

Dockerfile –partie 1

Introduction

Sources :

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

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

Docker peut créer des images automatiquement en lisant les instructions d'un Dockerfile.

- ✓ Un Dockerfile est un document texte qui contient toutes les commandes qu'un utilisateur devrait taper en ligne de commande pour assembler une image.

Il s'écrit sous se format :

```
# Comment  
INSTRUCTION arguments
```

- ✓ L'instruction n'est pas sensible à la casse.

Cependant, la convention veut qu'ils soient en MAJUSCULES pour les distinguer plus facilement des arguments.

- ✓ Docker exécute les instructions dans un Dockerfile dans l'ordre.
- ✓ Un Dockerfile doit commencer par une instruction FROM.

L'instruction FROM spécifie l'image parent à partir de laquelle vous construisez votre image.

Créons une image simple :

- ✓ Un environnement NodeJs pou exécuter un fichier JS.

Le dokerfile étant un simple fichier, sans extension, basé sur le système « clé:valeur »

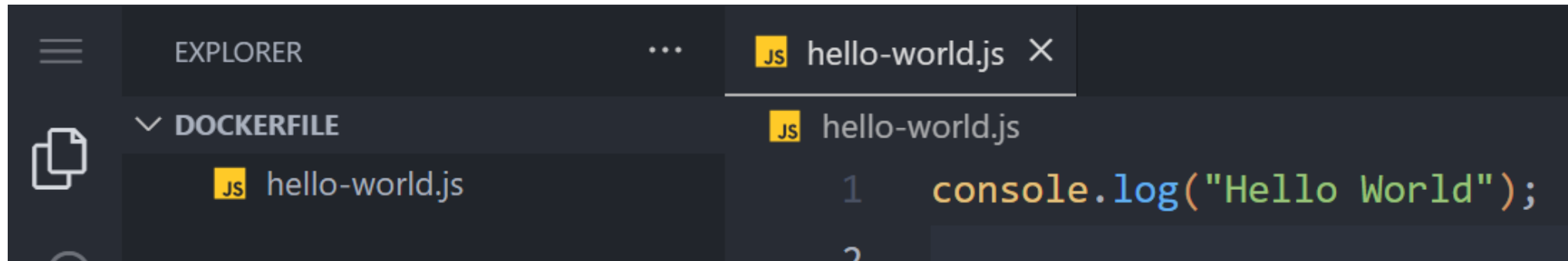
- ✓ Mettons en place une extension utile dans notre éditeur préféré : VsCode

L'extension :

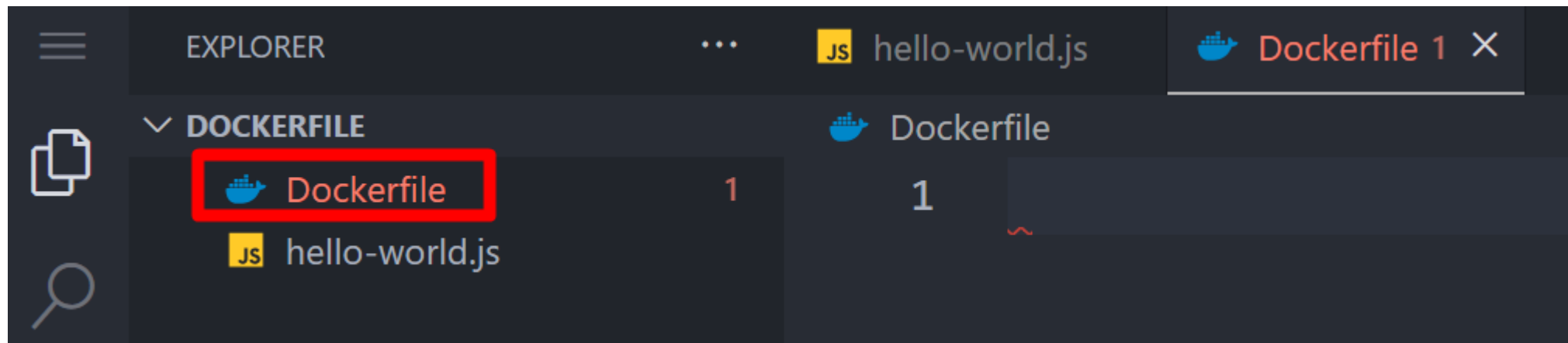


Source : <https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscod-docker>

Le projet :



This screenshot shows the VS Code interface. The Explorer sidebar on the left displays a folder named 'DOCKERFILE' containing a file 'hello-world.js'. The Editor pane on the right shows the content of 'hello-world.js', which contains a single line of JavaScript code: `console.log("Hello World");`. The file icon in the Explorer is a document with a yellow 'JS' tag.

