

EASS שיעור 5- חזרה על משימת הכנה לתרגיל 1 ו-FastAPI

השלמת חלקים 6+7 של משימת ההכנה לתרגיל 1:

6. ניצור dockerfile חדש באותה התיקייה עם שם אחר- localhost.dockerfile . קובץ זה יעתיק ויריץ קובץ python חדש שנקרא לו local_client.py

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cd ./http-api-demo-natalieafalo
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo-natalieafalo$ vi localhost.dockerfile
```

```
FROM python:3.8
RUN pip install httpx
COPY ./local_client.py .
CMD ["python", "./local_client.py"]
~
~
~
~
~
:wq
```

כעת ניצור את קובץ local_client.py שמבצע את פעולות get ו-post דרך localhost כך:

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo-natalieafalo$ vi local_client.py
```

```
import httpx

get_response = httpx.get("http://localhost/get")

post_response = httpx.post("http://localhost/post")

print("GET response: ",get_response.json())

print("POST response: ",post_response.json())
~
~
~
~
:wq
```

7. ניצור את הimage החדש. מכיוון שבנתיב (path) הזה יש כבר קובץ dockerfile צריך להודיע בפקודה שאנו רוצים לבנות את הקובץ localhost.dockerfile עם -f כך:

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo-natalieafalo$ docker build . -f localhost.Dockerfile -t local-http-api-demo
[+] Building 229.8s (8/8) FINISHED
=> [internal] load build definition from localhost.Dockerfile
    0.1s
=> => transferring dockerfile: 152B
    0.0s
=> [internal] load .dockerignore
    0.0s
=> => transferring context: 2B
    0.0s
```

כדי שפעולות get ו-post יוכלו לפעול לוקלית על <http://localhost> עלינו להריץ ברקע container של האימג' kennethreitz/httpbin בעזרת הפקודה שבאתר של httpbin בצירוף כך:

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo-natalieafalo$ docker run -d -p 80:80 kennethreitz/httpbin
87f1a16cc458699e2db8184bde11bab770aaf4591c346e8c379bc0774112169f
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo-natalieafalo$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
87f1a16cc458	kennethreitz/httpbin	"unicorn -b 0.0.0.0..."	3 seconds ago	Up 2 seconds	0.0.0.0:80->80/tcp	modest_payne

על מנת שהimage החדש שיצרנו יוכל לתקשר עם host שמריץ את kennethreitz/httpbin נצטרך לקשר ביניהם בעזרת פרמטר network:

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker run --network host local-http-api-demo
GET response: {'args': {}, 'headers': {'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate', 'Connection': 'keep-alive', 'Host': 'localhost', 'User-Agent': 'python-httpx/0.22.0'}, 'origin': '172.17.0.1', 'url': 'http://localhost/get'}
POST response: {'args': {}, 'data': '', 'files': {}, 'form': {}, 'headers': {'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate', 'Connection': 'keep-alive', 'Content-Length': '0', 'Host': 'localhost', 'User-Agent': 'python-httpx/0.22.0'}, 'json': None, 'origin': '172.17.0.1', 'url': 'http://localhost/post'}
```

ניתן לראות בjson של התגובה שהיא אכן הגיעה מhost:localhost.
נותר להעלות את הקבצים החדשים localhost.dockerfile ו-local_client.py לגיטהאב ולעדכן את ההוראות הפעלה בREADME (הוסבר בשיעור 4 חלק 2).

