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](https://docs.docker.com/registry/insecure/#/using-self-signed-certificates) 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](https://en.wikipedia.org/wiki/Basic_access_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](https://docs.docker.com/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](https://hub.docker.com/_/registry/) 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](https://docs.docker.com/registry/storage-drivers/) so the image data is saved to reliable share storage, and setting up your registry as a [caching proxy for Docker Store](https://docs.docker.com/registry/recipes/mirror/) are good next steps.