How To Set Up a Private Docker Registry on Ubuntu

Step 1 — Installing and Configuring the Docker Registry

Docker on the command line is useful when starting out and testing containers, but proves to be unwieldy for bigger deployments involving multiple containers running in parallel.

With Docker Compose, you can write one `.yml` file to set up each container's configuration and information the containers need to communicate with each other. You can use the `docker-compose` command-line tool to issue commands to all the components that make up your application, and control them as a group.

Docker Registry is itself an application with multiple components, so you will use Docker Compose to manage it. To start an instance of the registry, you'll set up a `docker-compose.yml` file to define it and the location on disk where your registry will be storing its data.

You'll store the configuration in a directory called `docker-registry` on the main server. Create it by running:

1. mkdir ~/docker-registry

Navigate to it:

1. cd ~/docker-registry

Then, create a subdirectory called `data`, where your registry will store its images:

1. mkdir data

Create and open a file called `docker-compose.yml` by running:

1. nano docker-compose.yml

Add the following lines, which define a basic instance of a Docker Registry:

~/docker-registry/docker-compose.yml

```
version: '3'

services:
  registry:
    image: registry:2
    ports:
    - "5000:5000"
    environment:
      REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY: /data
    volumes:
      - ./data:/data
```

First, you name the first service `registry`, and set its image to `registry`, version `2`. Then, under `ports`, you map the port `5000` on the host to the port `5000` of the container. This allows you to send a request to port `5000` on the server, and have the request forwarded to the registry.

In the `environment` section, you set the `REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY` variable to `/data`, specifying in which volume it should store its data. Then, in the `volumes` section, you map the `/data` directory on the host file system to `/data` in the container, which acts as a passthrough. The data will actually be stored on the host's file system.

Save and close the file.

You can now start the configuration by running:

1. docker-compose up

The registry container and its dependencies will be downloaded and started.

You'll address the `No HTTP secret provided` warning message later in this tutorial. Notice that the last line of the output shows it has successfully started listening on port `5000`.

You can press `CTRL+C` to stop its execution.

In this step, you have created a Docker Compose configuration that starts a Docker Registry listening on port `5000`. In the next steps, you'll expose it at your domain and set up authentication.

Step 2 — Setting Up Nginx Port Forwarding

As part of the prerequisites, you've enabled HTTPS at your domain. To expose your secured Docker Registry there, you'll only need to configure Nginx to forward traffic from your domain to the registry container.

You have already set up the `/etc/nginx/sites-available/your_domain` file, containing your server configuration. Open it for editing by running:

1. sudo nano /etc/nginx/sites-available/your_domain

Find the existing `location` block:

/etc/nginx/sites-available/your_domain

```
...
location / {
  ...
}
...
```

You need to forward traffic to port `5000`, where your registry will be listening for traffic. You also want to append headers to the request forwarded to the registry, which provides additional information from the server about the request itself. Replace the existing contents of the `location` block with the following lines:

/etc/nginx/sites-available/your_domain

```
...
location / {
    # Do not allow connections from docker 1.5 and earlier
    # docker pre-1.6.0 did not properly set the user agent on ping, catch "Go *" user agents
    if ($http_user_agent ~ "^(docker\/1\.(3|4|5(?!\.[0-9]-dev))|Go ).*$" ) {
      return 404;
    }

    proxy_pass                  http://localhost:5000;
```

```
    proxy_set_header  Host              $http_host;   # required for docker client's sake
    proxy_set_header  X-Real-IP         $remote_addr; # pass on real client's IP
    proxy_set_header  X-Forwarded-For   $proxy_add_x_forwarded_for;
    proxy_set_header  X-Forwarded-Proto $scheme;
    proxy_read_timeout                  900;
  }
  ...
```

The `if` block checks the user agent of the request and verifies that the version of the Docker client is above 1.5, as well as that it's not a `Go` application that's trying to access. For more explanation on this, you can find the `nginx` header configuration in Docker's registry Nginx guide.

Save and close the file when you're done. Apply the changes by restarting Nginx:

1. sudo systemctl restart nginx

If you get an error, double-check the configuration you've added.

