RX-M

Docker Images

Lab – 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 commands to interact with registries.

- login
- logout
- search
- push
- pull

The search command 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                                     STARS     OFFICIAL
AUTOMATED
thrift                          Thrift is a framework for generating clien...   44        [OK]
evarga/thrift                   This is a Docker image for Apache Thrift. ...   4
[OK]
whiteworld/thrift                                                               1
[OK]
randyabernethy/thrift-book      Companion container image for the Programm...   1
[OK]
iwan0/hbase-thrift-standalone   hbase-thrift-standalone                         0
[OK]
...
```

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:

```
docker run -p 5000:5000 -d registry:2
```

```
user@ubuntu:~$ docker run -p 5000:5000 -d registry:2
Unable to find image 'registry:2' locally
2: Pulling from library/registry

e110a4a17941: Pull complete
2ee5ed28ffa7: Pull complete
d1562c23a8aa: Pull complete
06ba8e23299f: Pull complete
802d2a9c64e8: Pull complete
Digest: sha256:1b68f0d54837c356e353efb04472bc0c9a60ae1c8178c9ce076b01d2930bcc5d
Status: Downloaded newer image for registry:2
2cc5047801439513d83ebf1cc80f933799e925b7dbe3273567bd3e081639065d
```

```
user@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                    NAMES
2cc504780143        registry:2          "/entrypoint.sh /etc/"   14 seconds ago      Up 13 seconds
0.0.0.0:5000->5000/tcp   reverent_mcnulty
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.

- What Linux distribution is the registry container using?
- What support libraries is the registry container using?
- What 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 -vO - localhost:5000
--2015-11-04 12:29:48--  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           --.-
K/s   in 0s

2015-11-04 12:29:48 (0.00 B/s) - written to stdout [0/0]
```

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 run -it ubuntu:14.04 /bin/bash
Unable to find image 'ubuntu:14.04' locally
14.04: Pulling from library/ubuntu
8387d9ff0016: Already exists
3b52deaaf0ed: Already exists
4bd501fad6de: Already exists
a3ed95caeb02: Already exists
Digest: sha256:0844055d30c0cad5ac58097597a94640b0102f47d6fa972c94b7c129d87a44b7
Status: Downloaded newer image for ubuntu:14.04

root@2782eab14d8d:/# apt-get update
...

root@2782eab14d8d:/# apt-get install -y git
...

root@2782eab14d8d:/# 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 command to create an image from your container (make sure to substitute the ID of your container):

```
user@ubuntu:~$ docker commit 2782 mydev/devenv
sha256:76ebce1f9fc6f867ffda86bfd9a9c4e2a4e1294ac70d7422029aa1bd70efd73c
```

Now display the image with the images command:

```
user@ubuntu:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mydev/devenv        latest              76ebce1f9fc6        22 seconds ago      225.6 MB
lab/websvr          v0.2                99def8f04893        3 hours ago         298.5 MB
lab/websvr          latest              4d336053cd17        4 hours ago         298.5 MB
registry            2                   94d4bcbaa8d6        11 days ago         165.7 MB
ubuntu              14.04               3876b81b5a81        3 weeks ago         187.9 MB
ubuntu              latest              3876b81b5a81        3 weeks ago         187.9 MB
ubuntu              12.04               8b1548cecef1        3 weeks ago         137.5 MB
fedora              rawhide             e3cae2cffbcd        6 weeks ago         227.9 MB
centos              latest              61b442687d68        7 weeks ago         196.6 MB
hello-world         latest              690ed74de00f        4 months ago        960 B
centos              6                   55d1ee70345b        4 months ago        190.6 MB
```

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 command to add a second name for the image created above and display the results:

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

user@ubuntu:~$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED              SIZE
localhost:5000/devenv   latest              76ebce1f9fc6        About a minute ago   225.6 MB
mydev/devenv            latest              76ebce1f9fc6        About a minute ago   225.6 MB
...
```

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

```
user@ubuntu:~$ docker push localhost:5000/devenv
The push refers to a repository [localhost:5000/devenv]
6320b08437a5: Pushed
5f70bf18a086: Pushed
0d81735d8272: Pushed
982549bd6b32: Pushed
```

```
8698b31c92d5: Pushed
latest: digest: sha256:881711599c80fcf7fdf1aba50d3841f38b5d9e2665598e0a258737ebdb93dfd9 size: 1341
```

4. Pull 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 rmi mydev/devenv
Untagged: mydev/devenv:latest

user@ubuntu:~$ docker rmi localhost:5000/devenv
Untagged: localhost:5000/devenv:latest
Deleted: sha256:76ebce1f9fc6f867ffda86bfd9a9c4e2a4e1294ac70d7422029aa1bd70efd73c
Deleted: sha256:0ad8b48c6a42d2a16a1511131fc95dcfcbce0849fb5335e2c28600337e47d3e5
```

Because both tags referred to the same image only the second rmi command 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 images` 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 pull localhost:5000/devenv
Using default tag: latest
latest: Pulling from devenv
8387d9ff0016: Already exists
3b52deaaf0ed: Already exists
4bd501fad6de: Already exists
a3ed95caeb02: Already exists
80b3e4a5fe5d: Pull complete
Digest: sha256:881711599c80fcf7fdf1aba50d3841f38b5d9e2665598e0a258737ebdb93dfd9
Status: Downloaded newer image for localhost:5000/devenv:latest
```

Success! List your local images to verify the download:

```
user@ubuntu:~$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
localhost:5000/devenv   latest              76ebce1f9fc6        14 minutes ago      225.6 MB
...
```

- 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 tag localhost:5000/devenv 192.168.131.144:5000/devenv

user@ubuntu:~$ docker 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 client
```

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 192.168.131.144:

```
--insecure-registry 192.168.131.144:5000
```

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 run -p 5000:5000 -d registry:2
399e9c55ccc90949da6472446a46a034b118c07613275d1e3f301dea06419b7c
```

Now try your push again with insecure access allowed:

```
user@ubuntu:~$ docker 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: sha256:881711599c80fcf7fdf1aba50d3841f38b5d9e2665598e0a258737ebdb93dfd9 size: 1341
```

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!