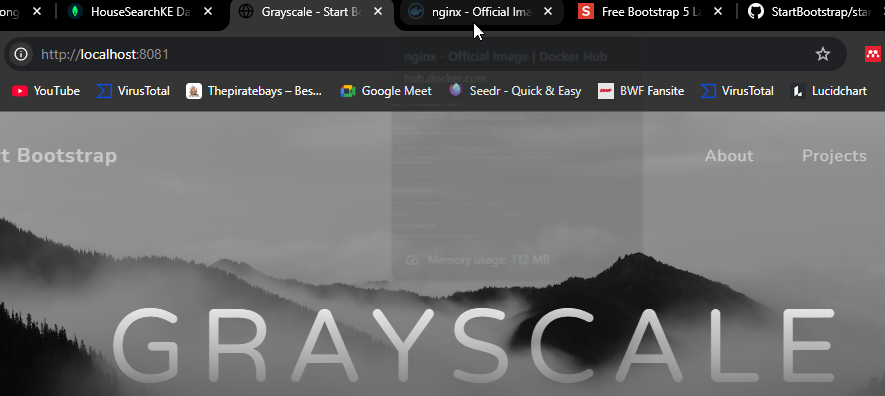
*The “bash****”*** *Specifies the command to execute inside the container. The nginx image is a linux-based image.*

## **Sharing volumes between containers**

Use the “—volume-from <parent\_container>” command to get a volume from an existing container and use the same as that for the new container.





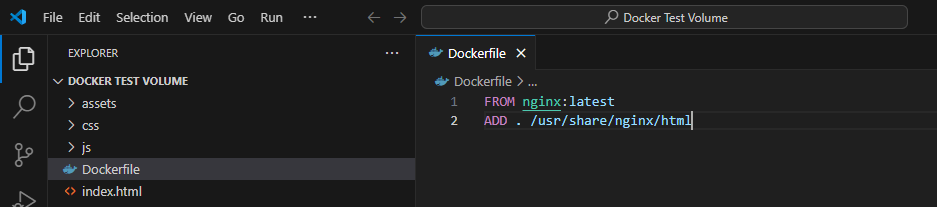


# **Build Custom Images**

## Dockerfile

This is a file that specifies the steps that should be followed to build our own custom image. It is place on the base directory of our project and it is what is called when we run the “docker build” command.

The instructions to be used in this dockerfile can be found at the official documentation: <https://docs.docker.com/reference/dockerfile/>.



**Note:**

I have opened the folder that we were using as the volume, created a dockerfile file within it, at the base of the project (the custom bootstrap website).

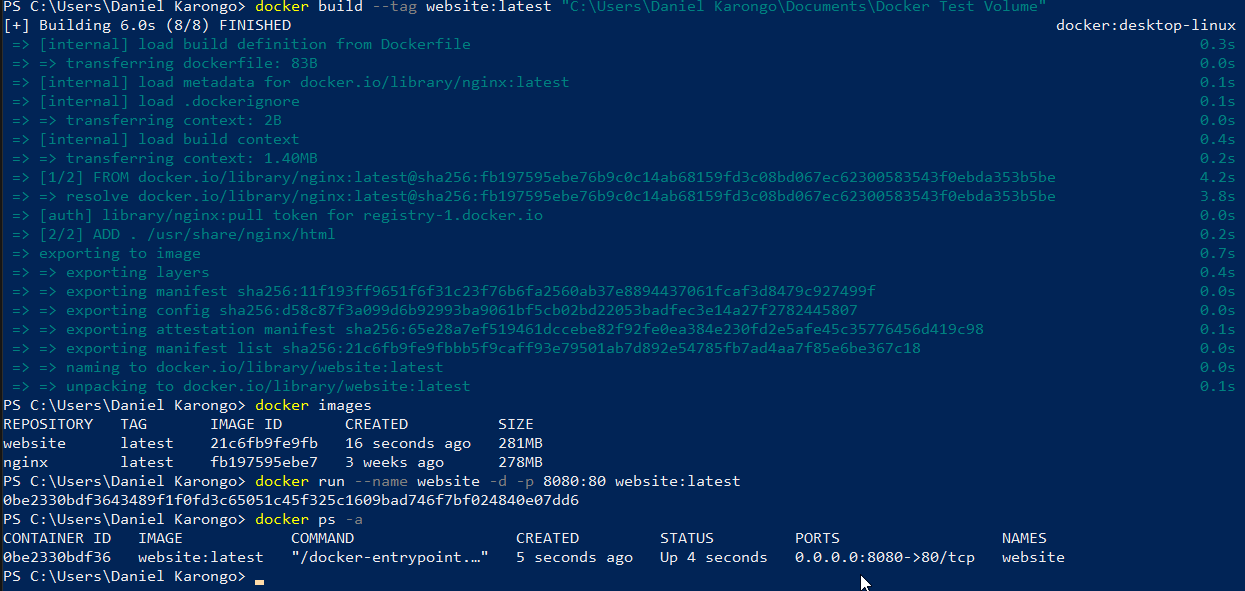
It is rare that we build out images from scratch. That is why I have used the “nginx:latest” as the parent image to base this new image on.

