22.3.22

EASS שיעור 6- תכנון הפרוייקט, חידוד על Dockerfile ובדיקות

- Draw.io - אתר שאיתו ניתן ליצור דיאגרמות בלוקים
- Load balancer – מציג ללקוח נקודת חיבור (כתובת VIP- Virtual IP) אחת למערכת ואז כשהלקוח ניגש לנקודת החיבור הזו ה-load balancer קובע לאיזו אפלקציה מסוימת לשלוח את החיבור (במקום לשלוח לשרת עצמו). באופן כללי, הוא מהווה reverse proxy. מוסיף גם אלמנט של אבטחה.
- DNS- פרוטוקול שמבצע המרה בין שם מילולי של רכיב לכתובת IP שלו.
- Caching – פעולת אחסון קבצים ומידע (שהשתמשו בהם לאחרונה) ברכיב אחסון זמני שניתן לגשת אליו במהירות.
- Replica – תהליך שכולל שיתוף מידע שמבטיח אחידות במערכת. דוגמאות:

1. Data replication, where the same data is stored on multiple storage devices
2. Computation replication, where the same computing task is executed many times. Computational tasks may be:
   - ❖ Replicated in space, where tasks are executed on separate devices
   - ❖ Replicated in time, where tasks are executed repeatedly on a single device

- Blue-green deployment – אפשרות לשנות ולהוסיף עדכונים עם פחות downtime (כלומר, ללא זמן שבו הלקוחות לא יכולים לגשת למערכת) ועם פחות סיכונים.

כיצד נתכנן מערכת?

- ✓ להבין מי הם הלקוחות
- ✓ איך הלקוחות יוכלו להשתמש במערכת?
- ✓ מה מנסים למכור? מה המטרה של המערכת?
- ✓ מה הקלט והפלט של המערכת?
- ✓ נבנה דיאגרמת בלוקים למערכת - חשוב שנתקבע בהתחלה על הארכיטקטורה של המערכת.
- ✓ נכנס לעובי הקורה עבור כל service בהתאם לסוג שלו. לדוגמה:
  האם צריך בסיס נתונים? איזה סוג? האם צריך יותר מבסיס נתונים אחד?
- ✓ מתחילים לבנות רכיב אחר רכיב, בצעדים קטנים, את המערכת. תחילה נבנה את המערכת עבור לקוח אחד בפשטות. בהצשך נבצע הרחבה לפי הצורך. משמעות הדבר היא לא להתקבע על מקרי קצה מסויימים אלא לתכנן את המערכת הכללית- השלד הבסיסי הנדרש בהתאם למטרת המערכת.
- ✓ Scaling – התאמת המערכת לכמות לקוחות גדולה יותר, התאמה לעומסים, הוספת replica , הוספת load balancer , שימוש ב-caching וכו'
- ✓ ביצוע בדיקות (טסטים)

22.3.22

יחידות מידה נפוצות:

Term (Abbreviation)	Approximate Size
Byte (B)	8 bits
Kilobyte (KB)	1024 bytes / 10^3 bytes
Megabyte (MB)	1024 KB / 10^6 bytes
Gigabyte (GB)	1024 MB / 10^9 bytes
Terabyte (TB)	1024 GB / 10^{12} bytes
Petabyte (PB)	1024 TB / 10^{15} bytes
Exabyte (EB)	1024 PB / 10^{18} bytes
Zettabyte (ZB)	1024 EB / 10^{21} bytes

Time	Symbol	Number in 1 second
1 second		1
1 millisecond	ms	1,000
1 microsecond	µs	1,000,000
1 nanosecond	ns	1,000,000,000
1 picosecond	ps	1,000,000,000,000