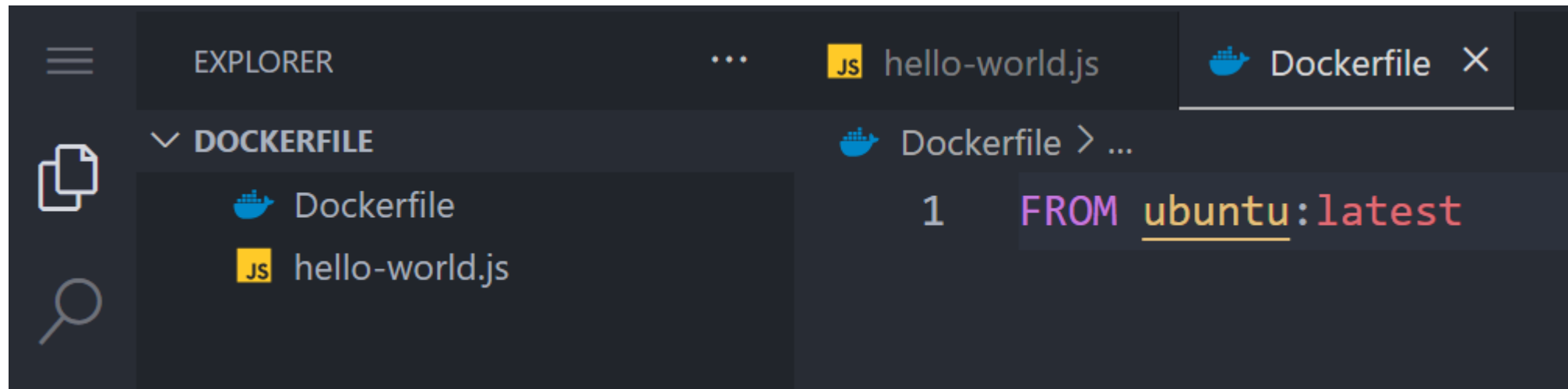


This screenshot shows the VS Code interface after creating a Dockerfile. The Explorer sidebar on the left shows the 'DOCKERFILE' folder containing two files: 'Dockerfile' and 'hello-world.js'. The 'Dockerfile' file is highlighted with a red rectangle. The Editor pane on the right shows the content of 'Dockerfile', which is currently empty. The file icon for 'Dockerfile' in the Explorer is a blue Docker logo.

Les instructions

Base de travail :

✓ Puller l'image ubuntu officielle



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar is open, showing a folder named 'DOCKERFILE' containing two files: 'Dockerfile' (with a Docker icon) and 'hello-world.js' (with a JS icon). The main editor area has two tabs: 'hello-world.js' (JS) and 'Dockerfile' (Docker icon). The 'Dockerfile' tab is active, showing a single line of code: 'FROM ubuntu:latest'. The word 'FROM' is in pink, 'ubuntu' is in yellow and underlined, and ':latest' is in red. The line number '1' is visible to the left of the code.

- Tag : latest => optionnel si pas de version précisée.

✓ Installation de l'environnement node :

Source : <https://deb.nodesource.com/> | <https://github.com/nodesource/distributions>

Install Node.js 20.x Current

Copy

```
~$ sudo apt-get update && sudo apt-get install -y ca-certificates curl gnupg
~$ curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | sudo
gpg --dearmor -o /etc/apt/keyrings/nodesource.gpg
~$ NODE_MAJOR=20
~$ echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg]
https://deb.nodesource.com/node_${NODE_MAJOR}.x nodistro main" | sudo tee
/etc/apt/sources.list.d/nodesource.list
~$ sudo apt-get update && sudo apt-get install nodejs -y
```

Après quelques modifications :

```
3 RUN apt-get update \  
4     && apt-get install -y ca-certificates curl gnupg \  
5     && curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | gpg --dearmor -o /etc/apt/keyrings/nodesource.gpg \  
6     && echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource.com/node\_20.x nodistro main" | tee /etc/apt/  
sources.list.d/nodesource.list \  
7     && apt-get update \  
8     && apt-get install nodejs -y
```

Ce sont les mêmes commandes que celles fournies par la doc officielle, simplement réadaptées au format dockerfile....

