Private docker registry with Nginx on CentOS 7

Unfortunately the docker registry does not care about authentication. You could use the docker hub to push your own images to the public docker registry but this is not a very good idea for non open source projects. Also there is the possibilty to buy private repositories. In most cases an own private docker registry with SSL and authentication is desired. It took me a lot of time until everything was working on my server because there were a lot of stumbling blocks. I hope you can save some time by reading my blog post "Private docker registry with Nginx on CentOS 7".

Installing the docker registry as a container

Basically you can install the registry in two different ways, either you can install it on a bare metal host or inside of a docker container. In this post we'll use the container way which is very easy. Just type:

```
$ docker pull registry
```

Next create the store location and give the needed permissions:

```
sudo mkdir -p /data/docker/private-registry/storage  sudo chmod 750 /data/docker/private-registry/storage  sudo chown 10000:10000 /data/docker/private-registry/storage
```

And now you can run a container from the downloaded registry image by entering the following command.

```
$ docker run \      -d \      --name private_registry \      -e SETTINGS_FLAVOUR=local \      -e STORAGE_PATH=/registry-storage \      -v /data/docker/private-registry/storage:/registry-storage \      -u 10000 \      -p 5000:5000 \      registry
```

The above command will run the registry on port 5000 and store images on the host filesystem under /data/docker/private-registry/storage. Interesting to know that the docker images are not stored inside of a container. You need an extra store (partition) where your registry stores the images. For example the docker container store can be on /dev/vda3 and your registry store could be on /dev/vda4. In this example /data is the mounting point for /dev/vda4. As I installed the registry for my first time I made only one big partition with btrfs because I was thinking the images will be stored inside of a docker container but this is wrong.

With the command

```
$ docker ps
```

you should see the running containers and I was expecting to see the registry container but nothing was shown. So I checked also the not running containers with

```
$ docker ps -a  CONTAINER ID        IMAGE               COMMAND
      CREATED             STATUS                      PORTS
   NAMES  6a78333cd3ec        registry:latest     "docker-registry"
2 minutes ago       Exited (3) 2 minutes ago                        pri
vate_registry
```

and could see my registry container. Actually I don't know why the container is exited because in the run command the -d switch for daemon is used. But anywhy I started the container again with:

```
  $ docker start private_registry
```

Then confirm again with

```
  $ docker ps  CONTAINER ID        IMAGE               COMMAND
     CREATED             STATUS              PORTS
NAMES  6a78333cd3ec        registry:latest     "docker-registry"   7 m
inutes ago       Up 53 seconds       0.0.0.0:5000->5000/tcp   private_
registry
```

Fortunately the container is now running and listening on port 5000. Use curl to double check everything is allright.

```
$ curl localhost:5000  "\"docker-registry server\""
```

If you can see the string "docker-registry server" everything is ok. But what will happen after a system reboot? Perhaps the docker daemon will start again, because

```
$ systemctl enable docker.service
```

was used. But for sure the registry container will not be running anymore. You can solve this by using a systemd unit file. As shown below.

```
[Unit]  Description=Private Docker Registry  Author=chris.koller@dro
pbit.ch  After=docker.service
   [Service]  Restart=always  ExecStart=/usr/bin/docker start -a priv
ate_registry  ExecStop=/usr/bin/docker stop -t 60 private_registry
[Install]  WantedBy=multi-user.target
```

and save it here /usr/lib/systemd/system/private-docker-registry.service. With the following command you should see your new created private-docker-registry service.

```
$ systemctl list-unit-files | grep docker  docker.service
                 enabled  private-docker-
registry.service            disabled  docker.socket
```

As you can see private-docker-registry.service is disabled currently, let's change it with

```
$ systemctl enable private-docker-registry.service  ln -s '/usr/lib/
systemd/system/private-docker-registry.service' '/etc/systemd/system/m
ulti-user.target.wants/private-docker-registry.service'
```

After the above command the docker registry will be started after a system reboot automatically. Use the next steps to confirm everything works as exprected:

```
$ docker ps  CONTAINER ID        IMAGE               COMMAND
   CREATED             STATUS              PORTS
NAMES  6a78333cd3ec        registry:latest     "docker-registry"   3 d
ays ago          Up 30 minutes       0.0.0.0:5000->5000/tcp   private_
```

```
registry
```

This tells us container private_registry is running and listening on port 5000. Now it's time to connect to the private registry for the second time:

```
  $ curl localhost:5000  "\"docker-registry server\""
```

If you can see the string "docker-registry server" then everything works as expected. At this time the private registry is completely open and insecure. If you use CentOS 7 however nobody can access the registry because the firewall firewalld will block port 5000.