סדרי גודל של זמן לביצוע פעולות:

```
Latency Comparison Numbers
--------------------------
L1 cache reference                    0.5 ns
Branch mispredict                       5 ns
L2 cache reference                      7 ns   14x L1 cache
Mutex lock/unlock                      25 ns
Main memory reference                 100 ns   20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy           10 us
Send 1 KB bytes over 1 Gbps network    10 us
Read 4 KB randomly from SSD*          150 us  ~1GB/sec SSD
Read 1 MB sequentially from memory    250 us
Round trip within same datacenter     500 us
Read 1 MB sequentially from SSD*        1 ms  ~1GB/sec SSD, 4X memory
HDD seek                               10 ms   20x datacenter roundtrip
Read 1 MB sequentially from 1 Gbps     10 ms   40x memory, 10X SSD
Read 1 MB sequentially from HDD        30 ms   120x memory, 30X SSD
Send packet CA->Israel->CA            200 ms
```

22.3.22

חידודים בנושא Dockerfile:

- פקודות כמו RUN, WORKDIR,COPY הן פקודות שיוצאות לפועל בשלב ה- docker build של האימג'. פעולות שרשומות תחת פקודת CMD הן פעולות שיוצאות לפועל רק אחרי שמבצעים docker run לאימג' (שנוצר ע"י ה-Dockerfile הזה).
- WORKDIR – פקודה שמטרתה לשנות את המיקום שבו הפקודות הבאות ירוצו. במידה והמיקום שניתן לה לא קיים היא יוצרת אותו. לדוגמה:

```
WORKDIR /app
COPY . .
CMD ["uvicorn","main:app", ……]
```

הפקודה WORKDIR מעבירה את המיקום לתיקיית app.
בפקודה הבאה COPY . . המיקום שאליו מעתיקים את הקבצים הוא app/ (כי כעת הנקודה השנייה הפכה להיות app/)

- ל-CMD יש שתי צורות כתיבה:

1. Shell – כתיבה פשוטה של המשפט אותו נרצה להריץ. המשפט ירוץ אוטומטית באמצעות /bin/sh

`CMD executable param1 param2`

When using the shell form, the specified binary is executed with an invocation of the shell using `/bin/sh -c`. You can see this clearly if you run a container and then look at the `docker ps` output:

```
$ docker run -d demo
15bfcddb11b5cde0e230246f45ba6eeb1e6f56edb38a91626ab9c478408cb615

$ docker ps -l
CONTAINER ID IMAGE COMMAND CREATED
15bfcddb4312 demo:latest "/bin/sh -c 'ping localhost'" 2 seconds ago
```

2. Exec – כתיבת הפקודה כשכל מילה היא string במערך שסביבה מרכאות. המשפט ירוץ באמצעות מה שאנחנו בוחרים

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

Note that the content appearing after the CMD instruction in this case is formatted as a JSON array.

When the *exec form* of the CMD instruction is used the command will be executed without a shell.

Let's change our Dockerfile from the example above to see this in action:

```
FROM ubuntu:trusty
CMD ["/bin/ping","localhost"]
```

Rebuild the image and look at the command that is generated for the running container:

```
$ docker build -t demo .
[truncated]

$ docker run -d demo
90cd472887807467d699b55efaf2ee5c4c79eb74ed7849fc4d2dbfea31dce441

$ docker ps -l
CONTAINER ID IMAGE COMMAND CREATED
90cd47288780 demo:latest "/bin/ping localhost" 4 seconds ago
```

22.3.22

- docker logs dockerID – מציג מידע מורחב על container שרץ כרגע בהינתן ID או שם של הcontainer. לדוגמה, מציג הדפסות שרשומות ב-Dockerfile (הן לא מוחזרות למשתמש)

Database מידע כללי:

- מסד נתונים גרפי הוא בסיס נתונים המשתמש במבני גרף לשאילתות סמנטיות עם צמתים, קשתות ומאפיינים כדי לייצג ולאחסן נתונים. הגרף מתייחס לפריטי הנתונים באכסון לאוסף צמתים וקשתות, הקשתות מייצגות את היחסים בין הצמתים.
- Redis מאפשר פתרון בעיות מורכבות באמצעות אופטימיזציה של מבנה הנתונים והפקודות המבוצעות באופן מהיר ובפשטות. אף על פי שכל בסיס הנתונים מאוחסן ב־RAM (מתבסס על cache), הוא עדיין מאפשר גיבויים ויציבות. Redis מבוסס על שליפת הנתונים על בסיס "Key-value" המאפשר שליפת נתונים באופן מהיר ביותר מתוך מאגרי מידע ענקיים.

