Docker Images

Lab 2 – Registries And Containers As Services

In this lab you will get a chance to work with and explore Docker registries and try running containers as networked
services. Registries are systems for saving, looking up and retrieving Docker images. Docker Hub is the default
registry. Private registries can also be used to save and retrieve images within an organization. In the steps below you
will run a private registry server and push/pull images to/from it.

Containers can act as networked services. This is one of the most common uses for containers, enabling users to reliably
deploy services in their own encapsulated containers. To run containers as networked services we will need to learn how
to run containers as background daemon processes as well as how to map ports from the host interface to the services
listening within containers.

1. Searching For images

Docker supplies several subcommands to interact with registries.

- login
- logout
- search
- push
- pull

The `search` subcommand provides a convenient way to quickly lookup Docker Hub repositories. Repositories in Docker act
like folders containing images. Images within a repository are assigned tags, which are simply arbitrary names used to
identify the various images within a repository.

You can perform term searches on Docker Hub to locate repositories with the term in the repository name or description.
For example to search for Apache Thrift images you might use the following command:

```
user@ubuntu:~$ docker search thrift

NAME                                   DESCRIPTION
thrift                                 Thrift is a framework for generating clien...
evarga/thrift                          This is a Docker image for Apache Thrift. ...
randyabernethy/thrift-book             Companion container image for the Programm...
whiteworld/thrift
apache/thrift                          Apache Thrift
itzg/thrift                            Provides the Apache Thrift generator tool
bufferoverflow/thrift                  Apache Thrift - *make cross*
thrift/thrift-compiler-test            thrift-compiler test run to verify https:/...
cjmay/thrift                           thrift builds
nsuke/thrift                           Temporary docker images for Apache Thrift CI
lulichn/thrift                         Thrift
jimdo/thrift                           Mirror of Apache Thrift
thrift/thrift-compiler                 Apache Thrift Compiler
thrift/ubuntu
iwan0/hbase-thrift-standalone          hbase-thrift-standalone
cjmay/thrift-4042                      MWE of THRIFT-4042 (egg extraction error)
thrift/debian                          build/docker/debian
zezuladp/thrift
thrift/thrift-build                    Apache Thrift build environment.
rzhilkibaev/spark-standalone-thrift    Standalone Spark with Thrift
andyfactual/jenkins-dind-thrift        Jenkins server with thrift compiler installed
reecerobinson/spark-thrift-server      Docker container based on spark-nosql that...
vanoise29/thrift                       Thrift App
zhoumingjun/thrift
masgari/thrift

user@ubuntu:~$
```

Try some of your own searches.

- Locate some mongodb images
- Display help on the docker search command at the command line
- Rerun the last search but use the *--limit* switch to display only the top 5 matches
- Locate some mysql images
- Rerun the last search with the following switch: `--filter is-automated=true`
- Locate some postgres images
- Rerun the last search witht he following switch: `--filter is-official=true`
- Locate some redis images
- rerun the last search with the following switch: `--filter stars=5`

Use a browser to navigate to https://hub.docker.com and lookup one of the repositories you searched for on the command
line.

2. Running a Private Registry

Setting up and running a simple service on a normal Linux system is often a daunting task. Even setting up a simple web
server can take more time than you would like to spend; install packages, mess with config files, fix dependencies,
test, change things some more, etc. Docker can reduce or even eliminate this repetitive work. Run the following command
to start a registry server on your Docker host:

```
user@ubuntu:~$ docker container run -p 5000:5000 -d registry:2

Unable to find image 'registry:2' locally
2: Pulling from library/registry
709515475419: Pull complete
df6e278d8f96: Pull complete
16218e264e88: Pull complete
16748da81f63: Pull complete
8d73e673c34c: Pull complete
Digest: sha256:28be0609f90ef53e86e1872a11d672434ce1361711760cf1fe059efd222f8d37
Status: Downloaded newer image for registry:2
7b8d0dc8c5f09a1076a586491d18863b94a49264d95f99126f5f074912653034

user@ubuntu:~$
```

Show the run time information:

```
user@ubuntu:~$ docker container ls

CONTAINER ID        IMAGE               COMMAND                  CREATED
7b8d0dc8c5f0        registry:2          "/entrypoint.sh /e..."   About a minute ago

user@ubuntu:~$
```

One Docker command and seconds later you have a running Docker registry server. The container almost always works
because it brings its own configuration environment with it.

- Which Linux distribution is the registry container using?
- What support libraries is the registry container using?
- Which language is the registry service written in?

You may not know the answers to these questions. The great thing about containers is that if the answers don't matter to
you, you don't need to know! The lion's share of the burden of service configuration goes away when you use containers.
The people who designed the service have already configured it optimally in the image (registry:2) you used to launch
the container. The only things we need to configure when we run a container are the external aspects of the service,
which programs outside the container will make use of, like the network port to use on the host.

The registry server runs on port 5000 by default in the container, however only software running on the Docker host can
reach the container through the container network. The Docker run command we used provided the `-p` switch to map the
container port 5000 to the host port 5000 so that we can contact the registry from the outside world. The `-d` switch ran
the container detached (in the background as a daemon).

