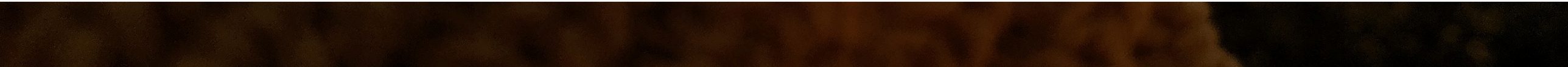
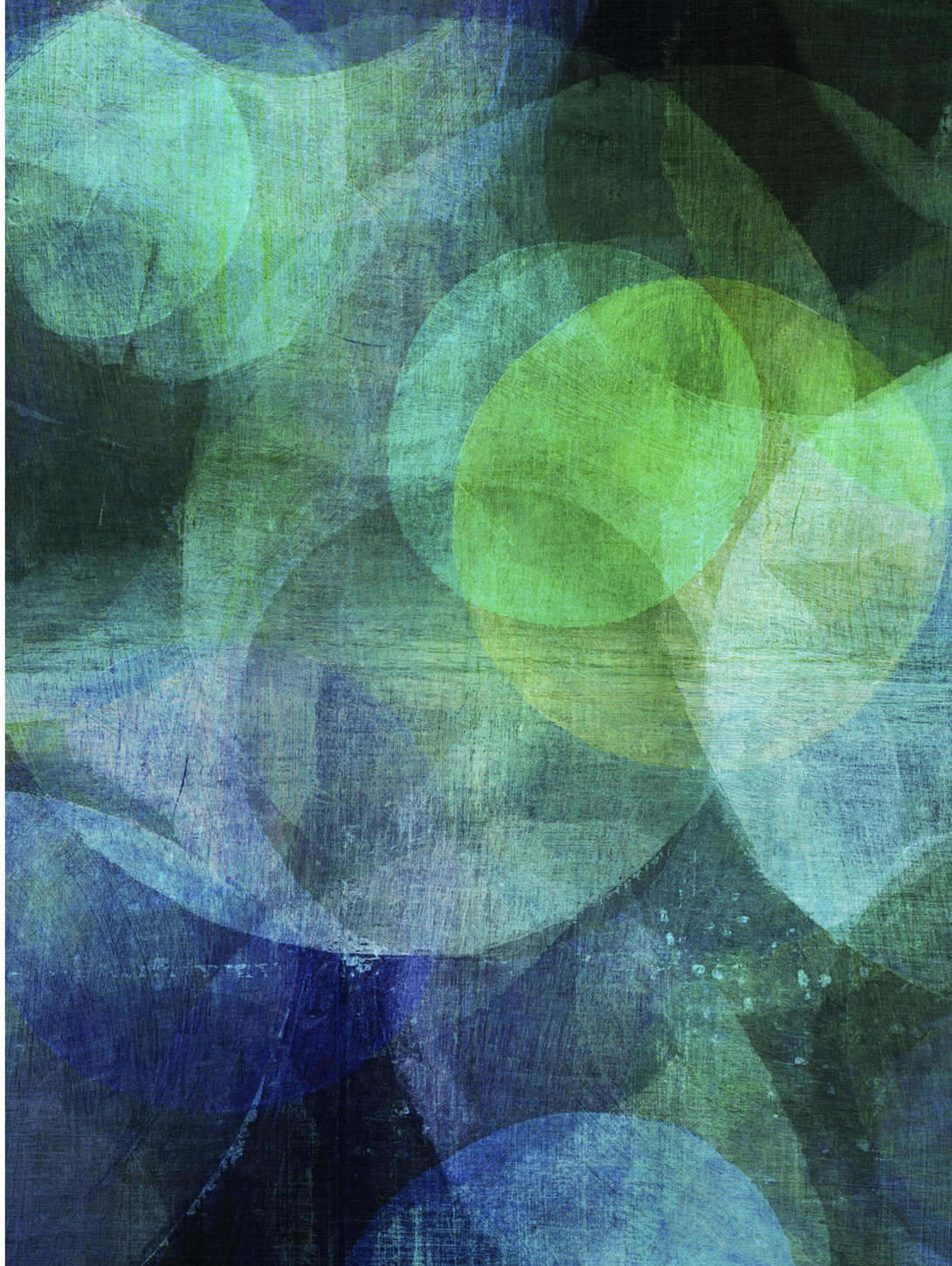
INTRODUCTION TO DOCKER

