

הנדסת פתרונות תוכנה מתקדמים - מבוא

- נלמד על כלי עבודה של מתכנתים:

Technical debt ✓

Business logic ✓

Bash and Commendline ✓

Git and Github ✓

AWS ✓

Canvas ✓

Discord ✓

LinkedIn ✓

Docker ✓

Stackoverflow ✓

Hackernews ✓

WSL- Linux kernel commendline ✓

הקורס הוא קורס אינטנסיבי שדורש מאמץ ושיתוף פעולה בין הסטודנטים. הציון מורכב מתרגילים והשתתפות. יש אפשרות לביצוע פרוייקט (כולל micro services)

- Technical debt חוב טכני-

עיקרון בפיתוח תוכנה, משקף את העלות של פיתוח הקוד, עלות תועלת של ההחלטות שאנו עושים בפיתוח (בפתרון בעיות בפיתוח)

- Business logic לוגיקה עסקית-

החלק בקוד שמיישם את business rules, שהם מה שהתוכנה צריכה לעשות מבחינת מפתחים שאינם מפתחי תוכנה. בנוסף אחר- חלק ממערכת תוכנה שתפקידו לממש את הכללים העסקיים מ"העולם האמיתי". הכללים קובעים כיצד נתונים יכולים: להיווצר, להשתנות, להיות מוצגים ומאוחסנים.

- :Shell / Bash (Commendline)

דרך אינטרקטיבית לתקשר עם המחשב. מאפשר פונקציות רבות כגון: יצירת תיקיות, חיפוש בקבצים (find), הצגת המשתמש שאני נמצא בו whoami, מציאת מיקום שלי pwd, sudo, echo, לולאות כמו for, ssh, הצגת היסטוריית פקודות history, שינוי מיקום לתיקייה אחרת cd (change directory)

- :Hackernews

מעין אתר חדשות של מתכנתים, מאמרים שניתן להגיב עליהם. פעם בחודש מעדכנים שם רשימת חברות שמחפשות עובדים בחו"ל.

- :AWS

ענן אמזון שבו חברות רבות בעולם שוכרות מיקום לאחסון ממוחשב. מבחינת ההכשרה אנחנו צריכים לבצע את ההכשרות על EBS, S3, ES2, RDS (SQL) שנמצאים במודלים 4,6

(המרצה ירשום אותנו להכשרה בעזרת כתובת המייל שלנו, יש לפתוח בנוסף חשבון canvasa בעזרת קישור לחשבון github)

- :Docker

פלטפורמת קוד פתוח של תוכנה, מאפשר אחזון של קוד בקונטיינרים (containers) תוכנה בינארית, מאפשרת הורדה של סביבות ועבודה בהן (כמו מערכות הפעלה OS שונות) אנו נבצע את הפקודות והעבודה עם docker על פלטפורמת WSL אם הסביבה אותה אנחנו בוחרים לא קיימת על המחשב, הdocker מבצע הורדה של הסביבה מdockerhub (ניתן להיכנס לאתר הזה מהדפדפן ולחפש סביבות וגרסאות שלהן).

הסביבה שאנחנו בוחרים היא נפרדת מהסביבה הלוקלית של המחשב (לא ניתן למצוא בסביבה שהחיצונית קבצים לוקלים ששמורים במחשב לדוגמה)

```
~docker run -ti ubuntu:18.04 /bin/bash
```

