Лекція 1. Віртуальна лабораторія.

1.1. Віртуальна машина

1.2. Контейнерне середовище.

1.3. Різниця між віртуальною машиною та контейнерним середовищем

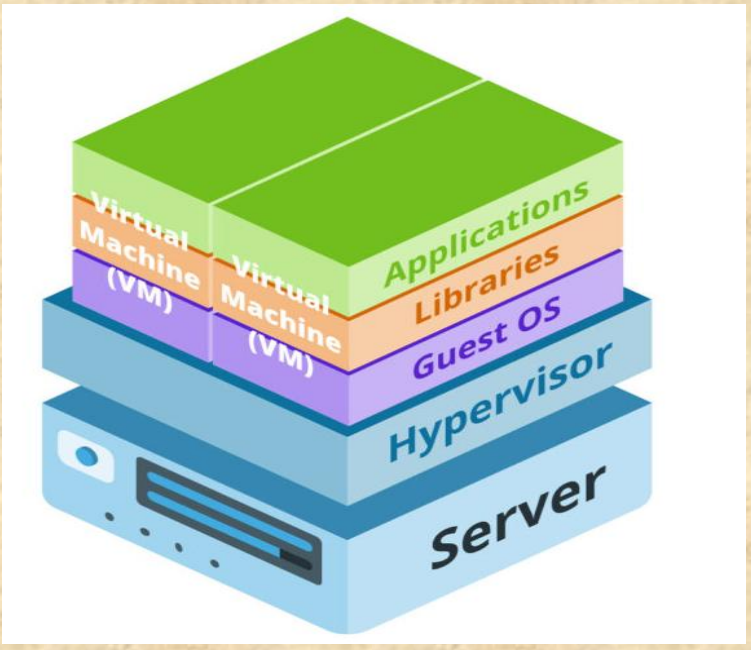
1.4. Приклади програм для створення віртуальних машин

1.5. Приклади програм для створення контейнерів

1.6. Гіпервізори

1.7. Приклад створення контейнера в docker

1.8. Висновки



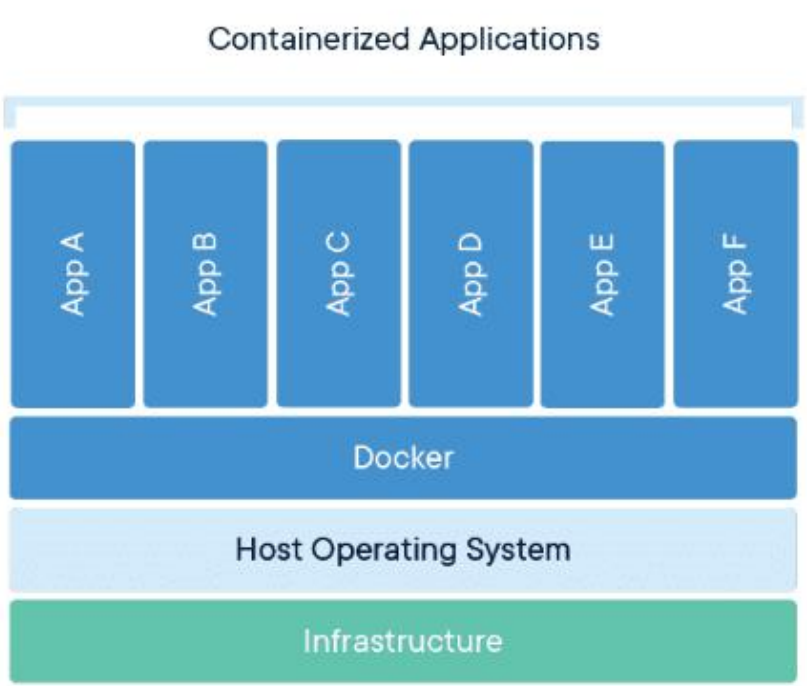
Віртуальна машина — це спеціальне програмне середовище, що працює за принципом окремого комп'ютера, але всередині сервера, на якому може бути запущено від однієї до кількох сотень ВМ з однаковими або різними параметрами продуктивності. За необхідністю віртуальні машини можна переносити з одного фізичного сервера на інший — це зручніше, ніж змінювати конфігурацію обладнання під запуск програми з певними вимогами. Технологія, завдяки якій став можливим запуск кількох віртуальних середовищ на основі одного фізичного сервера або кластера фізичних серверів, називається віртуалізацією.

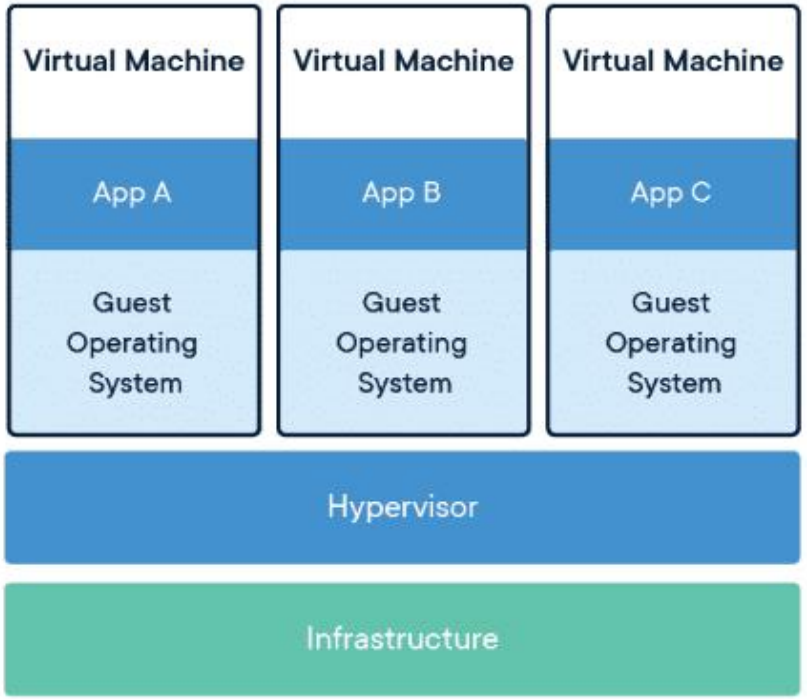
Віртуальна машина виступає як емулятор ПЗ. Припустимо, на комп'ютері вже є Windows і потрібно додатково встановити, наприклад, Ubuntu. Створюємо на тому ж комп'ютері віртуальну машину і вже на неї встановлюємо додаткову ОС. У результаті матимемо ніби два комп'ютери водночас. Однак для віртуальної машини потрібне місце на жорсткому диску, вдосталь оперативної пам'яті та ресурсів процесору. На противагу цьому з'явився принцип контейнерів, на якому побудований Docker та інші схожі інструменти. Принцип контейнерів у власній інфраструктурі та єдиній для всіх середовищ ОС. Зверху цієї основи встановлюється Docker, завдяки якому можна створювати безліч контейнерів, аплікацій тощо.

Docker розділяє єдине ядро ОС на окремі контейнери, під кожний із яких виділяється свій процес. Це набагато зручніше, ніж віртуальна машина. Вам не потрібно шукати ще й пам'ять, диск, оперативку, адже окремий процес витягає значно менше ресурсів.

Коли ви працюєте з багатьма віртуальними машинами, часто вони заважають одна одній. Docker же може запускати в ядрі будь-який застосунок, який буде повністю ізольований від інших. Окрім цього, на фоні конкурентів Docker виграє за рахунок простого синтаксису. Також його можна запустити на всіх популярних платформах: Linux, Windows, Mac.