INTRODUCTION TO DOCKER

Dockerfile

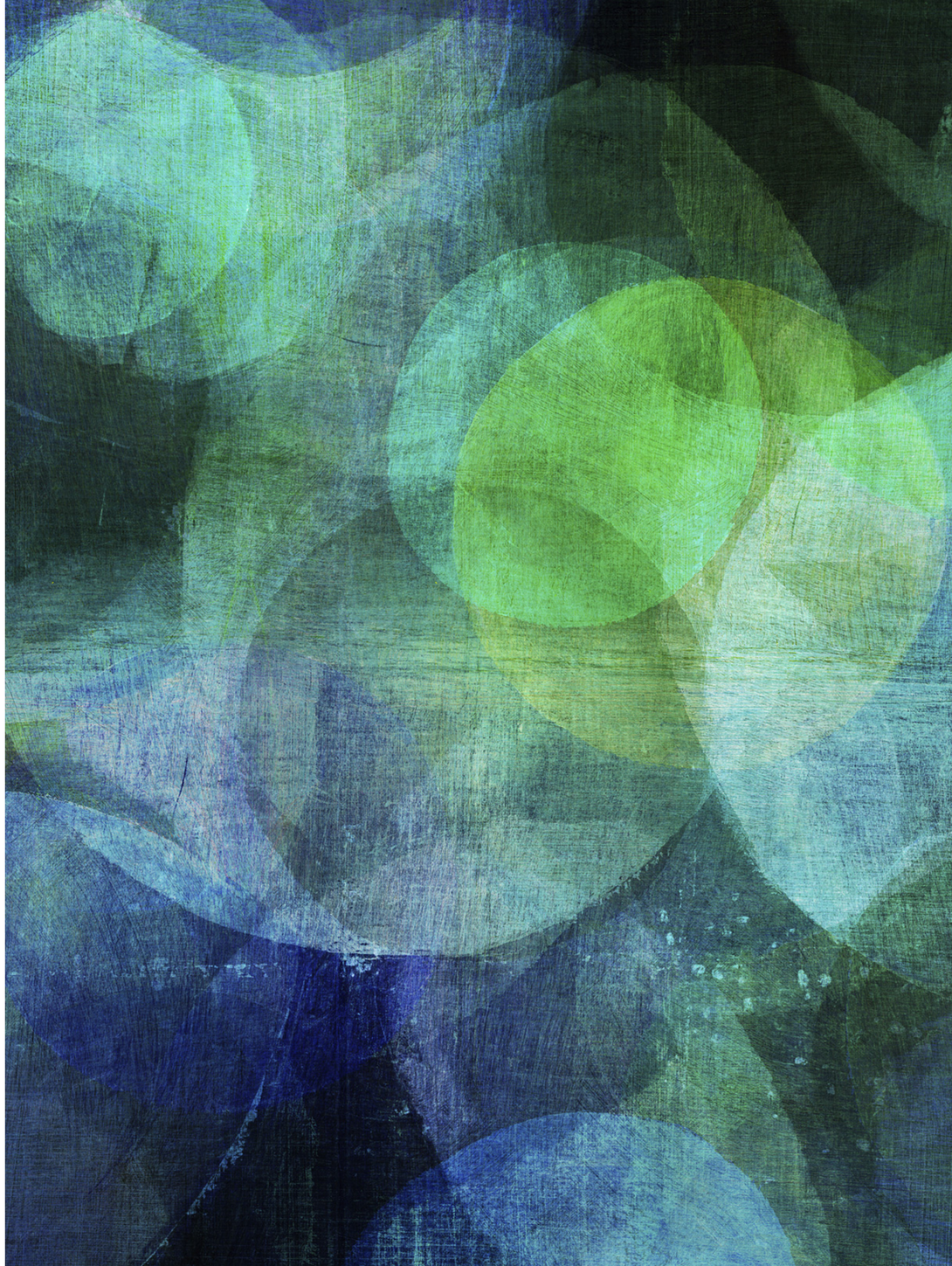


AGENDA



AGENDA

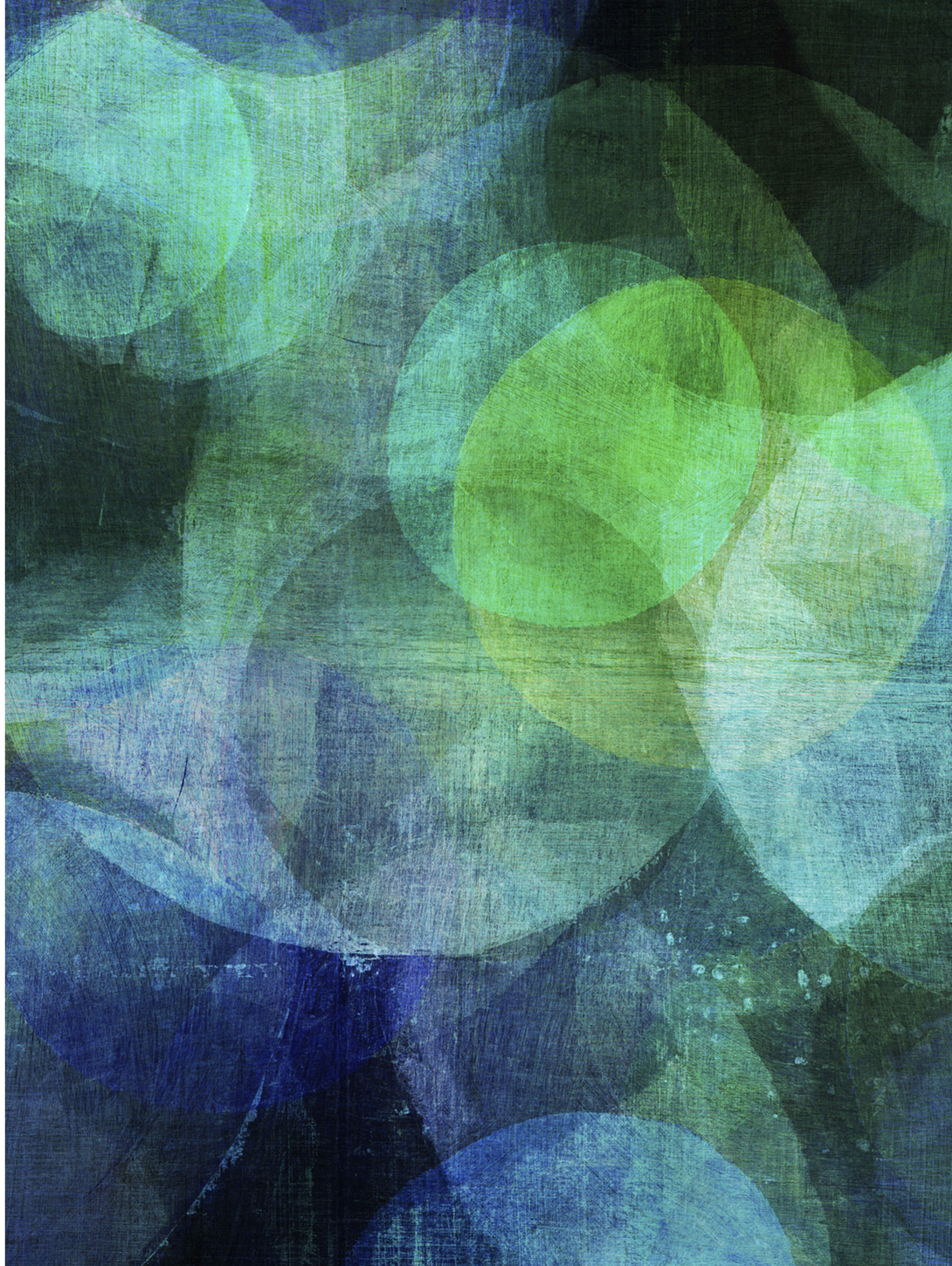
- Dockerfile



AGENDA

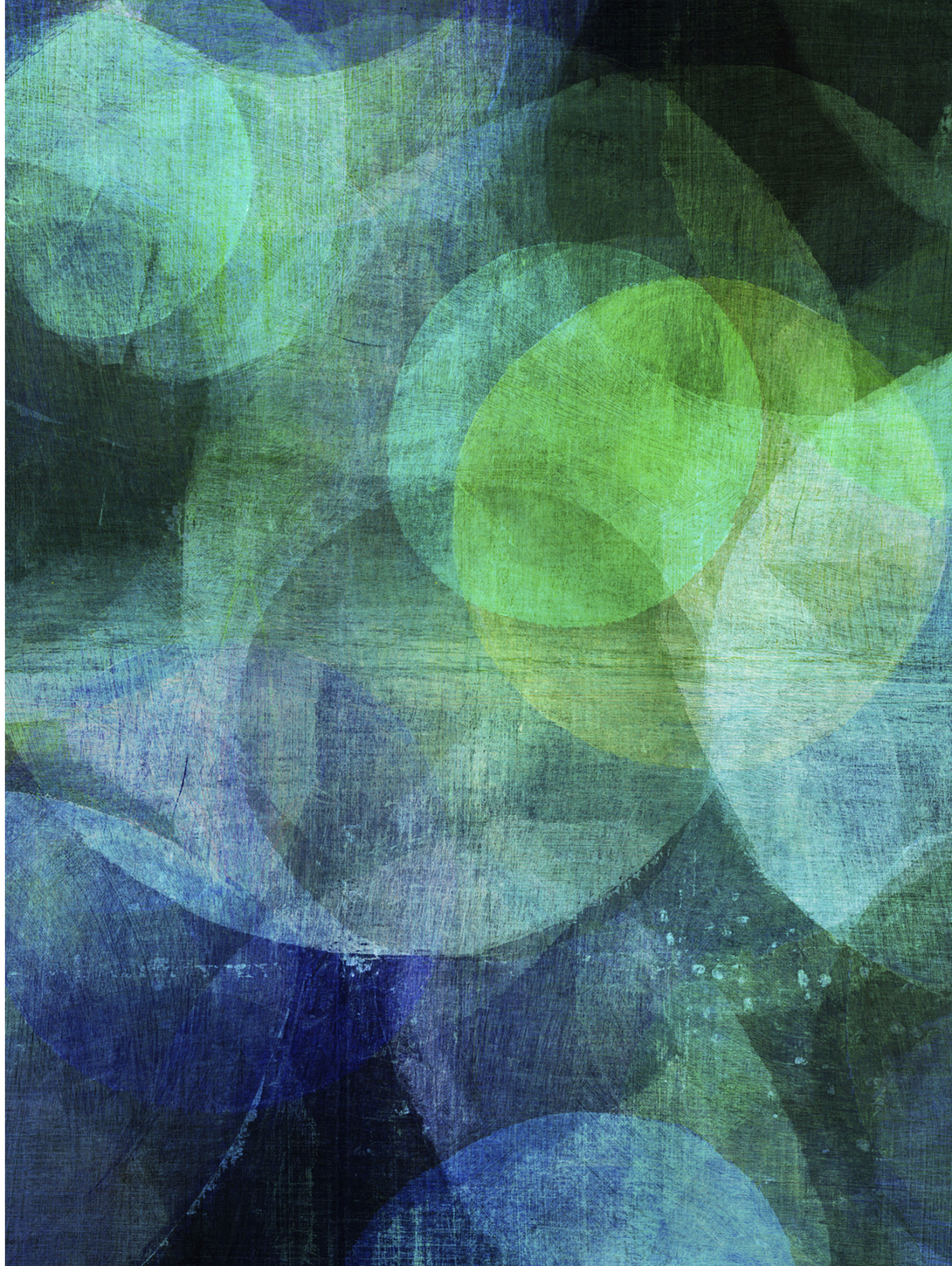
- Dockerfile

- Format



AGENDA

- Dockerfile
- Format
- Instruction



DOCKERFILE

INSTRUCTION ARGRUMENT

```
INSTRUCTION arguments
```

INSTRUCTION ARGRUMENT

The instruction is not case-sensitive. However, convention is for them to be UPPERCASE to distinguish them from arguments more easily.

```
INSTRUCTION arguments
```

INSTRUCTION ARGRUMENT

The instruction is not case-sensitive. However, convention is for them **to be UPPERCASE** to distinguish them from arguments more easily.

```
INSTRUCTION arguments
```

arguments are based on instruction

INSTRUCTION ARGRUMENT

The instruction is not case-sensitive. However, convention is for them to be UPPERCASE to distinguish them from arguments more easily.

```
INSTRUCTION arguments
```

arguments are based on instruction

Docker runs instructions in a Dockerfile in order. A Dockerfile must begin with a FROM instruction.

INSTRUCTION ARGRUMENT

The instruction is not case-sensitive. However, convention is for them **to be UPPERCASE** to distinguish them from arguments more easily.

```
INSTRUCTION arguments
```

arguments are based on instruction

Docker runs instructions in a Dockerfile in order. A Dockerfile must begin with a FROM instruction.