We ran the registry:2 image. In a repository the "latest" tag need not actually point to the image housing the most
recent version. For example, in the "registry" repository the "latest" tag pointed to a v1 era registry for quite some
time for backwards compatibility. Docker version 1.6 added support for the v2 registry API with parallel image download
support, not backwards compatible with the v1 API. The new Docker registry v2 service is written in Go and has since
eclipsed the old v1 registry server.

Lookup the official "registry" repository on Docker Hub.

- How many images are available through the "registry" repository?
- How many tags are defined in the "registry" repository?
- Which tag identifies the newest image in the repository?
- Which image does the "latest" tag refer to?
- Which OS Distribution and Version are the "registry" images based on?

The registry service, like most Docker services, exposes its interface using a REST API. We can use the `curl` or `wget`
tools to test our registry service:

```
user@ubuntu:~$ wget -v0- localhost:5000
```

```
--2017-03-07 22:05:46--  http://localhost:5000/
Resolving localhost (localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:5000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 0 [text/plain]
Saving to: 'STDOUT'

-                                       [ <=>                                ]       0  --.

2017-03-07 22:05:46 (0.00 B/s) - written to stdout [0/0]

user@ubuntu:~$
```

Note that because we have mapped port 5000 on the host (localhost in this case) to port 5000 in the container, we do not
need to know the IP address of the container on the container network.

3. Use the Registry

Imagine we want to create a container to do development work in. We use Windows at work, OSX at home, and three flavors
of Linux in production. To simplify our lives we could create a Linux dev environment container with all of our tools
installed that we could use everywhere.

Create a simple dev container with the following commands:

```
user@ubuntu:~$ docker container run -it ubuntu:14.04 /bin/bash

Unable to find image 'ubuntu:14.04' locally
14.04: Pulling from library/ubuntu
30d541b48fc0: Pull complete
8ecd7f80d390: Pull complete
46ec9927bb81: Pull complete
2e67a4d67b44: Pull complete
7d9dd9155488: Pull complete
Digest: sha256:62a5dce5ceccd7f1cb2672a571ebee52cad1f08eec9b57fe4965fb0968a9602e
Status: Downloaded newer image for ubuntu:14.04

root@c12ad2caab30:/# apt-get update
...

root@c12ad2caab30:/# apt-get install -y git
...

root@c12ad2caab30:/# exit
```

```
exit

user@ubuntu:~$
```

This is of course a trivial example, our container includes an Ubuntu 14.04 base and the Git version control system. In
a real scenario we might install many other tools (valgrind, cmake, g++, java, maven, ant, scala, Play, ruby, Rails,
python 2/3, Sinatra, etc.)

Now that we have our example dev container prepared let's create an image from the container that we can launch over and
over again. Execute the following `commit` subcommand to create an image from your container (make sure to substitute
the ID of your container):

```
user@ubuntu:~$ docker container commit c12a mydev/devenv

sha256:393440c87ab49d14b617199b95fffbfd9bb89166eed8ea4464c2ffd15500fe84

user@ubuntu:~$
```

Now display the image with the `image ls` subcommand:

```
user@ubuntu:~$ docker image ls mydev/devenv

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZ
mydev/devenv        latest              393440c87ab4        7 seconds ago       248

user@ubuntu:~$
```

We can push our devenv image to a registry to make it easy to access from any other Docker system on the network. Our
image has a userId/repository formatted name. Pushing such a repository image will cause it to be sent to the Docker
Hub. To push the image to a local repository we need to change the first part of the repository string to represent the
URL for the local repository. Fortunately we can create multiple names for a single image ID.

Use the `tag` subcommand to add a second name for the image created above and display

the results:

```
user@ubuntu:~$ docker image tag mydev/devenv localhost:5000/devenv

user@ubuntu:~$ docker image ls | grep devenv

localhost:5000/devenv   latest              393440c87ab4        About a minute ago
mydev/devenv            latest              393440c87ab4        About a minute ago

user@ubuntu:~$
```

Use the following command to push the image to your local registry:

```
user@ubuntu:~$ docker image push localhost:5000/devenv

The push refers to a repository [localhost:5000/devenv]
fd5ec982e7c8: Pushed
bd00cdbae641: Pushed
af43131c4039: Pushed
9bd4c7af882a: Pushed
04ab82f865cf: Pushed
c29b5eadf94a: Pushed
latest: digest: sha256:3083b0d5d114200b6a17cdcb9d5a301086440a3b1aff420457c639a54ece

user@ubuntu:~$
```

4. Pull an Image From the Registry

To fully test our registry we should try to pull images from it. Remove the two image names you created above:

```
user@ubuntu:~$ docker image rm mydev/devenv

Untagged: mydev/devenv:latest

user@ubuntu:~$ docker image rm localhost:5000/devenv

Untagged: localhost:5000/devenv:latest
Untagged: localhost:5000/devenv@sha256:3083b0d5d114200b6a17cdcb9d5a301086440a3b1aff
Deleted: sha256:393440c87ab49d14b617199b95fffbfd9bb89166eed8ea4464c2ffd15500fe84
Deleted: sha256:8b273f0f377c7974ef298e0a34c47401fc7eae9f9665bb4f11d17da95b210c2e

user@ubuntu:~$
```

Because both tags referred to the same image only the second `image rm` subcommand actually deleted the image. In the
example above two images are deleted. Under certain circumstances Docker creates unnamed intermediate images. When you
delete a named image Docker automatically removes any unused unnamed intermediate images in the ancestry.

