

Docker-compose

Partie 1

Introduction

Sources :

<https://docs.docker.com/compose/>

Compose est un outil permettant de définir et d'exécuter des applications Docker multi-conteneurs :

- ✓ Vous utilisez un fichier YAML pour configurer les services de votre application.

Ensuite, avec une seule commande :

- vous créez et démarrez tous les services de votre configuration.

Compose fonctionne dans tous les environnements :

- ✓ production,
- ✓ préparation,
- ✓ développement,
- ✓ tests,
- ✓ ainsi que les flux de travail CI.

Compose dispose également de commandes pour gérer l'ensemble du cycle de vie de votre application :

- ✓ Démarrer, arrêter et reconstruire les services,
- ✓ Afficher l'état des services en cours d'exécution,
- ✓ Diffusez la sortie du journal des services en cours d'exécution,
- ✓ Exécuter une commande ponctuelle sur un service.

Installation

Docker Desktop :

Le moyen le plus simple et recommandé d'obtenir Docker Compose consiste à installer Docker Desktop qui inclut :

- ✓ Docker Compose,
- ✓ Docker Engine,
- ✓ et Docker CLI.

qui sont des prérequis pour Compose.

Docker Desktop :

Si vous avez déjà installé Docker Desktop, vous pouvez vérifier de quelle version de Compose vous disposez en CLI :

```
C:\Users\conta>docker-compose version  
Docker Compose version v2.23.0-desktop.1
```

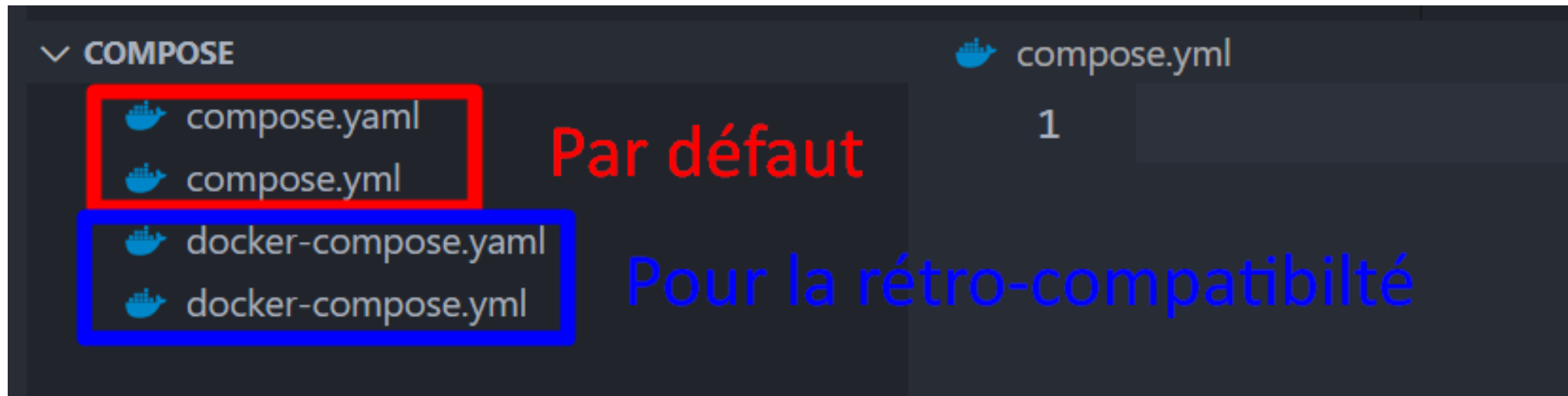
Installation manuelle :

<https://docs.docker.com/compose/install/>

Les bases

Le chemin par défaut d'un fichier Compose est `compose.yaml` (préfér ) ou `compose.yml` qui est plac  dans le r pertoire de travail.

- ✓ Compose prend  galement en charge `docker-compose.yaml` et `docker-compose.yml` pour la compatibilit  ascendante des versions ant rieures.
- Si les deux fichiers existent, Compose pr f re le canonique `compose.yaml`.



Structure du fichier :

```
1 version: 'version'
2
3 services:
4   name:
5   image: imageName
6   volumes:
7     - volume-name:containerPath:ro
8
9
10
11 ... Indentations
```

Système de clé:valeur

Indentations

Le contenu

Version :

La propriété de version de niveau supérieur est définie par la spécification Compose pour une compatibilité ascendante.

- C'est seulement informatif.

Compose n'utilise pas la version pour sélectionner un schéma exact afin de valider le fichier Compose, mais préfère le schéma le plus récent lorsqu'il est implémenté.

Compose valide s'il peut analyser complètement le fichier Compose.

- Si certains champs sont inconnus, généralement parce que le fichier Compose a été écrit avec des champs définis par une version plus récente de la spécification, vous recevrez un message d'avertissement.

Version :

<https://docs.docker.com/compose/compose-file/compose-versioning/#compatibility-matrix>

[https://github.com/docker/compose/blob/v1/docs/Compose%20file%20reference%20\(legacy\)/version-3.md](https://github.com/docker/compose/blob/v1/docs/Compose%20file%20reference%20(legacy)/version-3.md)

```
C:\Users\conta>docker version
Client:
  Cloud integration: v1.0.35+desktop.5
  Version:          24.0.6
  API version:      1.43
```

| Compose file format | Docker Engine release |
|-----------------------|-----------------------|
| Compose specification | 19.03.0+ |
| 3.8 | 19.03.0+ |
| 3.7 | 18.06.0+ |
| 3.6 | 18.02.0+ |

Version :