Example :

```
1  FROM ubuntu:22.04
```

FROM *[instruction]*

Name and tag of FROM Image *[arguments]*

ENVIRONMENT REPLACEMENT

ENVIRONMENT REPLACEMENT

```
FROM    busybox
ENV     FOO=/bar
WORKDIR ${FOO}        # WORKDIR /bar
ADD     . $FOO        # ADD . /bar
COPY    \$FOO /quux   # COPY $FOO /quux
```

ENVIRONMENT REPLACEMENT

```
FROM    busybox
ENV     FOO=/bar
WORKDIR ${FOO}          # WORKDIR /bar
ADD     . $FOO          # ADD . /bar
COPY    \$FOO /quux     # COPY $FOO /quux
```

Environment variables are supported by the following list of instructions in the Dockerfile:

- ADD
- COPY
- ENV
- EXPOSE
- FROM
- LABEL
- STOPSIGNAL
- USER
- VOLUME
- WORKDIR
- ONBUILD #

INSTRUCTION ON DOCKERFILE

FROM : SETS THE BASE IMAGE

FROM : SETS THE BASE IMAGE

The **FROM** instruction initializes a new build stage and sets the Base Image for subsequent instructions. As such, a valid Dockerfile must start with a FROM instruction.

FROM : SETS THE BASE IMAGE

The **FROM** instruction initializes a new build stage and sets the Base Image for subsequent instructions. As such, a valid Dockerfile must start with a FROM instruction.

```
FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]
```

FROM : SETS THE BASE IMAGE

The **FROM** instruction initializes a new build stage and sets the Base Image for subsequent instructions. As such, a valid Dockerfile must start with a FROM instruction.

```
FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]
```

Example :

```
1  FROM ubuntu:22.04
```

Image = ubuntu

Tag (Version) of Ubuntu Image = 22.04

FROM : SETS THE BASE IMAGE

The **FROM** instruction initializes a new build stage and sets the Base Image for subsequent instructions. As such, a valid Dockerfile must start with a FROM instruction.

```
FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]
```

Example :

```
1  FROM ubuntu:22.04
```

Image = ubuntu

Tag (Version) of Ubuntu Image = 22.04

Example :

```
1  FROM nginx:1.21.6
```

Image = nginx

Tag (Version) of nginx Image = 1.21.6

COPY : COPIES NEW FILE/DIRECTORY TO IMAGE (NEW)

COPY : COPIES NEW FILE/DIRECTORY TO IMAGE (NEW)

The **COPY** instruction copies new files, directories from <src> and adds them to the filesystem of the image at the path <dest>.

COPY : COPIES NEW FILE/DIRECTORY TO IMAGE (NEW)

The **COPY** instruction copies new files, directories from <src> and adds them to the filesystem of the image at the path <dest>.

```
COPY [--chown=<user>:<group>] <src>... <dest>
```

OR

```
COPY [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

COPY : COPIES NEW FILE/DIRECTORY TO IMAGE (NEW)

The **COPY** instruction copies new files, directories from <src> and adds them to the filesystem of the image at the path <dest>.

`--chown` for non root user

```
COPY [--chown=<user>:<group>] <src>... <dest>
```

OR

```
COPY [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

COPY : COPIES NEW FILE/DIRECTORY TO IMAGE (NEW)

The **COPY** instruction copies new files, directories from <src> and adds them to the filesystem of the image at the path <dest>.

`--chown` for non root user

```
COPY [--chown=<user>:<group>] <src>... <dest>
```

OR

```
COPY [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

Example :

```
2  COPY index.html /usr/share/nginx/html
```

<src> = relative path

<dest> = absolute path or relative from WORKDIR

ADD : COPIES NEW FILE/DIRECTORY/URL TO IMAGE (OLD)

ADD : COPIES NEW FILE/DIRECTORY/URL TO IMAGE (OLD)

The **ADD** instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.

ADD : COPIES NEW FILE/DIRECTORY/URL TO IMAGE (OLD)

The **ADD** instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.

```
ADD [--chown=<user>:<group>] <src>... <dest>
```

OR

```
ADD [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

ADD : COPIES NEW FILE/DIRECTORY/URL TO IMAGE (OLD)

The **ADD** instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.

--chown for non root user

```
ADD [--chown=<user>:<group>] <src>... <dest>
```

OR

```
ADD [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

ADD : COPIES NEW FILE/DIRECTORY/URL TO IMAGE (OLD)

The **ADD** instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.

--chown for non root user

```
ADD [--chown=<user>:<group>] <src>... <dest>
```

OR

```
ADD [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

Example :

```
2  ADD index.html /usr/share/nginx/html
```

<src> = relative path

<dest> = absolute path or relative from WORKDIR

ADD (OLD) VS COPY (NEW)

ADD (OLD) VS COPY (NEW)

ADD and **COPY** are similar one. ADD lets you do like a COPY, but it also supports 2 other sources.

