**https://www.youtube.com/watch?v=edPrPcgjTgw**

**Demo Steps**

**1. Create an app directory and use NPM to create a package.json file and add the Express framework.**

$ mkdir hello-world

$ cd hello-world

$ npm init

$ npm install —save express

**2. Create an index.js for our app code.**

$ touch index.js

**3. Open a code editor and add some code for a simple hello world app.**

var express = require('express')

var app = express()

app.get('/', function (req, res) {

res.send('Hello World!')

})

app.listen(8081, function () {

console.log('app listening on port 8081!')

})

**4. Note that the Node code is using port 8081. Then, run the app in Node and open it in a browser.**

$ node index.js

Node should tell you the app is running on port 8081. Test the app by opening [http://localhost:8081](http://localhost:8081/).

If you see "Hello World" all is good.

**5. Create a Dockerfile**

$ touch Dockerfile

Note: you can also just create a file from your code editor. The Dockerfile is just a plain text file named 'Dockerfile' with no extension.

**6. Add instructions for Docker into the Dockerfile**

FROM node:7

WORKDIR /app

COPY package.json /app

RUN npm install

COPY . /app

CMD node index.js

EXPOSE 8082

The first line tells Docker to use another Docker image as a template for the image that we’re creating.

We’re using is the official Docker image for Node.js and it’s version 7 of that image.

The second line, sets a working directory for where the app code will live inside the Docker container.

On lines 3, 4, and 5 we’re telling Docker to copy our local files into the container’s working directory and to run npm to install any package dependancies.

On line 6, we tell docker to execute our app inside the container by using node to run our index.js file.

Last, on line 7, we setup the port that Docker will expose when the container is running. Port 8082 in our case.

NOTE: The Node.js app is using port 8081 but it will be using that port inside the Docker container when the container is running.

**7. Build a Docker image**

To build the Docker image, we use the docker build command with the -t parameter to tag our image with the name hello-world and we provide a reference to the path for the directory where our Dockerfile exists.

In our case, it’s in the same directory that I’m running the docker build command from so, I’ll just use a dot.

$ docker build -t hello-world .

**8. Run our Docker container**

OK, now that we have our Docker image built, we can use it to launch a container.

To do that, we use the docker run command with the name of the image we want to use.

We’ll also provide the dash ‘p’ parameter with the run command to map the ports that our Docker container and Node app will be using.

Remember, in our Dockerfile we exposed port 8082 but in our code, Node is using port 8081.

So, the run command will look like this.

$ docker run -p 8082:8081 hello-world

When you run that command Docker will show you the respone from Node saying the app is running on port 8081. However, that's inside the container. So to see the app running on your localhost you'll need to open port 8082 - the port Docker is exposing. So, [http://localhost:8082](http://localhost:8082/)

That's it! Your Node.js app is running in Docker.

**Clean up**

When you're done walking through this demo you can use the following steps to remove the hello-world container and image.

**1. Get the container id**

Use the docker ps command to list the running containers and copy the container id.

$ docker ps

**2. Stop the container**

$ docker stop your-container-id

**3. Remove the container**

$ docker rm your-container-id

**4. Remove the image**

$ docker rmi hello-world