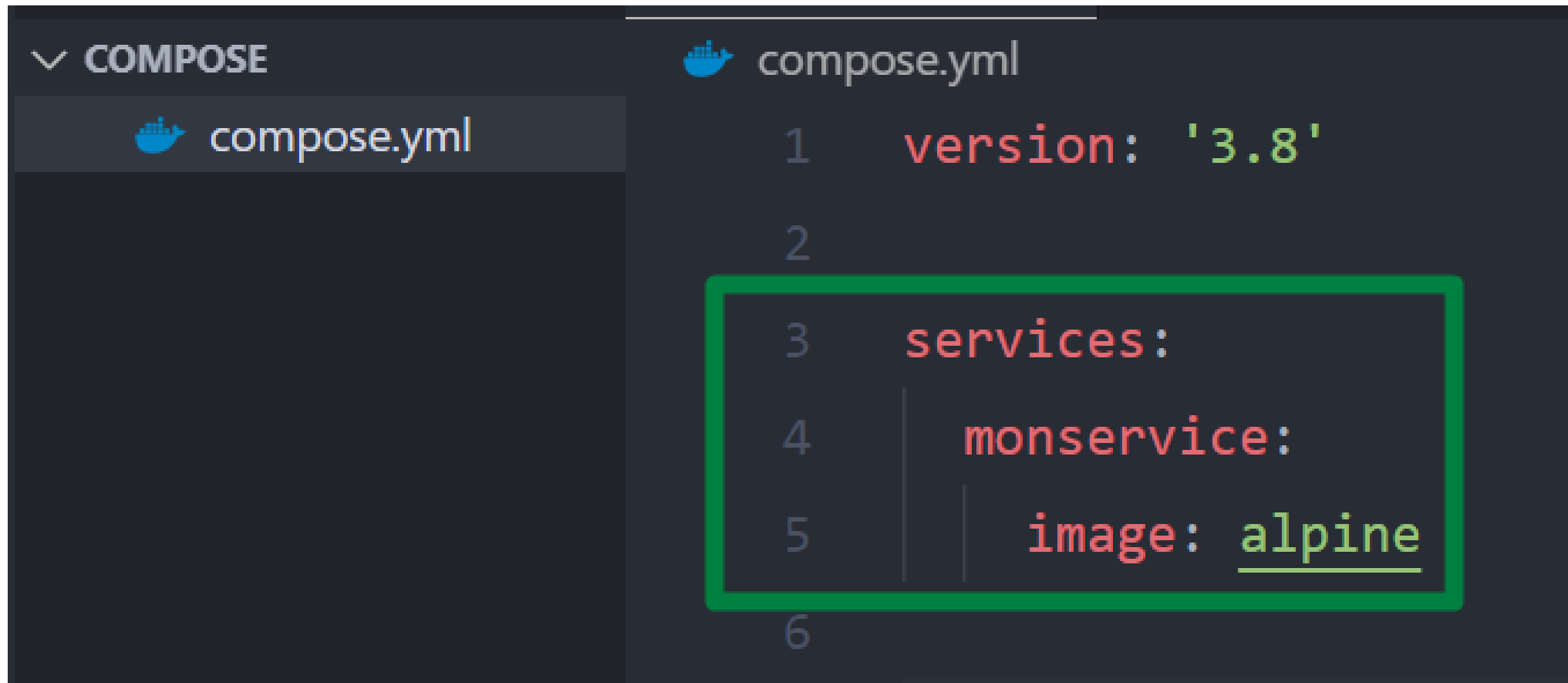
| Compose file format | Docker Engine release |
|-----------------------|-----------------------|
| Compose specification | 19.03.0+ |
| 3.8 | 19.03.0+ |
| 3.7 | 18.06.0+ |
| 3.6 | 18.02.0+ |

```
COMPOSE
compose.yml
compose.yml
1 version: '3.8'
2
```

Services :

Dans une application Docker distribuée, différentes parties de l'application sont appelées **services**.

➤ Les services ne sont en réalité que des conteneurs.



```
1  version: '3.8'
2
3  services:
4    monservice:
5      image: alpine
6
```

Services - test :

La commande pour notre composition

```
CHRISTOPHE@CHRISDELL MINGW64 /d/AFC1/Cours/8 - conteneurisation/cours/C
$ docker-compose up
[+] Running 2/2
  ✓ monservice 1 layers [::]      0B/0B      Pulled
    ✓ 661ff4d9561e Pull complete
[+] Building 0.0s (0/0)
[+] Running 2/2
  ✓ Network compose_default      Created
  ✓ Container compose-monservice-1 Created
Attaching to compose-monservice-1
compose-monservice-1 exited with code 0
```

Que c'est-il passé ?

L'image alpine est PULL

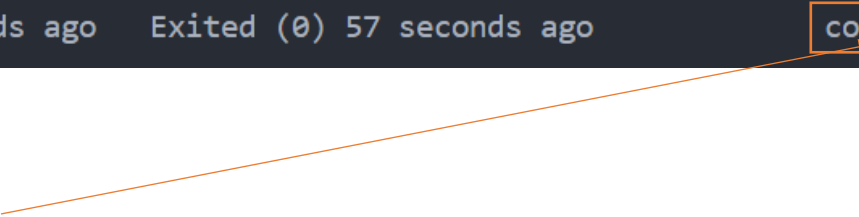
```
$ docker image ls
```

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|------------|--------|--------------|-------------|--------|
| alpine | latest | f8c20f8bbcb6 | 2 weeks ago | 7.38MB |

Un container est créé

```
$ docker container ls -a
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|--------|-----------|----------------|---------------------------|-------|----------------------|
| 680b902ea7a4 | alpine | "/bin/sh" | 59 seconds ago | Exited (0) 57 seconds ago | | compose-monservice-1 |



Nom_du_repertoire-nom_du_service

Commandes utiles

Quelques interactions :

✓ Mode interactif

```
$ docker-compose run monservice
[+] Building 0.0s (0/0)
[+] Building 0.0s (0/0)
/ # ls
bin  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

```
$ docker-compose run monservice
[+] Building 0.0s (0/0)
[+] Building 0.0s (0/0)
/ # ls
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS |
|--------------|-------------------------------------|-----------|----------------|--------------|
| 6abac61da5bf | alpine | "/bin/sh" | 33 minutes ago | Up 33 minute |
| s | compose-monservice-run-627bad4d55ea | | | |

Quelques interactions :

- ✓ Nettoyage

```
$ docker-compose down
```

```
[+] Running 2/2
```

```
✓ Container compose-monservice-1 Removed
```

```
✓ Network compose_default Removed
```

```
$ docker ps -a
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|-------|---------|---------|--------|-------|-------|
|--------------|-------|---------|---------|--------|-------|-------|

Quelques interactions :

✓ Lancer une commande

➤ command: command

```
3  services:
4      monservice:
5          image: alpine
6          command: ["ls"]
```

PROBLEMS OUTPUT **TERMINAL** PORTS

TERMINAL

Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneuri

\$ docker-compose up

[+] Running 2/2

✓ monservice 1 layers [::] 0B/0B Pulled

✓ 661ff4d9561e Pull complete

[+] Building 0.0s (0/0)

[+] Running 2/2

✓ Network compose_default Created

✓ Container compose-monservice-1 Created

Attaching to compose-monservice-1

compose-monservice-1 | bin

compose-monservice-1 | dev

compose-monservice-1 | etc

compose-monservice-1 | home

compose-monservice-1 | lib

compose-monservice-1 | media

compose-monservice-1 | mnt

compose-monservice-1 | opt

compose-monservice-1 | proc

compose-monservice-1 | root

Quelques interactions :

