

Dockerfile –partie 2

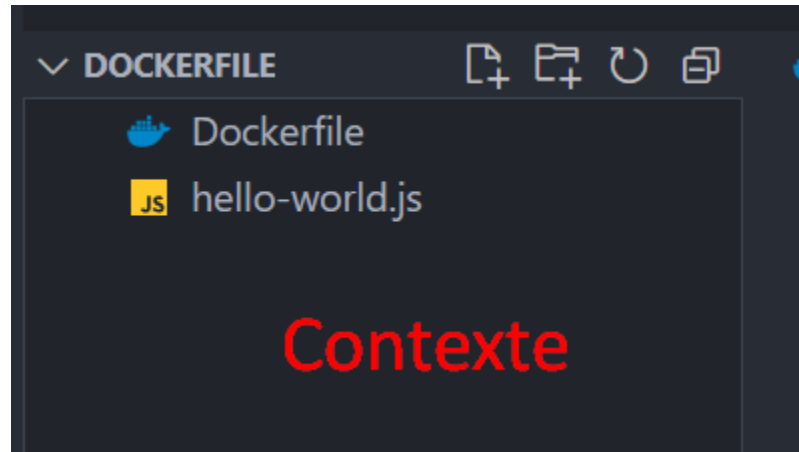
Introduction

Sources :

<https://docs.docker.com/engine/reference/builder/#dockerfile-reference>

<https://docs.docker.com/build/building/packaging/>

Le contexte :



Seuls les fichiers présents dans le contexte de départ sont accessibles par Docker.

Les commentaires :

- Ils ne sont pas traités lors du build.

Très pratiques / utiles pour :

- ✓ Aider vos collaborateurs / vous à comprendre le code
- ✓ Désactiver certaines exécutions sans les supprimées.

```
3 # installation de l'environnement Node
4 RUN apt-get update \
5     && apt-get install -y ca-certificates curl gnupg \
6     && curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key |
7     apt-keyrings/nodesource.gpg \
8     && echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource.com
9     nodistro main" | tee /etc/apt/sources.list.d/nodesource.list \
10    && apt-get update
11 # && apt-get install nodejs -y
```

COMMENTAIRES

L'instruction ADD :

- Alternative à l'instruction COPY.

COPY et ADD sont deux commandes Dockerfile qui ont la même finalité :

- permettre de copier des fichiers dans une image Docker.

- ✓ COPY prend en compte une source et une destination.

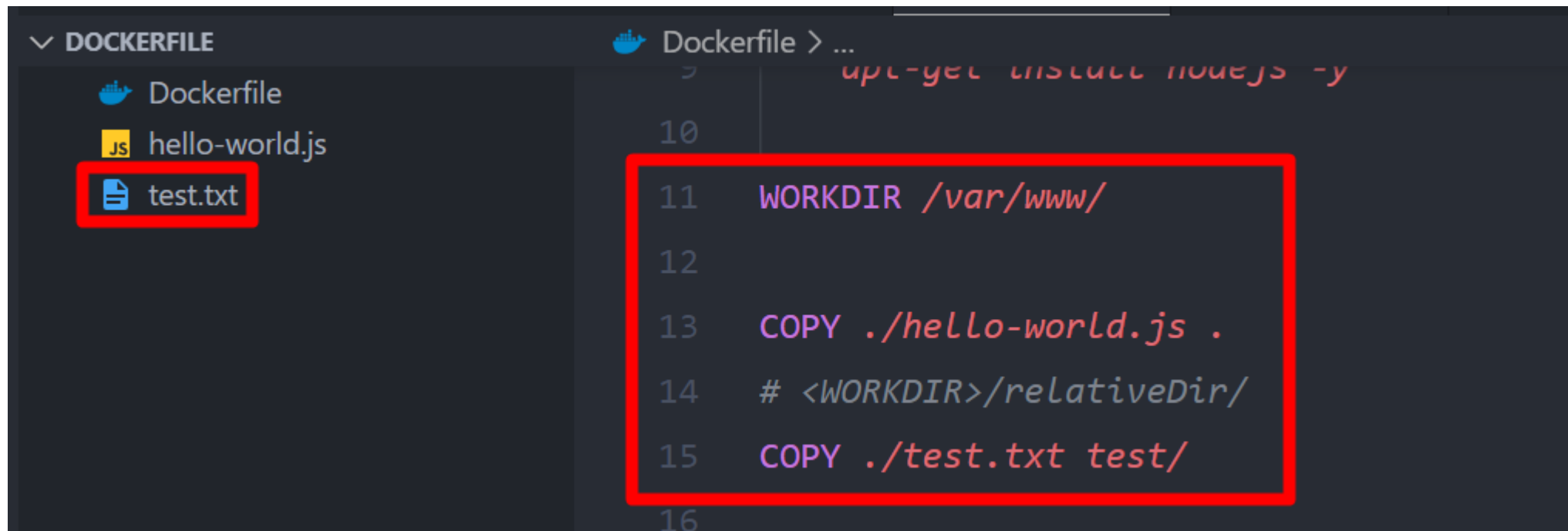
Elle permet de copier un fichier ou un répertoire local de la machine qui crée l'image Docker dans l'image Docker.