Docker
Virtual Machine
VS


Containerized Applications
App A
App B
App C
App D
App E
App F
Docker
Host Operating System
Infrastructure


Virtual Machine
Virtual Machine
Virtual Machine
App A
App B
App C
Guest Operating System
Guest Operating System
Guest Operating System
Hypervisor
Infrastructure

Віртуальні машини

- VirtualBox
- VMware Workstation
- Microsoft Virtual PC, Hyper-V
- Boot Camp (для MAC – WIN)
- Parallels Desktop
- Nox (емулятор Android)
- BlueStacks (емулятор Android)
- Appetixe.io (браузерний iOS та Android емулятор)
- Andy OS (емулятор Android)
- Kali Net Hunter(перетворює андроїд на повноцінний інструмент хакінгу, який можна поставити навіть на termux без рут прав)
- Java Virtual Machine
- Dalvik — частина мобільної платформи Android
- IBM VM (Система Віртуальних Машин, CBM)
- Xen
- Parrot Virtual Machine
- Sun VirtualBox
- Parallels Workstation
- QEMU
- Wine
- DOSEMU
- Mode Linux (UML)
- Termux – повноцінний емулятор linux на андроїд

Хмарна віртуалізація

- AWS
- GOOGLE CLOUD
- MICROSOFT AZURE
- ORACLE CLOUD
- DIGITALOCEAN
- UAR.NET
- Server.UA
- Інші VPS \ VDS хостінги

Контейнери

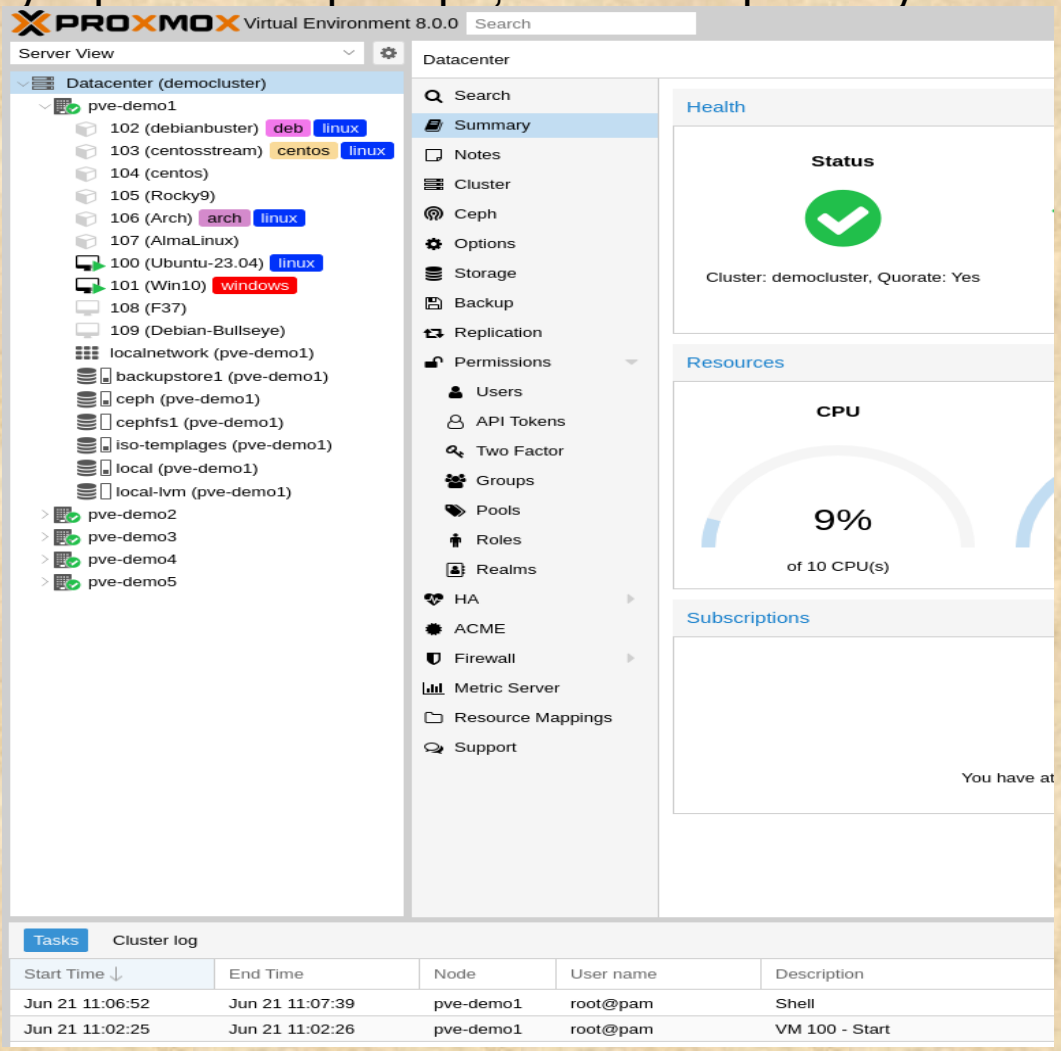
- Docker
- Virtuozzo
- Bochs
- Simics
- Serenity Virtual Station (SVISTA)
- Rocket
- LXC
- OpenVZ
- Linux VServer
- Solaris
- Windows Server Containers

Пісочниці

- ANY.RUN
- CROWDSTRIKE FALCON SANDBOX
- CISCO THREAT GRID GLOVEBOX
- CUCKOO SANDBOX
- VIRUSTOTAL
- CISCO THREAT GRID GLOVEBOX
- JOE SANDBOX
- Metadefender Cloud
- Jotty Malware Scan

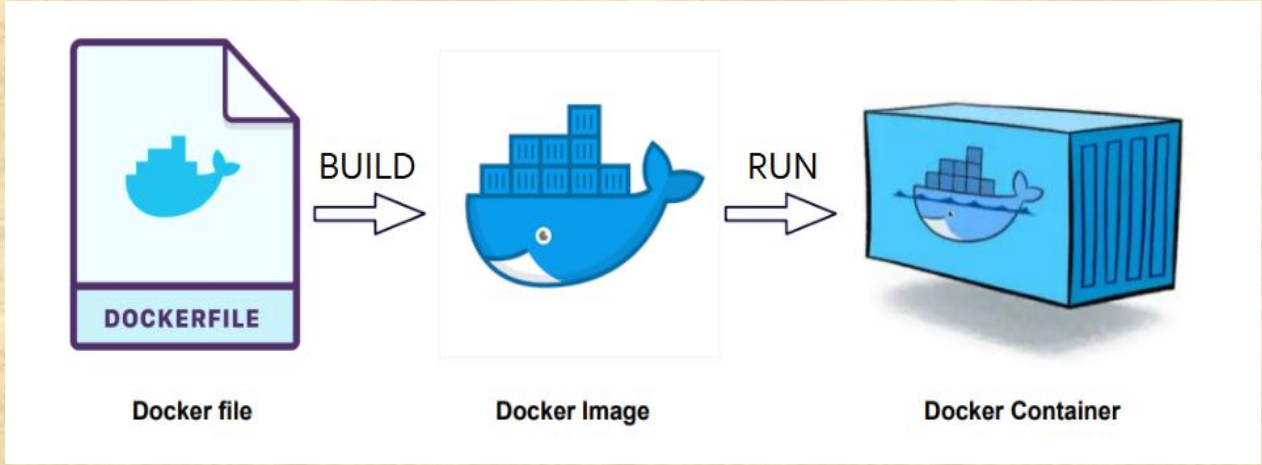
Доступ до апаратних фізичних ресурсів сервера віртуальні машини отримують завдяки гіпервізору. Він абстрагує віртуальні ресурси від апаратних і дозволяє запускати на одному фізичному обладнанні (хості) і керувати кількома ВМ, які називають гостьовими. Найпопулярніші гіпервізори, які використовуються в корпоративному середовищі:

- **Microsoft Hyper-V**
- **Proxmox**
- **Oracle VM VirtualBox**
- **ESXi Vmware**
- **QEMU KVM**
- **Xen**
- **Virtuozzo OpenVZ**



Головний принцип роботи з Docker можна описати у вигляді трьох слів: Build, Ship, Run. Кожне з них означає певний етап та елемент системи:

- **Build**. Тут мається на увазі створення Dockerfile. Це інструкція, за якою весь «корабель» будується та відправляється в «плавання». У файлі ви описуєте образ майбутнього продукту, принципи його роботи та створення.
- **Ship**. Наступний етап — Docker Image. Це готовий образ для створення та запуску контейнера. Якщо порівняти Dockerfile з планом кімнати, то Docker Image — це дизайн-проєкт із зображенням майбутнього ремонту, схемами розведення електрики та сантехніки тощо.
- **Run**. На останньому етапі з'являється Docker Container — вже запущений у роботу образ на основі Docker Image. І якщо раніше був дизайн-проєкт, то це вже готова до заселення кімната.



Так все і працює: будуєте Dockerfile, з нього створюєте Docker Image, а на його основі запускаєте Docker Container.

Dockerfile

Dockerfile — це звичайний текстовий документ, який ми називаємо Dockerfile без усіляких крапок в кінці. Цей файл розміщується у папці з проєктом. У ньому ви записуєте команди, за якими буде створюватися Image, а потім — запускатися Container.

На ілюстрації нижче прописані наступні команди:

•Беремо Python 3.10.2.

•Робимо директорію та встановлюємо туди Django.

•Копіюємо весь проєкт, відкриваємо порт 8000 та за допомогою CMD запускаємо runserver нашого Django.

Таким чином ми прописуємо в Dockerfile команди одна за одною. Так само покроково вони будуть виконуватися автоматично згідно цієї інструкції. Як бачите, синтаксис дійсно дуже простий:

Dockerfile

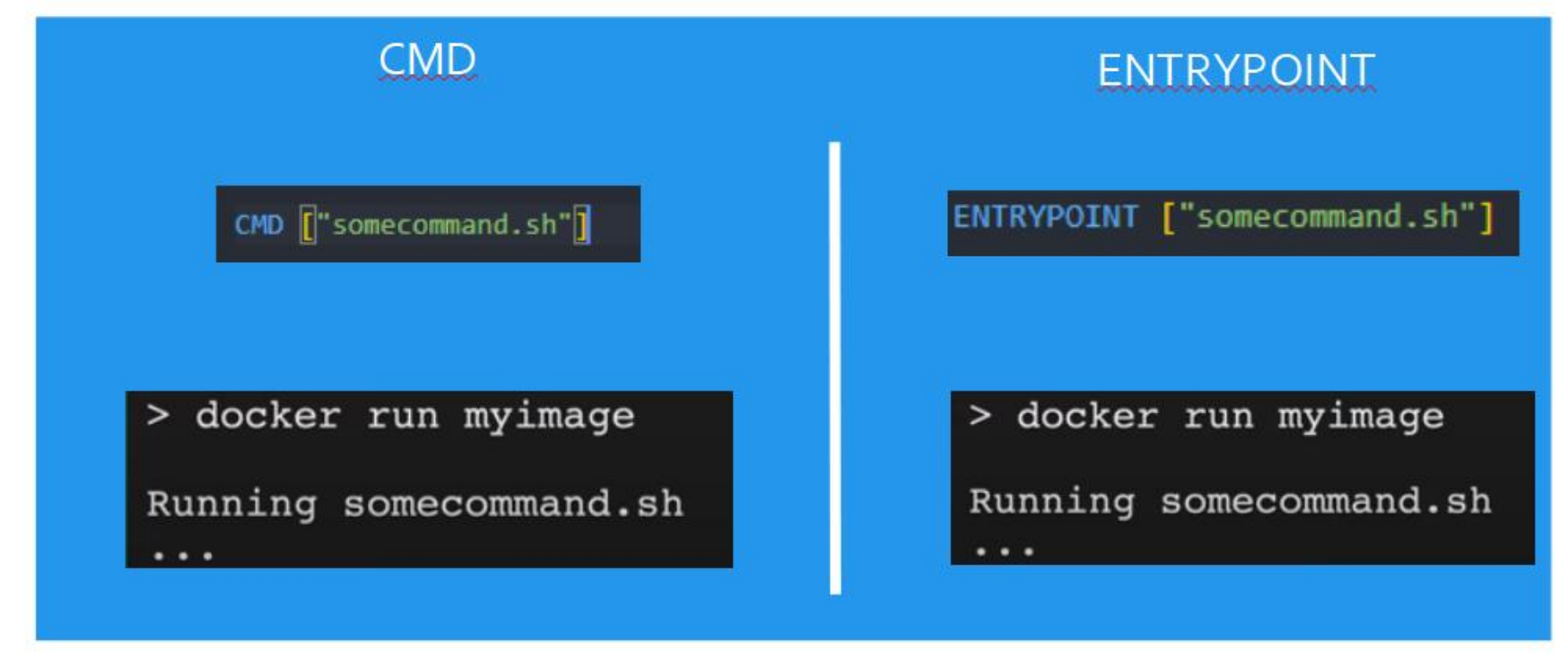
task > Dockerfile > ...

```
1   FROM python:3.10.2
2
3   WORKDIR /usr/src/task
4
5   RUN pip install Django
6
7   COPY . /usr/src/task/
8
9   EXPOSE 8000
10  CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
11
```

