Deploying a Registry Server

Objective: This is a short and fairly simple lab. The objective is to show you how we can set up our own registry, much like our local registry, secure it and push and pull images to and from.<br>

Preparation: Navigate to the apprororirate lab folder and then open a two terminal windows.<br>

Outcome: Students will set up a stand alone registry and even have the choice to establish one from a custom domain of their own.<br>

Data Files: `docker-compose.yaml`

A registry is a service for storing and accessing Docker images. Docker Cloud and Docker Store are the best-known hosted registries, which you can use to store public and private images. You can also run your own registry using the open-source Docker Registry, which is a Go application in a Alpine Linux container.

What You Will Learn

- run a local registry in a container and configure your Docker engine to use the registry;
- generate SSL certificates (using Docker!) and run a secure local registry with a friendly domain name;
- generate encrypted passwords (using Docker!) and run an authenticated, secure local registry over HTTPS with basic auth.

> *NOTE: The open-source registry does not have a Web UI, so there's no friendly interface like Docker Cloud or Docker Store. Instead there is a REST API you can use to query the registry. For a local registry which has a Web UI and role-based access control, Docker, Inc. has the Trusted Registry product.*

Part 1 - The Basics of Docker Registry

Step 1. Running a Registry Container in Linux

There are several ways to run a registry container. The simplest is to run an insecure registry over HTTP, but for that we need to configure Docker to explicitly allow insecure access to the registry.

Docker expects all registries to run on HTTPS. The next section of this lab will introduce a secure version of our registry container, but for this part of the tutorial we will run a version on HTTP. When registering a image, Docker returns an error message like this:

```
http: server gave HTTP response to HTTPS client
```

1. The Docker Engine needs to be explicitly setup to use HTTP for the insecure registry. Edit or create `/etc/docker/docker` file:

```
$ vi /etc/docker/docker
```

2. Now, add this line:

```
DOCKER_OPTS="--insecure-registry localhost:5000"
```

3. Close and save the file, then restart the docker daemon.

```
$ service docker restart
```

> *NOTE: In **Docker for Mac**, the `Preferences` menu lets you set the address for an insecure registry under the `Daemon` panel.*
> *In **Docker for Windows**, the `Settings` menu lets you set the address for an insecure registry under the `Daemon` panel:*
>

Step 2. Testing the Registry Image

1. First we'll test that the registry image is working correctly, by running it without any special configuration:

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

Step 3. Understanding Image Names

1. Typically we work with images from the Docker Store, which is the default registry for the Docker Engine. Commands using just the image repository name work fine, like this:

```
$ docker pull hello-world
```

> *NOTE: Remember, `hello-world` is the repository name, which we are using as a short form of the full image name. The full name is `docker.io/hello-world:latest`. That breaks down into three parts:*

- `docker.io` - the hostname of the registry which stores the image;
- `hello-world` - the repository name, in this case in `{imageName}` format;
- `latest` - the image tag.
  If a tag isn't specified, then the default `latest` is used. If a registry hostname isn't specified then the default `docker.io` for Docker Store is used. If you want to use images with any other registry, you need to explicitly specify the hostname - the default is always Docker Store, you can't change to a different default registry.

2. With a local registry, the hostname and the custom port used by the registry is the full registry address, e.g. `localhost:5000`.

```
$ hostname
```

Step 4. Pushing and Pulling from the Local Registry

Docker uses the hostname from the full image name to determine which registry to use. We can build images and include the local registry hostname in the image tag, or use the `docker tag` command to add a new tag to an existing image.

1. These commands pull a public image from Docker Store, tag it for use in the private registry with the full name `localhost:5000/hello-world`, and then push it to the registry:

```
$ docker tag hello-world localhost:5000/hello-world
$ docker push localhost:5000/hello-world
```

2. When you push the image to your local registry, you'll see similar output to when you push a public image to the Hub:

```
The push refers to a repository [localhost:5000/hello-world]
a55ad2cda2bf: Pushed
cfbe7916c207: Pushed
fe4c16cbf7a4: Pushed
latest: digest: sha256:79e028398829da5ce98799e733bf04ac2ee399
79b238e4b358e321ec549da5d6 size: 948
```

3. On the local machine, you can remove the new image tag and the original image, and pull it again from the local registry to verify it was correctly stored:

```
$ docker rmi localhost:5000/hello-world
$ docker rmi hello-world
$ docker pull localhost:5000/hello-world
```

That exercise shows the registry works correctly, but at the moment it's not very useful because all the image data is stored in the container's writable storage area, which will be lost when the container is removed. To store the data outside of the container, we need to mount a host directory when we start the container.

Step 5. Running a Registry Container with External Storage

1. Remove the existing registry container by removing the container which holds the storage layer. Any images pushed will be deleted:

```
$  docker kill registry
$  docker rm registry
```

In this example, the new container will use a host-mounted Docker volume. When the registry server in the container writes image layer data, it appears to be writing to a local directory in the container but it will be writing to a directory on the host.