Installing Nginx

Unfortunately the docker registry does not care about authentication you need to use a proxy to make your private registry secure. In this tutorial we use Nginx a
high concurrency, high performance and low memory usage open source reverse proxy server. You could also use an apache or something else. Use the following commands to install nginx and enable it for boot time:

```
  $ rpm -Uvh http://nginx.org/packages/centos/7/noarch/RPMS/nginx-rele
ase-centos-7-0.el7.ngx.noarch.rpm  $ yum install nginx  $ systemctl en
able nginx.service  $ systemctl start nginx.service
```

Because we want to access our private docker registry via http and https we neeed to open the http and https service in our firewall:

```
  $ firewall-cmd --zone=public --list-all  $ firewall-cmd --permanent
--zone=public --add-service=http   $ firewall-cmd --permanent --zone=p
ublic --add-service=https  $ firewall-cmd --reload  $ firewall-
cmd --zone=public --list-all
```

At this time you should be able to access your server's ip address in a browser and see something like:

```
  Welcome to nginx!    If you see this page, the nginx web server is s
uccessfully installed and working. Further configuration is required.
   For online documentation and support please refer to nginx.org.  Co
```

```
mmercial support is available at nginx.com.    Thank you for using ngi
nx.
```

Configure access through Nginx to your private docker registry

First create a file /etc/nginx/sites-available/secure.my.domain.ch, if the folder sites-available does not exist, then create it. Replace my.domain.ch with the domain which you want to use for accessing the private registry. Add the following content to the file afterwards.

```
  # For versions of Nginx > 1.3.9 that include chunked transfer encodi
ng support  # Replace with appropriate values where necessary    upstr
eam private-docker-registry {   server localhost:5000;  }    server {
  listen 443;   server_name my.domain.ch;     #ssl on;   #ssl_certific
ate /data/ssl/certs/my.domain.ch.crt;   #ssl_certificate_key /data/ssl
/private/my.domain.ch.key;     proxy_set_header Host       $http_host;
   # required for Docker client sake   proxy_set_header X-Real-IP  $re
mote_addr; # pass on real client IP     client_max_body_size 0; # disa
ble any limits to avoid HTTP 413 for large image uploads     # require
d to avoid HTTP 411: see Issue #1486 (https://github.com/dotcloud/dock
er/issues/1486)   chunked_transfer_encoding on;     location / {
 # let Nginx know about our auth file       auth_basic              "R
estricted";       auth_basic_user_file    /data/ssl/docker-registry.ht
passwd;         proxy_pass http://private-docker-registry;   }   locat
ion /_ping {       auth_basic off;       proxy_pass http://private-doc
ker-registry;   }   location /v1/_ping {       auth_basic off;       p
roxy_pass http://private-docker-registry;   }    }
```

Notice the configuration /data/ssl/docker-registry.htpasswd; in the file secure.my.domain.ch above. This is the location where nginx checks users and passwords for basic authentication change the path if needed and create the file docker-registry.htpasswd as follows:

```
  $ yum provides \*bin/htpasswd  $ yum install httpd-tools-2.4.6-18.el
7.centos.x86_64  $ mkdir -p /data/ssl/  $ htpasswd -c /data/ssl/docker-
registry.htpasswd myuser
```

Enter and notice a password if the prompt occurs. Next we have to make sure that our Nginx virtual host configuration file can be found. Open the file /etc/nginx/nginx.conf and add after the line "include /etc/nginx/conf.d/*.conf;" the following:

```
  include /etc/nginx/sites-enabled/*;
```