The “.” In the “ADD” instruction refers to all the files in the current directory.

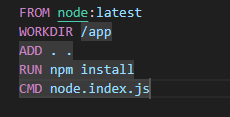
The “*/usr/share/nginx/html*” refers to where the files will be place in the container.

### **Docker Build**

This is the command that build the actual image, *“docker build –tag <image name>:<tag name>” “directory containing the docker file”.*



## **Caching when Building an Image**

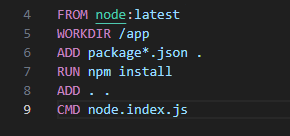
When docker is building an image, depending on the base image, it may take very long. For example, an image may use the “node” base image for the build. The docker file for such a build may be as:  


The “npm.install” command is what will take a lot of time. When the dockerfile is called upon multiple times, e.g. when creating updated to the image, this becomes inefficient.

This can, therefore, be overcome by adding the package.json and package-lock.json files files to the WORKDIR and then running npm install, before adding the rest of the source code.

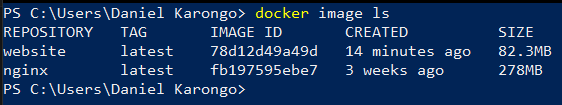
Because of hashing, the step described above will rely on the cache from the previous build because, if the project is mature enough, the package\*.json files are not changed very often, and there will be no need to run another npm install when the same command has been cached already.

The new docker file will be as:



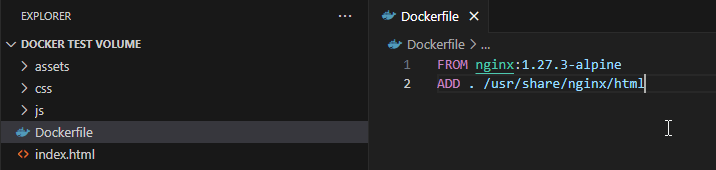
## **Alpine JS**

Consider the file sizes of the images below:



These are both created from the nginx base image. However, the smaller, more efficient one is built from the image “nginx:alpine” rather than “nginx:latest”. Any base image in the docker hub with an alpine tag is smaller and more efficient than its regular counterpart.

