

General instructions

Before proceeding, please fill the “Demographic Information” and “Skills and Experience” sections of the questionnaire linked below. Once you reach the “DF1” section, read the rest of this document.

<https://forms.gle/fXuQu3euNfGgic769>

In this experiment you will perform 3 tasks. In each task you must edit a Dockerfile using Visual Studio Code. To help you in these tasks, you’ll have access to a set of extensions which will provide feedback while you edit the Dockerfiles. The combination of Visual Studio Code with these extensions will be referred to as the “environment” in this document and in the questionnaire. The features available in these extensions are described in the next sections of this document.

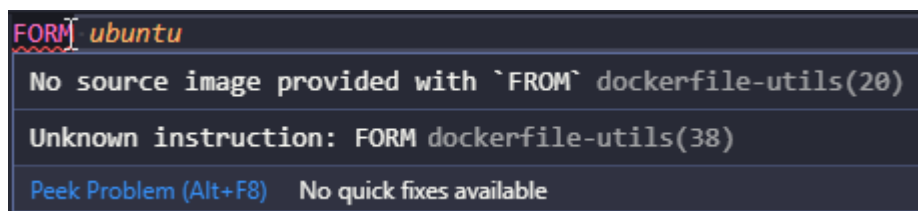
You have 15 minutes to read and understand the features described in this document. You are allowed to consult this document at any point during the task execution.

Available Features

This environment includes a VSCode extension which automatically compiles an image from the Dockerfile while it is being edited, raises a container with that image and executes some verifications. The result of those tests are shown to the developer in VSCode as explained in the following paragraphs.

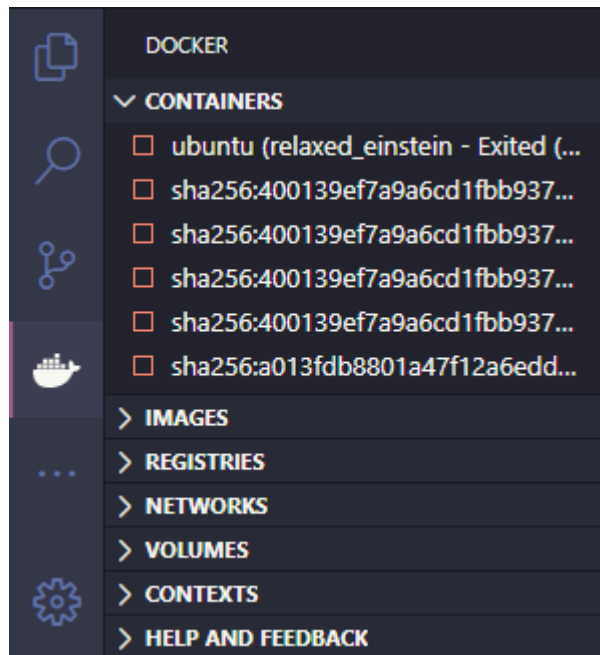
1- Static Analysis Errors

This environment analyses the Dockerfile’s syntax, underlining any syntax errors that it finds.



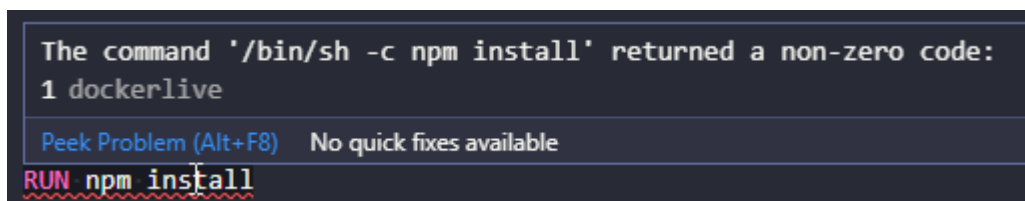
2- Docker Management

This environment provides a sidebar menu which allows the programmer to perform quick actions related to Docker containers, images, registries, networks, volumes and contexts (e.g. start container, stop container, view container logs, delete image).