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cd ./http-api-demo-natalieafalo
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo-natalieafalo$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    local_client.py
    localhost.dockerfile

nothing added to commit but untracked files present (use "git add" to track)
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo-natalieafalo$ git add local_client.py
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo-natalieafalo$ git add localhost.dockerfile
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo-natalieafalo$ git commit -m "Added localhost.dockerfile and local_client.py : parts 6+7"
[main 0e4594d] Added localhost.dockerfile and local_client.py : parts 6+7
 2 files changed, 13 insertions(+)
 create mode 100644 local_client.py
 create mode 100644 localhost.dockerfile
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo-natalieafalo$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 599 bytes | 33.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/EASS-HIT-2022/http-api-demo-natalieafalo.git
 0b23373..0e4594d  main -> main
```

(למי שמעוניין, הrepo שלי בנושא הוא ציבורי. זמין בקישור <https://github.com/EASS-HIT-2022/http-api-demo-natalieafalo>, ניתן לראות שם מבנה של README כמו שהמרצה הדגים בשיעור בנוסף לקבצים)

:FastAPI

- דרך קלה ומהירה לתכנת API באמצעות PYTHON.
- אנו נשתמש ב-fastapi כדי ליצור את התקשורת בין ה-frontend ל-backend. כלומר, ה-frontend ייצור בקשות (request) של HTTP כמו GET, POST וב-backend נגדיר את התשובות (response) לכל בקשה (בהתאם לפרמטרים וסוג הבקשה).
- ניתן להגדיר פונקציות ב-FastAPI שיהיו סינכרוניות (בצורה הרגילה def) או אסינכרוניות (בעזרת async def ובתוך הפונקציה להגדיר פעולה await)

If your utility function is a normal function (with def), it will be called directly (as you write it in your code), if the function is created with "async def" then you should await for that function when you call it in your code.

The "await" tells Python that it has to wait for the action to finish doing its thing. With that, Python will know that it can go and do something else in the meanwhile (like receiving another request).

- הגדרה לשפה אסינכרונית:

Asynchronous (async) programming lets you execute a block of code without stopping (or *blocking*) the entire thread where the action is being executed. A common myth about async code is that it improves performance, which isn't necessarily true. Instead, the major perk of async programming is that it increases the number of tasks (*throughput*) that can be executed concurrently without having to block the thread where these actions are taking place.

דוגמת הטבח- במסעדה ישנו מלצר אחד וטבח אחד הנותנים שירות ל-3 שולחנות. במצב סינכרוני, המלצר לוקח הזמנה משולחן 1 ומביא אותה לטבח, ולא ממשיך עד שהוא מקבל מהטבח את כל מה ששולחן 1 הזמין. במצב אסינכרוני, המלצר מביא לטבח את ההזמנה של שולחן 1 ובמקום להמתין הוא ממשיך ולוקח הזמנה מהשולחן הבא (כלומר, ממשיך לעבוד עד שהטבח יסיים להכין את הזמנה 1 או עד שהוא יהיה חייב שהטבח יסיים בשביל להמשיך).

- Pydantic : ספרייה שמבצעת PARSE DATA- המרה של DATA מפורמט אחד לאחר. נמצאת בסיפריית fastapi

אנו נשתמש בספרייה זאת כדי לבצע המרה של JSON שמתקבלים בrequest של frontend לאובייקט פייתוני שה-backend יכול להבין ולעבד את המידע שלו, ולאחר מכן נמיר את האובייקט של ה-response לJSON כדי שנוכל לשלוח אותו כתשובה ל-backend. לדוגמה- frontend שולח בקשת POST שמגיעה ל-backend כJSON. הספרייה יכולה להמיר את ה-JSON לאובייקט פייתוני כשהפרמטרים של הבקשה יומרו למשתנים רגילים, ה-backend יחזיר אובייקט בהתאם לפרמטרים והספרייה תמיר אותו ל-JSON שלבסוף יישלח ל-frontend.

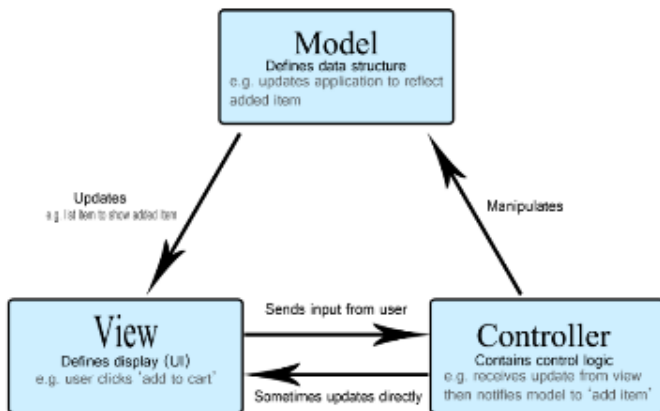
- Uvicorn : מאפשרת הרצת צד שרת server כשעובדים עם fastapi a minimal low-level server/application interface for async frameworks הערות- בפקודה יש להוסיף --reload כדי שהמערכת תעדכן את עצמה אוטומטית (לדוגמה אם נשנה את הקובץ main). בנוסף, הפורט חייב להיות מספר גדול יותר מ-1024 כי אסור לתת מספר קטן יותר ממנו למשהו שאינו ה-root.

- ניהול מידע: העבודה עם JSON יכולה להיות מבולגנת. ישנם מבני object שאיתם נוכל לעבוד בקוד במקום directory:

1. DTO -data transfer object , objects that carry data between processes in order to reduce the number of methods calls
2. ORM -object relational mapping , technique for converting data between incompatible type systems using object-oriented programming languages.

- MVC: תבנית Model-View-Controller היא תבנית עיצוב בהנדסת תוכנה המשמשת להפשטת יישום כלשהו. התבנית מתארת טכניקה לחלוקת היישום לשלושה חלקים: "מודל", "תצוגה" ו"בקר".

- **Model** The model manages the data, logic and rules of the application.
- **View** Any representation of information such as a chart, diagram or table.
- **Controller** Accepts input and converts it to commands for the model or view.



דוגמה להרצת fastapi בצד שרת באמצעות uvicorn:

1. תחילה, יש לבצע התקנה ראשונית של fastapi ו-uvicorn בעזרת pip install-