To confirm that Nginx is properly forwarding traffic to your registry container on port `5000`, run it:

1. docker-compose up

Then, in a browser window, navigate to your domain and access the `v2` endpoint, like so:

```
https://your_domain/v2
```

You will see an empty JSON object:

```
{}
```

In your terminal, you'll receive output similar to the following:

Output

```
registry_1  | time="2018-11-07T17:57:42Z" level=info msg="response completed" go.version=go1.7.6
http.request.host=cornellappdev.com http.request.id=a8f5984e-15e3-4946-9c40-d71f8557652f
http.request.method=GET http.request.remoteaddr=128.84.125.58 http.request.uri="/v2/"
http.request.useragent="Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2) AppleWebKit/604.4.7 (KHTML, like
Gecko) Version/11.0.2 Safari/604.4.7" http.response.contenttype="application/json; charset=utf-8"
http.response.duration=2.125995ms http.response.status=200 http.response.written=2 instance.id=3093e5ab-
5715-42bc-808e-73f310848860 version=v2.6.2
registry_1  | 172.18.0.1 - - [07/Nov/2018:17:57:42 +0000] "GET /v2/ HTTP/1.0" 200 2 "" "Mozilla/5.0 (Macintosh;
```

```
Intel Mac OS X 10_13_2) AppleWebKit/604.4.7 (KHTML, like Gecko) Version/11.0.2 Safari/604.4.7"
```

You can see from the last line that a `GET` request was made to `/v2/`, which is the endpoint you sent a request to, from your browser. The container received the request you made, from the port forwarding, and returned a response of `{}`. The code `200` in the last line of the output means that the container handled the request successfully.

Press `CTRL+C` to stop its execution.

Now that you have set up port forwarding, you'll move on to improving the security of your registry.

Step 3 — Setting Up Authentication

Nginx allows you to set up HTTP authentication for the sites it manages, which you can use to limit access to your Docker Registry. To achieve this, you'll create an authentication file with `htpasswd` and add username and password combinations to it that will be accepted.

You can obtain the `htpasswd` utility by installing the `apache2-utils` package. Do so by running:

1. sudo apt install apache2-utils -y

You'll store the authentication file with credentials under `~/docker-registry/auth`. Create it by running:

1. mkdir ~/docker-registry/auth

Navigate to it:

1. cd ~/docker-registry/auth

Create the first user, replacing `username` with the username you want to use. The `-B` flag orders the use of the `bcrypt` algorithm, which Docker requires:

1. htpasswd -Bc registry.password username

Enter the password when prompted, and the combination of credentials will be appended to `registry.password`.

Note: To add more users, re-run the previous command without `-c`, which creates a new file:

```
1. htpasswd -B registry.password username
```

Copy

Now that the list of credentials is made, you'll edit `docker-compose.yml` to order Docker to use the file you created to authenticate users. Open it for editing by running:

1. nano ~/docker-registry/docker-compose.yml

Add the highlighted lines:

~/docker-registry/docker-compose.yml

```
version: '3'

services:
  registry:
    image: registry:2
    ports:
    - "5000:5000"
    environment:
      REGISTRY_AUTH: htpasswd
      REGISTRY_AUTH_HTPASSWD_REALM: Registry
      REGISTRY_AUTH_HTPASSWD_PATH: /auth/registry.password
      REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY: /data
    volumes:
      - ./auth:/auth
      - ./data:/data
```

You've added environment variables specifying the use of HTTP authentication and provided the path to the file `htpasswd` created. For `REGISTRY_AUTH`, you have specified `htpasswd` as its value, which is the authentication scheme you are using, and set `REGISTRY_AUTH_HTPASSWD_PATH` to the path of the authentication file. `REGISTRY_AUTH_HTPASSWD_REALM` signifies the name of `htpasswd` realm.

You've also mounted the `./auth` directory to make the file available inside the registry container. Save and close the file.

You can now verify that your authentication works correctly. First, navigate to the main directory:

1. cd ~/docker-registry

Then, run the registry by executing:

1. docker-compose up

In your browser, refresh the page of your domain. You'll be asked for a username and password.

After providing a valid combination of credentials, you'll see an empty JSON object:

```
{}
```

This means that you've successfully authenticated and gained access to the registry. Exit by pressing `CTRL+C`.