הרצת redis בהתאם לקוד שנשלח בdiscord: (מה שמסומן זה מה שאנחנו רושמים)

תקשורת עם redis באמצעות קונטיינר redis-CLI שמהווה client-

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker run --rm --name redis-container -d redis
e78aa9b072fc6e8d8738742f173857b613ccf1e43bf1b406a00ea17af57d097f
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker run -it --name redis-cli --link redis-container:redis --rm redis redis-cli -h redis -p 6379
redis:6379> HELLO
 1) "server"
 2) "redis"
 3) "version"
 4) "6.2.6"
 5) "proto"
 6) (integer) 2
 7) "id"
 8) (integer) 3
 9) "mode"
10) "standalone"
11) "role"
12) "master"
13) "modules"
14) (empty array)
redis:6379> ping
PONG
redis:6379> ping [hello]
"[hello]"
redis:6379> ping [hello-EASS-2022]
"[hello-EASS-2022]"
redis:6379>
```

תקשורת עם redis באמצעות python שמהווה client-

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker run --rm --name redis-container -p1234:6379 -d redis
34c8e5df1e297bf6f5dbb411a22d6ad334a4b952260ff3ec4fa7c18daa42c130
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ python3
Python 3.10.2 (main, Jan 13 2022, 19:06:22) [GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis; r = redis.Redis(host='localhost', port=1234, db=0); r.set('foo', 'bar')
True
>>> r.get('foo')
b'bar'
>>>
```

22.3.22

בדיקות Testing:

- סוגי בדיקות:
  1. Integration test – בודק את המערכת מקצה לקצה. מדמה לקוח שמבצע פעולה כלשהי במערכת ובודק שתוצאת התהליך (שיכול לרוץ בכמה services) תקין.
  2. Unit test – בודק service ספציפי באופן לוקלי.
- Profiling- בדיקות זמן לביצוע פעולות במערכת.

הדגמת ביצוע unit tests באמצעות ספריית pytest:

קודם נתקין את pytest . למען הסדר, נעבור לתיקייה אחרת:

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ pip3 install pytest
Collecting pytest
  Downloading pytest-7.1.1-py3-none-any.whl (297 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 297.0/297.0 KB 1.2 MB/s eta 0:00:00
Collecting py>=1.8.2
  Downloading py-1.11.0-py2.py3-none-any.whl (98 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.7/98.7 KB 3.4 MB/s eta 0:00:00
Collecting iniconfig
  Downloading iniconfig-1.1.1-py2.py3-none-any.whl (5.0 kB)
Collecting attrs>=19.2.0
  Downloading attrs-21.4.0-py2.py3-none-any.whl (60 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.6/60.6 KB 1.8 MB/s eta 0:00:00
Collecting tomli>=1.0.0
  Downloading tomli-2.0.1-py3-none-any.whl (12 kB)
Collecting pluggy<2.0,>=0.12
  Downloading pluggy-1.0.0-py2.py3-none-any.whl (13 kB)
Requirement already satisfied: packaging in /home/linuxbrew/.linuxbrew/lib/pythor
rom pytest) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /home/linuxbrew/.linux
ite-packages (from packaging->pytest) (3.0.7)
Installing collected packages: iniconfig, tomli, py, pluggy, attrs, pytest
Successfully installed attrs-21.4.0 iniconfig-1.1.1 pluggy-1.0.0 py-1.11.0 pytest
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ mkdir demo-testing
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cd ./demo-testing
```

ניצור בתיקייה את קובץ הבדיקה- השם שלו חייב להתחיל ב "test_" כדי ש-pytest יידע להריץ אותו (וגם השמות של פונקציות הבדיקה). בקובץ נבנה פונקציה inc , שאותה נבדוק באמצעות שתי פונקציות נוספות:

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing$ vi test_hello.py
```

```
def inc(x):  #The function- simple addition by 1.
    return x+1

#Every function that it's name starts with "test_" pytest knows how to run it.

def test_pos(): #Fail
    assert inc(3) == 5

def test_neg(): #Succeed
    assert inc(-1) == 0
```