פקודת המעבר לסביבה אחרת

```
~docker images
```

מציג את כל האימג'ים המותקנים

```
~exit
```

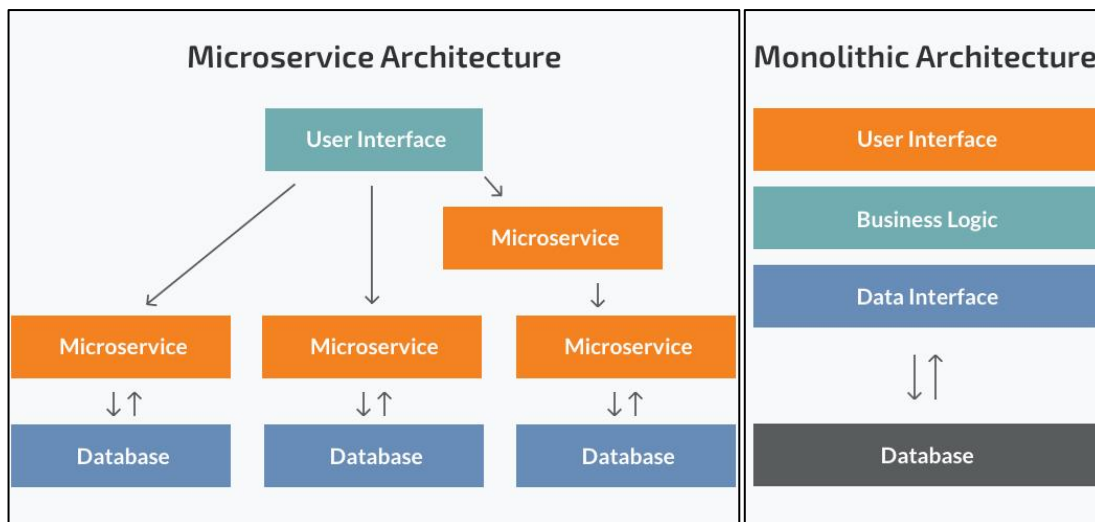
פקודת יציאה (מסביבה, מbash וכו')

ההבדל בין container ל-image: ניתן להריץ כמה containers על אותו image גם במקביל. לכל container יש ID שונה.

הערה- אנחנו נעבוד על docker compose.

הנדסת פתרונות תוכנה מתקדמים- המשך מבוא

- Stack Overflow: 3 כללים ליצירת שאלות
 1. Problem statement - פירוט הבעיה בפשטות.
 2. Sample code and data - דוגמת קוד שהיא stand-alone ולציין מה הקוד צריך לעשות.
 3. Spelling, grammar and formatting - ניסוח באנגלית תקינה וברורה.
- חלוקת הציון:
 1. 25% מהציון- ביצוע מודלים ב-AWS על הנושאים: S3, EC2, EBS, RDS. בונוס 10 נקודות למי שמסיים את כל ההכשרה. **תאריך סיום- 30/04/22**
 2. 25% מהציון- בניית צד שרת backend באמצעות REST/HTTP fastapi + Dockerization. **תאריך סיום- 01/04/22**
 3. UI באמצעות react/streamlit. **תאריך סיום- 01/05/22**
 4. Docker compose לכל מה שנבנה בסעיפים 2+3 וכתובת README עם git submodules. **תאריך סיום- 29/05/22**
 5. פרזנטציה של המערכת- רצוי ליצור סרטון ביוטיוב של 2-3 דקות עם כיתוב באנגלית תקינה, בנוסף חייב README. **תאריך סיום- 29/05/22**
- תוכנות שחשוב להוריד:
 - ✓ WSL - נשלח מדריך בדיסקורד.
 - ✓ Windows Terminal - דרך חנות Microsoft.
 - ✓ Docker Desktop - לחפש בגוגל "docker download".
 - ✓ Visual Studio Code - מהאינטרנט.
- עקרונות חשובים בהנדסת תוכנה:
 - ❖ Measure twice and cut once - תכנון מקדים לפני ביצוע.
 - ❖ Don't Repeat Yourself (DRY) - להמנע מחזרות של קוד.
 - ❖ Keep It Simple - Stupid- (KISS) - פשטות.
 - ❖ You Aren't Gonna Need It (YAGNI) - לא לבצע over engineering, אין צורך לממש הכל בשלבים הראשונים של פיתוח התוכנה.
 - ❖ Avoid Premature Optimization - לא לבצע אופטימיזציה לפני שמקבלים תמונה כללית של התוכנה. (הקשר לDonald Knuth)
 - ❖ Principle Of Least Astonishment - לכתוב קוד מובן ואינטואיטיבי.
 - ❖ Law of Demeter - לבצע היררכיה נכונה בקוד, חלוקה של המחלקות והמודות לרפואיטורים.
 - ❖ עקרונות SOLID.
- Monolithic vs. Microservices
 - Monolithic – base code אחד לכל המערכות, יחידה אחת שמרכזת את כל ה-logic.
 - Microservices - ארכיטקטורה ששוברת כל אפלקציה ליחידות קטנות ועצמאיות.