Your registry is now secured and can be accessed only after authentication. You'll now configure it to run as a background process while being resilient to reboots by starting automatically.

Step 4 — Starting Docker Registry as a Service

You can ensure that the registry container starts every time the system boots up, or after it crashes, by instructing Docker Compose to always keep it running. Open `docker-compose.yml` for editing:

1. nano docker-compose.yml

Add the following line under the `registry` block:

docker-compose.yml

```
...
  registry:
    restart: always
...
```

Setting `restart` to always ensures that the container will survive reboots. When you're done, save and close the file.

You can now start your registry as a background process by passing in `-d`:

1. docker-compose up -d

With your registry running in the background, you can freely close the SSH session, and the registry won't be affected.

Because Docker images may be very large in size, you'll now increase the maximum file size that Nginx will accept for uploads.

Step 5 — Increasing File Upload Size for Nginx

Before you can push an image to the registry, you need to ensure that your registry will be able to handle large file uploads.

The default size limit of file uploads in Nginx is `1m`, which is not nearly enough for Docker images. To raise it, you'll modify the main Nginx config file, located at `/etc/nginx/nginx.conf`. Open it for editing by running:

1. sudo nano /etc/nginx/nginx.conf

Find the `http` section, and add the following line:

/etc/nginx/nginx.conf

```
...
http {
        client_max_body_size 16384m;
        ...
}
...
```

The `client_max_body_size` parameter is now set to `16384m`, making the maximum upload size equal to 16GB.

Save and close the file when you're done.

Restart Nginx to apply the configuration changes:

1. sudo systemctl restart nginx

You can now upload large images to your Docker Registry without Nginx blocking the transfer or erroring out.

Step 6 — Publishing to Your Private Docker Registry

Now that your Docker Registry server is up and running, and accepting large file sizes, you can try pushing an image to it. Since you don't have any images readily available, you'll use the `ubuntu` image from Docker Hub, a public Docker Registry, to test.

From your second, client server, run the following command to download the `ubuntu` image, run it, and get access to its shell:

1. docker run -t -i ubuntu /bin/bash

The `-i` and `-t` flags give you interactive shell access into the container.

Once you're in, create a file called `SUCCESS` by running:

1. touch /SUCCESS

By creating this file, you have customized your container. You'll later use it to check that you're using exactly the same container.

Exit the container shell by running:

1. exit

Now, create a new image from the container you've just customized:

1. docker commit $(docker ps -lq) test-image

The new image is now available locally, and you'll push it to your new container registry. First, you have to log in:

1. docker login https://your_domain

When prompted, enter in a username and password combination that you've defined in step 3 of this tutorial.

The output will be:

Output

```
...
Login Succeeded
```

Once you're logged in, rename the created image:

1. docker tag test-image your_domain/test-image

Finally, push the newly tagged image to your registry:

1. docker push your_domain/test-image

You'll receive output similar to the following:

Output

```
The push refers to a repository [your_domain/test-image]
420fa2a9b12e: Pushed
c20d459170d8: Pushed
db978cae6a05: Pushed
aeb3f02e9374: Pushed
latest: digest: sha256:88e782b3a2844a8d9f0819dc33f825dde45846b1c5f9eb4870016f2944fe6717 size: 1150
```

You've verified that your registry handles user authentication by logging in, and allows authenticated users to push images to the registry. You'll now try pulling the image from your registry.

Step 7 — Pulling From Your Private Docker Registry

Now that you've pushed an image to your private registry, you'll try pulling from it.

On the main server, log in with the username and password you set up previously:

1. docker login https://your_domain

Try pulling the `test-image` by running:

1. docker pull your_domain/test-image

Docker should download the image. Run the container with the following command:

1. docker run -it your_domain/test-image /bin/bash

List the files present by running:

1. ls

You will see the `SUCCESS` file you've created earlier, confirming that its the same image you've created:

```
SUCCESS  bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

Exit the container shell by running:

1. exit

Now that you've tested pushing and pulling images, you've finished setting up a secure registry that you can use to store custom images.

Revision #2
Created 22 May 2023 11:56:07 by Alois
Updated 22 May 2023 12:02:36 by Alois