```
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ pip3 install fastapi
Requirement already satisfied: fastapi in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (0.62.0)
Requirement already satisfied: starlette<0.17.0,>=0.13.0 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from fastapi) (0.13.0)
Requirement already satisfied: pydantic<1.9.0,>=1.7.0 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from fastapi) (1.8.2)
Requirement already satisfied: anyio<4,>=3.0.0 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from starlette->fastapi) (3.2.1)
Requirement already satisfied: typing-extensions<3.7.4,>=3.7.4 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from anyio->starlette->fastapi) (3.7.4)
Requirement already satisfied: idna<2.8,>=2.5 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from anyio->starlette->fastapi) (2.8)
Requirement already satisfied: sniffio<1.1,>=1.0 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from anyio->starlette->fastapi) (1.0)
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ pip3 install uvicorn[standard]
Requirement already satisfied: uvicorn[standard] in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (0.16.0)
Requirement already satisfied: click>7.0 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from uvicorn[standard]) (7.1.2)
Requirement already satisfied: asgiref<3.4,>=3.3 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from uvicorn[standard]) (3.3.1)
Requirement already satisfied: h11>0.8 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from uvicorn[standard]) (0.11.0)
Requirement already satisfied: websockets>10.0 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from uvicorn[standard]) (10.3)
Requirement already satisfied: watchgod>0.6 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from uvicorn[standard]) (0.6)
```

(הערה חשובה- כשנכתוב את backend שלנו, ניצור קובץ requirements.txt שבו יהיו רשומות שתי השורות האלה, כדי שמי שירצה בעתיד להריץ את הקוד שלנו יידע שהוא צריך לבצע את ההתקנות שבקובץ זה על מנת שהוא יעבוד)

2. ניצור קובץ main.py שמגדיר (מבצע מימוש עבור) פקודות HTTP-

```
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ vi main.py
```

```
from fastapi import FastAPI
import time

app=FastAPI()

@app.get("/v1") # / is root path, and v1 represents version 1
def get_root():
    return {"hello-world":"demo-backend"} #When we send GET request to this
    backend, this will be the response