- **Docker**: חברה שמספקת שירות שבו נלקח כלי (כמו מערכת הפעלה, סביבה של שפת תכנות וכו') בתצורת image ואותו "עוטפים" ב-container שמריץ אותו. האימג'ים יכולים להיות לוקליים (מותקנים על המחשב של המשתמש), ב-dockerhub או בשרת פרטי.
 - ❖ Image - קובץ של הכלי (tool) כמו מערכת הפעלה.
 - ❖ Container - התהליך של הרצת image.
 - ❖ Alias - שמות שונים לאותו הדבר (בקורס זה- לאותה גרסה של ubuntu יש כמה שמות שונים שנתייחס אליהם כ-tags).
- **פקודות של Docker (נבצע ב-WSL)**:
 - Dockerhub/Registry : פנייה לענן שבו שמורים אימג'ים.
 - Dockerfile : מגדיר מה יהיה ב-docker, מחליט איזה אימג' ייבנה (קונפיגורציה).
 - docker build : בונה image מה-dockerfile.
 - docker pull : המשתמש נותן לפקודה כלי (וגרסה) שהוא רוצה לעבוד עם האימג' שלו. במידה והאימג' לא קיים לוקלית מתבצעת הורדה של האימג' מ-dockerhub. דוגמה- `docker pull ubuntu:20.04`
 - אם לא רושמים גרסה אז מתבצעת הורדה של הגרסה האחרונה latest.
 - docker run : יוצר container ל-image, מריץ את האימג'. אם האימג' לא קיים מבצע pull אוטומטית.
 - אם מוסיפים לפקודה -ti זה הופך את הטרמינל לאינטרקטיבי (פותח שורה חדשה שבה ניתן לתת פקודות ל-container).
 - אם מוסיפים בסוף הפקודה `/bin/sh` מתבצעת הרצה ב-shell (command שונה).
 - אם מוסיפים בסוף הפקודה `/bin/top` מוצג איזה services רצים.
 - docker ps : מציג את כל ה-containers שרצים כרגע.
 - docker images : מציג את כל ה-images שקיימים לוקלית, כל גרסה מקבלת שורה נפרדת.
 - docker network ls : מייצר לכל מודול network בפני עצמו, ls מציג את סוגי ה-networks.
 - docker volumes : מאפשר ל-container "לראות את העולם החיצוני" אליו. כלומר, נותן לו לגשת לקבצים מחוץ אליו. צריך בפקודה זו לרשום נתיב (path) למקום שאליו נרצה שהוא ייגש. (מבצע מיפוי של תיקייה חיצונית לתיקייה בתוך ה-container, לדוגמה).
 - Docker kill : מוחק את ה-container שה-ID שלו רשום בסוף הפקודה.
 - Docker exec -ti : יוצר טרמינל אינטרקטיבי ל-container שכבר רץ ברקע. יש לתת לפקודה ID או שם של ה-container.

- פקודות של Image (לאחר ששנכנסנו לסביבה, כלומר אחרי docker run):
 - ls : יוצר רשימה של כל התיקיות (directories) שנמצאים במיקום הנוכחי.
 - ls -l : נותן פירוט לרשימה של התיקיות.
 - ps : מציג את כל התהליכים שרצים (processes)
- פקודות נוספות:
 - exit : חוזר חזרה למקום שהיינו בו לפני. לדוגמה, אם מבצעים יציאה מהאימג' נשלחים חזרה לWSL.
 - clear : ניקוי מסך.
 - wsl -l -v : מריצים פקודה זו ב-power shell (PS) והיא מציגה את התהליכים (כמו ubuntu ,docker desktop) והסטטוס שלהם. (l זה list , v זה version).

1.3.22

הנדסת פתרונות תוכנה מתקדמים- המשך DOCKER

פקודות נוספות שמריצים בתוך wsl: (דוגמאות קודמות נמצאות בקובץ EASS-Lec02-Natalie)

- ~echo print_something

פקודה שמבצעת הדפסה

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ echo 1233
1233
```

- ~cat filename

פקודה בעלת 4 שימושים נפוצים:

1. הצגת תוכן של קובץ
2. שילוב של כמה קבצים
3. הצגת טקסט
4. יצירת קובץ חדש

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cat a.txt
123
```

- ~pwd

מציג את המיקום שאני נמצא בו בתיקיות.

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ pwd
/mnt/c/Users/natal
```

- ~cd /myDir/file.txt

Cd= change directory, לעבור למיקום של תיקייה או קובץ אחר

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ pwd
/mnt/c/Users/natal
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cd ./dance_dir
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/dance_dir$ pwd
/mnt/c/Users/natal/dance_dir
```

- ~<proc1>|<proc2>

לוקח STDOUT של proc1 ומזריק אותו לSTDIN של proc2. נקרא pipe

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ echo 123 | cat > a.txt
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cat a.txt
123
```

- ~command > file-disc/file

לוקח את ה-STDOUT של הפקודה ומכניס לקובץ file.

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ echo Just Dance > dance_file
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cat dance_file
Just Dance
```

- ~command >> file-disc/file

מוסיף את ה-STDOUT של הפקודה בסוף תוכן הקובץ file, כלומר מבצע append.