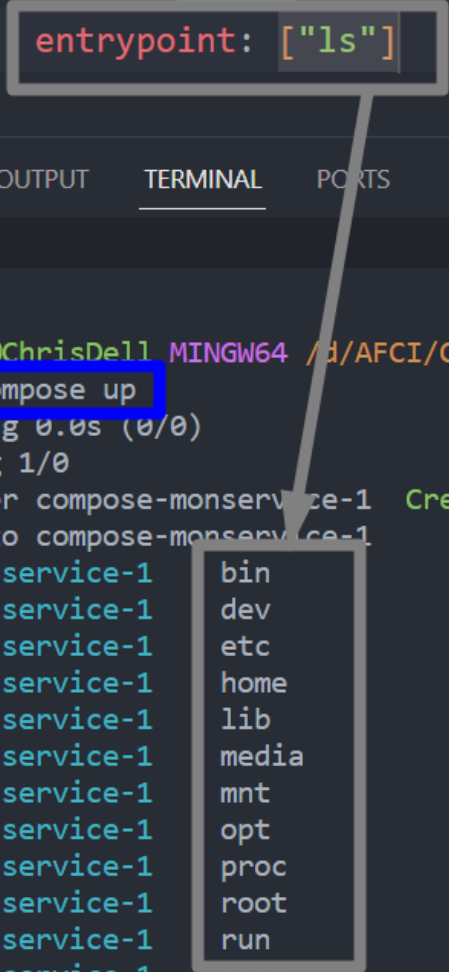
- ✓ Lancer une commande
- `entrypoint: entrypoint`

```
3  services:
4      monservice:
5          image: alpine
6          entrypoint: ["ls"]
7
```

PROBLEMS OUTPUT TERMINAL PORTS

✓ **TERMINAL**

```
Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/cours
$ docker-compose up
[+] Building 0.0s (0/0)
[+] Running 1/0
✓ Container compose-monservice-1 Created
Attaching to compose-monservice-1
compose-monservice-1 bin
compose-monservice-1 dev
compose-monservice-1 etc
compose-monservice-1 home
compose-monservice-1 lib
compose-monservice-1 media
compose-monservice-1 mnt
compose-monservice-1 opt
compose-monservice-1 proc
compose-monservice-1 root
compose-monservice-1 run
compose-monservice-1/sbin
compose-monservice-1/srv
```

A diagram consisting of a rectangular box around the `entrypoint: ["ls"]` line in the Docker Compose file and another rectangular box around the output of the `ls` command in the terminal. A line connects the two boxes, illustrating that the entrypoint configuration is being executed in the container.

Quelques interactions :

- ✓ Lancer une commande

```
2
3  services:
4  monservice:
5    image: alpine
6
```

PROBLEMS OUTPUT TERMINAL PORTS

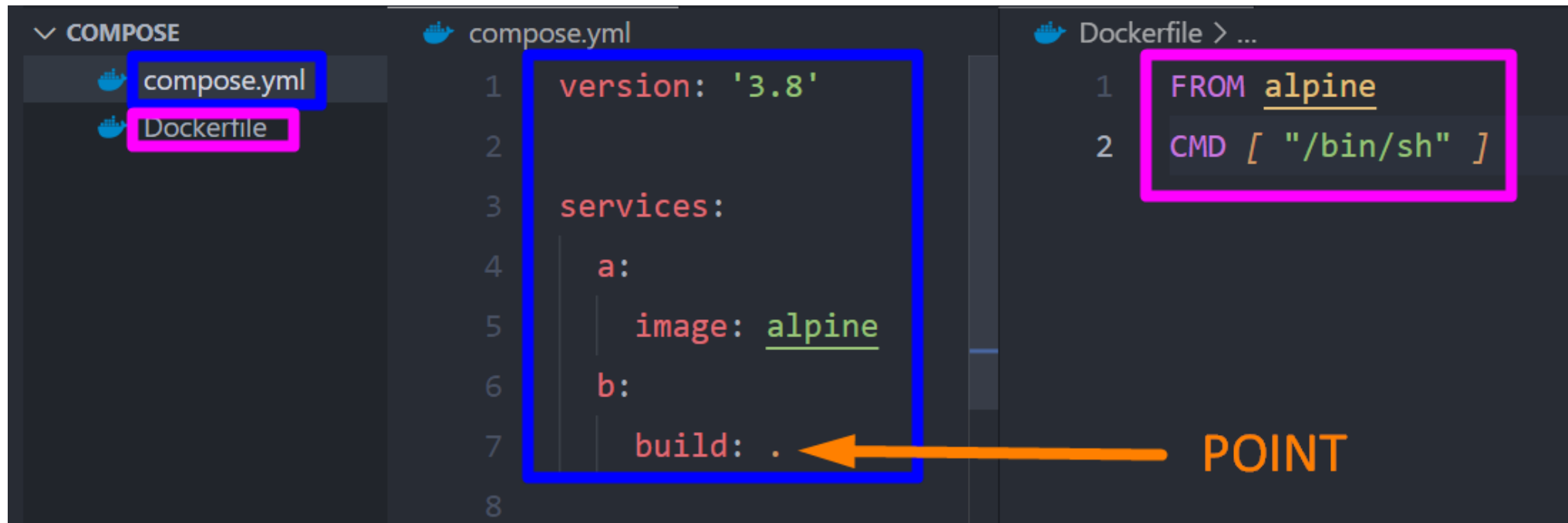
✓ **TERMINAL**

```
Christophe@ChrisDell MINGW64 /d/AFCI/Cours/8 - conteneurisation/cours/Compose
$ docker-compose run monservice ls
[+] Building 0.0s (0/0)
[+] Building 0.0s (0/0)
bin      etc      lib      mnt      proc     run      srv      tmp      var
dev      home    media    opt      root     sbin     sys      usr
```

Images personnalisées

Images personnalisées :

- ✓ Création d'un environnement avec un Dockerfile



Images personnalisées :

```
$ docker-compose up
[+] Running 2/2
 ✓ a 1 layers [::] 0B/0B Pulled
 ✓ 661ff4d9561e Pull complete
2023/12/26 12:32:31 http2: server: error reading preface from client //./pipe/docker_engine: fil
[+] Building 0.1s (5/5) FINISHED
=> [b internal] load .dockerignore
=> => transferring context: 2B
=> [b internal] load build definition from Dockerfile
=> => transferring dockerfile: 67B
=> [b internal] load metadata for docker.io/library/alpine:latest
=> [b 1/1] FROM docker.io/library/alpine
=> [b] exporting to image
=> => exporting layers
=> => writing image sha256:86d22d2633322d9bb545775462218d0d17ac65cc8f67f303273f57557ce9d528
=> => naming to docker.io/library/compose-b
[+] Running 3/3
 ✓ Network compose_default Created
 ✓ Container compose-b-1 Created
 ✓ Container compose-a-1 Created
Attaching to compose-a-1, compose-b-1
compose-b-1 exited with code 0
compose-a-1 exited with code 0
```

