# Secure Docker Registry With TLS and Authentication

The default registry install is rather bare bones, and is open by default, meaning anyone can push and pull images. You'll likely want to at least add TLS to it so you can work with it easily via HTTPS, and then also add some basic authentication.

These aren't actually that hard to setup, but do require some commands. You can learn the basics by creating a self-signed certificate for HTTPS, and then enabling **htpasswd** auth, which you'll add users too with basic cli commands.

For this assignment you'll use Play With Docker, a great resource for web-based docker testing and also has a library of labs built by Docker Captains and others, and supported by Docker Inc.

I'd like you to do the Part 2 and 3 of "Docker Registry for Linux" for this assignment. You can use their text to do this assignment on your own machine, or jump back to their Part 1 and run the container on their infrastructure using their web-based interface to a real docker engine and learn how "PWD" works!

For more extra credit labs, look through their growing list: http://training.play-with-docker.com/

Refs:

1) https://docs.docker.com/registry/deploying/

2) https://training.play-with-docker.com/linux-registry-part2/

# Docker registry for Linux Part 1

Feb 26, 2017 • @manomarks

# Introduction

A registry is a service for storing and accessing Docker images. Docker Hub 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

You'll learn how to:

- 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 interface like Docker Hub 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.

You'll need Docker running on in this tutorial, or on a Linux machine and be familiar with the key Docker concepts, and with Docker volumes:

- Docker concepts
- Docker volumes

# Part 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
```

The Docker Engine needs to be explicitly setup to use HTTP for the insecure registry. For this sample it has already been done, `127.0.0.1:5000` has already been added to the daemon.

** *Running on your own Linux machine instead of in this browser window* ** Edit or create `/etc/docker/docker` file:

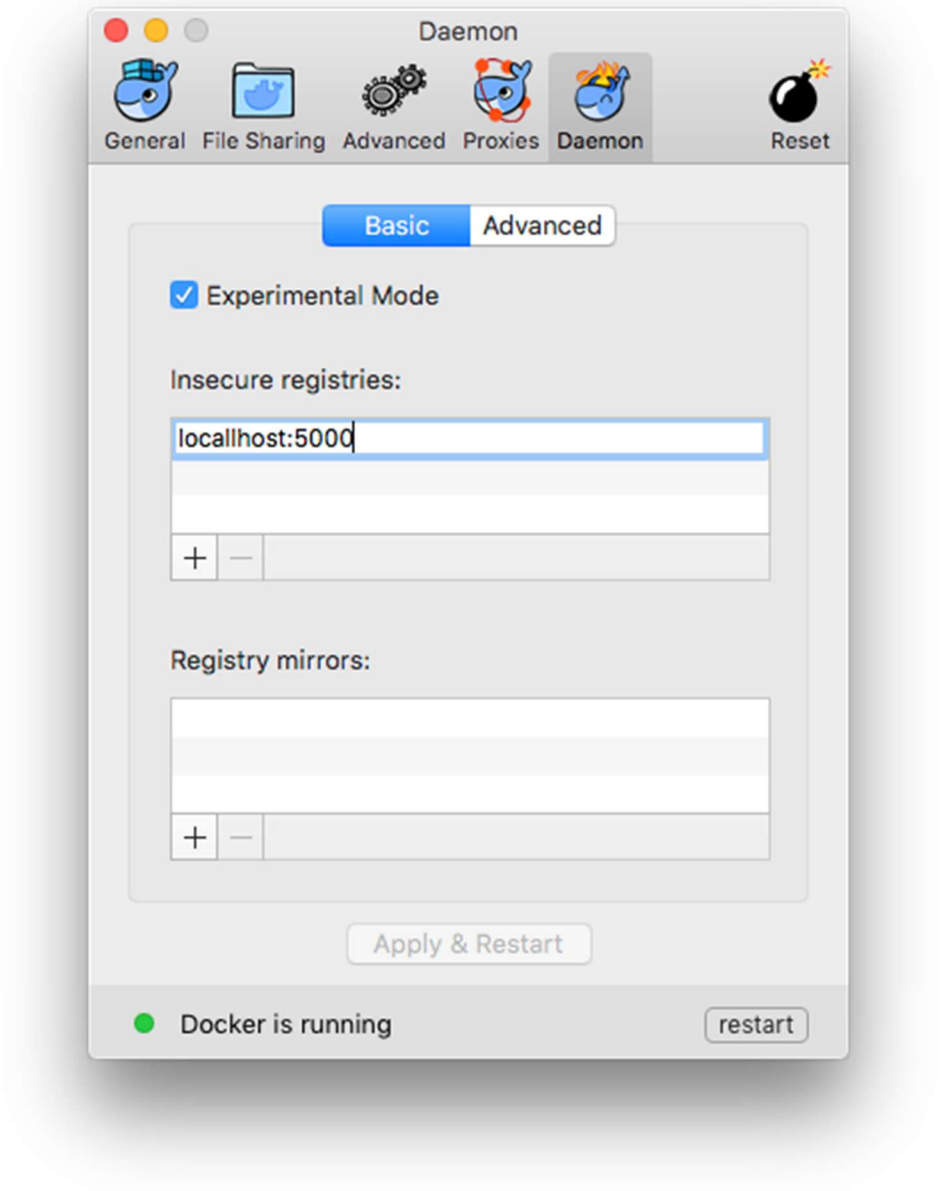
```
vi /etc/docker/docker