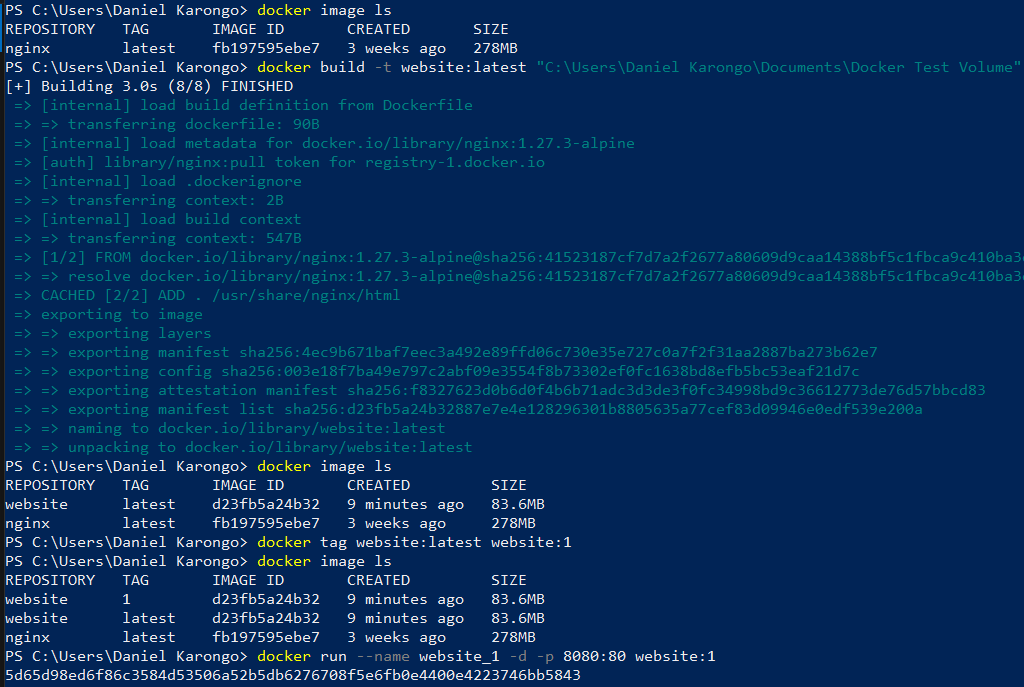




## **Tags and versioning**

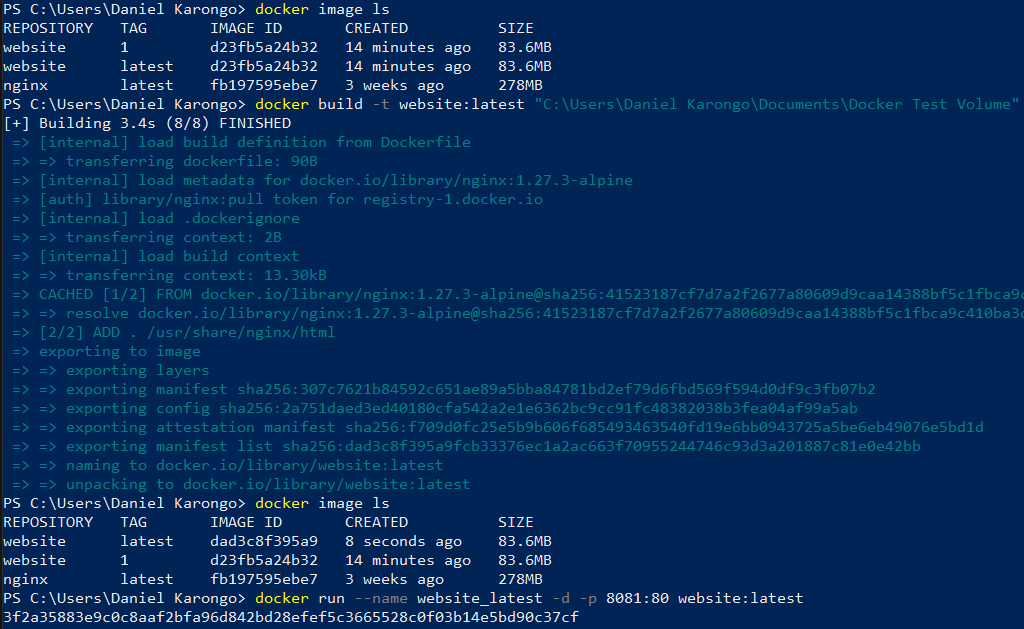
It is possible to restrict your base images to specific versions, as shown above by appending the version number to the tags portion of the image.