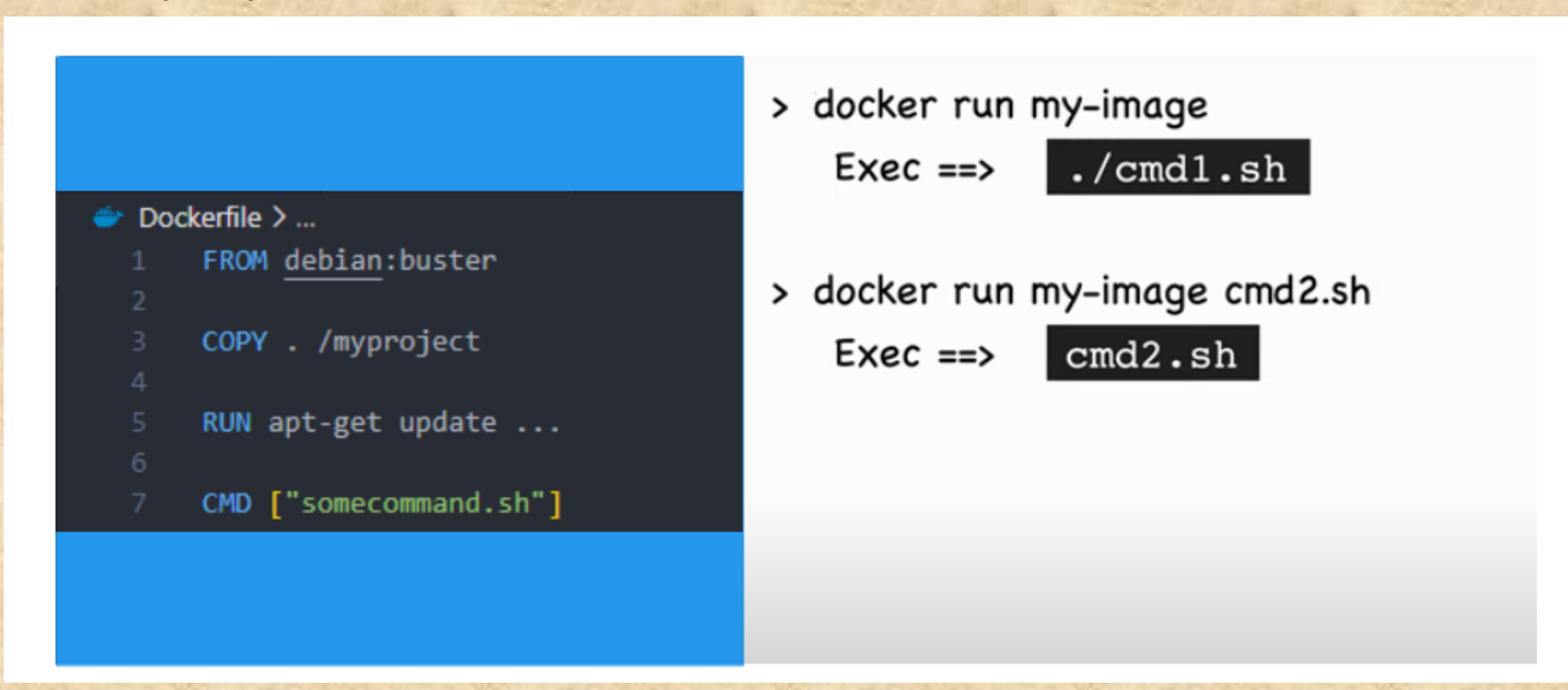
Команди CMD та ENTRYPOINT

Може здатися, що вони однакові за функціоналом, та це не зовсім так. **CMD — команда, яка прописується в консоль при запуску Dockerfile.** Якщо повернутися до попереднього прикладу, то для запуску сервера потрібні *python*, *manage.py*, *runserver*, IP та порт. І все це прописується саме в CMD, після чого запускається сервер.

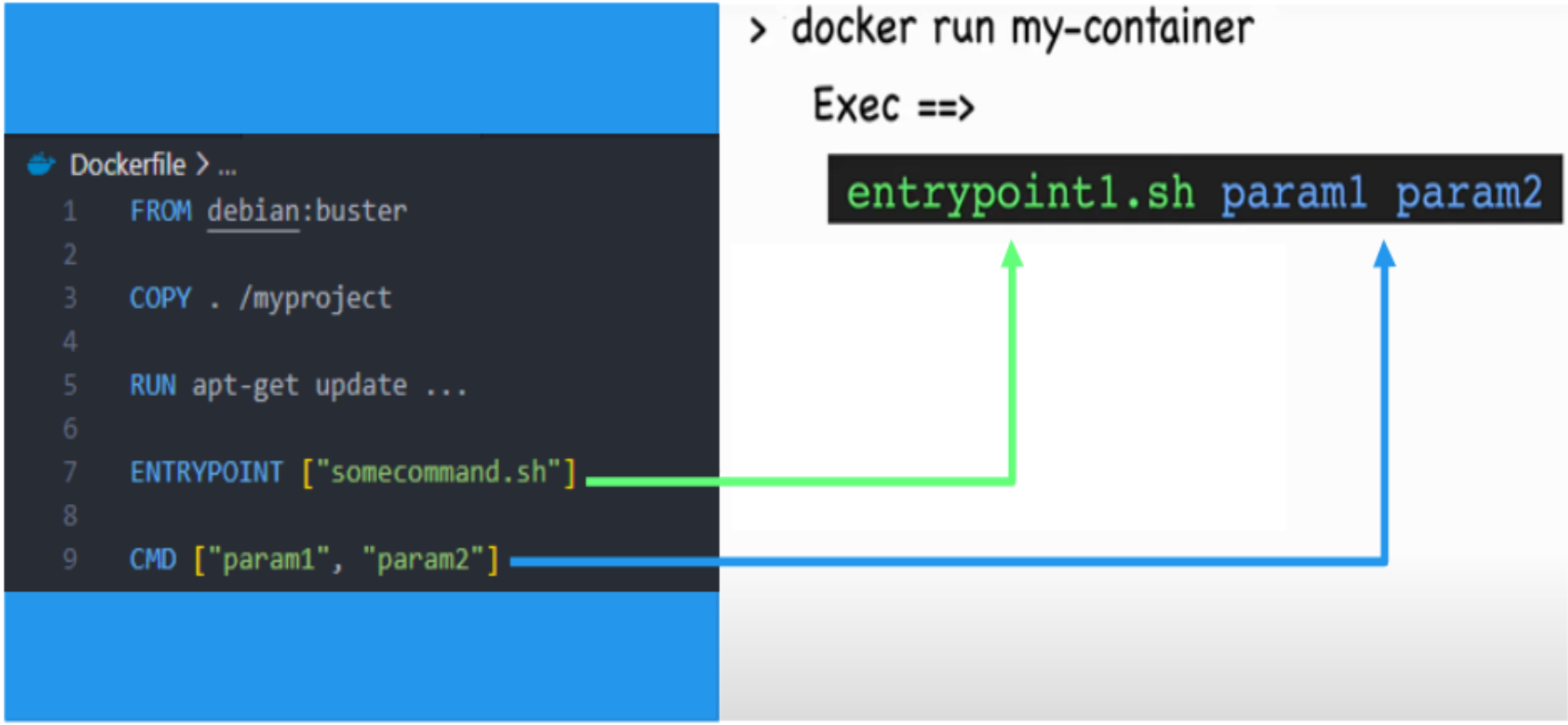
ENTRYPOINT так само запускає команду, яку ми хочемо прописати. Але він виконує це перед CMD, якщо ми використовуємо їх разом. Саме тому ENTRYPOINT і називається «точкою входу» — бо це буде перша команда.



Для розбору процесу Running CMD розглянемо приклад.
Беремо Debian, копіюємо проєкт, запускаємо `apt-get update` і зводимо CMD.
Якщо нам потрібно змінити цю команду, то при запуску Docker можна прописати `docker run image`. Тоді виконається команда, прописана в Dockerfile по замовчуванню.
Якщо ж ми хочемо виконати іншу команду, то пишемо `docker run my-image` і вказуємо нову команду. Це може бути, наприклад, вказівка перевірити список усіх елементів на Ubuntu. У результаті попередня команда буде замінена в CMD на ту, яку ми вказали:



Якщо ENTRYPOINT та CMD використовувати разом, то перша завжди стартує швидше. Вона може знадобитися, коли до основних команд з CMD потрібно запустити ще якусь команду. До прикладу, певна перевірка проєкту. В такому разі ви прописуєте відповідну команду ENTRYPOINT, а потім — основні параметри в CMD. Цей процес наведений на ілюстрації нижче:



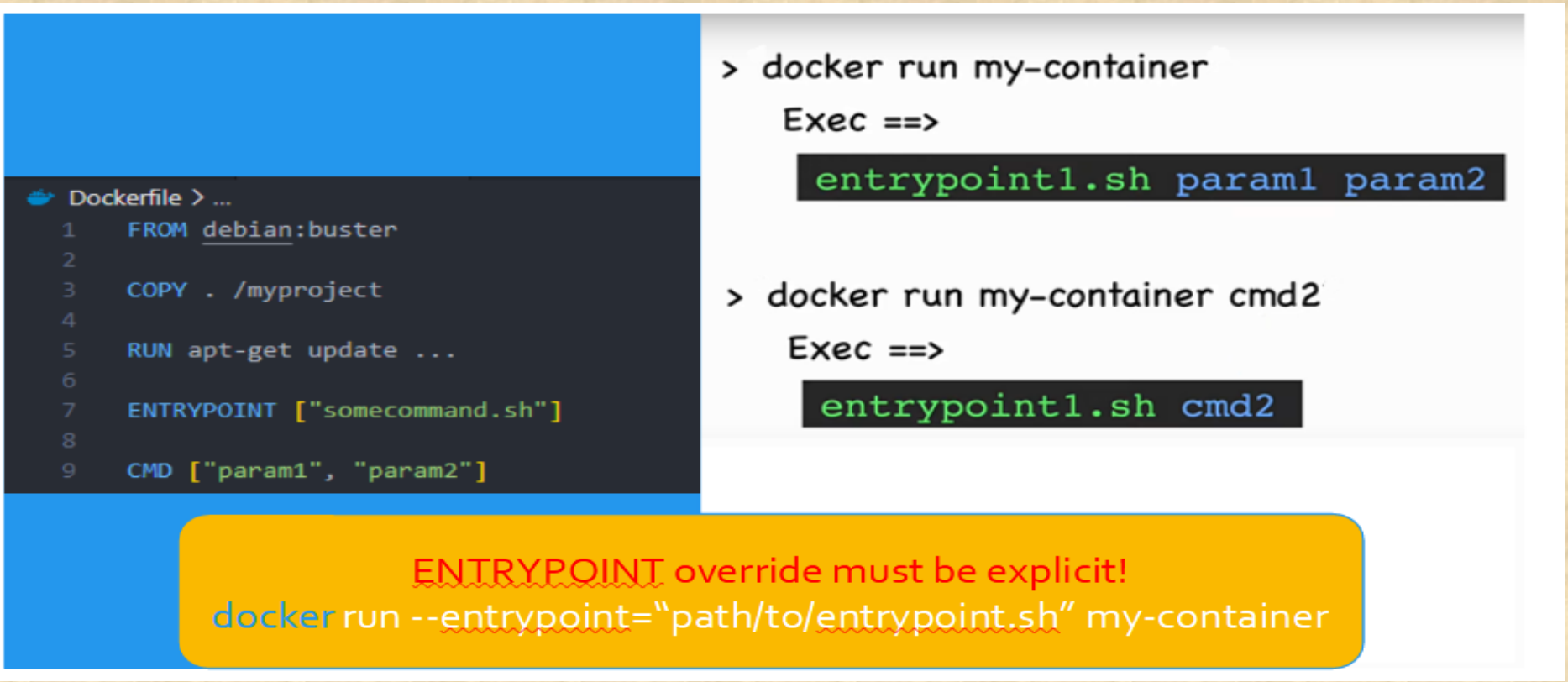
Також ви можете переписати певні параметри. При вказівці `docker run my-container` запишуться параметри та ENTRYPOINT, які вже є в Dockerfile. А якщо потрібно запустити інші CMD-команди, то знадобиться `docker run my-container cmd`:

Dockerfile > ...

```
1  FROM debian:buster
2
3  COPY . /myproject
4
5  RUN apt-get update ...
6
7  ENTRYPOINT ["somecommand.sh"]
8
9  CMD ["param1", "param2"]
```

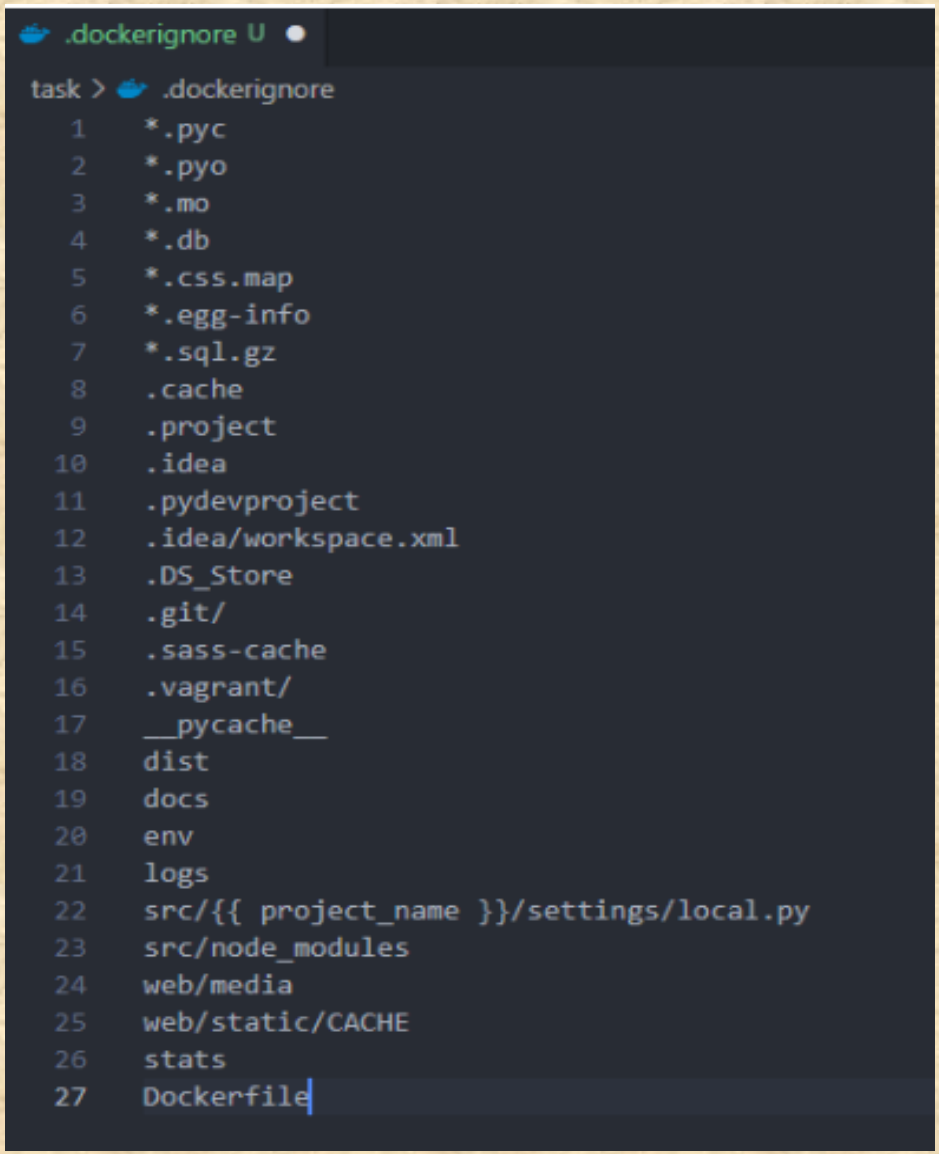
```
> docker run my-container
    Exec ==>
    entrypoint1.sh param1 param2

> docker run my-container cmd2
    Exec ==>
    entrypoint1.sh cmd2
```

Щоб переписати ENTRYPOINT, потрібно написати команду `docker run --entrypoint` і шлях до `entrypoint.sh` (це entrypoint, який буде змінюватися), а також назву контейнера. Зверніть увагу: ви маєте вказати, що переписуєте саме ENTRYPOINT:



Обираючи між CMD та ENTRYPOINT, краще використовувати CMD. З ним легше працювати. ENTRYPOINT краще залишити для тих випадків, коли контейнер має робити щось вже на старті: перевіряти процеси або чекати чогось. Тоді поєднання цих команд буде виправданим.