Furthermore we need a symbolic link from sites-enabled to sites-availalbe:

```
  $ mkdir -p /etc/nginx/sites-enabled  $ cd /etc/nginx/sites-enabled
$ ln -s /etc/nginx/sites-
available/secure.my.domain.ch secure.my.domain.ch
```

After the configuration changes we have to tell nginx to reload the configuration files.

```
  systemctl reload nginx.service
```

Now it's time to access our private registry through nginx.

```
  $ curl localhost:443
```

What a suprise we are not allowed to access the registry. That's correct because we told our nginx virtual host to use basic authentication. Provide now your user and password and try again.

```
  curl myuser:test@localhost:443
```

Mmmhh not a lot better now we get an Internal Server Error 500. What's going on here this should work. Let's see perhaps the nginx log file can tell us something.

```
  tail -f /var/log/nginx/error.log  2014/12/22 14:15:01 [crit] 3877#0:
 *6 open() "/data/ssl/docker-registry.htpasswd" failed (13: Permission
 denied), client: 127.0.0.1, server: my.domain.ch, request: "GET / HTT
P/1.1", host: "localhost:443"  2014/12/22 14:16:09 [crit] 3877#0: *7 o
pen() "/data/ssl/docker-registry.htpasswd" failed (13: Permission deni
ed), client: 127.0.0.1, server: my.domain.ch, request: "GET / HTTP/1.1
", host: "localhost:443"  2014/12/22 14:17:14 [crit] 3877#0: *8 open()
 "/data/ssl/docker-registry.htpasswd" failed (13: Permission denied),
client: 127.0.0.1, server: my.domain.ch, request: "GET / HTTP/1.1", ho
st: "localhost:443"
```

My friend SELinux blocks here, we need to add a rule first.

```
  $ yum install policycoreutils-python  $ grep nginx /var/log/audit/au
dit.log | audit2allow -m nginx > nginx.te  $ grep nginx /var/log/audit
/audit.log | audit2allow -M nginx  $ semodule -i nginx.pp  $ rm -rf ng
inx.pp  $ rm -rf nginx.te  $ systemctl reload nginx.service  $ systemc
tl restart nginx.service
```

You can find more information about the rule above here. Now let's try again.

```
  curl myuser:test@localhost:443
```

And again something blocks our request from fulfilling, this time we face 502 Bad Gateway. The log /var/log/nginx/error.log shows the following this time.

```
  2014/12/22 15:02:53 [crit] 4169#0: *7 connect() to 127.0.0.1:5000 fa
iled (13: Permission denied) while connecting to upstream, client: 127
.0.0.1, server: my.domain.ch, request: "GET / HTTP/1.1", upstream: "ht
tp://127.0.0.1:5000/", host: "localhost:443"  2014/12/22 15:02:54 [err
or] 4169#0: *10 no live upstreams while connecting to upstream, client
: 127.0.0.1, server: my.domain.ch, request: "GET / HTTP/1.1", upstream
: "http://private-docker-registry/", host: "localhost:443"
```

To avoiding this problem type:

```
  $ setsebool -P httpd_can_network_connect true
```

All good things come in threes, type again:

```
  curl myuser:test@localhost:443
```

And you should see again the famous string "docker-registry server". What does this mean so far?
1. Nginx receives the http request on port 443 and proxies to the private docker registry.
2. Without providing user and password nginx prevents access.
3. Basic Authentication is in place.

4. No HTTPS (SSL) configured so far.

Configure Nginx to use ssl

Basic Authentication without ssl is not secure because the connection is unencrypted therefore we add ssl to our configuration.
First we need a self-signed SSL certificate. Since Docker currently doesn't allow you to use self-signed SSL certificates this is a bit more complicated than usual. We will have to set up our system to act as our own certificate signing authority.
In the first step create a new root key with:

```
  $ mkdir /tmp/certs  $ cd /tmp/certs  $ openssl genrsa -out dockerCA.
key 2048
```