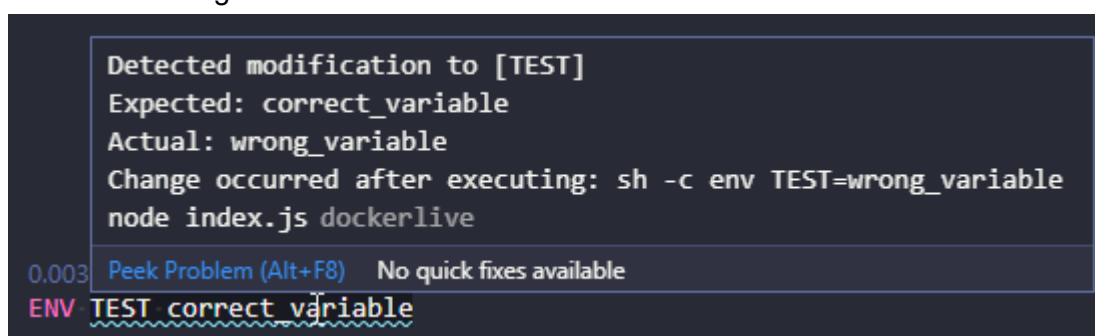
3- Image Build and Container Runtime Errors

Instructions which cause errors while building the image or instantiating the image in a container are underlined in **red**. Hovering the underlined instruction shows more details about the error.



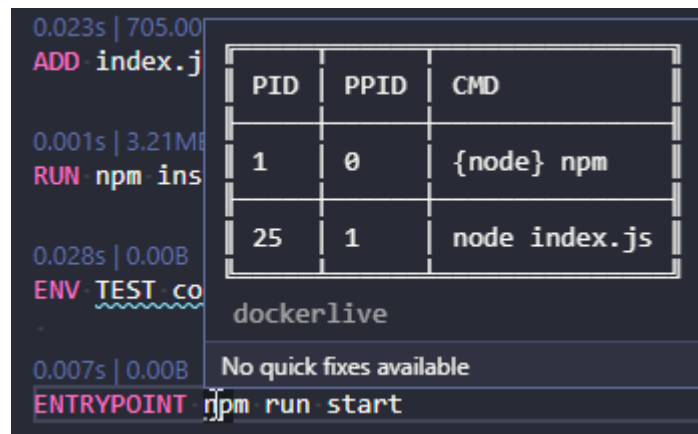
4- Changes to environment variables

ENV instructions are underlined in **blue** when the value that they set is changed during the container's execution. Hovering the underlined instruction shows more details, including the process which changed the environment variable.



5- Processes running in the container

By hovering over the ENTRYPOINT/CMD instruction you can see the processes running in the container.



PID	PPID	CMD
1	0	{node} npm
25	1	node index.js

dockerlive

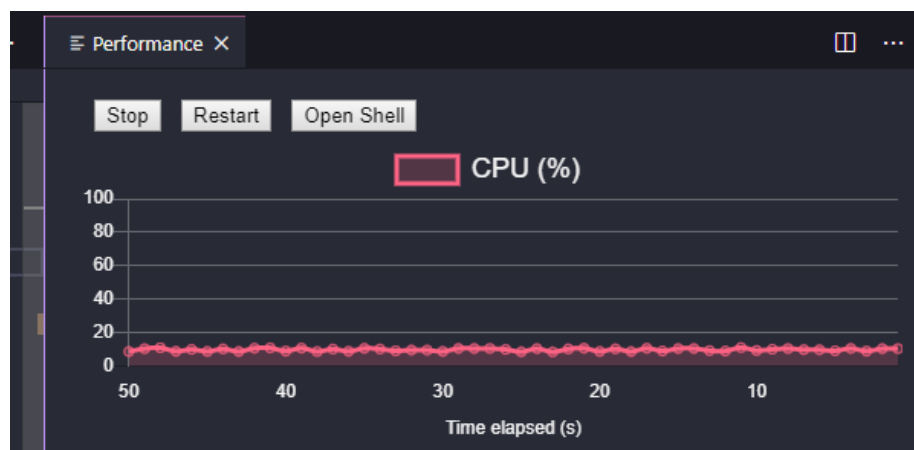
No quick fixes available

ENTRYPOINT npm run start

6- Container performance statistics



You can visualize performance statistics of the container by clicking the “CPU” button, located in the upper right corner of the editor tab which contains the Dockerfile. By clicking this button, a new tab with performance graphs will be visible. If the graphs are updated every second, then the container is running. If the graphs are stopped, then the container is stopped. When a new container starts (by editing the Dockerfile or pressing the restart button) data is erased from the graphs.