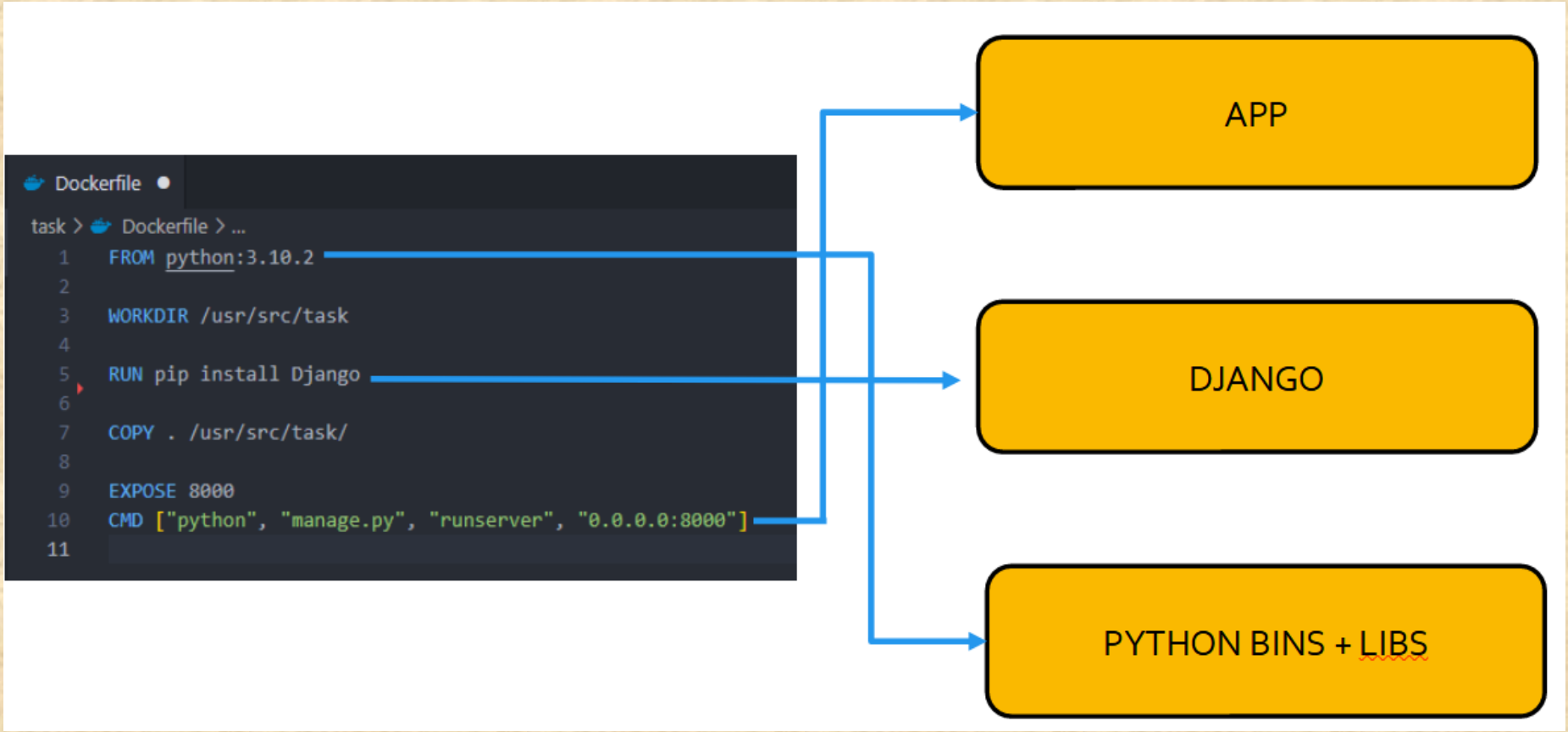
Dockerignore



Продовжуючи тему **Dockerfile** варто згадати за **Dockerignore.** Це аналог **`.gitignore`** в Git. Завдяки цьому функціоналу при копіюванні проекту в контейнер ви можете вказати певні файли, які не хочете переносити разом з іншими. Це може стосуватися pycache, логів, environments, docs тощо. У такому разі прописуєте назви файлів у dockerignore, і вони будуть просто ігноруватися при копіюванні проєкту.

Dockerfile створюється за принципом Layers — шар за шаром

Спочатку ви прописуєте основу майбутнього контейнера. Наприклад, при `FROM python` це буде Python певної версії, і він стає на базовий рівень усієї схеми. У нашому випадку далі йде Django, але можуть бути й потрібні для проекту бібліотеки, requirements.txt, залежності тощо. Все це формує середній шар документу. Наприкінці маємо верхній шар, який власне запускає застосунок — це CMD-команда:



Цей процес можна порівняти з будівництвом дому. У якості фундаменту виступає найбільша частина: Python, Ubuntu, Anaconda тощо. Далі йдуть основні поверхи, більш компактні за розміром — бібліотеки та залежності. І вже на горищі розміщується найменша та найлегша частина — run проекту контейнера.
Щоб усі шари були зрозумілими, раджу ретельно прописувати Layers.
На ілюстрації можна побачити приклад грамотного створення **Dockerfile.**

Dockerfile 1 ●

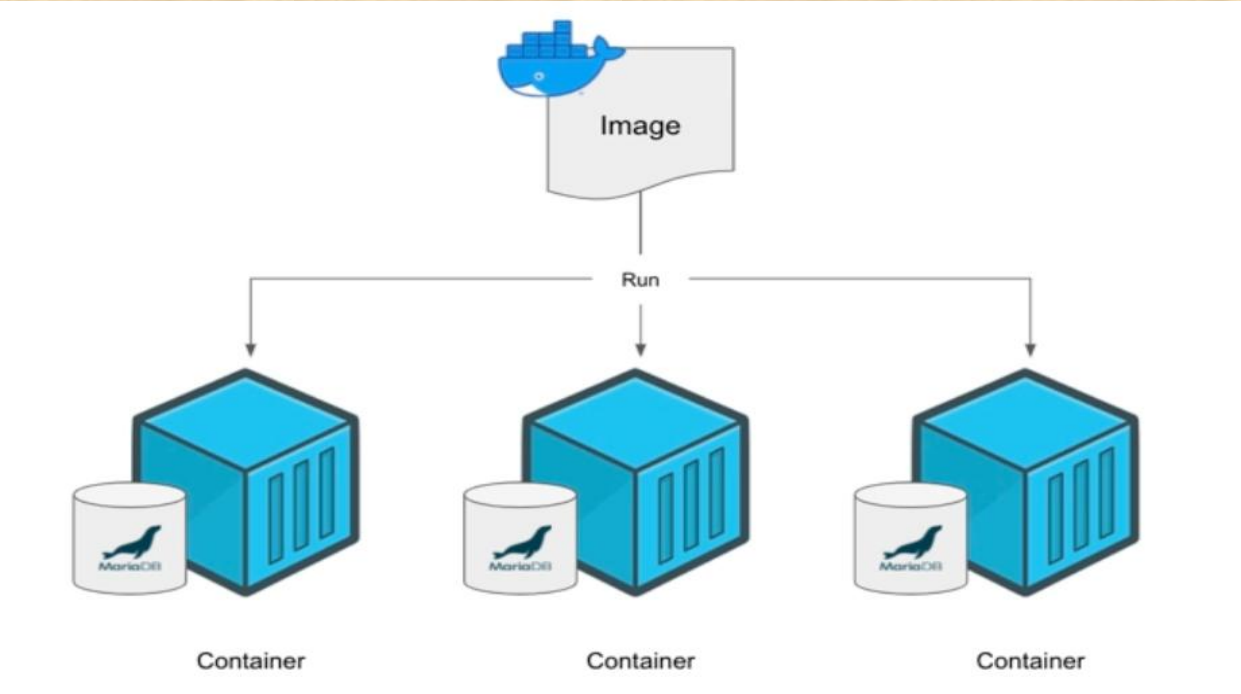
Dockerfile > ...

```
1   FROM continuumio/miniconda3:4.5.12
2   MAINTAINER Oleg Koval <oleh.koval@nixsolutions.com>
3
4   ENV JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
5   ENV PATH=/opt/conda/envs/my_venv/bin:$PATH
6
7   WORKDIR /project
8
9   COPY . /project
10
11  RUN apt-get update && \
12      apt-get -y upgrade && \
13      apt-get install -y openjdk-8-jdk && \
14      find . -name \*.pyc -delete && \
15      conda create -n my_venv python=3.7.9 && \
16      echo /bin/bash -c "source activate my_venv" && \
17      pip install --no-cache-dir -r requirements.txt
18
```