- ✓ ADD vous permet la même chose,

mais également de prendre en charge des sources distantes telles que des URLs ou un fichier archive qui sera extraite automatiquement dans l'image Docker.

L'instruction WORKDIR :

- Elle permet de définir le répertoire de travail dans le conteneur.
 - ✓ Permet de ne pas répéter le dossier sur plusieurs instructions
 - ✓ Ouvre le bash directement dans le dossier



The screenshot shows a Dockerfile editor with a sidebar on the left and a main editor area. The sidebar, titled 'DOCKERFILE', contains three items: 'Dockerfile' (with a Docker icon), 'hello-world.js' (with a JS icon), and 'test.txt' (with a document icon). The 'test.txt' item is highlighted with a red rectangular box. The main editor area shows the content of the Dockerfile, with line numbers 10 through 16 visible. A red rectangular box highlights the following lines:

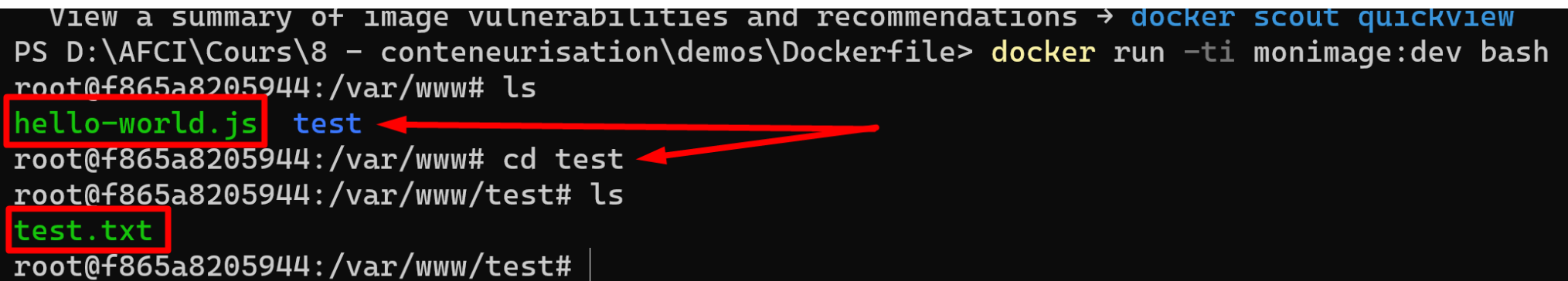
```
11 WORKDIR /var/www/  
12  
13 COPY ./hello-world.js .  
14 # <WORKDIR>/relativeDir/  
15 COPY ./test.txt test/  
16
```


Vérifions :

```
PS D:\AFCI\Cours\8 - conteneurisation\demos\Dockerfile> docker build -t monimage:dev .
[+] Building 0.7s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 635B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [1/5] FROM docker.io/library/ubuntu:latest@sha256:8eab65df33a6de2844c9aefd19efe8ddb87b7df5e9
=> [internal] load build context
=> => transferring context: 118B
=> CACHED [2/5] RUN apt-get update && apt-get install -y ca-certificates curl gnupg &&
=> CACHED [3/5] WORKDIR /var/www/
=> CACHED [4/5] COPY ./hello-world.js .
=> [5/5] COPY ./test.txt test/
=> exporting to image
=> => exporting layers
=> => writing image sha256:989833bb25bb7a3e6ab3de4929e75c30d7423ac60ee6ae7d13636d870208c7e4
=> => naming to docker.io/library/monimage:dev
```

