EXPERIMENT 10

AIM: Working with .dockerignore

Perform Docker Link and .dockerignore based scenario on Katacoda.

Working with .dockerignore

Step 1 - Docker Ignore

To prevent sensitive files or directories from being included by mistake in images, you can add a file named *.dockerignore*.

Example

A Dockerfile copies the working directory into the Docker Image. As a result, this would include potentially sensitive information such as a passwords file which we'd want to manage outside the image. View the Dockerfile with `cat Dockerfile`

Build the image with `docker build -t password .`

Look at the output using `docker run password ls /app`

This will include the passwords file.

Ignore File

The following command would include passwords.txt in our *.dockerignore* file and ensure that it didn't accidentally end up in a container. The *.dockerignore* file would be stored in source control and share with the team to ensure that everyone is consistent.

```
echo passwords.txt >> .dockerignore
```

The ignore file supports directories and Regular expressions to define the restrictions, very similar to *.gitignore*. This file can also be used to improve build times which we'll investigate in the next step.

Build the image, because of the Docker Ignore file it shouldn't include the passwords file.

```
docker build -t nopassword .
```

Look at the output using `docker run nopassword ls /app`

Protip

If you need to use the passwords as part of a *RUN* command then you need to copy, execute and delete the files as part of a single RUN command. Only the final state of the Docker container is persisted inside the image.

Step 2 - Docker Build Context

The *.dockerignore* file can ensure that sensitive details are not included in a Docker Image. However they can also be used to improve the build time of images.

In the environment, a 100M temporary file has been created. This file is never used by the *Dockerfile*. When you execute a build command, Docker sends the entire path contents to the Engine for it to calculate which files to include. As a result sending the 100M file is unrequired and creates a slower build.

You can see the 100M impact by executing following the command.

```
docker build -t large-file-context .
```

In the next step, we'll demonstrate how to improve the performance of the build.

Protip

It's wise to ignore *.git* directories along with dependencies that are downloaded/built within the image such as *node_modules*. These are never used by the application running within the Docker Container and just add overhead to the build process.

Step 3 - Optimised Build

In the same way, we used the *.dockerignore* file to exclude sensitive files, we can use it to exclude files which we don't want to be sent to the Docker Build Context during the build.

Optimizing

To speed up our build, simply include the filename of the large file in the ignore file.

```
echo big-temp-file.img >> .dockerignore
```

When we rebuild the image, it will be much faster as it doesn't have to copy the 100M file.

```
docker build -t no-large-file-context .
```

This optimisation has a greater impact when ignoring large directories such as *.git*.