Then create a root certificate, you don't have to answer the upcoming question, just hit enter.

```
  $ openssl req -x509 -new -nodes -key dockerCA.key -days 3650 -out do
ckerCA.crt
```

Then create a private key for your Nginx Server:

```
  $ openssl genrsa -out my.domain.ch.key 2048
```

Next a certificate signing request is needed. **Answer the upcoming question for "Common Name" with the domain of your server, e.g: my.domain.ch.** In this example you would access your private docker registry with my.domain.ch at the end. Don't provide a challenge password.

```
  $ openssl req -new -key my.domain.ch.key -out my.domain.ch.csr
```

Afterwards we need to sign the certificate request:

```
  $ openssl x509 -req -in my.domain.ch.csr -CA dockerCA.crt -CAkey doc
kerCA.key -CAcreateserial -out my.domain.ch.crt -days 3650
```

Now open the file /etc/nginx/sites-available/secure.my.domain.ch again and look for the lines:

```
  ...  #ssl on;  #ssl_certificate /data/ssl/certs/my.domain.ch.crt;  #
ssl_certificate_key /data/ssl/private/my.domain.ch.key;  ...
```

Remove the hashtags and make sure ssl_certificate and ssl_certificate_key points to your newly generated my.domain.ch.crt and my.domain.ch.key files.
Since the certificates we just generated aren't verified by any known certificate authority (e.g.: VeriSign), we need to tell any clients that are going to be using this Docker registry that this is a legitimate certificate. Let's do this locally so that we can use Docker from the Docker registry server itself:

```
  $ update-ca-trust enable  $ cp dockerCA.crt /etc/pki/ca-trust/source
/anchors/  $ update-ca-
trust extract  $ systemctl reload nginx.service
```

After that our host accepts the certificate and we should be able to access our private docker registry with https.

```
  curl https://myuser:test@my.domain.ch
```

If everything works as expected you will face the famouse string "docker-registry server" again. Notice you need to access with your domain. localhost for example will not work.

```
  $ curl https://myuser:test@localhost  curl: (51) Unable to communica
te securely with peer: requested domain name does not match the server
's certificate.
```

Use docker with your private remote docker registry

Now the time is ripe for end to end testing. I assume that you have installed a docker daemon somewhere and are able to enter docker commands, for example:

```
  $ docker info
```

Now let's try to requet the registry:

```
$ curl https://myuser:test@my.domain.ch
```

For sure you want to see something like:

```
curl: (60) Peer's certificate has an invalid signature.  More detail
s here: http://curl.haxx.se/docs/sslcerts.html    curl performs SSL ce
rtificate verification by default, using a "bundle" of Certificate Aut
hority (CA) public keys (CA certs). If the default bundle file isn't a
dequate, you can specify an alternate file using the --cacert option.
If this HTTPS server uses a certificate signed by a CA represented in
the bundle, the certificate verification probably failed due to a prob
lem with the certificate (it might be expired, or the name might not m
atch the domain name in the URL). If you'd like to turn off curl's ver
ification of the certificate, use the -k (or --insecure) option.
```

Because we used a self-signed certificate we have to copy the certificate to the new client host. Copy the file dockerCA.crt from the docker registry host to the new client host to the directory /etc/pki/ca-trust/source/anchors/ and enter the following commands:

```
$ update-ca-trust enable   $ update-ca-trust extract
```

And now try again:

```
$ curl https://myuser:test@my.domain.ch
```

If you can see the string "docker-registry server" it does mean you successfully connected to your private docker registry via https connection and basic authentication. Yes a secure connection is in place. Congratulation! Try to connect with docker now.

```
$ docker login --username='myuser' --password='test' --email="chris.
koller@dropbit.ch" https://my.domain.ch
```

Hopefully you can see "Login Succeeded" now. If that is the case it was well worth the effort. Use the next docker commands to pull a docker image, tag it and push it to your registry.

```
$ docker pull busybox  $ docker tag busybox:latest my.domain.ch/mybu
```

```
sybox:latest  $ docker push my.domain.ch/mybusybox:latest
```

If you have some questions, don't hesitate to contact me. Thanks for reading and share if you like it.