# add this line
DOCKER_OPTS="--insecure-registry 127.0.0.1:5000"
```

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

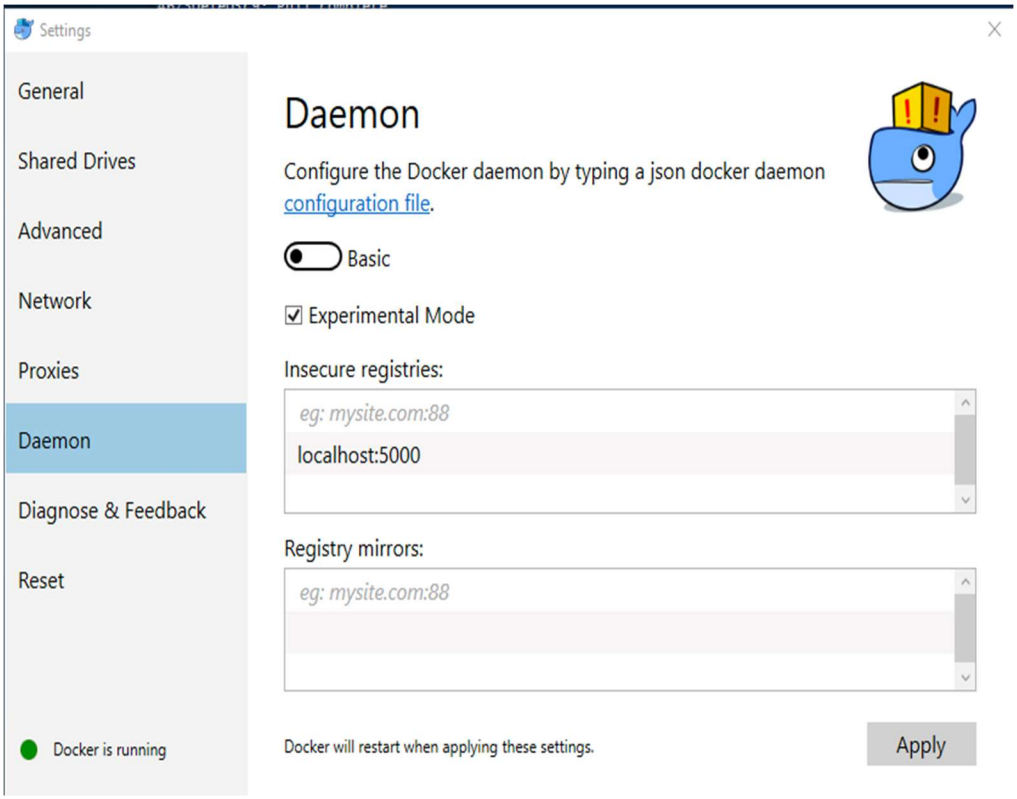
```
pkill dockerd
dockerd > /dev/null 2>&1 &
```

** *If you're running on your own Mac or Windows machine instead of in this browser window* ** 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:




# Testing the Registry Image

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
```

