Creating Your First Containerized Application

As industry continues to progress towards lightweight scalable workloads, containers are becoming more common replacement for virtual machines. In a previous post, we have described how containers in conjunction with Kubernetes can help manage the cluster orchestration, and in this guide, we will provide instructions on how to build, run, and manage a docker container to expose a simple application.

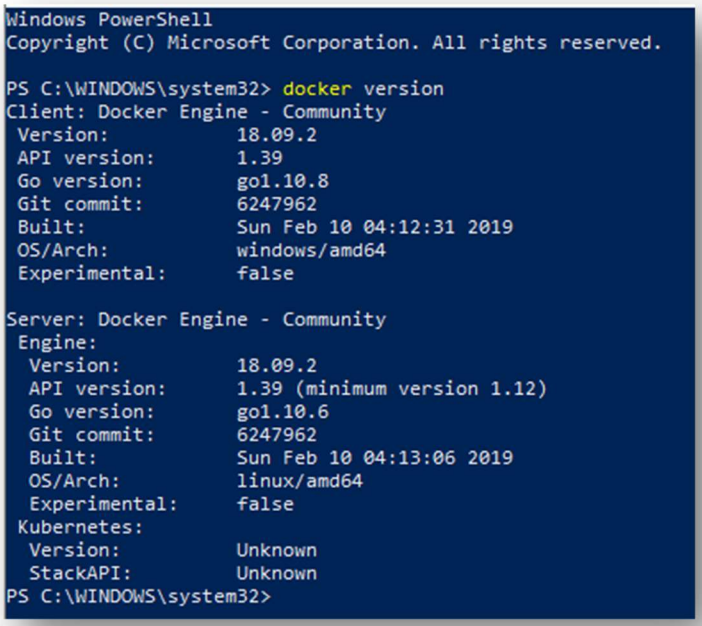
Set Up a Simple Application

While most applications are generally more complex, a simple JavaScript Tracker will be used for demo purposes. It consists of an HTML index page (including external links to Bootstrap, jQuery, and chance.js), and a custom java script file. The files are packaged together in a working directory labelled "html".

Docker Software

To create a container, the docker software will need to be installed on the environment. For this case, the Docker desktop app (on Windows) is being used with Linux containers. This could also be run on any other environment where you could download docker in either the Linux or Windows environment (this guide references a Linux container, but information about Windows containers can be found here.)

To verify that docker in installed and running, type **`docker version`** in your terminal session



Create a Dockerfile

The Dockerfile is a text file which contains all the commands necessary to build a docker image. The commands cover a wide variety of user defined steps and runs sequentially. In this case, the Dockerfile is configured as:

```
FROM httpd:2.4

COPY ./html/ /usr/local/apache2/htdocs/

EXPOSE 80
```

This is a very simple Dockerfile, and we will break down the components:

FROM – The FROM command specifies a pre-defined image pulled from the Docker Hub. In this case it is the Apache HTTP Web server (httpd), with tag (version) 2.4.

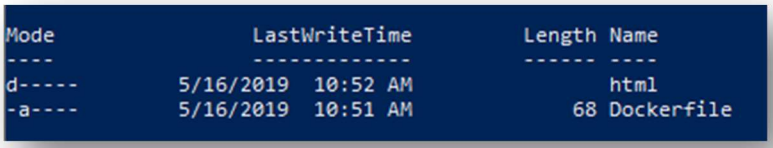
COPY – the COPY command will copy the directory where the application is locally housed to the working directory for the Apache web server

EXPOSE – The EXPOSE command informs Docker that the container listens on the specified port (80 in this case) at runtime.

There are many more build commands available based on the needs of the application, and they can be found here.

Build the Docker Image

Navigate to the working directory where the source application files and Dockerfile are located.