22.3.22

כעת נריץ את pytest בתיקייה עם הקובץ:

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing$ ls
test_hello.py
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing$ pytest
========================================= test session starts =========================================
platform linux -- Python 3.10.2, pytest-7.1.1, pluggy-1.0.0
rootdir: /mnt/c/Users/natal/demo-testing
plugins: anyio-3.5.0
collected 2 items

test_hello.py F.                                                                              [100%]

============================================= FAILURES ================================================
_____________________________________________ test_pos _____________________________________________

    def test_pos(): #Fail
>       assert inc(3) == 5
E       assert 4 == 5
E        +  where 4 = inc(3)

test_hello.py:7: AssertionError
===================================== short test summary info =====================================
FAILED test_hello.py::test_pos - assert 4 == 5
==================================== 1 failed, 1 passed in 0.36s ====================================
```

נוכל לראות שטסט אחד נכשל וטסט אחד הצליח כמו שציפינו. בנוסף במקום בו הפונקציה נכשלה רשום מה הוחזר.

<u>הדגמה של profiling:</u>

ניצור קובץ פייתון חדש:

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing$ vi profile_fibo.py
```

```
import cProfile

def fibonnaci(n):
    if n in [0, 1]:
        return n
    else:
        return fibonnaci(n-1) + fibonnaci(n-2)

if __name__ == '__main__':
  pr = cProfile.Profile()
  pr.enable()
  fibonnaci(30)
  pr.disable()
  pr.print_stats()
~
```

כאן אין צורך בהורדה באמצעות pip, שכן הספרייה cProfile כבר קיימת ומימשנו אותה בקובץ. התנאי if למטה מבצע ריצה לפונקציה ובוחן כמה זמן לקח לה לרוץ. כעת נריץ את הקובץ עצמו באמצעות python:

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing$ python3 profile_fibo.py
         2692538 function calls (2 primitive calls) in 0.763 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
2692537/1    0.763    0.000    0.763    0.763 profile_fibo.py:3(fibonnaci)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

22.3.22

דוגמה נוספת לשימוש ב-pytest עם benchmark:

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ pip3 install pytest-benchmark
```

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ pip3 install aspectlib
```

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing$ vi test_and_prof_foo.py
```

```
import time
import pytest

class Foo(object): #Inheriting from object
  def __init__(self, arg=0.01): # "__init__" is a constructor, it also declairs the class variable
    self.arg = arg # "self" is like "this" in c++

  def run(self):
    self.internal(self.arg)

  def internal(self, duration):
    time.sleep(duration)

#Test
def test_foo(benchmark):
  benchmark.weave(Foo.internal, lazy=True)
  f = Foo()
  f.run()
~
```

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing$ pytest test_and_prof_foo.py
========================================= test session starts =========================================
platform linux -- Python 3.10.2, pytest-7.1.1, pluggy-1.0.0
benchmark: 3.4.1 (defaults: timer=time.perf_counter disable_gc=False min_rounds=5 min_time=0.000005 ma
x_time=1.0 calibration_precision=10 warmup=False warmup_iterations=100000)
rootdir: /mnt/c/Users/natal/demo-testing
plugins: aspectlib-1.5.2, benchmark-3.4.1, anyio-3.5.0
collected 1 item

test_and_prof_foo.py .                                                                            [100%]


------------------------------------------------ benchmark: 1 tests -------------------------------------
---------
Name (time in ms)          Min      Max     Mean  StdDev   Median     IQR  Outliers      OPS  Rounds  I
terations
----------------------------------------------------------------------------------------------------------
---------
test_foo               10.2428  10.9964  10.6117  0.1820  10.6680  0.2431      31;0  94.2359      98
        1
----------------------------------------------------------------------------------------------------------
---------

Legend:
  Outliers: 1 Standard Deviation from Mean; 1.5 IQR (InterQuartile Range) from 1st Quartile and 3rd Qu
artile.
  OPS: Operations Per Second, computed as 1 / Mean
========================================== 1 passed in 2.31s ==========================================
```