La commande pour construire une image :

```
docker build -t node:dev .
```

✓ L'argument -t permet de donner un nom à votre image Docker.

Cela permet de retrouver plus facilement votre image par la suite.

✓ :tag (cf. :dev) ajoute un tag à notre image (facultatif)

✓ Le . est le répertoire où se trouve le Dockerfile

dans notre cas, à la racine de notre projet.

Testons en lançant un build notre image :

```
Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/demos/Dockerfile
$ docker build -t node:dev .
[+] Building 11.3s (11/11) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 467B                                0.0s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                       0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest   0.5s
=> [1/7] FROM docker.io/library/ubuntu:latest@sha256:8eab65df33a6de2844c9aefd19efe8ddb87b7d 0.0s
=> CACHED [2/7] RUN apt-get update                                0.0s
=> CACHED [3/7] RUN apt-get install -y ca-certificates curl gnupg 0.0s
=> CACHED [4/7] RUN curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | 0.0s
=> [5/7] RUN echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource. 0.2s
=> [6/7] RUN apt-get update                                       1.7s
=> [7/7] RUN apt-get install nodejs -y                           8.2s
=> exporting to image                                             0.8s
=> => exporting layers                                             0.8s
=> => writing image sha256:615eff1b09e020148b1ee4f72cb86fbca27d34e9628f67cfc10db6e18bae7c24 0.0s
=> => naming to docker.io/library/node:dev                        0.0s
```

What's Next?

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

```
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 467B
=> [internal] load .dockerignore
=> => transferring context: 2B
```

CONTEXT

➤ Le contexte de build est l'ensemble des fichiers auxquels votre build peut accéder.

L'argument positionnel que vous transmettez à la commande build spécifie le contexte que vous souhaitez utiliser pour la build :

- ✓ Le chemin relatif ou absolu vers un répertoire local (cf '.')
- ✓ Une URL distante d'un référentiel Git, d'une archive tar ou d'un fichier en texte brut
- ✓ Un fichier texte brut ou une archive tar transmis à la commande docker build via l'entrée standard


```
=> [1/7] FROM docker.io/library/ubuntu:latest@sha256:8eab65df33a6de2844c9aefd19efe8ddb87b7d 0.0s
=> CACHED [2/7] RUN apt-get update 0.0s
=> CACHED [3/7] RUN apt-get install -y ca-certificates curl gnupg 0.0s
=> CACHED [4/7] RUN curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | 0.0s
=> [5/7] RUN echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource. 0.2s
=> [6/7] RUN apt-get update 1.7s
=> [7/7] RUN apt-get install nodejs -y 8.2s
=> exporting to image 0.8s
=> => exporting layers 0.8s
```

LAYERS

➤ L'ordre des instructions Dockerfile est important.


Une build Docker se compose d'une série d'instructions de build ordonnées.

Chaque instruction d'un Dockerfile se traduit par une couche d'image :