@app.get("/v1/clock")
async def clock():
    return {"clock":time.asctime()}

~
~
~
~
~
~
~
:wq
```

3. נעביר את הקובץ לתיקייה נפרדת ונשנה את המיקום שלנו לתיקייה זו-

```
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ mkdir demo-fastapi
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ mv main.py demo-fastapi
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cd ./demo-fastapi
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-fastapi$ ls
main.py
```

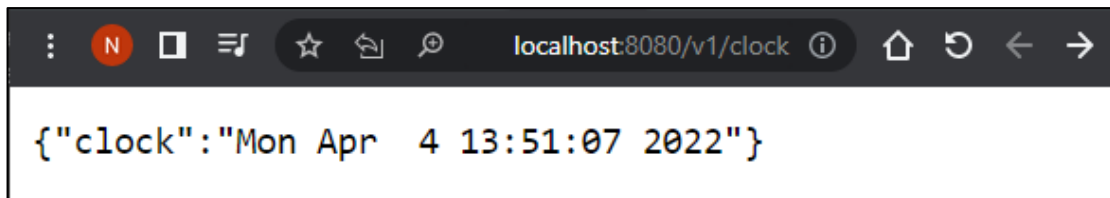
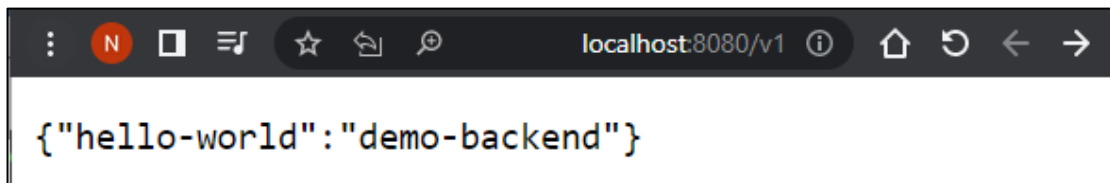
4. נריץ את uvicorn במצב reload עבור קובץ main.py-

```
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/demo-fastapi$ uvicorn main:app --host 0.0.0.0 --port 8080 --reload
INFO: Will watch for changes in these directories: ['/mnt/c/Users/natal/demo-fastapi']
INFO: Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
INFO: Started reloader process [300] using watchgod
INFO: Started server process [302]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

ב-Dockerfile נרשום את המשפט הזה כך: (לאחר RUN של pip install לכל הספריות הדרושות ו-COPY של הקבצים הרלוונטים, ביניהם main.py)

CMD ["uvicorn" , "main:app" , "--host" , "0.0.0.0" , "--port" , "8080"]

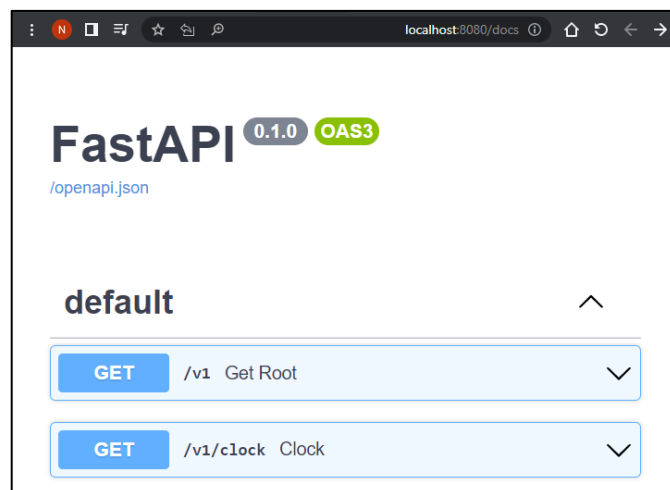
5. נבדוק האם הפקודות פועלות בעזרת הדפדפן. נריץ את localhost שהוא 0.0.0.0 דרך פורט 8080 ונוסיף את הנתוב path שבנו נמצאת הפקודה לפי הקובץ main.py-