Docker Image

Docker Image — це блок, створений за інструкцією з Dockerfile, що виступає шаблоном запуску контейнерів. За допомогою цього документу можна побудувати безліч однакових контейнерів.

Припустимо, створюється один шаблон, який має встановити Python і певні залежності на нього, і потрібно п’ять однакових контейнерів. У такому випадку не потрібно прописувати п’ять Dockerfile. Достатньо одного такого файлу для створення Docker Image, який можна запускати п’ять разів.



Основні команди Docker Image

При роботі з Docker Image виділяють декілька найбільш використовуваних команд:

•`docker image --help` показує основні команди в Image з їх коротким описом.

```
PS C:\Users\gokus\Desktop\Python_proggrams> docker image --help

Usage:  docker image COMMAND

Manage images

Commands:
  build       Build an image from a Dockerfile
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  inspect     Display detailed information on one or more images
  load        Load an image from a tar archive or STDIN
  ls          List images
  prune       Remove unused images
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rm          Remove one or more images
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

Run 'docker image COMMAND --help' for more information on a command.
```

Основні команди Docker Image

•`docker image build [OPTIONS] PATH` будує сам Image для подальшого запуску Container. За допомогою цієї команди можна назвати свій Image. Для цього необхідно додати тег -t, а за ним вказати бажану назву (наприклад test). Залишається прописати шлях до директорії (крапка у наведеному прикладі означає поточну директорію). Після запуску команди і почне будуватися Image: від створення директорії, встановлення Django, whitenoise, gunicorn до копіювання.

```
PS C:\Users\gokus\Desktop\Python_proggrams\task> docker image build -t test .
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 297B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/python:3.10.2
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load build context
=> => transferring context: 294.51kB
=> [1/7] FROM docker.io/library/python:3.10.2@sha256:3204faabc2f0b5e0939bdb8b29079a2a330c38dee92a22482a9ed449c5649a55
=> => resolve docker.io/library/python:3.10.2@sha256:3204faabc2f0b5e0939bdb8b29079a2a330c38dee92a22482a9ed449c5649a55
=> => sha256:17e2d81e5757980ee40742d77dd5d3e1a69ad0d6dacb13064e1b018a6664ec72 2.22kB / 2.22kB
=> => sha256:4ff1945c672b08a1791df62afaaf8aff14d3047155365f9c3646902937f7ffe6 5.15MB / 5.15MB
=> => sha256:ff5b10aec998344606441aec43a335ab6326f32aae331aab27da16a6bb4ec2be 10.87MB / 10.87MB
=> => sha256:3204faabc2f0b5e0939bdb8b29079a2a330c38dee92a22482a9ed449c5649a55 2.14kB / 2.14kB
=> => sha256:178dcaa62b393b539abc8b866c39be81e8ade01786880dc5d17ce3fe02426dbb 8.55kB / 8.55kB
=> => sha256:e4d61adff2077d048c6372d73c41b0bd68f525ad41f5530af05098a876683055 54.92MB / 54.92MB
=> => sha256:12de8c754e45686ace9e25d11bee372b070eed5b5ab20aa3b4fab8c936496d02 54.58MB / 54.58MB
=> => sha256:ada1762e76024dd216336649249fc2470257acc5af277bae3c71710382df345f 196.52MB / 196.52MB
=> => sha256:2f2b2e030155d32fbf69b4feeced1877ce0e3f21b5f27b98eb080c3ef01a70df 6.29MB / 6.29MB
=> => extracting sha256:e4d61adff2077d048c6372d73c41b0bd68f525ad41f5530af05098a876683055
=> => sha256:fa63e4e5310b6c6e8fc69825073a49a506de7e0dd1628b292cd59cf050744ef9 19.20MB / 19.20MB
=> => sha256:7c5288a5b77966ccab65aaa8529a1985574b516b5ff97c4c447fceb342c4933e 238B / 238B
=> => sha256:38121472aa0128f87b31fde5c07080418cc17b4a8ee224767b59e24c592ff7d3 2.34MB / 2.34MB
=> => extracting sha256:4ff1945c672b08a1791df62afaaf8aff14d3047155365f9c3646902937f7ffe6
=> => extracting sha256:ff5b10aec998344606441aec43a335ab6326f32aae331aab27da16a6bb4ec2be
=> => extracting sha256:12de8c754e45686ace9e25d11bee372b070eed5b5ab20aa3b4fab8c936496d02
=> => extracting sha256:ada1762e76024dd216336649249fc2470257acc5af277bae3c71710382df345f
=> => extracting sha256:2f2b2e030155d32fbf69b4feeced1877ce0e3f21b5f27b98eb080c3ef01a70df
=> => extracting sha256:fa63e4e5310b6c6e8fc69825073a49a506de7e0dd1628b292cd59cf050744ef9
=> => extracting sha256:7c5288a5b77966ccab65aaa8529a1985574b516b5ff97c4c447fceb342c4933e
=> => extracting sha256:38121472aa0128f87b31fde5c07080418cc17b4a8ee224767b59e24c592ff7d3
=> [2/7] WORKDIR /usr/src/task
=> [3/7] RUN pip install Django
=> [4/7] RUN pip install whitenoise
=> [5/7] RUN pip install gunicorn
=> [6/7] RUN pip install dj-database-url
=> [7/7] COPY . /usr/src/task/
=> exporting to image
=> => exporting layers
=> => writing image sha256:4eed0b356e17c3bd6e273e6573654a3611b976ba1bf6d2098c56b3f01d1afbbd
=> => naming to docker.io/library/test
```

Основні команди Docker Image

- `docker image inspect [OPTIONS] IMAGE.`

Ця команда видає всю інформацію про Image. Це можуть бути ID, дата створення, пов'язані контейнери та енвайроменти, версія Python, наявні pip і get pip, CMD і ENTRYPOINT, Volumes, директорії тощо. При запуску команди необхідно вказати назву Image, який вас цікавить.

```
PS C:\Users\gokus\Desktop\Python_proggrams\task> docker image inspect test
[
    {
        "Id": "sha256:4eed0b356e17c3bd6e273e6573654a3611b976ba1bf6d2098c56b3f01d1afbbd",
        "RepoTags": [
            "test:latest"
        ],
        "RepoDigests": [],
        "Parent": "",
        "Comment": "buildkit.dockerfile.v0",
        "Created": "2022-08-02T07:47:02.338075521Z",
        "Container": "",
        "ContainerConfig": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": null,
            "Cmd": null,
            "Image": "",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": null
        },
```