It is also possible to change the versions of already existing images as well as create some new versions of a same-named image.



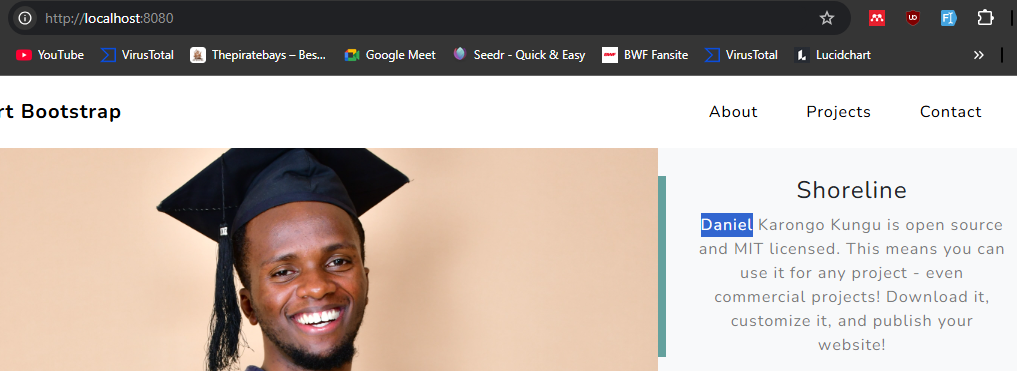
This:

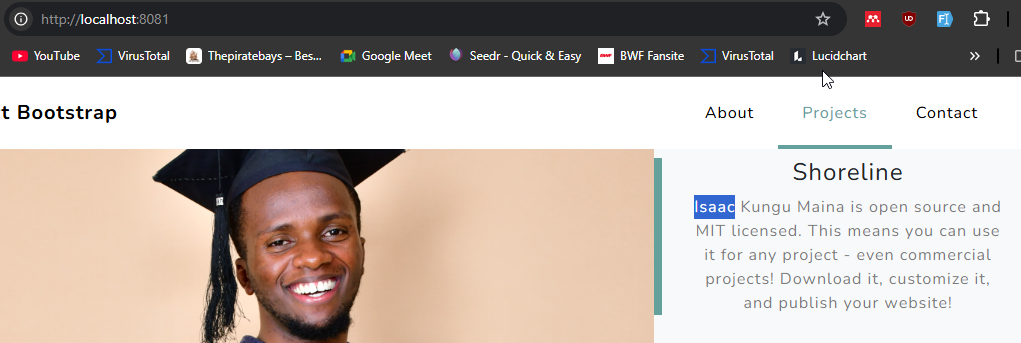
* create a new image “website:latest”.
* Changes the tag of the new image to “website:1” but create a new image in the process known as “website:1”
* Create a container from the “website:1” image running on port 8080.



This:

* Create a new image called “website:latest” overwriting the old “website:latest”
* Create a new container from the new “website:latest” image running on port 8081.





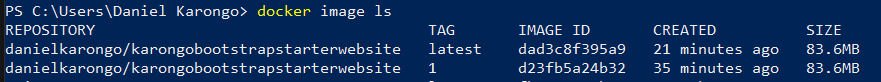
# **Docker Registries**

These are remote docker servers than can hold our images and run them. They include docker hub, quay.io and Amazon.ECR. The images can be set to be public or private. To move our local images to a docker registry we use the “push” command, similar to github.