הערה חשובה- עד כה בדוגמאות הבדיקה הייתה באותו קובץ עם מה שהיא בודקת. מדובר ב- bad practice , כיוון שמקובל להפריד בינהם. בדוגמה הבאה נפריד בין קובץ main לקובץ הבודק אותו ונראה כיצד הם מתקשרים.

22.3.22

דוגמה לשימוש ב-pytest עם fastapi:

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing$ mkdir fastapi-test
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing$ cd fastapi-test
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing/fastapi-test$ vi main.py
```

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def read_main():
  return {"msg": "Hello World"}
~
```

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing/fastapi-test$ vi test_main.py
```

בקובץ הבדיקה נקרא לapp שנמצא בmain

```
from fastapi.testclient import TestClient
from main import app

client = TestClient(app)

def test_read_main():
  response = client.get("/")
  assert response.status_code == 200
  assert response.json() == {"msg": "Hello World"}
```

נריץ pytest

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing/fastapi-test$ pytest -s
================================================= test session starts =================================================
platform linux -- Python 3.10.2, pytest-7.1.1, pluggy-1.0.0
benchmark: 3.4.1 (defaults: timer=time.perf_counter disable_gc=False min_rounds=5 min_time=0.000005 max_time=1.0 calibration_pre
cision=10 warmup=False warmup_iterations=100000)
rootdir: /mnt/c/Users/natal/demo-testing/fastapi-test
plugins: aspectlib-1.5.2, benchmark-3.4.1, anyio-3.5.0
collected 1 item

test_main.py .

================================================== 1 passed in 0.61s ==================================================
```

22.3.22

Linting/Formatting using Black:

Black מבצע סידור נעים לקריאה של קוד. לדוגמה:

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing/fastapi-test$ pip3 install black
```

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing/fastapi-test$ cat test_main.py
from fastapi.testclient import TestClient
from main import app
client=TestClient(app)
def test_read_main():
    response=client.get("/")
    assert response.status_code==200
    assert response.json()=={"msg": "Hello World"}
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing/fastapi-test$ black test_main.py
reformatted test_main.py

All done! ✨ 🍰 ✨
1 file reformatted.
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing/fastapi-test$ cat test_main.py
from fastapi.testclient import TestClient
from main import app

client = TestClient(app)


def test_read_main():
    response = client.get("/")
    assert response.status_code == 200
    assert response.json() == {"msg": "Hello World"}
```

ניתן לראות שנוסף ריווח בין השורות ובין סימני = על מנת שהקוד יהיה יותר קריא.

Async IO:

תזכורת- עבודה אסינכרונית משמעה לבצע פעולה, ובמקום לחכות עד שהיא תסתיים להמשיך לפעולה הבאה (שלא תלויה בקודמת) ולבאה אחריה. לאורך הזמן מבצעים "דגימות" כדי לראות אם פעולות קודמות החזירו תשובה (הסתיימו) וכך ממשיכים. כלומר, אין תלות בסיום של פעולה כדי להמשיך להתקדם.

