# An example of using Dockerfile with a Python app

To get more practice with using a Dockerfile, in this part of the lab, you will create a Docker file to install a basic Python App.

1. **mkdir python-docker**
2. Change directoryto it and create a python file called `**coolApp.py**`

1. Enter the following in `**coolApp.py**`

**from flask import Flask**

**app = Flask(\_\_name\_\_)**

**@app.route('/home')**

**@app.route('/')**

**def home():**

**return "Hello world!"**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(port=5000, host='0.0.0.0', debug=True)**

1. Create a file called **Dockerfile** (no extension) in the same directory and add the following text:

**# Python as a base image**

FROM python:3.7

**# Create and set work directory as coolApp**

WORKDIR /coolApp

**# Execute a pip install command to install flask**

RUN pip install flask

**# Copies the python file into /coolApp**

COPY coolApp.py .

**# Documents the port to use**

EXPOSE 5000

**# What do you want the container to do when it starts? Main process**

ENTRYPOINT ["python", "coolApp.py"]

1. Build the image with   
   **docker build -t coolapp .**

(there is a dot at the end)

1. Tag the image with your username   
   **docker tag coolapp <username>/coolapp**

1. Push the image up to Docker Hub with   
   **docker push <username>/coolapp**

1. Run a container with   
   **docker run -d -p 5001:5000 --name *coolAppContainer* coolapp**(if you don’t set a name, Docker will give it a random name)
2. Test the setup in a browser  
   [**http://localhost:5001**](http://localhost:5001)

## **Make a change to the flask app**

1. Change the line **return "Hello world!"**to **return "<h1>Hello world!</h1>"**
2. Stop the coolApp docker using the **docker stop** command
3. Rebuild the app: **docker build -t coolapp .**
4. Run the app: **docker run -d -p 5001:5000 coolapp**
5. Test the new changes in a browser  
   **http://localhost:5001**

## Part 2: Install a docker for a Node server with MySQL Objectives:

In this part of the lab, you will create a more complex app with Node server which uses a local MySQL database for storing the tables in a database called **qa**. You will install the required MySQL server from Docker Hub. To test the API, you will also write a simple HTML page to execute a fetch command to call the server.  
  
**Please be aware that this is a new project in VS Code with a fresh directory structure. It's recommended to open a new instance of VS Code.**

1. Create a **new directory** for your code and then change directory to that folder and then start writing you app. You can choose any directory name you like.

**mkdir** myApp

**cd** myApp

1. Intitialise a new Node app. This command also creates the **package.json** file.  
   **npm init**  and then answer yes to all the questions by pressing enter. You will replace the generated lines later with a supplied package.json
2. Create a new file called **server.js** to host you Node server. We will write some JavaScript code later on in this lab.
3. Open a Terminal window in vs-code and then install the required packages for your project.

|  |  |
| --- | --- |
| **Express** | for creating the server and routing |
| **mysql2** | for interacting with MySQL. This module is mostly API compatible with Node MySQL and supports majority of features and offers better performance than **mysql** module |
| **body-parser** | used to get information from an incoming HTTP requests |
| **cors** | to enable you to define HTTP headers that specify which external clients' scripts can access your API. |

**npm install** express mysql2 body-parser cors

1. **Install MySQL server.**

In this part, you will run a docker command to host a MySQL server with the root user name of **root**  and password of **password123**.

**docker run --name mysql -e MYSQL\_ROOT\_PASSWORD=password123 -p 3306:3306 -d mysql:latest**

(type/copy all on one line)

1. In this step, you will create a database with a table for your newly created MySQL docker. Please follow these command to go into (exec) the docker and install a database:

**docker exec** -it mysql mysql -uroot -p  
  
Type **password123** when prompted for a password.   
(Chars will not be displayed)

You can now use MySQL’s CLI (command line interface).

**CREATE DATABASE qa;**

**USE qa;**

Execute the SQL code below to create a table called users:  
Tip: to save you typing, please copy from the line below.  
CREATE TABLE users (id INT AUTO\_INCREMENT PRIMARY KEY,username VARCHAR(50) NOT NULL,email VARCHAR(100) NOT NULL);

**CREATE TABLE users (**

**id** INT AUTO\_INCREMENT PRIMARY KEY,

**username** VARCHAR(50) NOT NULL,

**email** VARCHAR(100) NOT NULL

**);**

Let’s fill the users table with a user record **INSERT INTO** users (username, email) **VALUES**   
 ('john', 'john@qa.com');

Try getting data from the database by typing: **SELECT** \* **FROM** users**;  
  
A black background with white text

Description automatically generated**

You can now type **exit** to end the MySQL session**;**

## Node server code

Currently you're running MySQL in a Docker container and you will be trying to connect to it from another Docker container, you can't use localhost or 127.0.0.1 to connect to MySQL. This is because each Docker container has its own network stack, and localhost or 127.0.0.1 refers to the container itself.

