**What simple project can I work on to get a hands on experience in Docker?**

That’s the beauty of Docker - really anything can work. If you want some experience that will stick, instead of walking through the Docker Docs and simple todos, I suggest the following:

a) Pick a project you’re currently working on or have

So let’s say you have an application that’s a react web app. It uses node, yarn, etc. And all of it is on your local machine.

b) Create a Docker image that can act as your development environment as opposed to using dependencies on your local machine

So in this case, you’d create a Dockerfile with something really simple like:

1. FROM node:6.9.4
3. RUN npm install -g <any packages you need globally>
5. WORKDIR /usr/src/app

Which in this case says, build an image from the official Node image and whenever it’s run, inside of it, use “/usr/src/app” as the working directory.

Build your image with something like

1. $ docker build -t name-of-your-image directory/ofDockerFile/

Where name-of-your-image is, well, the name of your image. The directory of your dockerfile should be just that - the directory that contains the Dockerfile.

c) Use the development image

The goal here is to use the Node version inside of the Docker file / container instead of using the ones on your local machine.

Let’s assume you have a directory “app” and inside is all of your code. The way you’d then run your local code would be to “mount” your local code into an instance of your Docker Image (a container). Think of mounting like inserting a CD rom into a computer, where the CD rom is your local folder and the container is like the computer that’s going to run the CD rom. When this computer saves changes to the CD, and you remove it the changes will still be there.

The way you’d do this is, inside of your “app” directory, is run something like

1. $ docker run -p 3000:3000 -it -v $(pwd):/usr/src/app --rm name-of-your-built-image bash

To talk through it

a) `docker run` - this creates a container of your image

b) `-p 3000:3000` - assuming your app listens on localhost 3000, you’d need to map your container’s port 3000 to your local machines port 3000.

c) `-it` - this runs your container in interactive mode, so that you’ll actually be able to be inside of it and work with bash

d)`—rm` - this will remove the container once you exit it

e) `-v $(pwd):/usr/src/app` - this mounts your present working directory, in this case your “app” directory, to the working directory within your node container

f) `name-of-your-built-image` - self explanatory

g) `bash` - this tells docker to run the command bash, which will allow you to navigate about inside of your container once it’s up and running

Once this command is run you’ll shown a command line interface that is IN your container, but has access to the files of your local app folder. In fact if you do a

1. $ ls

You’ll see that the files are all there

From there, you can also run your app, assuming it’s only global dependency is Node. If your project has local dependencies (node\_modules), those will be in there. So the next step would be:

1. node .

Or whatever your app’s begin command is. At this point, your app will go up, be live on localhost:3000 and you’ll still be able to edit and work with the files as normally.

The above is obviously very simplified, but the whole point is this:

Take a project. Dockerize it.

I have a tutorial on using docker and docker-compose on react style apps that might help here:

[No Eject - Create React App with SASS, Storybook and Yarn in a Docker Environment](http://start.jcolemorrison.com/no-eject-create-react-app-with-sass-storybook-and-yarn-in-a-docker-environment/)

That builds up a react app from docker images and containers.

**YOUR TITLE HERE**