First: you can use a URL instead of a local file / directory.

Second: you can extract a tar file (not for zip) from the source directly into the destination.

ADD (OLD) VS COPY (NEW)

ADD and **COPY** are similar one. ADD lets you do like a COPY, but it also supports 2 other sources.

First: you can use a URL instead of a local file / directory.

Second: you can extract a tar file (not for zip) from the source directly into the destination.

Recommendation from Docker Team :

> *Using COPY in almost all cases. Ultimately, the rule is this: use COPY (unless you're absolutely sure you need ADD).*

RUN: EXECUTE ANY COMMAND IN A NEW LAYER

RUN: EXECUTE ANY COMMAND IN A NEW LAYER

The **RUN** instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.

RUN: EXECUTE ANY COMMAND IN A NEW LAYER

The **RUN** instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.

```
RUN <command>
```

OR

```
RUN [“executable", "param1", “param2"]
```

RUN: EXECUTE ANY COMMAND IN A NEW LAYER

The **RUN** instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.

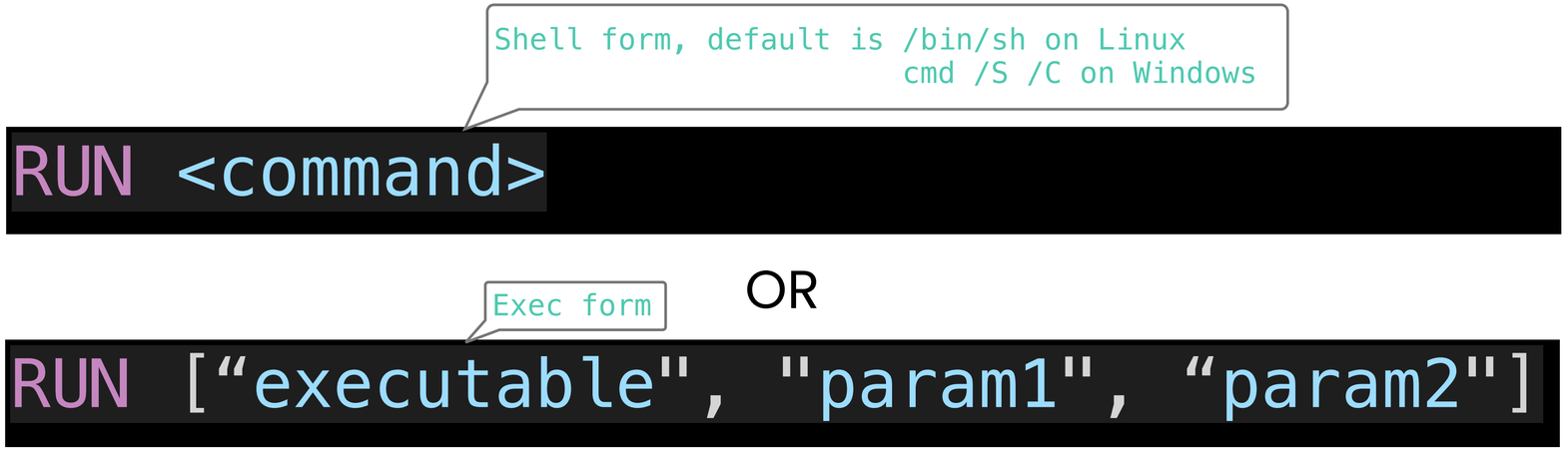
```
Shell form, default is /bin/sh on Linux
                      cmd /S /C on Windows
RUN <command>
```

OR

```
RUN [“executable", "param1", “param2"]
```

RUN: EXECUTE ANY COMMAND IN A NEW LAYER

The **RUN** instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.



RUN: EXECUTE ANY COMMAND IN A NEW LAYER

The **RUN** instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.

