Deploying a Simple and Secure Docker Registry (http://tech.paulcz.net/2016/01/deploying-a-secure-docker-registry/)

January 10, 2016

There comes a time in everybody's life where they realize they have to run their own Docker Registry (https://www.docker.com/docker-registry). Unfortunately there's not a lot of good information on how to run one. Docker (http://docker.com)'s documentation is pretty good, but is verbose and across a lot of different pages which means having half a dozen tabs open and searching for the right information.

While it's pretty common to run the Docker Registry (https://www.docker.com/docker-registry) itself with little to no security settings and fronting it with NGINX or Apache to provide this security I wanted to show how it can be done with just the Docker (http://docker.com) Registry. If you need to do really clever stuff like authenticate against LDAP then you'll want to go down the reverse proxy road.

This example will demonstrate using just the Docker Registry (https://www.docker.com/docker-registry) itself with both TLS certificate backed encryption and Certificate based endpoint authorization.

For simplicity it will assume a single registry running on the local filesystem and will avoid using OS specific init (systemd/upstart/etc) systems by focusing just on the docker commands themselves. This should work on any system capable of running Docker (http://docker.com).

Preparation

Boot a server that has Docker (http://docker.com) installed. For an OS with Docker (http://docker.com) already installed I recommend CoreOS (http://coreos.com/). However you could just as easily boot Ubuntu or CentOS and run `curl -sSL get.docker.com | sudo bash` if you're into that sort of thing.

SSH into the server and ensure Docker (http://docker.com) is working:

```
$ ssh core@xx.xx.xx.xx
$ docker info
Containers: 0
Images: 0
Server Version: 1.9.1
Storage Driver: overlay
 Backing Filesystem: extfs
Execution Driver: native-0.2
Logging Driver: json-file
Kernel Version: 4.3.3-coreos
Operating System: CoreOS 899.1.0
CPUs: 1
Total Memory: 997.4 MiB
Name: core-01
ID: C5XV:CZ3H:EAO4:ATJ3:ARSO:UOGD:XH3X:UKLZ:V3FO:2LRF:6E3X:CV5K
```

Create Certificates

To keep this as simple as possible I will demonstrate using the paulczar/omgwtfssl (https://github.com/paulczar/omgwtfssl) image to create certificates. If you would rather create them manually via the `openssl` cli see my blog post on Securing Docker with TLS (http://tech.paulcz.net/2016/01/secure-docker-with-tls/).

We need to create a place on the filesystem to store the data for the registry as well as certificates and config data:

```
$ sudo mkdir -p /opt/registry/{data,ssl,config}
```

Now we can create the certificates, add any IPs and DNS that you might address your registry with including that of any loadbalancer or floating IP that you might have:

```
$ docker run --rm \
  -v /opt/registry/ssl:/certs \
  -e SSL_IP=172.17.8.101 \
  -e SSL_DNS=registry.local \
  paulczar/omgwtfssl
--------------------------
| OMGWTFSSL Cert Generator |
--------------------------

--> Certificate Authority
====> Generating new CA key ca-key.pem
Generating RSA private key, 2048 bit long modulus
.................+++
...............................+++
e is 65537 (0x10001)
====> Generating new CA Certificate ca.pem
====> Generating new config file openssl.cnf
====> Generating new SSL KEY key.pem
Generating RSA private key, 2048 bit long modulus
.....................................................+++
.........................................+++
e is 65537 (0x10001)
====> Generating new SSL CSR key.csr
====> Generating new SSL CERT cert.pem
Signature ok
subject=/CN=example.com
Getting CA Private Key

core@core-01 ~ $ ls /opt/registry/ssl/
ca-key.pem  ca.pem  ca.srl  cert.pem  key.csr  key.pem  openssl.cnf
```

Our next step is to create a config file `/opt/registry/config/registry.env` which will contain a list of Environment Variables that will be passed into the container:

For this example I'm using the same CA certificate for clients as I did for the server, in reality it should probably be a different CA.

```
# location of registry data
REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY=/opt/registry/data

# location of TLK key/cert
REGISTRY_HTTP_TLS_KEY=/opt/registry/ssl/key.pem
REGISTRY_HTTP_TLS_CERTIFICATE=/opt/registry/ssl/cert.pem

# location of CA of trusted clients
REGISTRY_HTTP_TLS_CLIENTCAS_0=/opt/registry/ssl/ca.pem
```

