How to create your own private Docker registry and secure it

Docker lets you bundle your application into different containers, which makes it easy to develop and deploy your applications. But where do these container images come from and how can you deploy your own.

In this article, we will take a look at what a registry is, why it is essential and how you can create your own private registry. We will also take a look at some security and storage options that can help you customize your configuration.

So without wasting any further time, let's get started.

What is a Container Registry?

A container registry is a stateless, highly scalable central space for storing and distributing container images. They provide secure image management and a fast way to pull and push images with the right permissions.

The most well-known container registry is DockerHub, which is the standard registry for Docker and Kubernetes. The only problem with public registries is that you don't have full control over their actions and that they can get expensive if you need multiple private images.

That is why hosting your own private registry could come in useful in many cases. Private registries provide multiple different storage and authentification options and can be customized to your individual requirements.

Why use a private registry?

Here are some essential reasons why you should use your own private registry instead of a public registry like DockerHub.

- Control where your images are stored - A private registry gives you full control over the storage location of your images and how you can access them
- Custome image pipeline
- More privacy for proprietary and private images
- Custome configuration options e.g. logging, authentification, load balancing, etc..

Creating your own registry

Now that you have an overview of registries and what they are used for let's continue by creating a private registry using docker-compose.

```
version: '3'

services:
  registry:
    image: registry:2
    ports:
    - "5000:5000"
```

The configuration uses the official registry image and forwards the port 5000 of the container to the host machine. This allows us to send requests to port 5000 on the server that runs the registry.

You can now run the container using the following command:

```
docker-compose up -d
```

After the download of the image has completed, and the container is running, we can continue with pushing an image to the registry.

Pushing an image

You are now ready to push an image to the registry, but first, you need to create a local image and provide it with the right tag. For that, we are going to use the alpine Linux image because it is small and downloads fast.

First, we need to pull the image and then tag it with the address of our registry as a prefix (localhost:5000 in our case).

```
docker pull alpine
docker tag alpine localhost:5000/my-alpine
```

The newly labeled image should now appear:

```
docker images

# output
alpine                                      latest              e7d92cdc71fe        6
localhost:5000/my-alpine                    latest              e7d92cdc71fe        6
```

Now we can push the image using the push command:

```
docker push localhost:5000/my-alpine
```

Note: This only works if you host your registry on your local machine. If you host it on a server, you will need a secure SLL connection, which we will look at in a later section.

Pulling an image

Pulling an image from the registry is also straight forward and can be done using a single command.

```
docker pull localhost:5000/my-alpine
```

You should get a message that the image already exists. You can remove the image and pull it again if you want to make sure that it functions correctly.

After that, you can run the image as follows.

```
docker run -it localhost:5000/my-alpine sh
```

Note: Most registries will require you to log in before pulling and pushing images for authentification purposes. That is also what we will implement in the next section.

Setting up Authentification

All registries which are not located in a secure local network that only authorized people can access will need some kind of authentification to keep it safe from abuse.

The simplest way to achieve basic registry security and access restriction is through some kind of basic authentification tool like htpasswd, which stores a secret that helps you authenticate.

That is the method we will focus on in this article, but I will also provide a few more advanced options that you can look at on your own.

Creating the authentification file:

First, we start by installing the *htpasswd* package by running the following command:

```
sudo apt install apache2-utils
```

Next, we will create a folder that will hold our password files.

```
mkdir auth
```

After that, we will continue by creating a user using the following command:

```
htpasswd -Bc registry.password testuser
```

The last parameter is the name of the user in this case testUser. After executing the command, you will be prompted to enter your password.

Updating the docker-compose file:

Now that we have created the user using htpasswd, it is time to edit our docker-compose.yaml file.

```
version: '3'

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

For REGISTRY_AUTH you have to provide the authentification scheme you are using. REGISTRY_AUTH_HTPASSWD_PATH is the path of the authentification file we just created above.

You can now restart your Docker set up to make the changes accessible.

```
docker-compose up --force-recreate
```

Log in to the registry:

Now that the registry is running with basic authentification, you can test it by logging in using the user you created above.

```
docker login localhost:5000
```

You will be prompted to enter your username and password. After successfully logging into your registry ,you can push and pull images the same way as we did above.

Advanced authentication:

There are also more advanced ways to provide authentification for your registry. The most popular is to create a proxy and put it in front of your registry. This approach requires a more complex configuration and set up but also gives you more control over the access of your registry.

Here is an official guide by Docker on how to use Nginx as your authentification proxy.

Adding SSL Certificates

A registry on localhost has limited functionality and can not be accessed from external sources. That is why adding an SSL certificate for a secure connection is vital when hosting a registry.

This section assumes you have the following requirements:

- You have your own secure domain e.g. https://mydomain.com

- Your DNS configuration allows accessing the registry on port 443
- You have obtained a certificate from a certificate authority (CA) e.g. Let's Encrypt

There are different ways of adding a certificate to your registry. We will look at the most common one which will cover most use-cases.

Adding a certificate:

If you already have a *.crt* and *.key* file from your CA, then you just need to copy them into a directory named certs in your project and add the following lines to your docker-compose file.

```
version: '3'

services:
  registry:
    image: registry:2
    ports:
    - "443:443"
    environment:
      REGISTRY_HTTP_ADDR=0.0.0.0:443
      REGISTRY_HTTP_TLS_CERTIFICATE=certs/domain.crt
      REGISTRY_HTTP_TLS_KEY=certs/domain.key
    volumes
    - ./certs:/certs
```

These environment variables tell the container where to find the certificates. The registry should now be secure and run on port 443 which is the default HTTPS port.

Customize Storage

The docker registry also lets you customize the location where the data of the registry is saved. For that, you just have to add an extra environment variable that defines the path the data should be saved to.

```
version: '3'

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

Further steps

There are many more configuration steps you can take to improve the security and functionality of your registry. Here is a list of things that might be of interest:

- Authentification using a proxy before your service
- Load balancing requests
- Logging error messages
- Monitoring performance and container health

Source

Here is a list of the sources I used for this article:

- How to use a private Registry
- Docker Registry Docs
- Setting up a private registry

- Katacode exercise
- Private Registry Ubuntu 18.04

Conclusion

You made it all the way until the end! I hope that this article helped you understand the basics of a container registry and how you can create your own.

If you have found this useful, please consider recommending and sharing it with other fellow developers. If you have any questions or feedback, let me know using my contact form or contact me on twitter.