On this page there are also 3 buttons available:

- Stop - Stops the running container

- Restart - Restarts/Starts the container
- Open Shell - Open an interactive shell inside the container

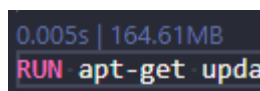
7- Base image OS information

By hovering the image name on the FROM instruction, you can see some information about the OS that the container is running (including the OS Family, Version and Kernel version).



8- Layer Size and Build Time

Above every instruction you can see a small text indicating the time it took to build that layer and the size that the layer takes.



9- Explore each layer's filesystem



You can explore each layer's filesystem by clicking in the "FS" button, located in the upper right corner of the editor tab which contains the Dockerfile. By clicking this button a new tab will allow the developer to navigate through the aggregated filesystem of each of the layers.

On the top of the table there is a dropdown which allows you to select the layer that is currently being displayed. Files which were created/edited/removed in the selected layer are marked with a yellow square.

Layer ID: 4862f462c0c7 - 333.49KB ▾ Up Down

C	Type	Size	Mode	UID	GID	Name
	directory	75 items	rwxr-xr-x	0	0	▶ bin
	directory	0.00B	rwxr-xr-x	0	0	▶ boot
	directory	0.00B	rwxr-xr-x	0	0	▶ dev
	directory	72 items	rwxr-xr-x	0	0	▶ etc
	directory	1 item	rwxr-xr-x	0	0	▶ home
	directory	6 items	rwxr-xr-x	0	0	▶ lib
	directory	1 item	rwxr-xr-x	0	0	▶ lib64
	directory	0.00B	rwxr-xr-x	0	0	▶ media
	directory	0.00B	rwxr-xr-x	0	0	▶ mnt
	directory	0.00B	rwxr-xr-x	0	0	▶ opt
	directory	0.00B	rwxr-xr-x	0	0	▶ proc
	directory	2 items	rwX-----	0	0	▶ root
	directory	2 items	rwxr-xr-x	0	0	▶ run
	directory	77 items	rwxr-xr-x	0	0	▶ sbin
	directory	0.00B	rwxr-xr-x	0	0	▶ srv
	directory	0.00B	rwxr-xr-x	0	0	▶ sys
	directory	0.00B	rwXrwxrwx	0	0	▶ tmp
	directory	8 items	rwxr-xr-x	0	0	▶ usr
	directory	11 items	rwxr-xr-x	0	0	▶ var

You can expand/collapse folders by clicking on their name.

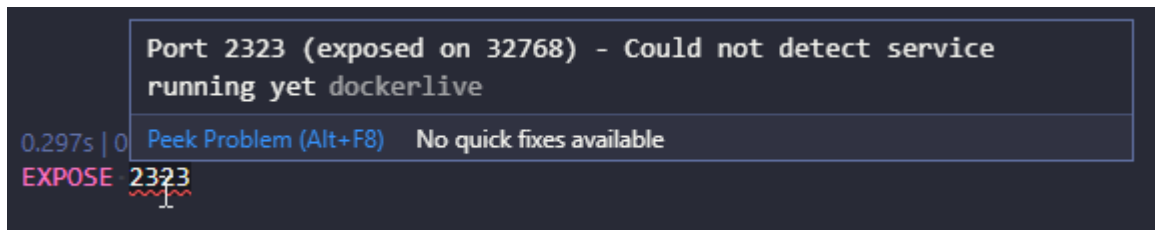
By hovering the permissions, you can see a small window which will help you interpret the permissions for each of the entities. These are in the UNIX permissions format.