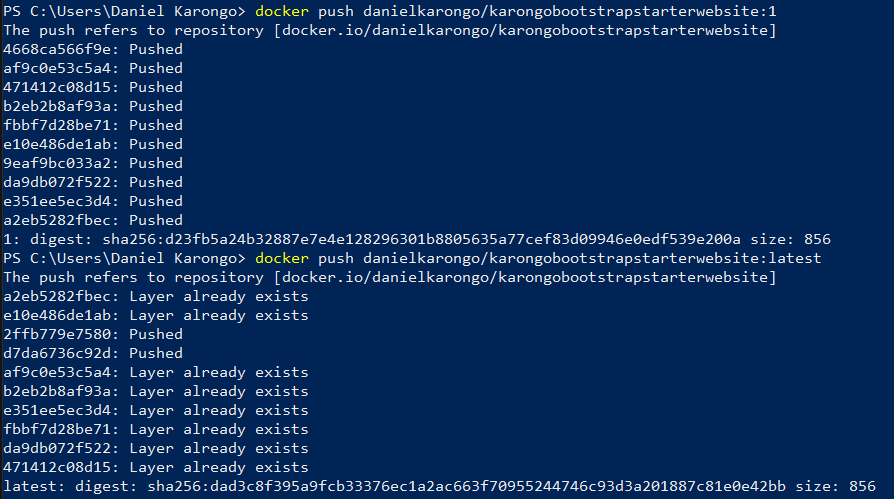
## **Pushing our images to docker hub**

* Create a docker hub repo. You will need to have an account.
* Create a repository on the docker hub.
* Rename your local images to match the pattern on docker hub.



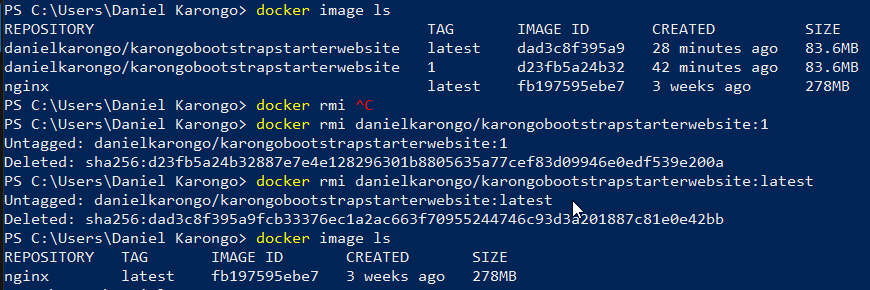


* Push the images, one tag at a time.

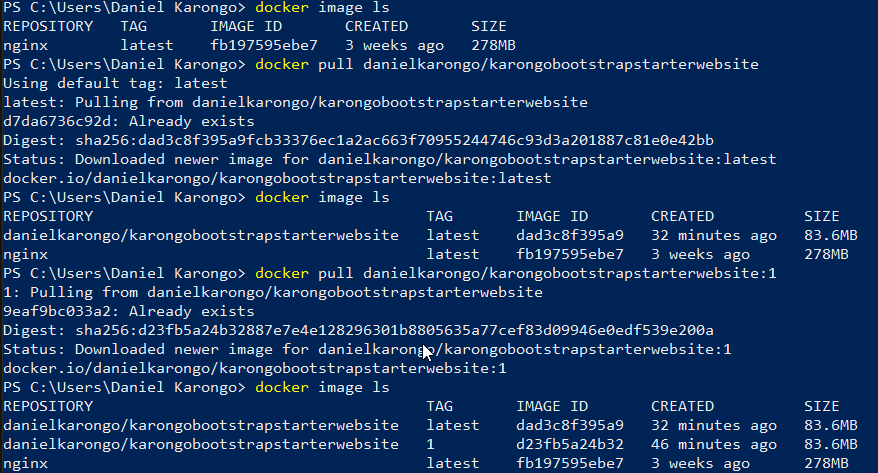


## **Pulling images from docker hub to the local docker**

Use the pull command



Delete the old local images. For demonstration purposes.



# **Container Monitoring**

We can use “docker inspect <container id>”, “docker logs <container id>” or “docker exec –it <container Id>/<container name> <container command>”