2. Create the registry:

```
$  mkdir registry-data
$  docker run -d -p 5000:5000 \
--name registry \
-v `pwd`/registry-data:/var/lib/registry \
registry:2
```

3. Tag and push the container with the new IP address of the registry.

```
docker tag hello-world localhost:5000/hello-world
docker push localhost:5000/hello-world
```

4. Repeating the previous `docker push` command uploads an image to the registry container, and the layers will be stored in the container's `/var/lib/registry` directory, which is actually mapped to the `$(pwd)/registry-data` directory on you local machine. The `tree` command will show the directory structure the registry server uses:

```
$ tree registry-data
```

Output:

```
.
|____docker
| |____registry
| | |____v2
| | | |____blobs
| | | | |____sha256
| | | | | |____1f
| | | | | | |____1fad42e8a0d9781677d366b1100defcadbe653280300
cf62a23e07eb5e9d3a41
...
```

Storing data outside of the container means we can build a new version of the registry image and replace the old container with a new one using the same host mapping - so the new registry container has all the images stored by the previous container.

Using an insecure registry also isn’t practical in multi-user scenarios. Effectively there’s no security so anyone can push and pull images if they know the registry hostname. The registry server supports authentication, but only over a secure SSL connection. We’ll run a secure version of the registry server in a container next.

Part 2 - Running a Secured Registry Container in Linux

We saw how to run a simple registry container in Part 1, using the official Docker registry image. The registry server con be configured to serve HTTPS traffic on a known domain, so it’s straightforward to run a secure registry for private use with a self-signed SSL certificate.

Step 1. Generating the SSL Certificate in Linux

1. The Docker docs explain how to generate a self-signed certificate on Linux using OpenSSL:

```
$ mkdir -p certs
$ openssl req \
  -newkey rsa:4096 -nodes -sha256 -keyout certs/domain.key \

  -x509 -days 365 -out certs/domain.crt
```

Output:

```
Generating a 4096 bit RSA private key
........++
.............................................................+
+
writing new private key to 'certs/domain.key'
-----
You are about to be asked to enter information that will be i
ncorporated
into your certificate request.
What you are about to enter is what is called a Distinguished
 Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Do
cker
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:localhost
Email Address []:
```

If you are running the registry locally, be sure to use your host name as the

CN.

2. To get the docker daemon to trust the certificate, copy the `domain.crt` file.

```
$ sudo su
$ mkdir /etc/docker/certs.d
$ mkdir /etc/docker/certs.d/<localhost>:5000
$ cp `pwd`/certs/domain.crt /etc/docker/certs.d/<localhost>:5
000/ca.crt
```

3. Make sure to restart the docker daemon.

```
$ sudo service docker restart
```

Now we have an SSL certificate and can run a secure registry.

Step 2. Running the Registry Securely

The registry server supports several configuration switches as environment variables, including the details for running securely. We can use the same image we’ve already used, but configured for HTTPS.

1. If you have an insecure registry container still running, remove it:

```
$ docker kill registry
$ docker rm registry
```

2. For the secure registry, we need to run a container which has the

SSL certificate and key files available, which we'll do with an additional volume mount (so we have one volume for registry data, and one for certs). We also need to specify the location of the certificate files, which we'll do with environment variables:

```
$ mkdir registry-data
$ docker run -d -p 5000:5000 --name registry \
  --restart unless-stopped \
  -v $(pwd)/registry-data:/var/lib/registry -v $(pwd)/certs:/
certs \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
  -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
  registry
```

> *The new parts to this command are:*

- `--restart unless-stopped` - restart the container when it exits, unless it has been explicitly stopped. When the host restarts, Docker will start the registry container, so it's always available.
- `-v $pwd\certs:c:\certs` - mount the local `certs` folder into the container, so the registry server can access the certificate and key files;
- `-e REGISTRY_HTTP_TLS_CERTIFICATE` - specify the location of the SSL certificate file;
- `-e REGISTRY_HTTP_TLS_KEY` - specify the location of the SSL key file.

We'll let Docker assign a random IP address to this container, because we'll be accessing it by host name. The registry is running securely now, but we’ve used a self-signed certificate for an internal domain name.

Step 3. Accessing the Secure Registry

We’re ready to push an image into our secure registry.

```
$ docker push localhost:5000/hello-world
$ docker pull localhost:5000/hello-world
```

We can go one step further with the open-source registry server, and add basic authentication - so we can require users to securely log in to push and pull images.

Part 3 - Using Basic Authentication with a Secured Registry in Linux

From Part 2 we have a registry running in a Docker container, which we can securely access over HTTPS from any machine in our network. We used a self-signed certificate, which has security implications, but you could buy an SSL from a CA instead, and use that for your registry. With secure communication in place, we can set up user authentication.

Step 1. Usernames and Passwords

The registry server and the Docker client support basic authentication over HTTPS. The server uses a file with a collection of usernames and encrypted passwords. The file uses Apache’s `htpasswd`.