✓ Layers

Gestion du cache optimisation

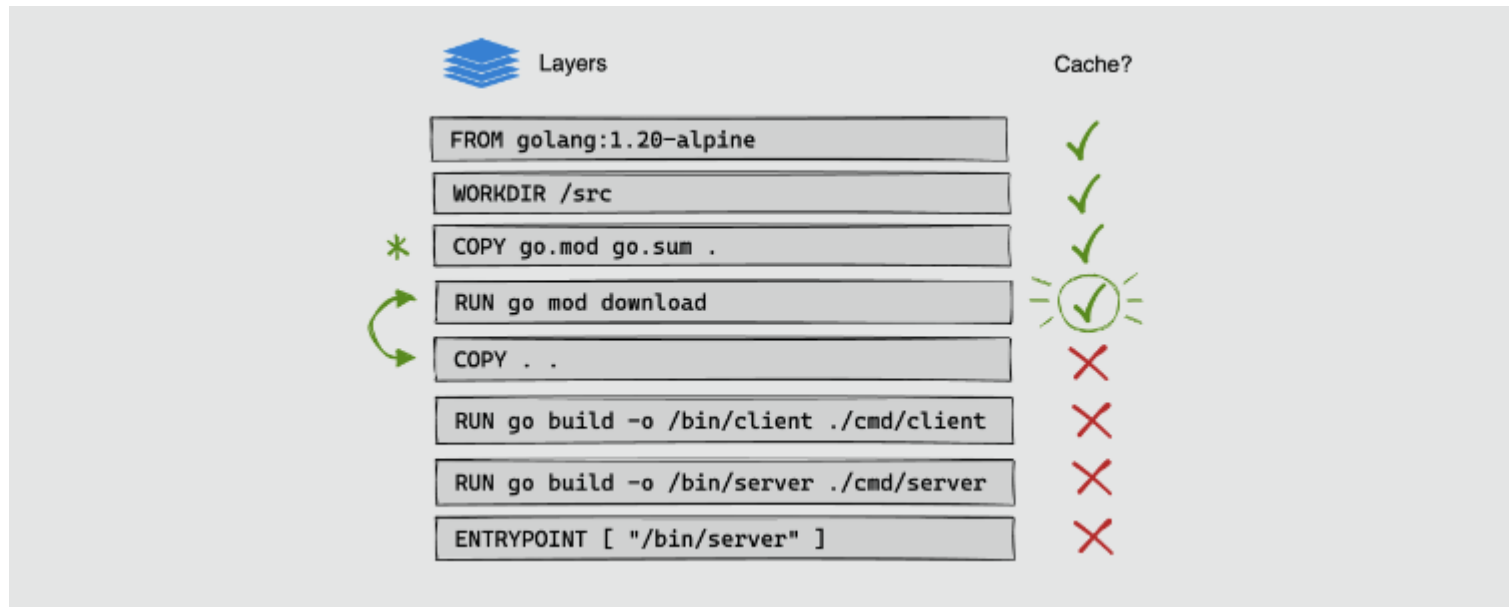
Ce dockerfile peut être optimisé :

 Layers	Cache?
FROM golang:1.20-alpine	✓
WORKDIR /src	✓
COPY . .	✗
RUN go mod download	✗
RUN go build -o /bin/client ./cmd/client	✗
RUN go build -o /bin/server ./cmd/server	✗
ENTRYPOINT ["/bin/server"]	✗

Lorsque vous exécutez une build, le générateur tente de réutiliser les couches des builds précédentes.

✓ Si une couche d'une image reste inchangée :

le constructeur la récupère dans le cache de construction. Si un calque a changé depuis la dernière génération, ce calque et tous les calques qui suivent doivent être reconstruits.



Retour projet ...

Vérifions l'installation de NodeJs dans un conteneur basé sur notre image :

```
Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - connected to docker isolation
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
node                 dev                 615eff1b09e0       10 seconds ago    327MB
nginx                latest              a6bd71f48f68       2 weeks ago       187MB
```

```
Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - co
$ docker run -ti node:dev ← 1
root@92dcd127dea9:/# node -v ← 2
v20.10.0 ←
root@92dcd127dea9:/#
```

Copie de notre fichier local dans l'image :

```
5  RUN curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo  
   keyrings/nodesource.gpg  
6  RUN echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https  
   main" | tee /etc/apt/sources.list.d/nodesource.list  
7  RUN apt-get update  
8  RUN apt-get install nodejs -y  
9  
10 COPY ./hello-world.js /var/www/  
11  
12
```

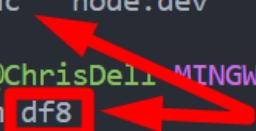
Vérifions :

✓ Nettoyage

```
Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/demos/Dockerfile
$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
df88f41a9bdc   node:dev   "/bin/bash"             3 minutes ago  Exited (0)    14 seconds ago
heuristic_mcclintock
```

Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/demos/Dockerfile
\$ docker rm df8

Les 3 premiers suffisent



✓ Rebuild de l'image (nécessaire à chaque modification!)

```
Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/demos/Dockerfile
$ docker build -t node:dev .
[+] building 1.5s (14/14) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 506B
=> [internal] load .dockerignore
```


Création d'un conteneur (mode terminal interactif):

```
Christophe@ChrisDell MTN64 /d/AFCI/Cours/8 - conteneurisation/demos/Dockerfile
$ docker run -ti node:dev
root@b046dfe2def9:/#
```

Vérification de la copie du fichier :

```
$ docker run -ti node:dev
root@b046dfe2def9:/# cd var/www/ && ls
hello-world.js
root@b046dfe2def9:/var/www#
```

Exécution du fichier

L'instruction :

```
main" | tee /etc/apt/sources.list.d/nodesource.list
7 RUN apt-get update --fix-missing
8 RUN apt-get install -y nodejs
9
10 COPY . /usr/src/app
11
12 CMD ["node", "index.js"]
```

^
1/2
v

CMD ["executable", "parameter", ...]