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ echo Just Dance Again >> dance_file
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cat dance_file
Just Dance
Just Dance Again
```

- ~cp oldFile newFile

מעתיק תוכן קובץ אחד לקובץ אחר (יוצר את הקובץ המועתק אם לא קיים)

```
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cp dance_file copied_file
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cat copied_file
Just Dance
Just Dance Again
```

- ~mv fileName newFileName \ ./myDir

משנה שם של קובץ או משנה את המיקום שלו.

```
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ mv dance_file ./dance_dir
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ ls ./dance_dir
dance_file
```

- ~mkdir directoryName

יצירת תיקייה חדשה, make directory.

```
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/dance_dir$ ls
dance_file
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/dance_dir$ mkdir newDirectory
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/dance_dir$ ls
dance_file  newDirectory
```

- ~docker kill ContainerID

"הורג" את ה-container.

```
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                    NAMES
8bcc5cbb5e7a   ubuntu   "bash"                  13 seconds ago Up 12 seconds                stoic_hopper
945c681b4a13   nginx    "/docker-entrypoint..." 52 minutes ago Up 52 minutes    0.0.0.0:8080->80/tcp      web
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker kill 8bcc5cbb5e7a
8bcc5cbb5e7a
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                    NAMES
945c681b4a13   nginx    "/docker-entrypoint..." 53 minutes ago Up 53 minutes    0.0.0.0:8080->80/tcp      web
```

- ~docker run -ti --rm -d -v/home/username/filename : /from_host --name hi ubuntu

-ti טרמינל אינטרקטיבי, מאפשר לנו לתת פקודות לcontainer.

-d מאפשר להתהליך לרוץ ברקע

--rm מסיר את ה-container אחרי שיוצאים ממנו אוטומטית.

--name נותן שם לcontainer.

-p, port, נקודת גישה למחשב

-v מבצע מיפוי דו כיווני בין תיקייה של host (המחשב הלוקלי) לבין תיקייה של ה-

container. קיצור של Volume.

```
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/dance_dir$ ls
dance_file  newDirectory
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/dance_dir$ docker run -ti -v/mnt/c/Users/natal/dance_dir:/contDir ubuntu
root@b22925a96740:/# ls
bin boot tmpdir dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr
root@b22925a96740:/# cd ./contDir
root@b22925a96740:/contDir# ls
dance_file  newDirectory
```

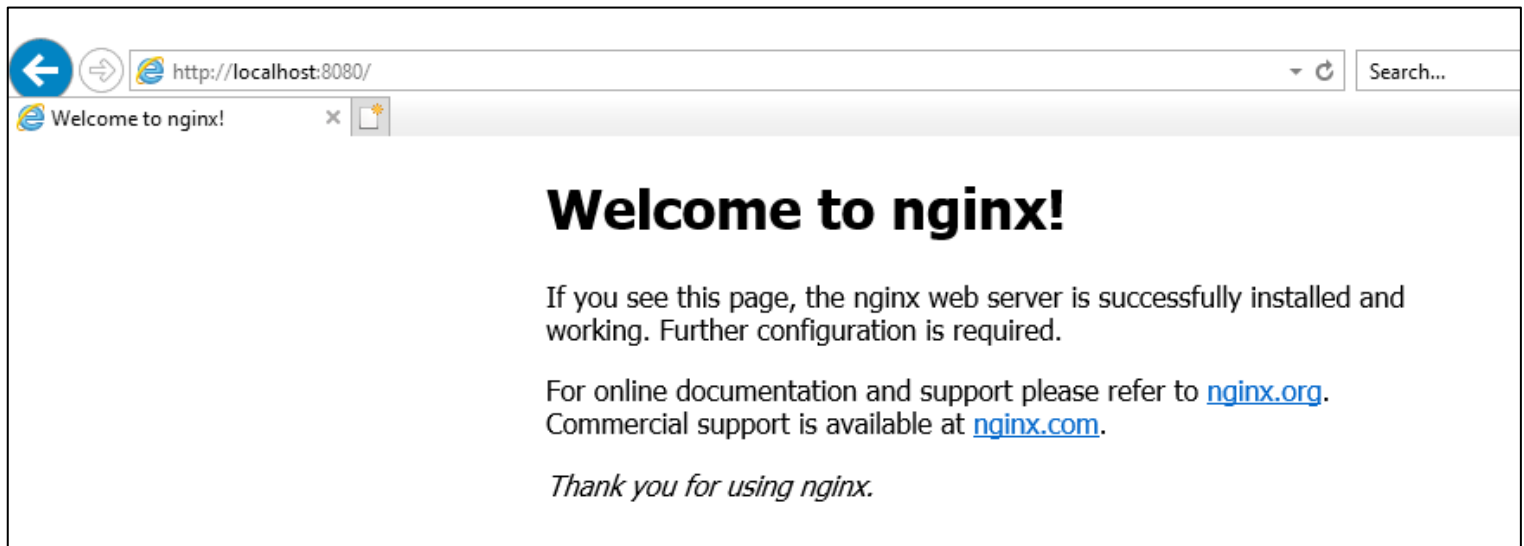
- ~docker logs

מציג פירוט לוגים של container. אמצעי לזיהוי שגיאות (debugging)

nginx

מדובר על סוג image. כאשר docker מריץ אותו ניתן לראות בפורט שרשמנו את דף ה-html שנמצא ב-image. נראה כך:

```
natalieaf1alo@LAPTOP-1GN00VKD:/mnt/c/Users/natal$ docker run -ti --rm -d -p 8080:80 --name web nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
f7a1c6dad281: Pull complete
4d3e1d15534c: Pull complete
9ebb164bd1d8: Pull complete
59baa8b00c3c: Pull complete
a41ae70ab6b4: Pull complete
e3908122b958: Pull complete
Digest: sha256:6a9f18391bb80b75ce50472168ea38e4050f7f56aff8ad08cb4d38b43df1aab2
Status: Downloaded newer image for nginx:latest
945c681b4a1385a25be167092fef84523ca19ae58b9347fd57f9f6f0f4e04a66
natalieaf1alo@LAPTOP-1GN00VKD:/mnt/c/Users/natal$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                NAMES
945c681b4a13   nginx    "/docker-entrypoint...." 3 minutes ago  Up 3 minutes  0.0.0.0:8080->80/tcp   web
```



כדי שנוכל לתת פקודות בתוך ה-container של nginx יש להשתמש בפקודת exec בפורמט bash. נראה כך:

```
natalieaf1alo@LAPTOP-1GN00VKD:/mnt/c/Users/natal$ docker exec -ti web_edit bash
root@56689d70a4cc:/# ls
bin boot dev docker-entrypoint.d docker-entrypoint.sh etc home lib lib64
root@56689d70a4cc:/# ls /user/share/nginx/html
index.html
root@56689d70a4cc:/# cat /user/share/nginx/html/index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx-natalie editing!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx-Natalie editing!</h1>
<p>This file is saved on my computer and I edited it.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@56689d70a4cc:/#
```

פקודת curl מקבלת כתובת url ומציגה לנו מה נמצא בקובץ ה-html. נראה כך:

```
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal$ curl http://localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

קובץ הhtml הזה נקרא index.html והוא חלק מהimage שנקרא nginx.

אם נשתמש במקום בפקודת wget ייווצר גם עותק של index.html לוקלית במחשב.

במקום docker kill ניתן להשתמש בdocker stop ולתת את השם של הcontainer.

```
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                    NAMES
9377a751c9fd   nginx    "/docker-entrypoint...." 26 seconds ago Up 25 seconds  0.0.0.0:8080->80/tcp      web
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal$ docker stop web
web
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                    NAMES
```

אם port מסוים כבר נבחר לשימוש בcontainer מסוים, לא ניתן להשתמש בו בcontainer אחר במקביל. אם ננסה נקבל שגיאה מה-demon.

עריכת קבצים דרך WSL:

בהמשך לפקודת wget, לאחר שיצרנו את הקובץ אצלנו (ניתן להעביר אותו לתיקייה מסוימת) ניתן לערוך אותו. פקודת vi מאפשרת הצגת תוכן הקובץ ושינוי שלו.