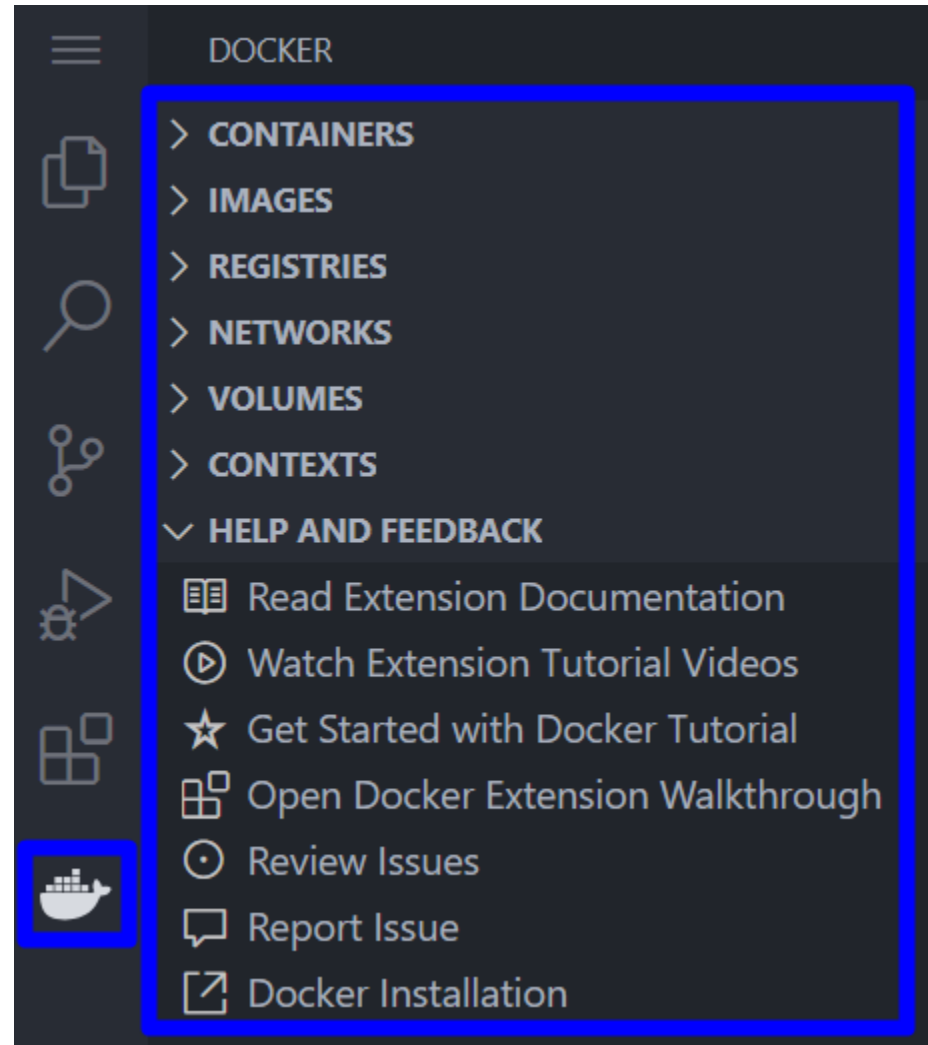
```
$ docker image ls
```

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|------------|--------|--------------|-------------|--------|
| alpine | latest | f8c20f8bbcb6 | 2 weeks ago | 7.38MB |
| compose-b | latest | 86d22d263332 | 2 weeks ago | 7.38MB |

Le plugin VsCode

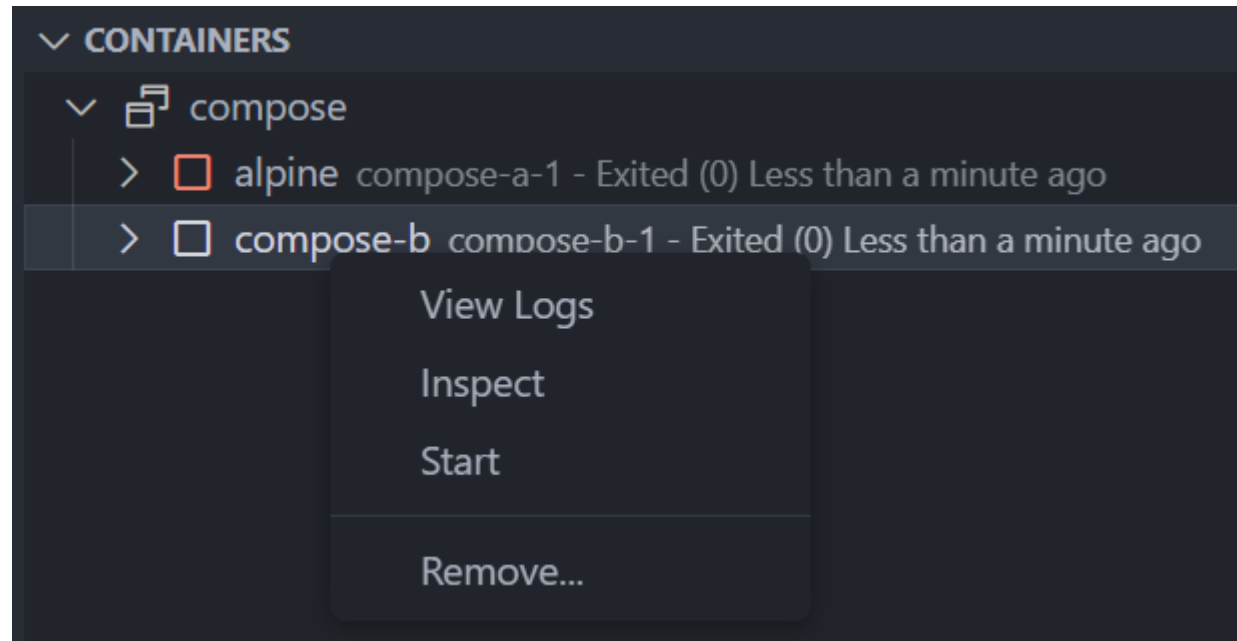
Docker VsCode:

✓ Présentation



Docker VsCode:

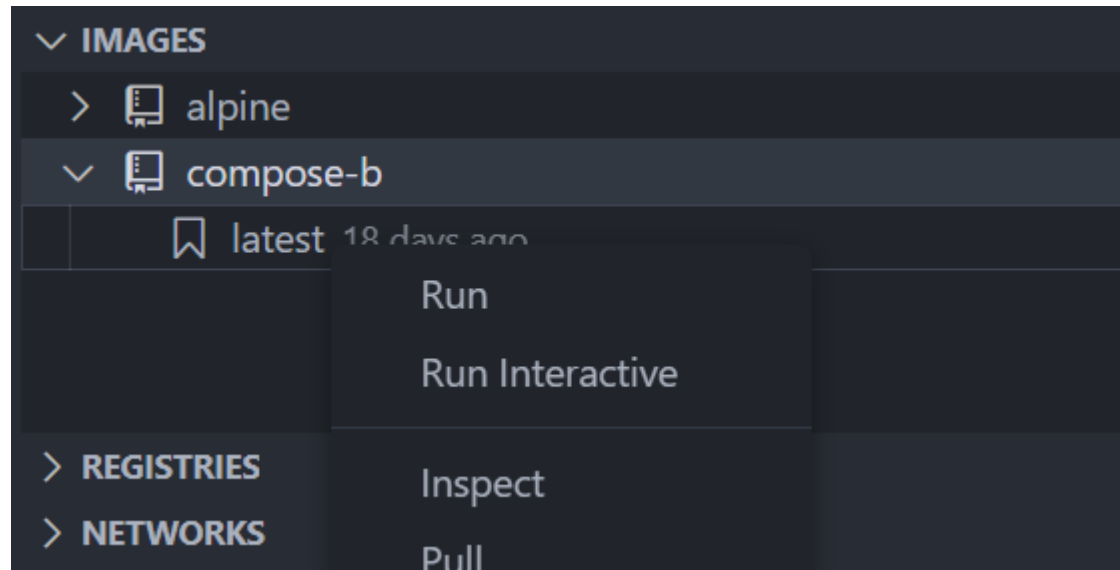
✓ Container



➤ Possibilité d'interagir en GUI (idem CLI)