To connect to MySQL from another Docker container, you need to use the IP address of the MySQL container.

Later on in this course we can put several dockers on the same network, but for now, please follow these instructions to get the IP address of your MySQL server.

**docker inspect mysql | findstr IPAddress**

Or for Linux:

**docker inspect mysql | grep IPAddress**

Here is an example of output (IP address may be different on your PC)

A black screen with white text

Description automatically generated

Copy the IP address and insert it in the code for your Node server as seen later.  
  
View the line in server.js file which you will create later:

**host: '<the IP address of your MySQL>',**

You will soon set the **host:** property to the above IP address

1. Write code for your Node server in the **server.js** file which you created earlier  
   In this code you will connect to the MySQL docker which you set up earlier and therefore you will need its IP address. There would be

**const express = require('express');**

**const bodyParser = require('body-parser');**

**const mysql = require('mysql2');**

**const cors = require("cors");**

**const app = express();**

**const port = 3000;**

**app.use(cors());**

**app.use(bodyParser.json());**

**const db = mysql.createConnection({**

**host: '<the IP address of your MySQL>',**

**user: 'root',**

**password: 'password123',**

**database: 'qa'**

**});**

**db.connect((err) => {**

**if (err) {**

**throw err;**

**}**

**console.log('Connected to MySQL database');**

**});**

**// API endpoint example:**

**app.get('/api/users', (req, res) => {**

**db.query('SELECT \* FROM users', (err, results) => {**

**if (err) {**

**throw err;**

**}**

**res.json(results);**

**});**

**});**

**app.listen(port, () => {**

**console.log(`Server is running on port ${port}`);**

**});**

* Create a file called **Dockerfile** in the same folder as **server.js**
* Include the following instructions for basing the image on node version 16 and installing the files on the Docker.

**FROM node:16**

**# Create app directory**

**WORKDIR /usr/src/app**

**# Install app dependencies**

**COPY package\*.json ./**

**RUN npm install**

**# Bundle app source**

**COPY . .**

**EXPOSE 3000**

**CMD [ "node", "server.js" ]**

* Create a **.dockerignore** (dot dockerignore) file in the same directory as your Dockerfile with the following content:

**node\_modules**

**npm-debug.log**

We will not copy the module files into the image because they can be created by the line **RUN npm install** in the Dockerfile.

* Type/copy the following to the **package.json** file:

**{**

**"name": "qa\_node\_sql\_app",**

**"version": "1.0.0",**

**"description": "A Node app using MySQL",**

**"author": "mike <mike@qa.com>",**

**"main": "server.js",**

**"scripts": {**

**"start": "node server.js"**

**},**

**"dependencies": {**

**"express": "^4.16.1",**

**"cors": "^2.8.5",**

**"mysql2": "^2.3.0"**

**}**

**}  
}**

### **Build the Docker image**

* Open a terminal window, navigate to the root directory of your Node.js application
* Run the following command to build the Docker image with the tag of   
  **<your dockerhub username>/qa-node-sql-app**   
    
  **docker build . -t <your Dockerhub username>/qa-node-sql-app**

Please note the dot ‘**.**’ In the middle of the command.  
  
This command will use the Dockerfile to build a Docker image of your Node.js application and tag it with the name that you specify.

### **Run the Docker container:**

* After the Docker image has been built, you can run a Docker container using the following command:   
  **docker run -p 3000:3000 <your Dockerhub username>/qa-node-sql-app**

This command will start a new Docker container based on the node-docker-app image and map port **3000 in the container** to port **3000 on your local machine** (the host).

So the format is **-p <host port>:<container port>**.

1. Create an HTML page to call your Node server. You may name this file as **index.html**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<title>API</title>**

**</head>**

**<body>**

**<script>**

**fetch('http://localhost:3000/api/users')**

**.then(response => response.json())**

**.then(data => {**

**// Handle the data returned from the server**

**console.log(data);**

**})**

**.catch(error => {**

**console.error('Error:', error);**

**});**

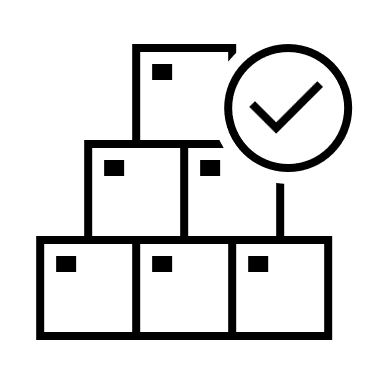
**</script>**

**</body>**

**</html>**

**Test the Docker container:**

* Browse the HTML page and look up values in the Console window.

Congratulations, you have successfully created a Docker to host a complex Node application that uses a database. This was a complex task which involved many  
 manual stages. We will simplify and automate the process in the later labs.