Shell form, default is /bin/sh on Linux
cmd /S /C on Windows

```
RUN <command>
```

OR

Exec form

```
RUN [“executable", "param1", “param2"]
```

Example :

```
3  RUN npm install
```

<command> = install package dependency from package.json

RUN: MORE EXAMPLE

RUN: MORE EXAMPLE

Example : More Readable with backslashes

```
3   RUN /bin/bash -c 'source $HOME/.bashrc; \
        echo $HOME'
```

RUN: MORE EXAMPLE

Example : More Readable with backslashes

```
3   RUN /bin/bash -c 'source $HOME/.bashrc; \
        echo $HOME'
```

OR

Example : Exec Form

```
3   RUN ["/bin/bash", "-c", "echo hello"]
```

RUN: MORE EXAMPLE

Example : More Readable with backslashes

```
3   RUN /bin/bash -c 'source $HOME/.bashrc; \
        echo $HOME'
```

OR

Example : Exec Form

```
3   RUN ["/bin/bash", "-c", "echo hello"]
```

OR

Example : Merge apt-get update and install to AVIOD Caching Problem

```
3   RUN apt-get update && apt-get install -y \
        package-bar \
        package-baz \
        package-foo  \
        && rm -rf /var/lib/apt/lists/*
```

CMD: PROVIDE DEFAULTS FOR AN EXECUTING CONTAINER

CMD: PROVIDE DEFAULTS FOR AN EXECUTING CONTAINER

The main purpose of a **CMD** is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an **ENTRYPOINT** instruction as well.

CMD: PROVIDE DEFAULTS FOR AN EXECUTING CONTAINER

The main purpose of a **CMD** is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an **ENTRYPOINT** instruction as well.

```
CMD [“executable", "param1", “param2"]
```

OR

```
CMD ["param1", “param2"]
```

OR

```
CMD command param1 param2
```

CMD: PROVIDE DEFAULTS FOR AN EXECUTING CONTAINER

The main purpose of a **CMD** is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an **ENTRYPOINT** instruction as well.

Exec form : this is the **preferred form**

```
CMD [“executable", "param1", “param2"]
```

OR

```
CMD ["param1", “param2"]
```

OR

```
CMD command param1 param2
```

CMD: PROVIDE DEFAULTS FOR AN EXECUTING CONTAINER

The main purpose of a **CMD** is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an **ENTRYPOINT** instruction as well.

Exec form : this is the **preferred form**

```
CMD [“executable", "param1", “param2"]
```

OR

As default params for **ENTRYPOINT**

```
CMD ["param1", “param2"]
```

OR

```
CMD command param1 param2
```

CMD: PROVIDE DEFAULTS FOR AN EXECUTING CONTAINER

The main purpose of a **CMD** is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an **ENTRYPOINT** instruction as well.

Exec form : this is the **preferred form**

```
CMD [“executable", "param1", “param2"]
```

OR

As default params for **ENTRYPOINT**

```
CMD ["param1", “param2"]
```

OR

Shell form

```
CMD command param1 param2
```

CMD: PROVIDE DEFAULTS FOR AN EXECUTING CONTAINER

The main purpose of a **CMD** is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an **ENTRYPOINT** instruction as well.

Exec form : this is the **preferred form**

```
CMD [“executable", "param1", “param2"]
```

OR

As default params for **ENTRYPOINT**

```
CMD ["param1", “param2"]
```

OR

Shell form

```
CMD command param1 param2
```

Noted : There **can only be one CMD** instruction in a Dockerfile.
If you list more than one CMD then only the **last CMD** will take effect.

ENTRYPOINT: CONFIGURE EXECUTION POINT

ENTRYPOINT: CONFIGURE EXECUTION POINT

An **ENTRYPOINT** allows you to configure a container that will run as an executable.

ENTRYPOINT: CONFIGURE EXECUTION POINT

An **ENTRYPOINT** allows you to configure a container that will run as an executable.

```
ENTRYPOINT [“executable", "param1", “param2”]
```

OR

```
ENTRYPOINT command param1 param2
```

ENTRYPOINT: CONFIGURE EXECUTION POINT

An **ENTRYPOINT** allows you to configure a container that will run as an executable.

```
ENTRYPOINT [“executable", "param1", “param2”]
```

OR

```
ENTRYPOINT command param1 param2
```

Example :

```
5  ENTRYPOINT ["java","-jar","/app.jar"]
```

HOW ENTRYPOINT AND CMD INTERACT

HOW ENTRYPOINT AND CMD INTERACT

Both **CMD** and **ENTRYPOINT** instructions define what command gets executed when running a container. There are *few rules that describe their co-operation.*