directory	8 items	rwxr-xr-x	0	0	▶ usr
directory	11 items	rwxr-xr-x			

	Owner	Group	Other
Read	Yes	Yes	Yes
Write	Yes	No	No
Exec	Yes	Yes	Yes

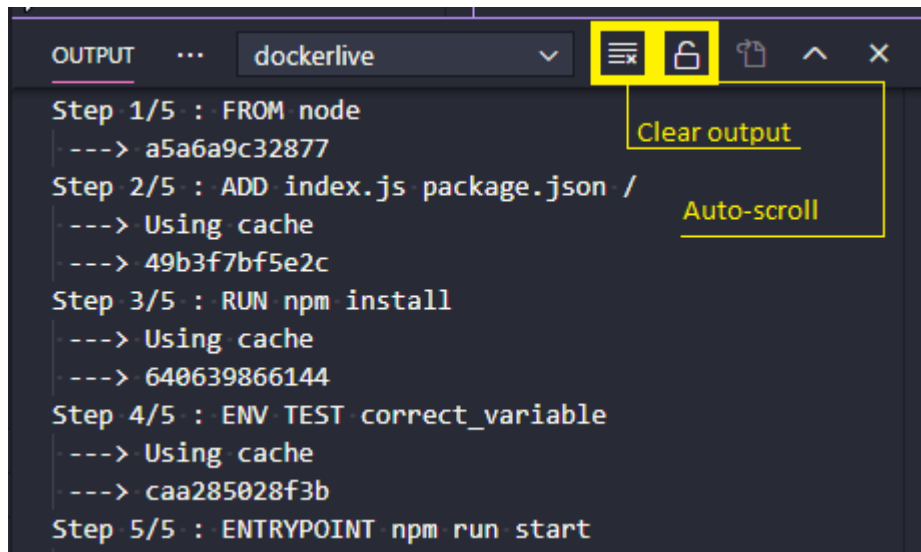
10- Service discovery

This environment will automatically try to detect any services running on ports that are exposed with the EXPOSE instruction. If no service is detected, an error underline will be displayed. By hovering over the port number on the instruction, you can see the name and protocol of the detected service.



11- Container log output

The output of the docker build and the docker container is displayed in the Output pane in VSCode. This pane opens automatically when a Dockerfile is opened and may also be opened in the VSCode top bar (View -> Output). It's advised to enable auto-scrolling (open padlock icon, as displayed below).



Tasks

Global Rules

- In each task, you'll be given a Dockerfile which you must edit until the container has the desired behaviour.
- Each task ends once you notify the experiment observer that you have reached the desired behaviour.
- **You may only edit the Dockerfile.** No other files (such as .py or .js) need to be edited in order to achieve the desired behaviour. However, you are allowed to make temporary changes to the code (e.g. print a variable or comment a line) if you think it may help you diagnose the issue. Note that if you make a temporary change to the code you must restore the code to its original state for the task to be validated.
- You have a maximum of 20 minutes to complete each task.
- Feel free to consult this document at any time.
- All the information that you need to solve the tasks is present in this document. However, feel free to consult any documentation and perform any web searches you may need, at any time, **in the remote computer where you're performing the tasks.**
- If something isn't clear in these instructions or in the descriptions of the tasks themselves please alert the experiment observer immediately.

Instructions

- In the C:/Users/DockerliveTest/Desktop folder, you'll find 3 folders - "DF1", "DF2", "DF3". Each of these folders contains a task.
- Do these 3 tasks in order and fill the respective section of the questionnaire after each task.
- Instructions for each of the tasks are available in the remaining sections of this document.
- At the end of each task please run the following command on a powershell window:
docker-cleanup-script
- You can now read the instructions and start the task "DF1".

DF1

1. Open Visual Studio Code using the shortcut in the desktop.
2. In Visual Studio Code open the "DF1" folder using the menu "File" -> "Open Folder..."
3. Edit the Dockerfile until all of the following properties are true (Note that you must verify all of these conditions):
 - Container stdout must show: "some data"
 - 'node' process must be running in container
4. Alert the experiment observer once you've met the criteria in point 3.
5. Copy, paste and execute the following command on a powershell window:
docker-cleanup-script

6. Fill the "DF1" section of the questionnaire.
7. Move to the next task.

DF2

1. In Visual Studio Code open the "DF2" folder using the menu "File" -> "Open Folder..."
2. Edit the Dockerfile until all of the following properties are true (Note that you must verify all of these conditions):
 - Container stdout must NOT show: "Error downloading file" nor "Error writing file"
 - Container stdout must show: "Success!"
 - Container must download 9MB-15MB of data
3. Alert the experiment observer once you've met the criteria in point 2.
4. Copy, paste and execute the following command on a powershell window:
docker-cleanup-script
5. Fill the "DF2" section of the questionnaire.
6. Move to the next task.

DF3

1. In Visual Studio Code open the "DF3" folder using the menu "File" -> "Open Folder..."
2. Edit the Dockerfile until all of the following properties are true (Note that you must verify all of these conditions):
 - Container stdout must show: "Listening!"
 - Container stdout must not show: "Could not bind"
 - Container must have a TCP service running on the exposed port 3000
3. Alert the experiment observer once you've met the criteria in point 2.
4. Copy, paste and execute the following command on a powershell window:
docker-cleanup-script
5. Fill the remaining sections of the questionnaire.