Docker VsCode:

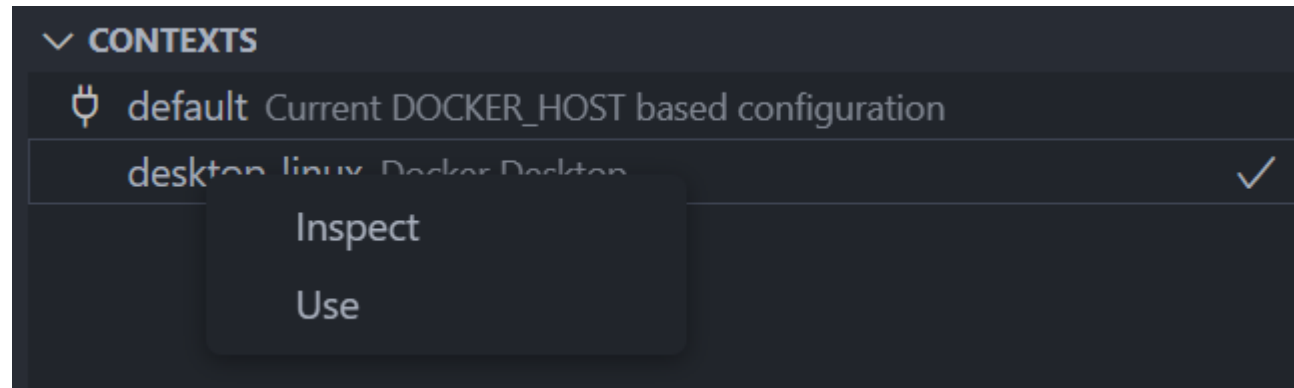
✓ Images



➤ Possibilité d'interagir en GUI (idem CLI)

Docker VsCode:

✓ Context

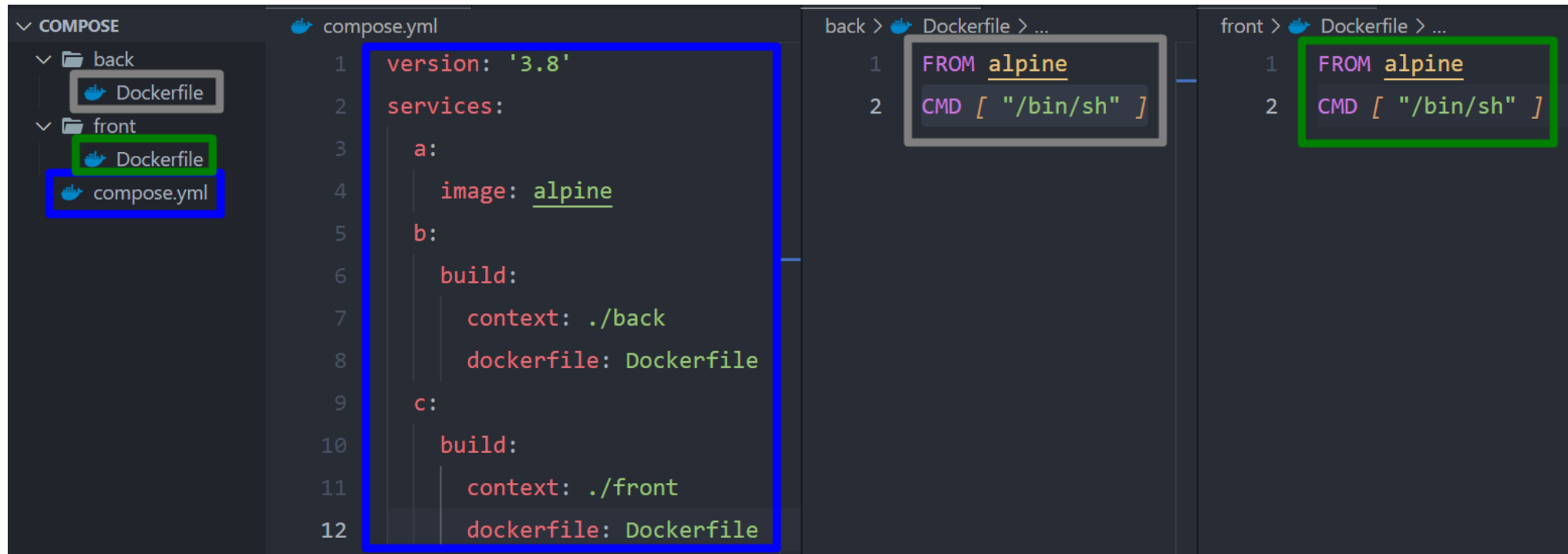


➤ Possibilité d'interagir en GUI (idem CLI)

Le context

Context :

- ✓ Il est possible d'utiliser plusieurs Dockerfile en fonction des besoins



The screenshot shows the Visual Studio Code interface with a Docker Compose project. The left sidebar displays the file explorer with a 'COMPOSE' section containing 'back' and 'front' folders. Each folder contains a 'Dockerfile' and a 'compose.yml' file. The 'back' folder's 'Dockerfile' is highlighted with a grey box, and the 'front' folder's 'Dockerfile' is highlighted with a green box. The 'compose.yml' file is highlighted with a blue box. The main editor area shows the 'compose.yml' file, which defines three services: 'a', 'b', and 'c'. Service 'a' uses the 'alpine' image. Service 'b' builds from the 'back' context using the 'Dockerfile'. Service 'c' builds from the 'front' context using the 'Dockerfile'. The 'back' and 'front' Dockerfiles are also visible in the editor, both using 'alpine' as the base image and running '/bin/sh' as the command. The 'back' Dockerfile is highlighted with a grey box, and the 'front' Dockerfile is highlighted with a green box.

```
COMPOSE
└─ back
   └─ Dockerfile
└─ front
   └─ Dockerfile
   └─ compose.yml
```

```
compose.yml
1 version: '3.8'
2 services:
3   a:
4     image: alpine
5   b:
6     build:
7       context: ./back
8       dockerfile: Dockerfile
9   c:
10    build:
11      context: ./front
12      dockerfile: Dockerfile
```

```
back > Dockerfile > ...
1 FROM alpine
2 CMD [ "/bin/sh" ]
```




```
front > Dockerfile > ...
1 FROM alpine
2 CMD [ "/bin/sh" ]
```