The default executable for this executing container.

Set the default executable and parameters for this executing container.

- ✓ Executable : NodeJs pour nous...
- ✓ Paramètre : notre fichier à exécuter dans le conteneur.

Notre dockerfile :

```
7  RUN apt-get update
8  RUN apt-get install nodejs -y
9
10 COPY ./hello-world.js /var/www/
11
12 CMD ["node", "/var/www/helo-world.js"]
```

Vérifions :

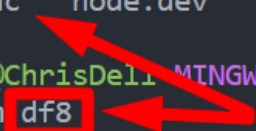
✓ Nettoyage

```
Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/demos/Dockerfile
$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS          NAMES
df88f41a9bdc   node:dev   "/bin/bash"             3 minutes ago Exited (0) 14 seconds ago

```

Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/demos/Dockerfile
\$ docker rm df8

Les 3 premiers suffisent

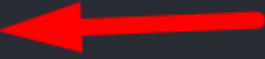


✓ Rebuild de l'image (nécessaire à chaque modification!)

```
Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/demos/Dockerfile
$ docker build -t node:dev .
[+] building 1.5s (14/14) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 506B
=> [internal] load .dockerignore
```

Création d'un conteneur (mode détaché) :

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



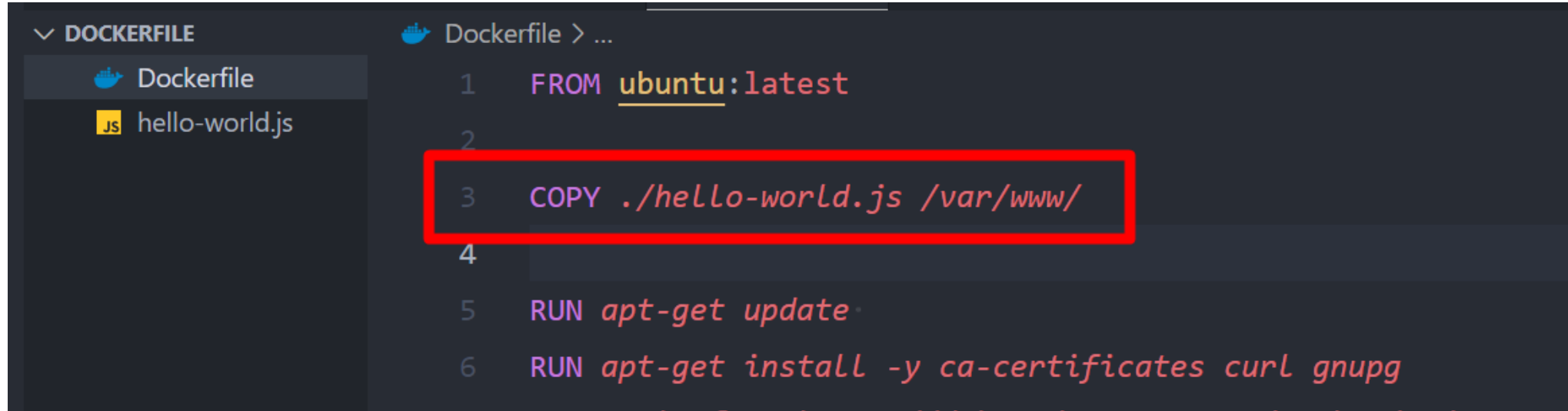
➤ Le fichier est bien exécuté.