Vérifions :

```
View a summary of image vulnerabilities and recommendations → docker scout quickview  
PS D:\AFCI\Cours\8 - conteneurisation\demos\Dockerfile> docker run -ti monimage:dev bash  
root@f865a8205944:/var/www# ls  
hello-world.js test  
root@f865a8205944:/var/www# cd test  
root@f865a8205944:/var/www/test# ls  
test.txt  
root@f865a8205944:/var/www/test# |
```



WORKDIR étant défini, nous pouvons également modifier le CMD :

```
15
```

```
16  CMD ["node", "hello-world.js"]
```

Vérifions :

```
PS D:\AFCI\Cours\8 - conteneurisation\demos\Dockefile> docker build -t monimage:dev .
[+] Building 0.9s (10/10) FINISHED
=> [internal] load build definition from Dockefile
=> => transferring dockefile: 661B
=> [internal] load .dockignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [1/5] FROM docker.io/library/ubuntu:latest@sha256:8eab65df33a6de2844c9aefd19efe8ddb8
=> [internal] load build context
=> => transferring context: 111B
=> CACHED [2/5] RUN apt-get update && apt-get install -y ca-certificates curl gnupg
=> CACHED [3/5] WORKDIR /var/www/
=> CACHED [4/5] COPY ./hello-world.js .
=> [5/5] COPY ./test.txt test/
=> exporting to image
=> => exporting layers
=> => writing image sha256:1364819f377dfa7765904716f9b79fd93e6a1ebecc3881cc52679f920e84
=> => naming to docker.io/library/monimage:dev
```

What's Next?

View a summary of image vulnerabilities and recommendations → `docker scout quickview`

```
PS D:\AFCI\Cours\8 - conteneurisation\demos\Dockefile> docker run monimage:dev
```

```
Hello World
```

```
PS D:\AFCI\Cours\8 - conteneurisation\demos\Dockefile> |
```

CMD vs ENTRYPOINT

commandes alternatives

Le CMD – pourquoi ? :

- ✓ Il permet à l'utilisateur de passer des commandes alternatives au lancement du conteneur

Prend la main sur le CMD.

```
View a summary of image vulnerabilities and recommendations - Docker Scout quickview
PS D:\AFCI\Cours\8 - conteneurisation\demos\Dockerfile> docker run monimage:dev
Hello World
PS D:\AFCI\Cours\8 - conteneurisation\demos\Dockerfile> docker run monimage:dev echo "autre chose"
autre chose
PS D:\AFCI\Cours\8 - conteneurisation\demos\Dockerfile> |
```

L'instruction ENTRYPOINT :

- ✓ Empêche le passage d'arguments par l'utilisateur

```
16  
17  # CMD ["node", "hello-world.js"]  
18  ENTRYPOINT ["node", "hello-world.js"]
```

Vérifions :

```
PS D:\AFCI\Cours\8 - conteneurisation\demos\Dockefile> docker build -t monimage:dev .
[+] Building 1.6s (11/11) FINISHED                                docker:default
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                  0.0s
=> [internal] load build definition from Dockerfile             0.0s
=> => transferring dockerfile: 702B                             0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest 1.4s
=> [auth] library/ubuntu:pull token for registry-1.docker.io   0.0s
=> [1/5] FROM docker.io/library/ubuntu:latest@sha256:8eab65df33a6de2844c9aefd19efe8ddb87b7df5e9185a4ab73af936225 0.0s
=> [internal] load build context                                0.1s
=> => transferring context: 111B                                 0.0s
=> CACHED [2/5] RUN apt-get update && apt-get install -y ca-certificates curl gnupg && curl -fsSL https: 0.0s
=> CACHED [3/5] WORKDIR /var/www/                               0.0s
=> CACHED [4/5] COPY ./hello-world.js .                         0.0s
=> CACHED [5/5] COPY ./test.txt test/                           0.0s
=> exporting to image                                           0.0s
=> => exporting layers                                          0.0s
=> => writing image sha256:9dab44c96ba2afb0a3f3996c48233b85e829477e413149f0f61975d0ab182262 0.0s
=> => naming to docker.io/library/monimage:dev                 0.0s
```

What's Next?