Context :

- ✓ Il est possible d'utiliser plusieurs Dockerfile en fonction des besoins

```
$ docker-compose up
[+] Building 0.0s (0/0)
[+] Running 3/0
✓ Container compose-a-1 Created
✓ Container compose-c-1 Created
✓ Container compose-b-1 Created
Attaching to compose-a-1, compose-b-1, compose-c-1
compose-a-1 exited with code 0
compose-b-1 exited with code 0
compose-c-1 exited with code 0
```

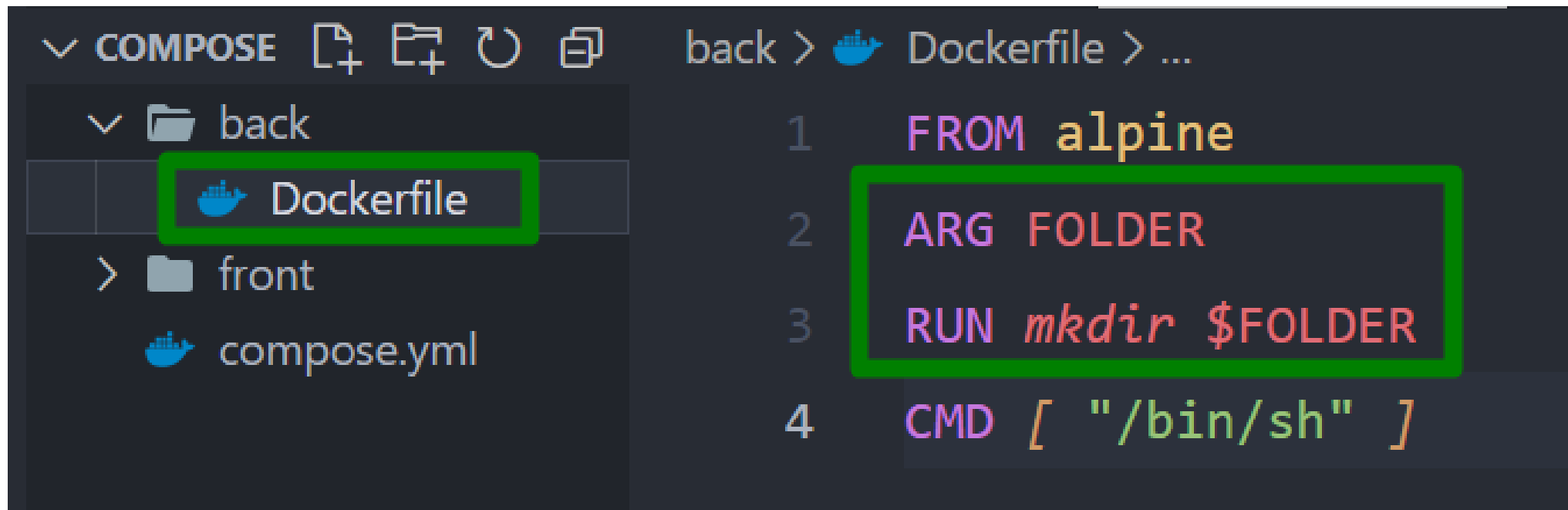
IMAGES

- >  alpine
- >  compose-b
- >  compose-c

Passage d'arguments

Arguments :

- ✓ Il est possible de passer des arguments au moment du build



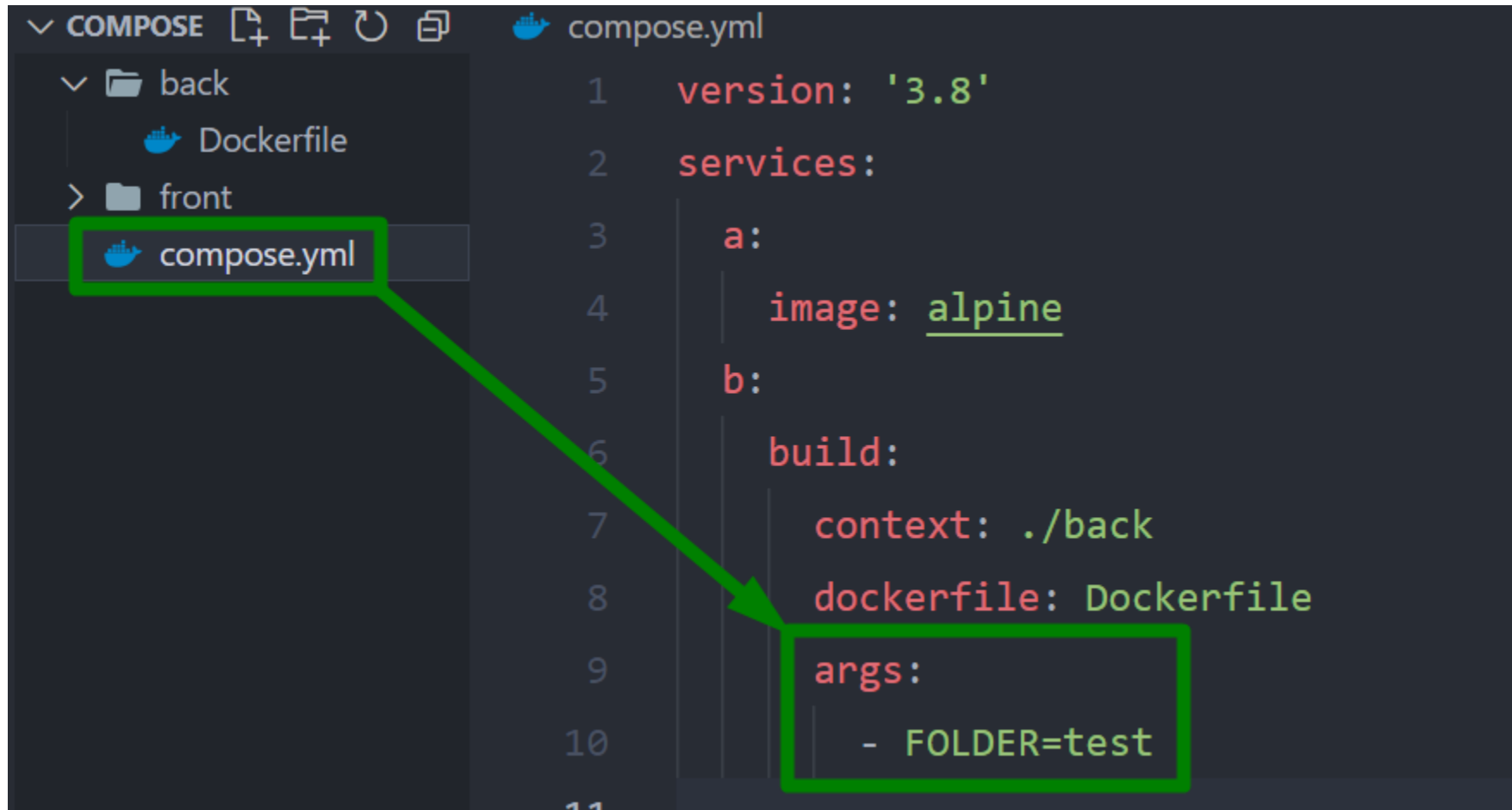
The screenshot shows a code editor interface. On the left, a file explorer displays a project structure with a 'back' folder containing a 'Dockerfile' (highlighted with a green box) and a 'front' folder containing a 'compose.yml' file. The main editor area shows the content of the 'Dockerfile' with the following lines:

```
back > Dockerfile > ...  
1 FROM alpine  
2 ARG FOLDER  
3 RUN mkdir $FOLDER  
4 CMD [ "/bin/sh" ]
```

