Building an own Docker Image

Above you have seen when we ran the docker run hello-world command a prebuilt image was used. Here we will learn to build our own image of the client application.

1. Go into the root directory of your project, where you have kept all the files of the client application.
2. Create a new file with the name “**Dockerfile5”**, Docker engine will use this file to build the image.
3. Open the file and start writing the following commands in the file:
   1. FROM node:boron  The FROM keyword tells Docker which image your image is based on. Here the  node:boron is the latest Linux image with npm and node installed into it.
   2. RUN mkdir -p /usr/src/”name\_of\_your\_application” WORKDIR /usr/src/”name\_of\_your\_application” Here we are creating a directory which will contain our application inside the container and set the directory as the working directory.
   3. COPY package.json /usr/src/”name\_of\_your\_application” RUN npm install

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image



The next task is to install all the dependencies of our application into the container. As you already know all dependencies are inside the package.json file. So first copy that file into your application folder and then install them by running “npm install”

1. COPY . /usr/src/”name\_of\_your\_application” This command will bundle up your source files into your application directory.
2. EXPOSE “port\_number” This command exposes the port number of your application for outside world to access it.
3. CMD [ "node", "clientServer.js"]  This tells the image the final command to run after its environment is set up

Save your file, your docker file is ready.

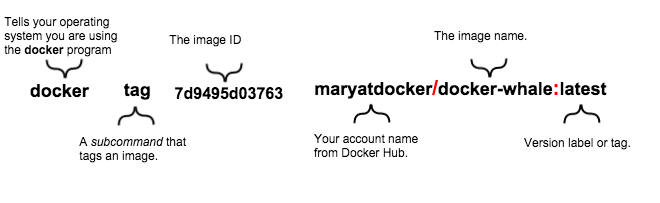
1. The next task is to add the .dockerignore file. This file tells the docker engine which files/directories to ignore while building the image. Create a new file with the name “.dockerignore” inside the project directory. Open it up and add the following in different lines
   1. node\_modules  To ignore all the npm\_modules installed in your project directory.
   2. npm-debug.log  To ignore all the debug logs.
2. Now we are all set to build our docker image. Inside the project directory run the following  command  $ docker build -t “image\_name” .  In the end, you will see it to be successfully built.
3. Now to check your local images you can run the command  $ docker images  You will see your image in the list.
4. Now your image is built, it’s time to run it. For running the image type the following  command  $ docker run -p ExternalPORT-NUMBER:InternalDockerPORT-NUMBER “image \_name”
5. After this command your application will start in the docker container. You can view all the running containers by typing the command  $ docker ps

Tag, Push and Pull your Image

As you already have built the image and created the docker hub account, so now it’s the time to tag your image and push it to your docker hub account. Tags are used to identify different versions.

Tag and Push

1. Open the terminal and run the $ docker images command.
2. Find the image ID for the “image\_name” image, in the third column.
3. Tag the “image\_name” image using the docker tag command and **the image ID**. The  command you have to type looks like this:



*Figure 6: Docker tag [6]*

**Make sure to use your own Docker Hub account name and the name of your image.**

1. Run docker images again to verify that the “image\_name” image has been tagged. The same image ID now exists in two different repositories.
2. Before you can push the image to Docker Hub, you need to log in, using the docker login command. The command doesn’t take any parameters, but prompts you for the username and password, as below:  $ docker login  Username: \*\*\*\*\* Password: \*\*\*\*\* Login Succeeded
3. Push your tagged image to Docker Hub, using the docker push command. $ docker push “your\_account\_id”/”your\_application\_image\_name”
4. Go back to the Docker Hub website to see the newly-pushed image in your account.

Pull the Image

You have your application image pushed into docker hub, now you can access it from any Docker host using docker pull. First, though, you need to remove the local copy. Otherwise, docker pull will not have any work to do, because it will see that you already have the latest version of the image locally. Open the terminal and run the following commands:

1. Use docker images to list the images you have locally.
2. Use the docker rmi command to remove the images. You can refer to an image by its ID  or its name. Since they share an ID, if you wanted to keep one of them, you’d need to refer to the other one by name. For this example, use the ID to remove both.  $ docker rmi -f “your\_application\_image\_id”
3. When you use docker run it automatically downloads (pulls) images that don’t yet exist  locally, creates a container, and starts it. So after removing the image run the following command

$ docker run yourusername/”your\_application\_image\_name”

Since the image is no longer available on your local system, Docker pull and run it.

Application Deployment

Now you have learned how to run a node.js application in the docker container. So, if you want to deploy this application on a VM, instead of running on the VM directly you should run it inside the docker container. This will deploy your application on the VM inside the docker container and make the management and deployment very easy.