ואם נסתכל בטרמינל לאחר מכן נוכל לראות את האישור לכך שהוחזרה תשובה-

```
INFO: 127.0.0.1:41778 - "GET /v1 HTTP/1.1" 200 OK
INFO: 127.0.0.1:41780 - "GET /v1/clock HTTP/1.1" 200 OK
```

באמצעות swagger-swagger כלומר כשנכניס localhost:8080/docs בדפדפן, נוכל לראות ולהריץ את כל הפקודות שכתבנו בפורמט נוח יותר-



דוגמה לשימוש בpydantic:

1. נשתמש בקובץ שייצרנו בדוגמה הקודמת ונכניס בו את השינויים הבאים: נייבא את pydantic, ממנו נייבא את BaseModel, ניצור מחלקה שמקבלת שני פרמטרים id, zone וניצור פקודת POST שתקבל את הפרמטרים ותחזיר תשובה בהתאם-

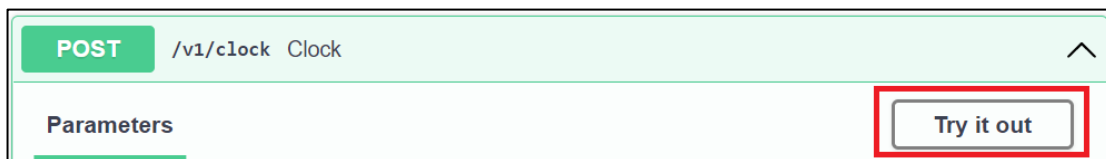
```
from fastapi import FastAPI
import time
from pydantic import BaseModel

class RequestClock(BaseModel):
    id: int
    zone: str

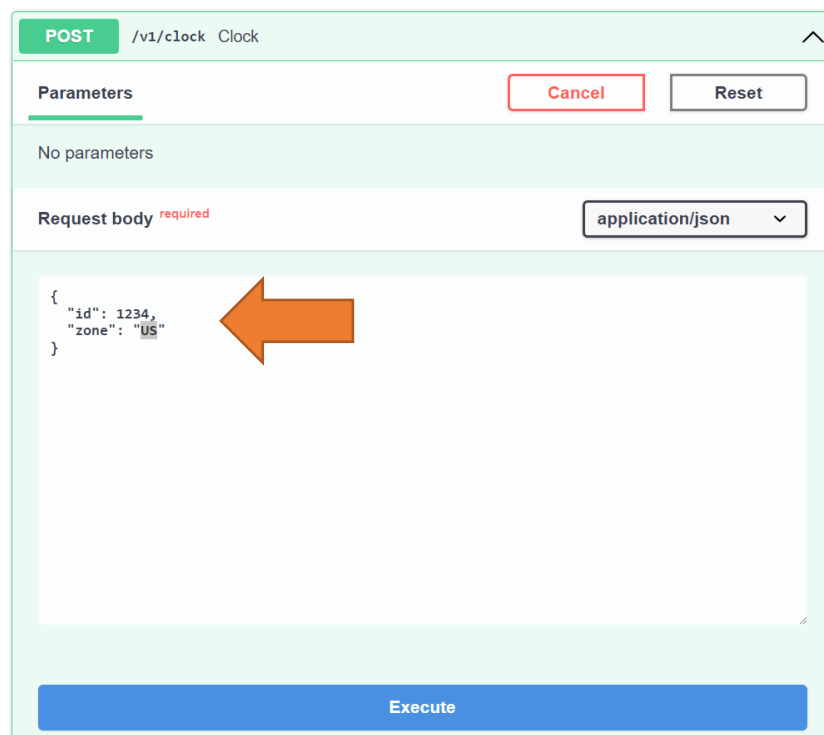
app=FastAPI()

@app.post("/v1/clock")
async def clock(req: RequestClock):
    return {"clock":time.asctime(), "req.zone": req.zone}
```

2. נשמור את הקובץ, נריץ את uvicorn כמו בדוגמה הקודמת, ניגש לswagger (localhost:8080/docs) ונבצע הרצה בעזרתו-