The lines 2, 3, and 4 are grouped together in a green box, highlighting the argument definition and its usage in the build command.

Arguments :

- ✓ Il est possible de passer des arguments au moment du build



The image shows a code editor with a dark theme. On the left, a file explorer shows a project structure with folders 'back' and 'front', and files 'Dockerfile' and 'compose.yml'. The 'compose.yml' file is selected and highlighted with a green box. A green arrow points from this box to the 'args' section of the 'b' service in the main editor. The main editor shows the following YAML code:

```
1  version: '3.8'
2  services:
3    a:
4      image: alpine
5    b:
6      build:
7        context: ./back
8        dockerfile: Dockerfile
9      args:
10        - FOLDER=test
```

The 'args' section is also highlighted with a green box.

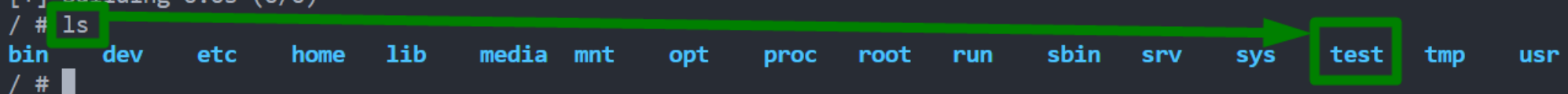
Arguments :

```
$ docker-compose build
```

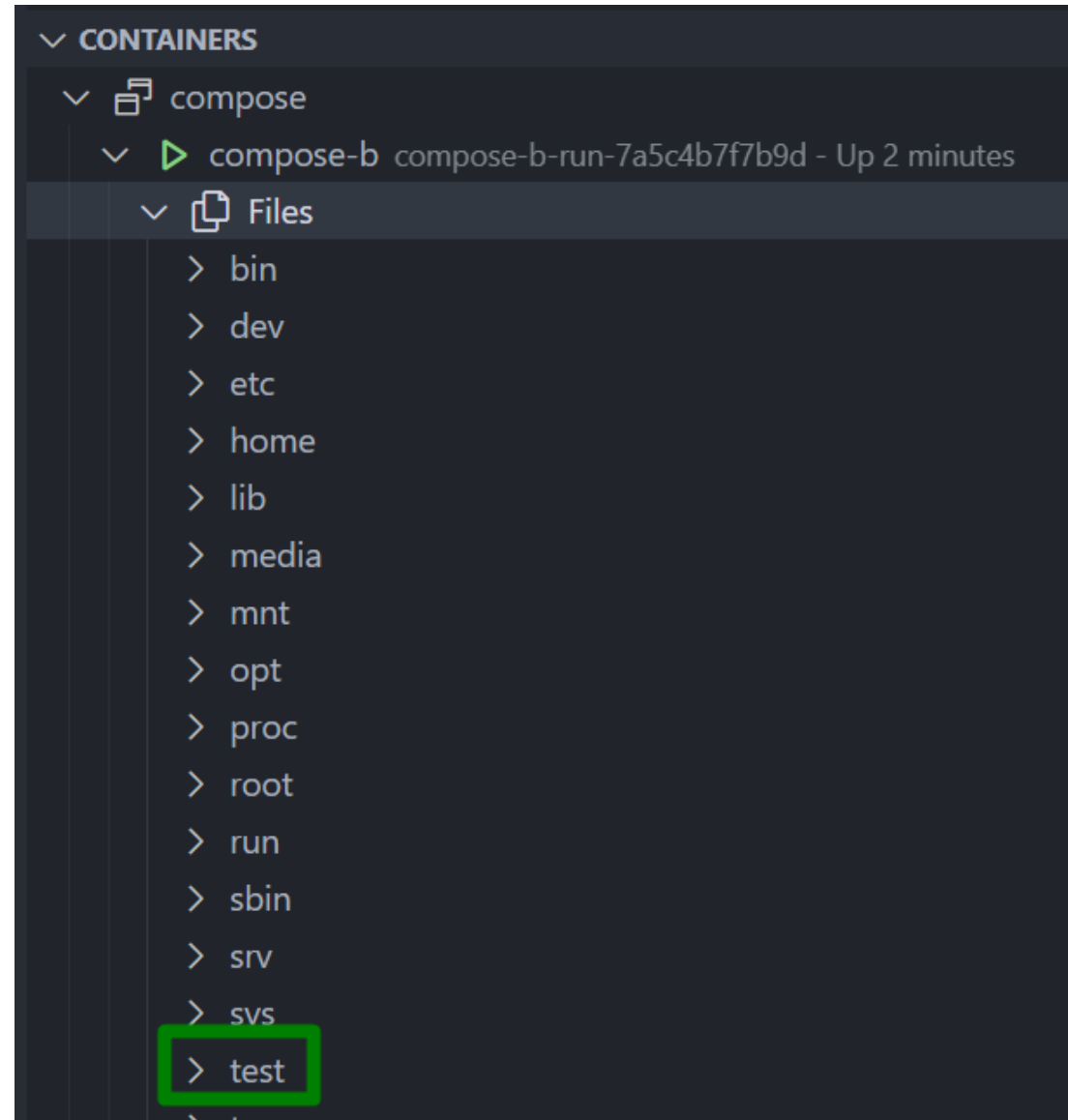
```
[+] Building 0.3s (6/6) FINISHED
=> [b internal] load .dockerignore
=> => transferring context: 2B
=> [b internal] load build definition from Dockerfile
=> => transferring dockerfile: 98B
=> [b internal] load metadata for docker.io/library/alpine:latest
=> CACHED [b 1/2] FROM docker.io/library/alpine
=> [b 2/2] RUN mkdir test
=> [b] exporting to image
=> => exporting layers
=> => writing image sha256:32e49c01796b9381398703cb356a1bcf62b24cf94eeb86c37152e2cc5335f72f
=> => naming to docker.io/library/compose-b
```

```
$ docker-compose run b
```

```
[+] Building 0.0s (0/0)
[+] Creating 1/0
✓ Network compose_default Created
[+] Building 0.0s (0/0)
/ # ls
bin    dev    etc    home  lib    media mnt    opt    proc   root   run    sbin   srv    sys    test  tmp    usr
```

