A

Share Docker Images Without Using the Docker Hub | Baeldung

Last modified: September 28, 2021

1. Overview

Let's suppose we need to share a Docker image that is present locally on our machine. To solve this problem, Docker Hub comes to the rescue.

Docker Hub is a cloud-based central repository where Docker images can be stored. So all we need to do is push our Docker image to the Docker Hub, and later, anyone can pull the same Docker image.

Being a cloud-based repository, Docker Hub requires an additional network bandwidth to upload and download the Docker images. Also, as the image

size increases, the time needed to upload/download the image also increases. Hence, this method of sharing the Docker images is not always useful.

In this tutorial, we'll discuss a way to share Docker images without using the Docker Hub. This approach proves out to be handy when sender and receiver are connected to the same private network.

2. Save Docker Image as a *tar* Archive

Suppose there is a Docker image *baeldung* which we need to transfer from machine A to machine B. To achieve this, first, we'll convert the Docker image to a *.tar* file using the *docker save* command:

```
$ docker save --output baeldung.tar
baeldung
```

The above command will create a tar archive named *baeldung.tar.* Alternatively, we can also use file redirection to achieve similar results:

```
$ docker save baeldung >
baeldung.tar
```

The *docker save* command can create a single tar archive using multiple Docker images:

```
$ docker save -o ubuntu.tar
ubuntu:18.04 ubuntu:16.04
ubuntu:latest
```

3. Transfer the *tar* Archive

The tar archive that we created is present on machine A. Let's now transfer the *baeldung.tar* file to machine B. We can use the protocols like *scp* or *ftp*.

This step is highly flexible and depends significantly on the environment where machine A and machine B are present.

4. Load *tar* Archive into the Docker Image

So far, we have created the tar archive of the Docker image and moved it to our target machine B.

Now, we'll create the actual Docker image from the tar archive *baeldung.tar* using the *docker load* command:

```
$ docker load --input baeldung.tar
Loaded image: baeldung:latest
```

Again, we can also use redirection from the file to convert the *tar* archive:

```
$ docker load < baeldung.tar
Loaded image: baeldung:latest
```

Let's now verify whether the image is successfully loaded by running the *docker images* command:

```
$ docker images
baeldung
latest
277bcd6563ce        About a minute
ago       466MB
```

Note that if the Docker image, *baeldung*, is already present on the target machine (machine B in our example), then the *docker load* command will rename the tag of the existing image to an empty string <none>:

```
$ docker load --input baeldung.tar
cfd97936a580: Loading layer
```

```
[==================================
==============>]  466MB/466MB
The image baeldung:latest already
exists, renaming the old one with ID
sha256:

277bcd6563ce2b71e43b7b6b7e12b830f5b3
29d21ab690d59f0fd85b01045574 to
empty string
```

5. Drawbacks

Using this approach, we lose the freedom to reuse the cached layers of the Docker image. So, **each time we run the *docker save* command, it'll create the *tar* archive of the entire Docker image.**

Another drawback is that we need to maintain the Docker image versions manually by saving all the *tar* archives.

Hence, it is recommended to use this approach in the testing environment or when we have restricted access to Docker Hub.

6. Conclusion

In this tutorial, we learned about the *docker save* and *docker load* commands and how to transfer a Docker image using these commands.

We also went through the downsides involved and the ideal situations where this approach could prove out to be efficient.