View a summary of image vulnerabilities and recommendations → `docker scout quickview`

```
PS D:\AFCI\Cours\8 - conteneurisation\demos\Dockefile> docker run monimage:dev
```

Hello World

```
PS D:\AFCI\Cours\8 - conteneurisation\demos\Dockefile> docker run monimage:dev echo "autre chose"
```

Hello World

CMD – ENTRYPOINT (intérêt) :

✓ Les 2 instructions sont cummulables.

Permet des configurations avancées ...

Exemple 1 (rebuilder l'image) :

```
17  ENTRYPOINT ["node"]
18  CMD ["hello-world.js"]
```

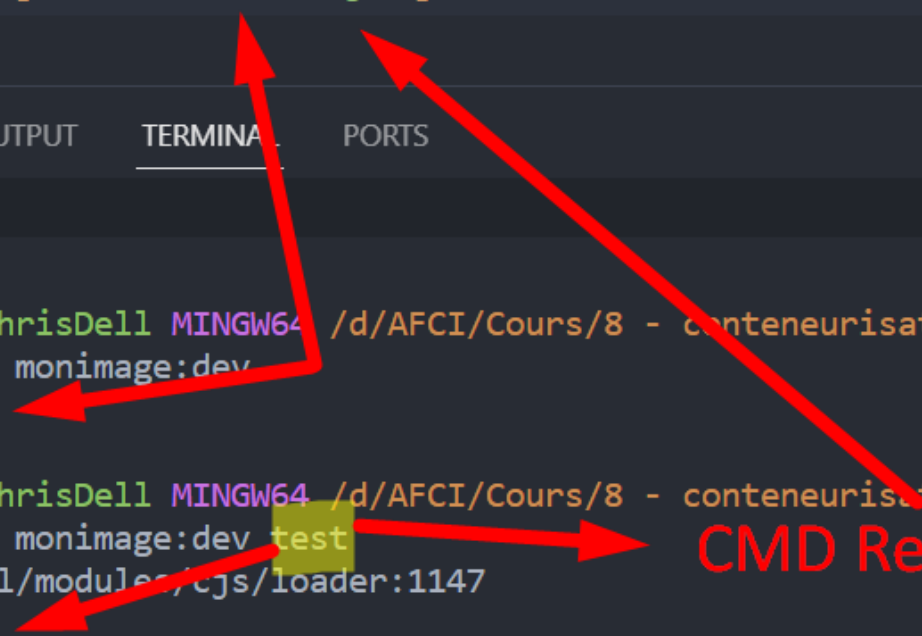
PROBLEMS OUTPUT TERMINAL PORTS

✓ **TERMINAL**

```
Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/demos/Dockerfile
$ docker run monimage:dev
Hello World

Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/demos/Dockerfile
$ docker run monimage:dev test
node:internal/modules/cjs/loader:1147
  throw err;
  ^
```

CMD Remplacée



Exemple 2 (rebuilder l'image) :

```
17 ENTRYPOINT ["echo"]
```

```
18 CMD ["Bonjour"]
```

PROBLEMS

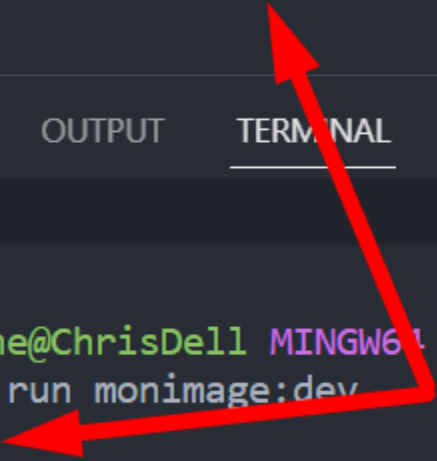
OUTPUT

TERMINAL

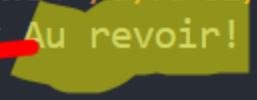
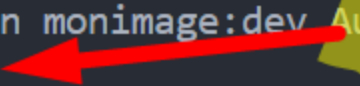
PORTS

✓ **TERMINAL**

```
Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/demos/Dockerfile
$ docker run monimage:dev
Bonjour
```



```
Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/demos/Dockerfile
$ docker run monimage:dev Au revoir!
Au revoir!
```



Questions ??

Prochain chapitre: Le Dockerfile – partie 3.