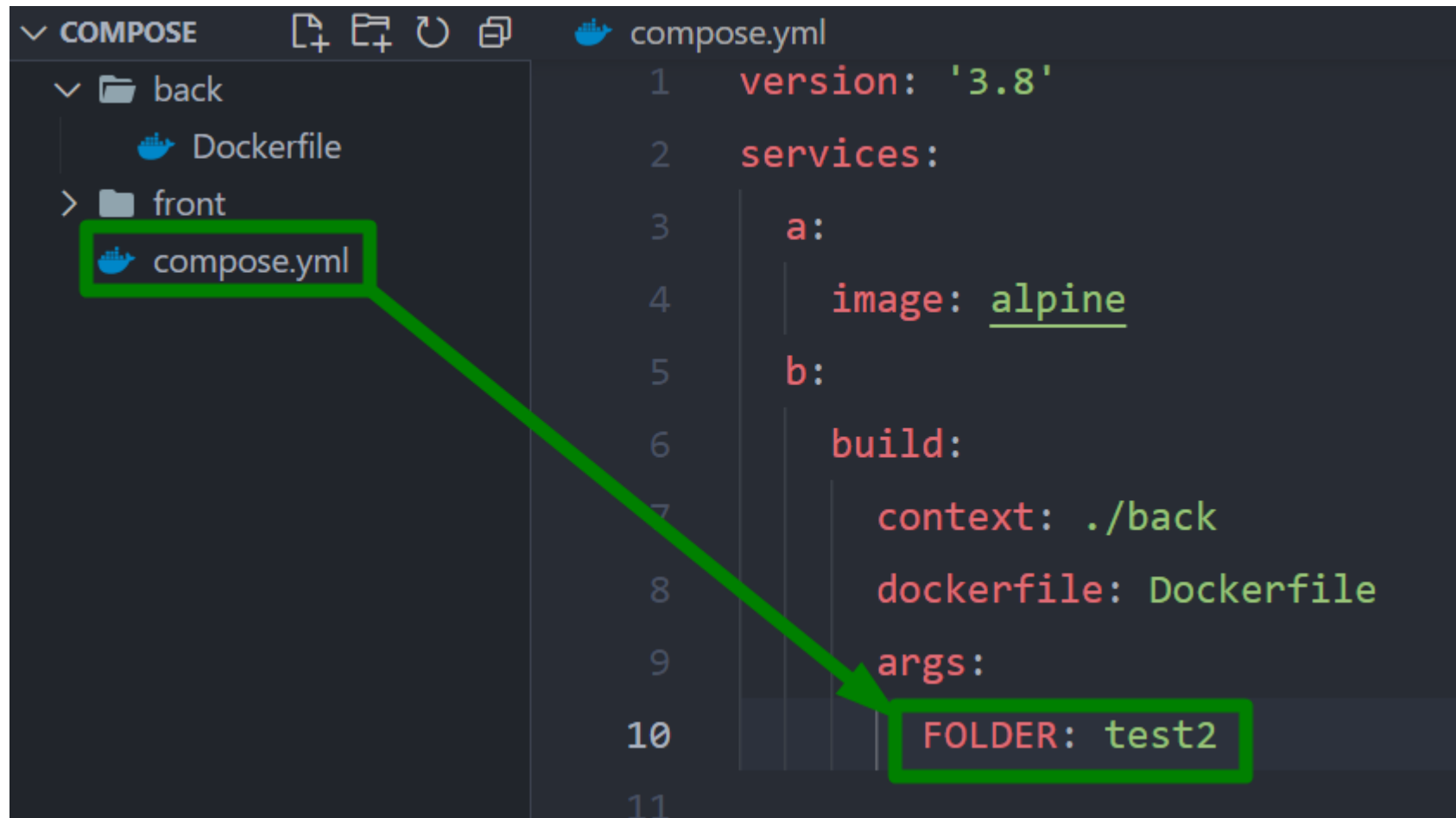


Arguments :



Arguments :

- ✓ Il est possible de passer des arguments au moment du build



The image shows a code editor with a dark theme. On the left, a file explorer shows a project structure with folders 'back' and 'front'. Inside 'front', a file named 'compose.yml' is highlighted with a green box. A green arrow points from this box to the 'args' section of the 'compose.yml' file in the main editor. The main editor shows the following YAML content:

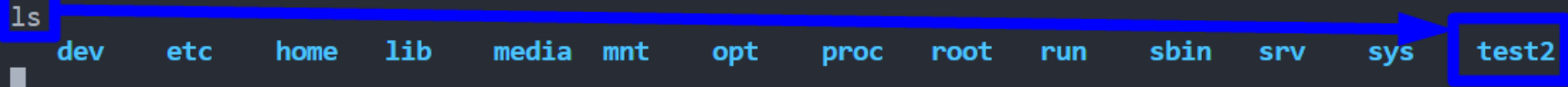
```
1  version: '3.8'
2  services:
3    a:
4      image: alpine
5    b:
6      build:
7        context: ./back
8        dockerfile: Dockerfile
9        args:
10         FOLDER: test2
11
```

The 'args' section is highlighted with a green box, and the line 'FOLDER: test2' is also highlighted with a green box.

Arguments :

```
chris@chris-15b11: /usr/local/src/docker-compose$ docker-compose build
2023/12/26 15:45:46 http2: server: error reading preface from client //./pipe/docker_engine: file has already been closed
[+] Building 1.6s (7/7) FINISHED
=> [b internal] load build definition from Dockerfile
=> => transferring dockerfile: 98B
=> [b internal] load .dockerignore
=> => transferring context: 2B
=> [b internal] load metadata for docker.io/library/alpine:latest
=> [b auth] library/alpine:pull token for registry-1.docker.io
=> CACHED [b 1/2] FROM docker.io/library/alpine@sha256:51b67269f354137895d43f3b3d810bfacd3945438e94dc5ac55fdac340352f48
=> [b 2/2] RUN mkdir test2
=> [b] exporting to image
=> => exporting layers
=> => writing image sha256:2232e17c667ef2a72dceaae23465488eb198701ff462741e37fddb50635a9238
=> => naming to docker.io/library/compose-b

chris@chris-15b11: /usr/local/src/docker-compose$ docker-compose run b
[+] Building 0.0s (0/0)
[+] Building 0.0s (0/0)
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    test2  tmp    usr
```



Questions ??

Prochain chapitre: Docker-compose – partie 2.