1. Dockerfile should specify *at least one* of **CMD** or **ENTRYPOINT** commands.

2. **ENTRYPOINT** should be defined when using *the container as an executable.*

3. **CMD** should be used as a way of *defining default arguments for an* **ENTRYPOINT** command *or* for executing an *ad-hoc command in a container.*

4. **CMD** will be *overridden* when running the container *with alternative arguments.*

HOW ENTRYPOINT AND CMD INTERACT

	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT ["exec_entry", "p1_entry"]
No CMD	**error, not allowed**	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD ["exec_cmd", "p1_cmd"]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD ["p1_cmd", "p2_cmd"]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry p1_cmd p2_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

HOW ENTRYPOINT AND CMD INTERACT

The table shows what command is executed for different **ENTRYPOINT** / **CMD** combinations:

	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT [“exec_entry”, “p1_entry”]
No CMD	error, not allowed	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD [“exec_cmd”, “p1_cmd”]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD [“p1_cmd”, “p2_cmd”]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry p1_cmd p2_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

WORKDIR: SETS THE WORKING DIRECTORY FROM IMAGE/CONTAINER

WORKDIR: SETS THE WORKING DIRECTORY FROM IMAGE/CONTAINER

The **WORKDIR** instruction sets the working directory for any **RUN**, **CMD**, **ENTRYPOINT**, **COPY** and **ADD** instructions that follow it in the Dockerfile.

If the **WORKDIR** doesn't exist, it will be created even if it's not used in any subsequent Dockerfile instruction.

WORKDIR: SETS THE WORKING DIRECTORY FROM IMAGE/CONTAINER

The **WORKDIR** instruction sets the working directory for any **RUN**, **CMD**, **ENTRYPOINT**, **COPY** and **ADD** instructions that follow it in the Dockerfile.

If the **WORKDIR** doesn't exist, it will be created even if it's not used in any subsequent Dockerfile instruction.

```
WORKDIR /path/to/workdir
```

WORKDIR: SETS THE WORKING DIRECTORY FROM IMAGE/CONTAINER

The **WORKDIR** instruction sets the working directory for any **RUN**, **CMD**, **ENTRYPOINT**, **COPY** and **ADD** instructions that follow it in the Dockerfile.

If the **WORKDIR** doesn't exist, it will be created even if it's not used in any subsequent Dockerfile instruction.

```
WORKDIR /path/to/workdir
```

Example :

```
2  WORKDIR /application
```

EXPOSE: LISTENS ON PORT

EXPOSE: LISTENS ON PORT

The **EXPOSE** instruction informs Docker that the container listens on the specified network ports at runtime. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified.

EXPOSE: LISTENS ON PORT

The **EXPOSE** instruction informs Docker that the container listens on the specified network ports at runtime. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified.

```
EXPOSE <port> [<port>/<protocal>]
```

EXPOSE: LISTENS ON PORT

The **EXPOSE** instruction informs Docker that the container listens on the specified network ports at runtime. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified.

```
EXPOSE <port> [<port>/<protocal>]
```

Example :

```
2  EXPOSE 80
```

ENV: SET ENVIRONMENT VARIABLE

ENV: SET ENVIRONMENT VARIABLE

The **ENV** instruction sets the environment variable <key> to the value <value>. This value will be in the environment for all subsequent instructions in the build stage and can be replaced inline in many as well.

ENV: SET ENVIRONMENT VARIABLE

The **ENV** instruction sets the environment variable <key> to the value <value>. This value will be in the environment for all subsequent instructions in the build stage and can be replaced inline in many as well.

```
ENV <key>=<value> ...
```

ENV: SET ENVIRONMENT VARIABLE

The **ENV** instruction sets the environment variable <key> to the value <value>. This value will be in the environment for all subsequent instructions in the build stage and can be replaced inline in many as well.

```
ENV <key>=<value> ...
```

Example :

```
2  ENV MY_NAME="John Doe"
3  ENV MY_DOG=Rex\ The\ Dog
4  ENV MY_CAT=fluffy
```

VOLUME: CREATE MOUNT POINT FOR IMAGE

VOLUME: CREATE MOUNT POINT FOR IMAGE

The **VOLUME** instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.

VOLUME: CREATE MOUNT POINT FOR IMAGE

The **VOLUME** instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.

```
VOLUME ["/data"]
```

VOLUME: CREATE MOUNT POINT FOR IMAGE

The **VOLUME** instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.

```
VOLUME ["/data"]
```

Example :

```
2 VOLUME ["/var/www", "/var/log/apache2"]
```

THANK YOU