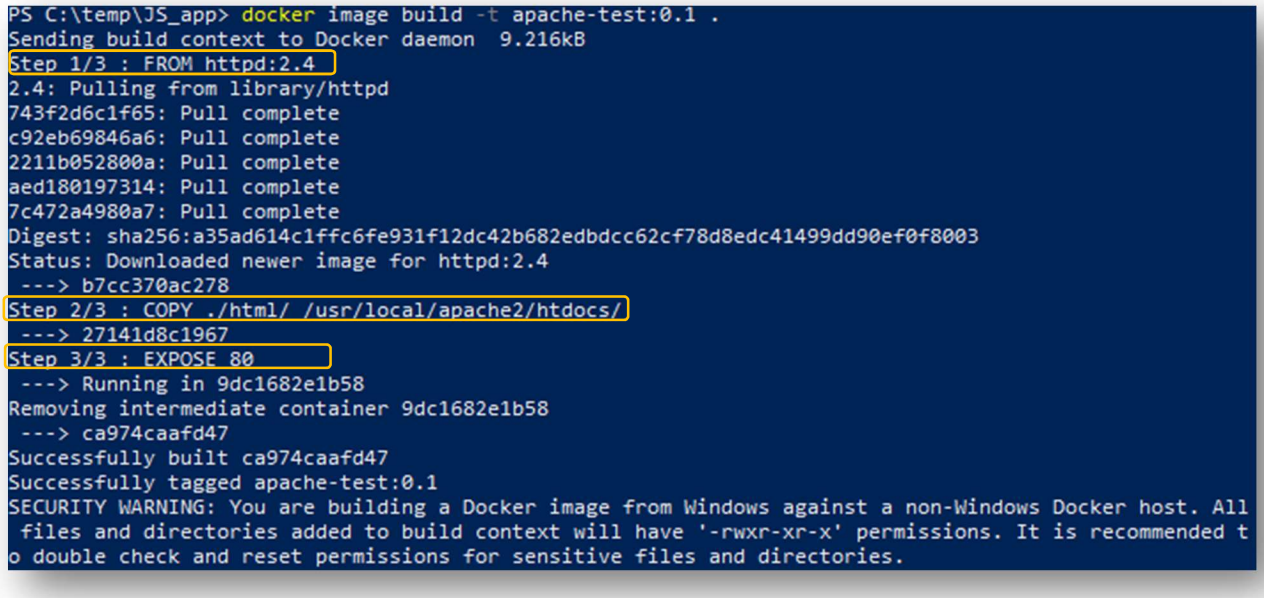


The following command can then be run to build the docker image:

```
docker image build -t apache-test:0.1 .
```

The "-t" option signifies the "name:tag" format; in this case the name is "apache-test" and the "tag" is 0.1. If no tag is defined, it will default to "latest". Assigning actual tag values will help keep track of versions and revisions. There are other build options available as well. The trailing "." signifies that the Dockerfile is located in the current directory.

The output of the build command will look like the following:



Step 1 - Pull the Apache Web server image from Docker Hub and extract it.

Step 2 - Copy the files to the working directory

Step 3 - Expose port 80

Verify the images by running:

```
Docker images
```

```
PS C:\temp\JS_app> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
apache-test         0.1                 ca974caafd47        2 minutes ago       132MB
httpd               2.4                 b7cc370ac278        2 weeks ago         132MB
```

This shows that the Apache Web server image was pulled down, and the new apache-test image was created.

[Optional] Store image in container registry (GCR, Docker Hub, AWS, Azure)

For development purposes, the image created can be stored on your local machine. However, if this needs to be shared (to be used by other team members, shared across environments, or promoted through the dev/test/prod lifecycle), the image can be stored in a central repository such as Docker Hub, Google Container Registry, AWS, or Azure. For this case, the image will be stored in Google Container Registry (gcr.io) via the Google Cloud SDK Shell.