Run a `docker image ls` command to be sure the images have been removed. Now try to pull your image back down to the
Docker host from the registry:

```
user@ubuntu:~$ docker image ls | grep devenv

user@ubuntu:~$ docker image pull localhost:5000/devenv

Using default tag: latest
latest: Pulling from devenv
d24bc6f195c3: Already exists
a994fca0b2a3: Already exists
958046a70c9d: Already exists
0e4fa347e95d: Already exists
2f0af262ade7: Already exists
4b7dcc92c295: Pull complete
Digest: sha256:3083b0d5d114200b6a17cdcb9d5a301086440a3b1aff420457c639a54ecedcef
Status: Downloaded newer image for localhost:5000/devenv:latest

user@ubuntu:~$
```

Success! List your local images to verify the download:

```
user@ubuntu:~$ docker image ls | grep devenv

localhost:5000/devenv   latest              393440c87ab4        4 minutes ago

user@ubuntu:~$
```

- Is the image you downloaded from the registry the same exact set of bits that you had before?
- What makes you think so?

Congratulations, you have completed the registry lab!

[OPTIONAL] Docker Registry TLS

The Docker daemon does not trust registries not found on the localhost. For example if you were to try to push your
image in the lab steps above to a host other than localhost it would fail. For example:

```
user@ubuntu:~$ ip a | grep "global e"

    inet 192.168.131.199/24 brd 192.168.131.255 scope global ens33

user@ubuntu:~$ docker image tag localhost:5000/devenv 192.168.131.144:5000/devenv

user@ubuntu:~$ docker image push 192.168.131.199:5000/devenv
The push refers to a repository [192.168.131.199:5000/devenv]
Get https://192.168.131.199:5000/v1/_ping: http: server gave HTTP response to HTTPS
```

In the example above, we discover our host's IP address and then tag an image to push to the IP rather than "localhost".
The Docker daemon accepts the push request but attempts to use HTTPS to communicate with the registry (which looks like
a remote system identified by IP address). The registry service we ran was given no keys or certificates to use for TLS
and does not support the request.

There are two way to address this situation. One way is to configure the registry server to support TLS. The registry
server we are running provides TLS support, but each registry server is different and requires different configuration
steps.

The second approach is to tell the Docker daemon to trust this particular registry host, allowing HTTP without TLS. The
Docker daemon configuration file supports passing command line arguments to the Docker daemon. Adding an switch like the
following in our Docker daemon configuration file would allow an exception for the host referred to with
192.168.131.199:

`--insecure-registry 192.168.131.199:5000`

Modify your Docker daemon configuration to allow insecure access to your local registry IP address. First discover the
location of the systemd configuration file used by the Docker service:

```
user@ubuntu:~$ systemctl show --property=FragmentPath docker

FragmentPath=/lib/systemd/system/docker.service
```

Now add the insecure registry exception switch for your registry service to the ExecStart line:

```
user@ubuntu:~$ sudo vi /lib/systemd/system/docker.service

user@ubuntu:~$ grep ExecStart /lib/systemd/system/docker.service

ExecStart=/usr/bin/dockerd -H fd:// --insecure-registry 192.168.131.199:5000
```

Now we need to tell SystemD to reload the new configuration then we need to restart the Docker daemon:

```
user@ubuntu:~$ sudo systemctl daemon-reload

user@ubuntu:~$ sudo systemctl restart docker

user@ubuntu:~$
```

Restarting, Docker kills our previous registry service so now we must run it again before we can push to it:

```
user@ubuntu:~$ docker container run -p 5000:5000 -d registry:2

399e9c55ccc90949da6472446a46a034b118c07613275d1e3f301dea06419b7c
```

Now try your push again with insecure access allowed:

```
user@ubuntu:~$ docker image push 192.168.131.199:5000/devenv

The push refers to a repository [192.168.131.199:5000/devenv]
```

```
6320b08437a5: Pushed
5f70bf18a086: Pushed
0d81735d8272: Pushed
982549bd6b32: Pushed
8698b31c92d5: Pushed
latest: digest: sha256:881711599c80fcf7fdf1aba50d3841f38b5d9e2665598e0a258737ebdb93
```

You can also tell the Docker daemon to trust a secure registry by supplying the daemon with the CA certificate which was
used to sign the registry server's certificate. For example, if you wanted to connect to a registry server with the
hostname registry.example.com you would supply its CA certificate here:

`/etc/docker/certs.d/registry.example.com:5000/ca.crt`

Congratulations, you have completed the optional Docker registry lab!