Real Time Group

RT Embedded Linux Solutions

Docker file Exercise

Exercise 1:
Create a basic Dockerfile that simply outputs "Hello, world!" to the console.

Exercise 2:
Modify your Dockerfile to set a working directory of /app.

Exercise 3:
Create a shell script named hello.sh that outputs "Hello, world!". Modify your Dockerfile to add this script to the container and run it.

Exercise 4:
Now change your Dockerfile to use the COPY instruction instead of ADD.

Exercise 5:
Add an environment variable NAME with the value "Docker" to your Dockerfile. Modify hello.sh to output "Hello, $NAME!".

Exercise 6:
Create a Docker container based on the Python image and run a script that displays "Hello, World!" in the terminal(CLI).

Exercise 7:
Mount a local directory as a volume inside a Docker container(CLI).

Exercise 8:
Set environment variables in a Docker container(CLI).

Exercise 9:
Rename a Docker container(CLI).

Exercise 10:
Copy files between the host machine and a Docker container.

Exercise 11:
Share data between containers using volumes.

Exercise 12
On the CLI, Create a Docker container from the official Ubuntu image and run the command echo "Hello, World!"

Exercise 13
On the CLI, Create a Docker container from the official Ubuntu image, start a shell session, and update the package list.

Exercise 14
Create a Dockerfile that creates an image for a Python application. The application should listen on port 5000, and the source code is located in the host's /app directory.(don't need real code, it's just for demonstration)

Exercise 15
Build an image from a Dockerfile and run a container from the image.

Exercise 16
Create a Docker container from the official redis image and run it in the background

Exercise 17
Create a Docker container from the official nginx image, serve a custom index.html file from the host's /app directory and map it to the container's /usr/share/nginx/html directory.

Exercise 18
Create a Docker container from the official node:14 image and run a simple NodeJS application that listens on port 3000 and prints "Hello World!" on the browser.

*Create a file called `app.js` with the following code :

```
var http = require('http');
var server = http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type': 'text/plain'});
res.end('Hello World\n');
});
server.listen(3000, '0.0.0.0');
console.log('Server running at http://0.0.0.0:3000/');
```

*To run this app you need to do: node app.js

Exercise 19
Create a Docker container that runs a simple Python application that listens on port 8080 and prints "Hello World!" on the browser.
*Create a file called `app.py` with the following code:

```
from flask import Flask
app = Flask(name)
@app.route("/")
def hello():
return "Hello World!"
if name == "main":
app.run(host='0.0.0.0', port=8080)
```

*Create a file called `requirements.txt` with the following:

```
flask
```

In the docker file you will need to install -r requirements.txt.

Exercise 20
Create Dockerfile with the following requirements:
Use the official Python base image.
Set the working directory in the container.
Copy the Python requirements file.
Install the Python dependencies.
Copy the Python application code.
Expose the port(5000) on which the Flask app will run.
Set the entry point command to run the Flask app using Gunicorn: gunicorn --bind 0.0.0.0:5000 app:app

Python file :

```
from flask import Flask, request
import requests

app = Flask(__name__)

@app.route('/')
def index():
    response = requests.get('https://example.com')
    return response.text

if __name__ == '__main__':
    app.run()
```

Requirements.txt :

```
flask==1.1.2
requests==2.24.0
gunicorn==20.0.4
```