1. Create the password file with an entry for user “moby” with password “gordon”;

```
$ mkdir auth
$ docker run --entrypoint htpasswd registry:latest -Bbn moby
gordon > auth/htpasswd
```

> *The options are:*

- –entrypoint Overwrite the default ENTRYPOINT of the image <<<<<<< HEAD
- -B to force bcrypt vs default md5 =======
- -B Use bcrypt encryption (required)

> *master*

- -b run in batch mode
- -n display results

2. We can verify the entries have been written by checking the file contents - which shows the user names in plain text and a cipher text password:

```
$ cat auth/htpasswd
```

Output:

```
moby:$2y$05$Geu2Z4LN0QDpUJBHvP5JVOsKOLH/XPoJBqISv1D8Aeh6LVGvjWWVC
```

Step 2. Running an Authenticated Secure Registry

Adding authentication to the registry is a similar process to adding SSL - we need to run the registry with access to the `htpasswd` file on the host, and configure authentication using environment variables.

1. As before, we'll remove the existing container and run a new one with authentication configured:

```
$ docker kill registry
$ docker rm registry
$ docker run -d -p 5000:5000 --name registry \
  --restart unless-stopped \
  -v $(pwd)/registry-data:/var/lib/registry \
  -v $(pwd)/certs:/certs \
  -v $(pwd)/auth:/auth \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
  -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
  -e REGISTRY_AUTH=htpasswd \
  -e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" \
  -e "REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd" \
  registry
```

> *The options for this container are:*

- `-v $(pwd)/auth:/auth` - mount the local `auth` folder into the container, so the registry server can access `htpasswd` file;
- `-e REGISTRY_AUTH=htpasswd` - use the registry's `htpasswd` authentication method;
- `-e REGISTRY_AUTH_HTPASSWD_REALM='Registry Realm'` - specify the authentication realm;
- `-e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd` - specify the location of the `htpasswd` file.

Now the registry is using secure transport and user authentication.

Step 3. Authenticating with the Registry

1. With basic authentication, users cannot push or pull from the registry unless they are authenticated. If you try and pull an image without authenticating, you will get an error:

```
$ docker pull localhost:5000/hello-world
```

Output:

```
Using default tag: latest
Error response from daemon: Get https://localhost:5000/v2/hel
lo-world/manifests/latest: no basic auth credentials
```

2. The result is the same for valid and invalid image names, so you

can't even check a repository exists without authenticating.

Logging in to the registry is the same `docker login` command you use for Docker Store, specifying the registry hostname:

```
$ docker login registry.local:5000
```

Output:

```
Username: moby
Password:
Login Succeeded
```

> *NOTE: If you use the wrong password or a username that doesn't exist, you get a `401` error message:*

```
Error response from daemon: login attempt to https://registry
.local:5000/v2/ failed with status: 401 Unauthorized
```

3. Now you're authenticated, you can push and pull as before:

```
$ sudo docker pull localhost:5000/hello-world
```

Output:

```
Using default tag: latest
latest: Pulling from hello-world
Digest: sha256:961497c5ca49dc217a6275d4d64b5e4681dd3b2712d949
```

```
74b8ce4762675720b4
Status: Image is up to date for registry.local:5000/hello-wor
ld:latest
```

> *NOTE: The open-source registry does not support the same authorization model as Docker Store or Docker Trusted Registry. Once you are logged in to the registry, you can push and pull from any repository, there is no restriction to limit specific users to specific repositories.*

Step 4. Using Docker Compose to Start the Registry

1. Typing in all the options to start the registry can become tedious. An easier and simpler way is to use Docker Compose. Here’s an example of a `docker-compose.yml` file that will start the registry. The YAML file has already been created for you. Just look it over and then continue to the next step.<br>

docker-compose.yaml

```
registry:
  restart: always
  image: registry:2
  ports:
    - 5000:5000
  environment:
    REGISTRY_HTTP_TLS_CERTIFICATE: /certs/domain.crt
    REGISTRY_HTTP_TLS_KEY: /certs/domain.key
    REGISTRY_AUTH: htpasswd
    REGISTRY_AUTH_HTPASSWD_PATH: /auth/htpasswd
    REGISTRY_AUTH_HTPASSWD_REALM: Registry Realm
  volumes:
    - /path/registry-data:/var/lib/registry
    - /path/certs:/certs
    - /path/auth:/auth
```

2. To start the registry, type:

```
$ docker-compose up
```

Conclusion

Docker Registry is a free, open-source application for storing and accessing Docker images. You can run the registry in a container on your own network, or in a virtual network in the cloud, to host private images with secure access. For Linux hosts, there is an official registry image on Docker Store.

We've covered all the options, from running an insecure registry, through adding SSL to encrypt traffic, and finally adding basic authentication to restrict access. By now you know how to set up a usable registry in your own environment, and you've also used some key Docker patterns - using containers as build agents and to run basic commands, without having to install software on your host machines.