נעדכן בגוף ההודעה את הפרמטרים שנרצה ונלחץ Execute-



Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/v1/clock' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 1234,
    "zone": "US"
  }'
```

Request URL

```
http://localhost:8080/v1/clock
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "clock": "Mon Apr 4 14:46:45 2022", "req.zone": "US" }</pre> <p>Response headers</p> <pre>content-length: 52 content-type: application/json date: Mon, 04 Apr 2022 11:46:45 GMT server: uvicorn</pre>

Responses

Code	Description	Links
200	Successful Response	No links

(הערה חשובה- כשניצור את הפרוייקט שלנו, המחלקות של pydantic יהיו בקובץ נפרד מהפיקודות HTTP של fastapi)

מושגים נוספים שהועלו במהלך ההרצאה:

- Microservice : ארכיטקטורה ששוברת כל אפליקציה ליחידות קטנות ועצמאיות. (צד שרת, צד לקוח, מאגר נתונים SQL וכו')
- Backend : חלק באתר, בתוכנה או באפליקציה שהמשתמש לא רואה. מדובר בד"כ בקוד ותהליכים שקורים מאחורי הקלעים במחשב מרוחק (ה-Server).
- Frontend : אחראי על איסוף ועיבוד הקלט מהמשתמש, על מנת להביא אותו לצורה שה-backend יכול להשתמש בה. ממשק שבין המשתמש ל-backend.
- I/O : קלט/פלט
- Kernel : הרכיב המרכזי של מרבית מערכות ההפעלה. זהו הגשר שבין תוכניות המחשב לבין עיבוד הנתונים עצמו שמבוצע ברמת החומרה.
- Embedded programming : קוד שרץ על מכשירים שאינם PC (כלומר לא פועלים על מערכת הפעלה סטנדרטית כמו windows או android)
- Latency : זמן השהיה, הזמן בין יזימת תהליך עד לרגע שהוא מתחיל להתבצע.
- Open source licenses : חוזים חוקיים בין יוצר של רכיבי תוכנה למשתמש שמגדירים כללים לשימוש בקוד של היוצר באופן מסחרי. דוגמאות:
 - ❖ GPL - כל תוכנה שמתבססת או מכילה רכיב בעל רישוי GPL חייבת להיות open source, כלומר חייב לפרסם את כל הקוד שלה לציבור.
 - ❖ MIT - רישוי עם מעט מאוד הגבלות- ניתן להעתיק ולעשות שינויים, כל עוד מוסיפים עותק של רישוי MIT המקורי והתראת זכויות יוצרים.
- Infrastructure : (תשתית) תהליך של ניהול והקצאת מרכזי נתונים ממוחשבים. לרוב משתמשים במושג כשמתייחסים לרכיבי התשתית, כמו שרתים וכתובות IP.
- Kubernetes : סביבת ענן שמוקצים לה מספר שרתים עליהם רצים קונטיינרים ולמעשה מוגדרת כמערכת קוד פתוח.
- Docker compose : כלי להגדרת והרצת אפליקציית docker בעלת כמה קונטיינרים (multi-container Docker applications)
- Dockerignore : קובץ שמכיל כללים ותוכן שלא רוצים לכלול כשיוצרים אימג'. יעיל ביצירת אימג'ים עם גודל קטן יותר, ומונע חשיפה של מידע שנרצה להשאיר חסוי.
- Swagger : דוקומנטציה אינטרקטיבית של API למשתמש, מאפשרת לו להריץ פקודות API ישירות מהדפדפן (בשיעור הודגם באמצעות הוספת docs / בסוף ה-URL)
- Pickle : מודל שמאפשר המרה של אובייקט לרצף של ביטים ולהפך (בין היתר, יכול להמיר JSON לרצף של ביטים, ולהמיר חזרה רצף של ביטים לJSON)
- Enum : class שמייצר סט של שמות בעלי ערכים קבועים ויחודיים.