Retour optimisation CACHE

Problème :

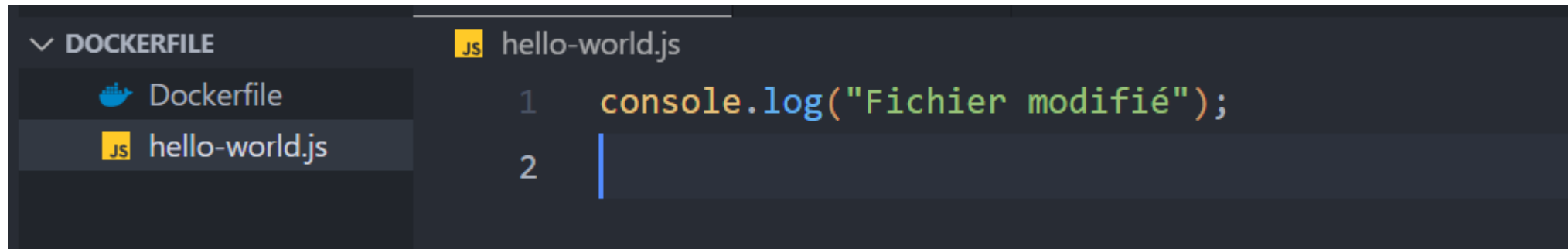
```
$ docker build -t node:dev .
[+] Building 1.0s (14/14) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 545B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [auth] library/ubuntu:pull token for registry-1.docker.io
=> [1/8] FROM docker.io/library/ubuntu:latest@sha256:8eab65df33a6de2844c9aefd19efe8ddb87b7df5e9185a4ab73af936225685bb
=> [internal] load build context
=> => transferring context: 35B
=> CACHED [2/8] RUN apt-get update
=> CACHED [3/8] RUN apt-get install -y ca-certificates curl gnupg
=> CACHED [4/8] RUN curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | gpg --dearmor -o /etc/apt/keyrings/
=> CACHED [5/8] RUN echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource.com/node_20.x nodistro main"
=> CACHED [6/8] RUN apt-get update
=> CACHED [7/8] RUN apt-get install nodejs -y
=> CACHED [8/8] COPY ./hello-world.js /var/www/
=> exporting to image
=> => exporting layers
```


Modifions nos fichiers en local et reconstruisons l'image :



The screenshot shows a code editor with a sidebar on the left containing a 'DOCKERFILE' section with 'Dockerfile' and 'hello-world.js' (marked with a JS icon). The main editor area shows the Dockerfile content with line numbers 1 through 6. Line 3, 'COPY ./hello-world.js /var/www/', is highlighted with a red rectangular box. The other lines are: 1 FROM ubuntu:latest, 2 (empty), 4 (empty), 5 RUN apt-get update, and 6 RUN apt-get install -y ca-certificates curl gnupg.

```
1 FROM ubuntu:latest
2
3 COPY ./hello-world.js /var/www/
4
5 RUN apt-get update
6 RUN apt-get install -y ca-certificates curl gnupg
```



The screenshot shows a code editor with a sidebar on the left containing a 'DOCKERFILE' section with 'Dockerfile' and 'hello-world.js' (marked with a JS icon). The main editor area shows the content of 'hello-world.js' with line numbers 1 and 2. Line 1 contains the code 'console.log("Fichier modifié");'. Line 2 is empty with a blue cursor at the start.

```
1 console.log("Fichier modifié");
2
```

Résultat :

✓ Nos run sont réexécutés

```
Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/demos/Dockerfile
$ docker build -t node:dev .
[+] Building 35.7s (14/14) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 547B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [auth] library/ubuntu:pull token for registry-1.docker.io
=> [internal] load build context
=> => transferring context: 35B
=> CACHED [1/8] FROM docker.io/library/ubuntu:latest@sha256:8e1b65df33a6de2844c9aefd19efe8ddb87b7df5e9185a4ab73
=> [2/8] COPY ./hello-world.js /var/www/
=> [3/8] RUN apt-get update
=> [4/8] RUN apt-get install -y ca-certificates curl gnupg
=> [5/8] RUN curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | gpg --dearmor -o /etc/apt/key
=> [6/8] RUN echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource.com/node_20.x nodine
```

Refaire la manipulation en :

1. Remettant la copie en avant dernier
2. Reconstruisant l'image
3. Modifiant le fichier js
4. Reconstruisant l'image...

```
=> CACHED [5/8] RUN echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource.com/no
=> CACHED [6/8] RUN apt-get update
=> CACHED [7/8] RUN apt-get install nodejs -y
=> [8/8] COPY ./hello-world.js /var/www/
=> exporting to image
=> => exporting layers
=> => writing image sha256:4c350b342303abcf49961b264d5767b4b9706396d147f3acada232ab9e7cf377
=> => naming to docker.io/library/node:dev
```

What's Next?

Questions ??

Prochain chapitre: Le Dockerfile – partie 2.