1. Authenticate into the registry

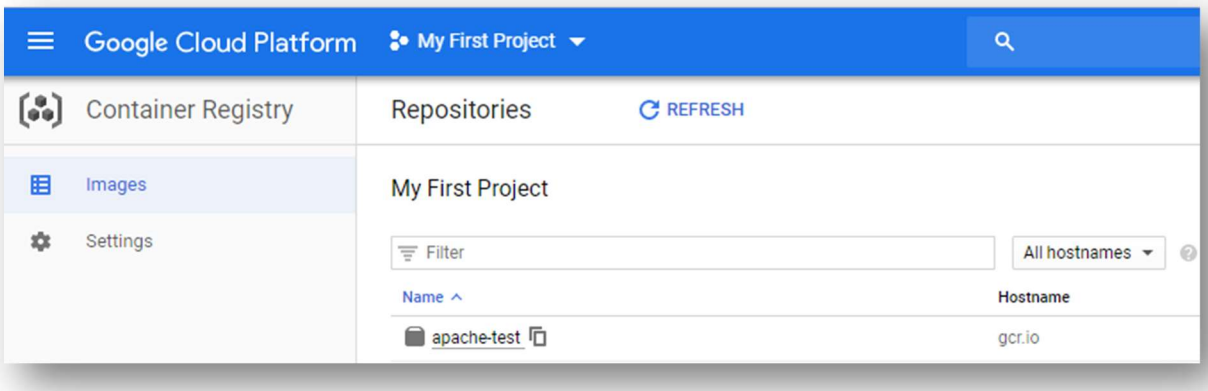
```
gcloud auth configure-docker
```

2. Tag the image with the registry name:

```
docker tag apache-test:0.1 gcr.io/<project ID>/apache-test:0.1
```

3. Push the image to the registry:

```
docker push gcr.io/<project ID>/apache-test:0.1
```



4. To pull the image back down to another environment, re-authenticate, and then run:

```
docker pull gcr.io/<project ID>/apache-test:0.1
```

Run the Image

Now that the image is built, it can be run inside an isolated container. This container has its own file system, networking, and process tree separate from the host. The command to run the container:

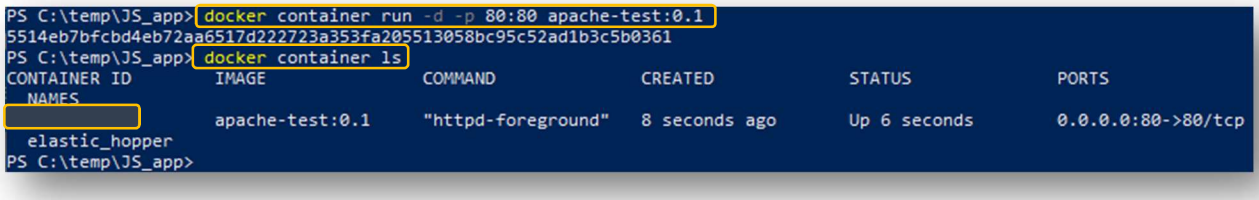
```
docker container run -d -p 80:80 apache-test:0.1
```

Breaking down the command:

-d – signifies to run in detached mode

-p – publishes the containers ports to the host (in this case port 80 on the container maps to port 80 on the host)

After running the command, the `docker container ls` command will show the available containers:



Navigating to "Localhost:80" on your browser will bring up the application running in the container! NOTE: this container has not been exposed to the internet, so only internal requests (localhost, cURL, Wget, etc.) will work to access the app.

To stop the container:

```
Docker stop <container ID>
```

Update the Image

As development progresses, if there are changes to the application, the process to update the image is:

1. Build new image version

```
docker image build -t apache-test:0.2 .
```

NOTE: since the Apache image was the same as before, docker used a cached version, and only updated the application files

```
PS C:\temp\JS_app> docker image build -t apache-test:0.2 .
Sending build context to Docker daemon  9.216kB
Step 1/3 : FROM httpd:2.4
 ---> b7cc370ac278
Step 2/3 : COPY ./html/ /usr/local/apache2/htdocs/
 ---> 1358c201a2a9
Step 3/3 : EXPOSE 80
 ---> Running in 0da0c1c51cce
Removing intermediate container 0da0c1c51cce
 ---> 1ad12d13a335
Successfully built 1ad12d13a335
Successfully tagged apache-test:0.2
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and director
ies added to build context will have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions f
or sensitive files and directories.
```

2. Run **`docker image ls`** command to verify images

```
PS C:\temp\JS_app> docker image ls
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
apache-test                         0.2                 1ad12d13a335        4 minutes ago       132MB
apache-test                         0.1                 ca974caafd47        20 hours ago        132MB
```

3. To run the new image in a container, specify the newer tag:

```
docker container run -d -p 80:80 apache-test:0.2
```

About OnPoint

OnPoint Consulting, Inc. (OnPoint) delivers secure IT infrastructure, enterprise systems, cybersecurity and program management solutions for the U.S. federal government. Our specialized strategy, cyber and technology capabilities are changing the way our clients improve performance, effectively deliver results and manage risk. OnPoint holds ISO 9001:2015, ISO 20000-1:2011, ISO 27001:2013 certifications and a CMMI Maturity Level 3 rating.

OnPoint is a part of the Publicis Sapient platform, with access to industry leading AI tools and teams. Contact us at innovation@onpointcorp.com or visit onpointcorp.com to learn more about us and our services.