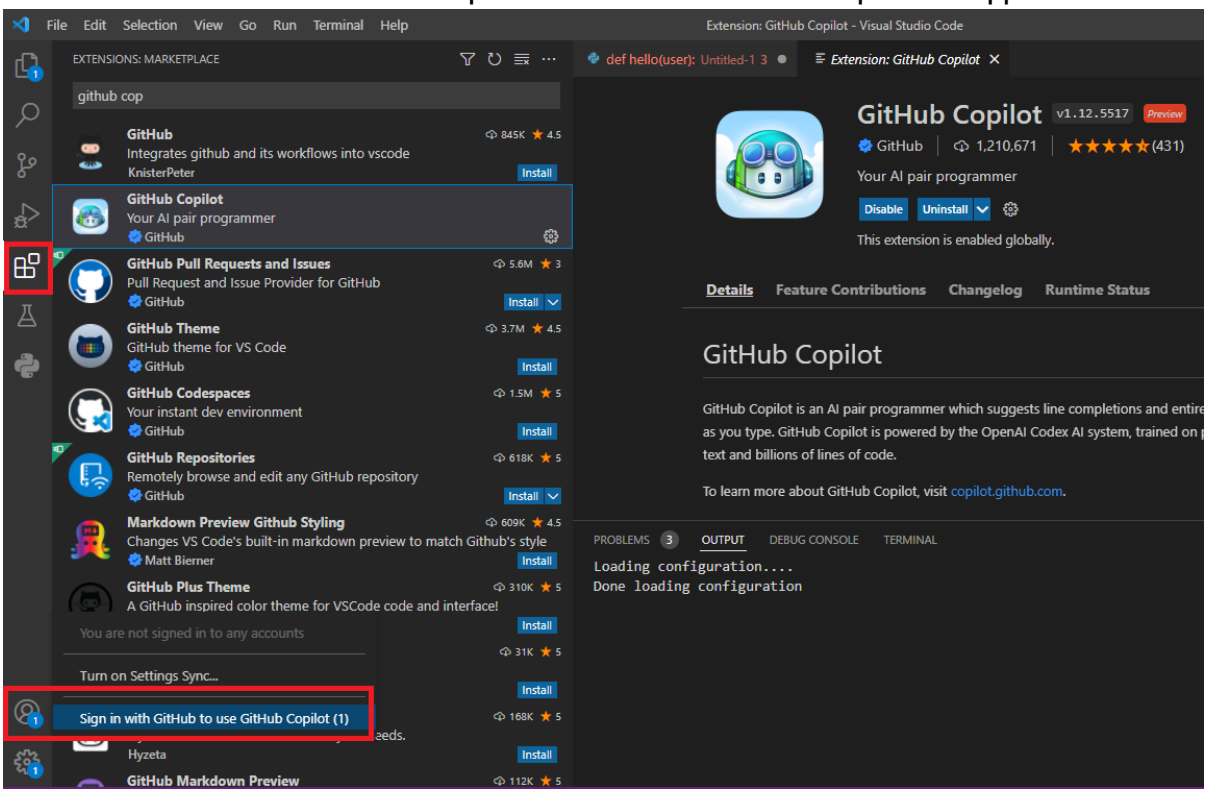
Async IO זה עבודה אסינכרונית מול קלט/פלט. לדוגמה- המתנה לתשובה מmicroservice אחר, משרת אחר ברשת (network), מהדיסק, מהזיכרון וכו'.

22.3.22

Github copilot:

כלי שנמצא בvisual studio code, מאפשר לבצע חיפוש של קטעי קוד בכל הגיטהאב לפי שמות של פונקציות שנרשום.

תחילה יש להתקין את התוסף ולבצע התחברות לגיטהאב דרך VSCode



לאחר הסנכרון יש להירשם לרשימת ההמתנה של התוסף הזה, רק לאחר שהם יאשרו יהיה ניתן להשתמש בו.

מסמך README שמדגים כיצד להשתמש בתוסף:
https://github.com/github/copilot-docs/blob/main/docs/visualstudiocode/gettingstarted.md#enabling