All that is left to do now is start the registry container, bind mount in the `/opt/registry` directory, pass in the config file, and expose port `443` to the internal registry port:

```
$ docker run -d --name registry \
  -v /opt/registry:/opt/registry \
  -p 443:5000 --restart always \
  --env-file /opt/registry/config/registry.env \
  registry:2
Unable to find image 'registry:2' locally
2: Pulling from library/registry
Digest: sha256:a842b52833778977f7b4466b90cc829e0f9aae725aebe3e32a5a6c407acd2a03
Status: Downloaded newer image for registry:2
d0106555b2d0aa30691c75c50b279e6a8bd485aa4ba2f203773e971988253169
```

We can check that we can access it from the server itself by tagging and pushing the `alpine` image to it:

```
$ docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
Digest: sha256:78a756d480bcbc35db6dcc05b08228a39b32c2b2c7e02336a2dcaa196547a41d
Status: Downloaded newer image for alpine:latest
$ docker tag alpine 127.0.0.1/alpine
$ docker push 127.0.0.1/alpine
The push refers to a repository [127.0.0.1/alpine] (len: 1)
74e49af2062e: Pushed
latest: digest: sha256:a96155be113bb2b4b82ebbc11cf1b511726c5b41617a70e0772f8180af
c72fa5 size: 1369
```

To check the security settings worked we'll try to access the docker registry from a remote host:

Anywhere you see `172.17.8.101` you will want to replace it with the IP or hostname of your docker registry.

```
docker pull 172.17.8.101/alpine
Using default tag: latest
Error response from daemon: unable to ping registry endpoint https://172.17.8.10
1/v0/
v2 ping attempt failed with error: Get https://172.17.8.101/v2/: x509: certificat
e signed by unknown authority
 v1 ping attempt failed with error: Get https://172.17.8.101/v1/_ping: x509: cert
ificate signed by unknown authority
```

On the server we can see this failure in the docker logs:

```
$ docker logs registry
2016/01/10 16:18:47 http: TLS handshake error from 172.17.8.1:44096: remote erro
r: bad certificate
2016/01/10 16:18:47 http: TLS handshake error from 172.17.8.1:44098: remote erro
r: bad certificate
2016/01/10 16:18:47 http: TLS handshake error from 172.17.8.1:44099: remote erro
r: bad certificate
```

There are two things causing this failure. The first is that the remote server does not trust the client because it cannot provide the trusted CA certificate as specified in `REGISTRY_HTTP_TLS_CLIENTCAS_0`. The second reason for failure is that the client doesn't trust the `CA` of the server.

If we didn't have `REGISTRY_HTTP_TLS_CLIENTCAS_0` set we could simply add `--insecure-registry 172.17.8.101` to `DOCKER_OPTS` in `/etc/default/docker`, however since we do have this set we'll want to take the `CA.pem` and save it as `/etc/docker/certs.d/172.17.8.101/ca.crt` on the remote machine that you want to trust the registry server.

I do this with the following commands, you may need to do it differently based on how your server is set up for access:

```
$ sudo mkdir -p /etc/docker/certs.d/172.17.8.101
$ sudo scp core@172.17.8.101:/opt/docker/registry/ca.pem \
    /etc/docker/certs.d/172.17.8.101/ca.crt
```

Now we have established trust in both directions we can try to access the docker registry again:

```
$ docker pull 172.17.8.101/alpine
Using default tag: latest
latest: Pulling from alpine

340b2f9a2643: Already exists
Digest: sha256:a96155be113bb2b4b82ebbc11cf1b511726c5b41617a70e0772f8180afc72fa5
Status: Downloaded newer image for 172.17.8.101/alpine:latest
```

Success! We know have a Docker Registry (https://www.docker.com/docker-registry) that is secured both with Encryption and an authorization based on each client having a specific CA certificate. This setup is ideal for providing secure access to a private registry for remote servers.

If you want to do this in a more automated fashion you can look at the various configuration management communities such as chef (https://supermarket.chef.io/cookbooks/docker_registry) for examples.

Related Posts