# Understanding Image Names

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

```
docker pull hello-world
```

`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.

With a local registry, the hostname and the custom port used by the registry is the full registry address, e.g. `127.0.0.1:5000`. In this sample we'll just be using `127.0.0.1:5000` as that's already been added to the daemon.

# 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.

These commands pull a public image from Docker Store, tag it for use in the private registry with the full name `127.0.0.1:5000/hello-world`, and then push it to the registry:

```
docker tag hello-world 127.0.0.1:5000/hello-world
docker push 127.0.0.1:5000/hello-world
```

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 [127.0.0.1:5000/hello-world]
a55ad2cda2bf: Pushed
cfbe7916c207: Pushed
fe4c16cbf7a4: Pushed
latest: digest:
sha256:79e028398829da5ce98799e733bf04ac2ee39979b238e4b358e321ec549da5d6 size:
948
```

On your 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 127.0.0.1:5000/hello-world
docker rmi hello-world
docker pull 127.0.0.1:5000/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.

# Running a Registry Container with External Storage

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.

Create the registry:

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

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

```
docker pull hello-world
docker tag hello-world 127.0.0.1:5000/hello-world
docker push 127.0.0.1:5000/hello-world
```

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 your machine. 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 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.

# Docker registry for Linux Parts 2 & 3

Feb 27, 2017 • @manomarks

# 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 can 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.

# Generating the SSL Certificate in Linux

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
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 incorporated
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]:Docker
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:127.0.0.1
Email Address []:
```

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

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

```
mkdir /etc/docker/certs.d
mkdir /etc/docker/certs.d/127.0.0.1:5000
cp $(pwd)/certs/domain.crt /etc/docker/certs.d/127.0.0.1:5000/ca.crt
```

Make sure to restart the docker daemon.

```
pkill dockerd
dockerd > /dev/null 2>&1 &
```

The */dev/null* part is to avoid the output logs from docker daemon.

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

# 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.

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.

## Accessing the Secure Registry

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

```
docker pull hello-world
docker tag hello-world 127.0.0.1:5000/hello-world
docker push 127.0.0.1:5000/hello-world
docker pull 127.0.0.1:5000/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.

*** We have added Part 3 to the end of this section to allow you to continue to use the set-up we have above ***

# 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.

# 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.

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
- -B Use bcrypt encryption (required)
- -b run in batch mode
- -n display results

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

# 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.

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.

# Authenticating with the Registry

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 127.0.0.1:5000/hello-world
Using default tag: latest
Error response from daemon: Get https://127.0.0.1:5000/v2/hello-
world/manifests/latest: no basic auth credentials
```

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 127.0.0.1:5000
Username: moby
Password:
Login Succeeded
```

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
```

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

```
docker pull 127.0.0.1:5000/hello-world
Using default tag: latest
latest: Pulling from hello-world
Digest:
sha256:961497c5ca49dc217a6275d4d64b5e4681dd3b2712d94974b8ce4762675720b4
Status: Image is up to date for registry.local:5000/hello-world: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.

# 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 Hub.

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.

There is still more you can do with Docker Registry - using a different storage driver so the image data is saved to reliable share storage, and setting up your registry as a caching proxy for Docker Store are good next steps.