Основні команди Docker Image

- `docker image ls.`

Ця команда показує список усіх Image, які є в Docker. На прикладі зображено перелік із декількох тестових Docker Image:

```
PS C:\Users\gokus\Desktop\Python_proggrams\task> docker image ls
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
test            latest    4eed0b356e17   21 minutes ago   959MB
dockertest69    latest    a4de5b19aeb7   7 days ago       13.6GB
dockertest8     latest    125b8ca377d4   10 days ago      13.4GB
dockertest7     latest    acf337a7b7b6   10 days ago      13.4GB
dockertest6     latest    0a86f7141881   10 days ago      13.4GB
dockertest5     latest    ac3a15269419   10 days ago      13.4GB
dockertest2     latest    f63906d544f7   10 days ago      13.4GB
PS C:\Users\gokus\Desktop\Python_proggrams\task> docker image tag test owenzbs/first-repo:test
PS C:\Users\gokus\Desktop\Python_proggrams\task> docker image ls
REPOSITORY           TAG       IMAGE ID       CREATED          SIZE
test                 latest    4eed0b356e17   22 minutes ago   959MB
owenzbs/first-repo   test      4eed0b356e17   22 minutes ago   959MB
dockertest69         latest    a4de5b19aeb7   7 days ago       13.6GB
dockertest8          latest    125b8ca377d4   10 days ago      13.4GB
dockertest7          latest    acf337a7b7b6   10 days ago      13.4GB
dockertest6          latest    0a86f7141881   10 days ago      13.4GB
dockertest5          latest    ac3a15269419   10 days ago      13.4GB
dockertest2          latest    f63906d544f7   10 days ago      13.4GB
```

Основні команди Docker Image

- `docker image rm [OPTIONS] IMAGE`

призначена для видалення Image. Нижче на ілюстрації я зобразив типовий сценарій такої процедури. Спочатку за допомогою `docker image ls` отримав перелік усіх Image. Далі прописав `docker image rm`, зазначивши назву Image, який потрібно видалити — `dockertest2`. Ще одна перевірка `docker image ls` показала: того Image більше немає у Docker.

```
PS C:\Users\gokus\Desktop\Python_proggrams\task> docker image ls
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
test                    latest    4eed0b356e17   52 minutes ago   959MB
owenzbs/first-repo      test      4eed0b356e17   52 minutes ago   959MB
dockertest69            latest    a4de5b19aeb7   7 days ago       13.6GB
dockertest8             latest    125b8ca377d4   10 days ago      13.4GB
dockertest7             latest    acf337a7b7b6   10 days ago      13.4GB
dockertest6             latest    0a86f7141881   10 days ago      13.4GB
dockertest5             latest    ac3a15269419   10 days ago      13.4GB
dockertest2             latest    f63906d544f7   10 days ago      13.4GB
PS C:\Users\gokus\Desktop\Python_proggrams\task> docker image rm dockertest2
Untagged: dockertest2:latest
Deleted: sha256:f63906d544f7e1cb82775f563d9f6ebd5dc816b7a9769227434e95f655462710
Deleted: sha256:42d744e590bb291ee10daa28bee861b9745223683946b62766037ae72811fc40c
PS C:\Users\gokus\Desktop\Python_proggrams\task> docker image ls
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
test                    latest    4eed0b356e17   52 minutes ago   959MB
owenzbs/first-repo      test      4eed0b356e17   52 minutes ago   959MB
dockertest69            latest    a4de5b19aeb7   7 days ago       13.6GB
dockertest8             latest    125b8ca377d4   10 days ago      13.4GB
dockertest7             latest    acf337a7b7b6   10 days ago      13.4GB
dockertest6             latest    0a86f7141881   10 days ago      13.4GB
dockertest5             latest    ac3a15269419   10 days ago      13.4GB
```

Основні команди Docker Image

- `docker image rm [OPTIONS] IMAGE`

призначена для видалення Image. Нижче на ілюстрації типовий сценарій такої процедури. Спочатку за допомогою `docker image ls` отримується перелік усіх Image. Далі при виконанні `docker image rm`, зазначено назву Image, який потрібно видалити — `dockertest2`. Ще одна перевірка `docker image ls` показала: того Image більше немає у Docker.

```
PS C:\Users\gokus\Desktop\Python_proggrams\task> docker image ls
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
test                    latest    4eed0b356e17   52 minutes ago   959MB
owenzbs/first-repo      test      4eed0b356e17   52 minutes ago   959MB
dockertest69            latest    a4de5b19aeb7   7 days ago       13.6GB
dockertest8             latest    125b8ca377d4   10 days ago      13.4GB
dockertest7             latest    acf337a7b7b6   10 days ago      13.4GB
dockertest6             latest    0a86f7141881   10 days ago      13.4GB
dockertest5             latest    ac3a15269419   10 days ago      13.4GB
dockertest2             latest    f63906d544f7   10 days ago      13.4GB
PS C:\Users\gokus\Desktop\Python_proggrams\task> docker image rm dockertest2
Untagged: dockertest2:latest
Deleted: sha256:f63906d544f7e1cb82775f563d9f6ebd5dc016b7a9769227434e95f655462710
Deleted: sha256:42d744e590bb291ee10daa28bee861b9745223683946b62766037ae72811fc40c
PS C:\Users\gokus\Desktop\Python_proggrams\task> docker image ls
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
test                    latest    4eed0b356e17   52 minutes ago   959MB
owenzbs/first-repo      test      4eed0b356e17   52 minutes ago   959MB
dockertest69            latest    a4de5b19aeb7   7 days ago       13.6GB
dockertest8             latest    125b8ca377d4   10 days ago      13.4GB
dockertest7             latest    acf337a7b7b6   10 days ago      13.4GB
dockertest6             latest    0a86f7141881   10 days ago      13.4GB
dockertest5             latest    ac3a15269419   10 days ago      13.4GB
```

Висновки

У межах першої лекції ми розглянули ключові засади побудови ізольованих середовищ для вивчення та практичного застосування інструментів кібербезпеки. Було з'ясовано, що **віртуальна машина (ВМ)** — це програмна емуляція фізичного комп'ютера, яка дозволяє запускати окрему операційну систему поверх основної, забезпечуючи високий рівень ізоляції. Натомість **контейнерне середовище** використовує спільне ядро хост-системи, що робить його легшим і швидшим у запуску, але менш ізольованим порівняно з ВМ.

Ми детально порівняли ці два підходи, виокремивши їхні сильні та слабкі сторони. Також було наведено **приклади програмного забезпечення** для роботи з віртуальними машинами (зокрема, VirtualBox, VMware) та контейнерами (Docker, Podman). Окрему увагу приділено **гіпервізорам** — програмному або апаратному забезпеченню, що керує роботою віртуальних машин. Наприкінці лекції ми розглянули **практичний приклад створення контейнера в Docker**, що дало змогу наочно побачити, як контейнеризація працює на практиці.

Отже, ми отримали базове розуміння двох основних технологій віртуалізації, які є фундаментом для побудови сучасних лабораторних стендів у кібербезпеці. Розуміння їхніх відмінностей і правильний вибір інструмента є критичним для ефективного та безпечного тестування.