Reading: Review of Docker Concepts and Understanding a Dockerfile

Estimated time needed: 30 minutes

This reading is designed to enhance your understanding of Docker concepts and guide you through the process of comprehending a Dockerfile.

Reading Overview:

- **Docker Concepts**
- **Understanding a Dockerfile**

Docker Concepts

Overview

Docker simplifies the process of creating, deploying, and managing applications by using containers. Containers allow you to package an application with its dependencies into a standardized unit for software development. Docker provides tools and a platform to build, ship, and run containers across various environments.

Dockerfile

A text file that contains instructions for building a Docker image. It specifies the base image, sets the working directory, installs dependencies, copies application code, exposes ports, and defines commands to run the application.

Container

An instance of a Docker image that runs as a process on the host machine. Containers are lightweight, portable, and isolated, making them ideal for deploying and scaling applications.

Docker Image Storage

Docker images are stored in registries, which can be public or private. Public registries like Docker Hub host millions of images, while organizations often use private registries for security and control.

Where Docker Images Exist

Local Machine: When you build a Docker image, it's initially stored locally on your machine. You can list local Docker images using the `docker images` command.

Registry: After building, you can push Docker images to a registry, making them accessible from anywhere with access to that registry.

Dockerfile

Below is a sample Dockerfile for reference. The subsequent explanation details the commands used, providing a comprehensive understanding of how to write a Dockerfile.

▶ Dockerfile

Understanding the Dockerfile

1: Specify the Base Image

To begin, specify the base image using the `FROM` instruction. In this case, the official Node.js image version 14 is used.

```
FROM node:14
```

`FROM`: Specifies the base image for the Docker container.
`node:14`: Pulls the official Node.js image with version 14 from the Docker registry.

2: Set Environment Variables

Next, define the necessary environment variables with the `ENV` instruction. Here, `NODE_ENV` is set to production and `PORT` is set to 3000.

```
ENV NODE_ENV=production
ENV PORT=3000
```

`ENV`: Sets environment variables inside the Docker container.
`NODE_ENV=production`: Sets the Node.js environment to production.
`PORT=3000`: Sets the port on which the Node.js application will listen.

3: Set the Working Directory

Then, specify the working directory inside the container using the `WORKDIR` instruction. This sets /app as the working directory.

```
WORKDIR /usr/src/app
```

`WORKDIR`: Defines the working directory for subsequent instructions, simplifying file path references.
`/usr/src/app`: The directory inside the container to store your application code, keeping it organized and easily accessible.

4: Copy Package Files

After defining the working directory, use the `COPY` instruction to copy the `package.json` and `package-lock.json` files into the container. These files are essential for installing the application dependencies.

```
COPY package*.json ./
```

`COPY`: Copies files from your local machine to the container.
`package*.json ./`: Copies both `package.json` and `package-lock.json` files

5: Install Dependencies

With the package files copied, execute the `RUN` instruction to install the production dependencies listed in `package.json`.

```
RUN npm install --production
```

- **RUN:** Executes commands in the container during the build process.
- **npm install –production:** Installs dependencies listed in `package.json` without devDependencies.

6: Copy Application Code

Once the dependencies are installed, copy the rest of the application code into the container using the `COPY` instruction.

```
COPY . .
```

`COPY . .`: Copies all files and directories from the current directory on the local host to the current directory in the Docker container.

7: Add Additional File(s)

Additionally, use the `ADD` instruction to include any extra files. Here, the index.html file is added to the public directory.

```
# ADD <source_path> <destination_path>
ADD public/index.html /app/public/index.html
```

- `public/index.html`: Path to the file or directory on the host machine.
- `/app/public/index.html`: Path where you want to add the file or directory inside the Docker image.

8: Expose the Application Port

Next, inform Docker that the container listens on the specified port with the `EXPOSE` instruction. Here, it exposes the port defined by the `PORT` environment variable.

```
EXPOSE $PORT
```

- `EXPOSE`: Informs Docker that the container will listen on the specified port at runtime.
- `$PORT`: The environment variable representing the port number defined earlier.

9: Specify the Default Command

Define the default command to run when the container starts using the `CMD` instruction. Here, it runs node app.js.

```
CMD ["node", "app.js"]
```

`CMD`: This instruction specifies the default command and/or parameters to execute at the entry point of the container.
`["node", "app.js"]`: This array specifies the command to run any arguments for that command. In this case, it tells Docker to execute the node command with `app.js` as the argument.

Note: The filename `app.js` is commonly used as the main file for Node.js applications. However, depending on your project structure or naming conventions, the main file could have a different name.

10: Label the Image

Then, add metadata to the image with the `LABEL` instruction, including the version, description, and maintainer information.

```
LABEL version="1.0"
LABEL description="Node.js application Docker image"
LABEL maintainer="Your Name"
```

`LABEL`: Adds metadata to the Docker image.
`version="1.0"`: Specifies the version of the Docker image.
`description="Node.js application Docker image"` : Provides a description of the Docker image.
`maintainer="Your Name"`: Specifies the maintainer of the Docker image.

11: Add a Healthcheck

For ensuring the container is running correctly, define a health check using the `HEALTHCHECK` instruction. This command checks the application's health by sending a request to the specified port.

```
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 CMD curl -fs http://localhost:$PORT || exit 1
```

`HEALTHCHECK`: Configures a health check to ensure the container is running correctly.
`--interval=30s`: Specifies the interval between health checks.
`--timeout=10s`: Sets the timeout for each health check.

`--start-period=5s`: Defines the start period during which the container must initialize before health checks begin.
`--retries=3`: Sets the number of retries before considering the container unhealthy.
`CMD curl -fs http://localhost:$PORT || exit 1`: Specifies the command to run for health checks. It checks if a request to `http://localhost:$PORT` succeeds; otherwise, it exits with code 1, indicating failure.

12: Set a Non-Root User

Finally, for security purposes, set a non-root user with the `USER` instruction. Here, the user is set to node

```
USER node
```

`USER`: Sets the user that will run the subsequent instructions in the Dockerfile.
`node`: Specifies the user named `node` to run the commands for security purposes.

This step-by-step explanation shows the flow of creating a Dockerfile, highlighting how each instruction builds upon the previous one to create a fully functional Docker image.

Summary

In this reading, you learned about Docker concepts and the process of creating a Dockerfile. The provided Dockerfile and the outlined steps demonstrate how to build a Docker image for a Node.js application. It starts with the official Node.js base image, sets up environment variables, defines a working directory, and installs the necessary dependencies. The application code is then copied into the container, and a health check is included to ensure the container runs correctly. Finally, a non-root user is set for enhanced security.

Author: Nikesh Kumar

Skills Network