תחילה נקליד את הפקודה ונלחץ enter.

```
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal$ vi html_edit/index.html
```

לאחר מכן ייפתח לנו תוכן הקובץ לתצוגה. כדי שנוכל לערוך, יש ללחוץ על מקש i במקלדת (כך שיופיע בתחתית הדף -- INSERT)

נבצע את השינויים שנרצה. בסוף העריכה נלחץ על מקש esc במקלדת. כדי לשמור את השינויים ולצאת מתצוגת הקובץ חזרה ל'wsl' יש לרשום wq: ואז ללחוץ enter. נראה כך:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx-natalie editing!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx-Natalie editing!</h1>
<p>This file is saved on my computer and I edited it.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

:wq

לאחר העריכה:

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ vi html_edit/index.html
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cat html_edit/index.html
<!DOCTYPE html>

<html>

<head>

<title>Welcome to nginx-natalie editing!</title>

<style>

html { color-scheme: light dark; }

body { width: 35em; margin: 0 auto;

font-family: Tahoma, Verdana, Arial, sans-serif; }

</style>

</head>

<body>

<h1>Welcome to nginx-Natalie editing!</h1>

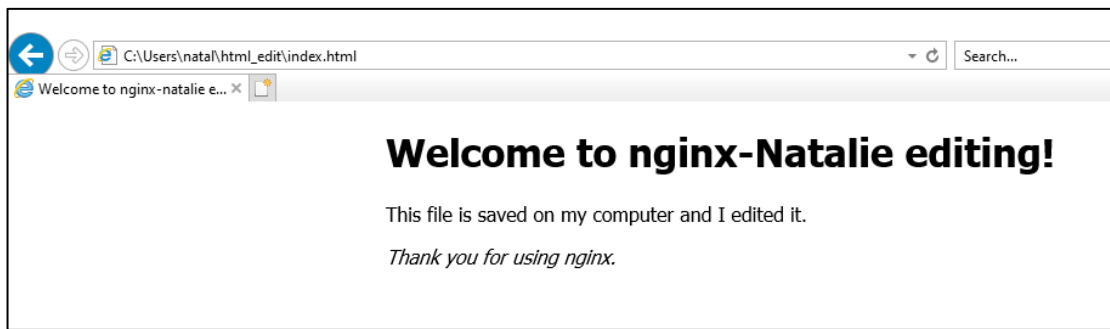
<p>This file is saved on my computer and I edited it.</p>

<p><em>Thank you for using nginx.</em></p>

</body>

</html>
```

נראה כך בדפדפן:



שינוי Image קיים:

יש באפשרותינו לשנות image קיים, במקרה זה אנחנו יכולים להחליף את קובץ index.html המקורי בקובץ שערכנו (ששמור בתיקייה לוקלית במחשב). כדי לבצע זאת נשתמש ב -v . נראה כך:

```
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker run -ti --rm -d -p 8080:80 -v /mnt/c/Users/natal/html_edit:/usr/share/nginx/html --name web_edit nginx
05490250d4122ebf612a82eef4eda5268a25cee2d9d549cee046302ab4fc6199
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ curl localhost:8080/
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx-natalie editing!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx-Natalie editing!</h1>
<p>This file is saved on my computer and I edited it.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

כעת כשנריץ את localhost:8080 בדפדפן נקבל את הדף שערכנו (ולא את המקורי).

יצירת Image חדש:

באמצעות Dockerfile אנו יכולים לכתוב image. הוא יכול להתבסס על image קיים, ניתן להתקין בו תוכנות נוספות, להוסיף לו קבצים ועוד.

נתחיל בעריכת dockerfile באמצעות הפקודה vi Dockerfile. לאחר לחיצה על enter נכנס למסך תצוגה. יש ללחוץ על i במקלדת כדי להיכנס למצב עריכה. כעת נכניס את תוכן האימג':

FROM נספק לdockerfile אימג' קיים שממנו הוא יורש. חייב כאן לרשום את הגרסה של האימג'.

COPY העתקה מה-host (מחשב לוקלי) לתיקייה מסוימת בdockerfile

RUN התקנת תוכנה. לפני ההתקנות חייב לרשום שורה apt-get update

בסוף העריכה נלחץ על esc במקלדת וכדי לשמור את השינויים:wq

ליצירת האימג' מdockerfile נשתמש בפקודת docker build, חשוב לתת לה את נתיב התיקייה באמצעות -t. ולתת לה שם באמצעות -t

לבסוף ניתן ליצור container באמצעות docker run של האימג' שיצרנו.

נראה כך:

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ vi Dockerfile
```

```
FROM nginx:latest
RUN apt-get update
RUN apt-get -y install vim
COPY ./html_edit /usr/share/nginx/html
```

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

~

:wq

```

natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cat Dockerfile
FROM nginx:latest
RUN apt-get update
RUN apt-get -y install vim
COPY ./html_edit /usr/share/nginx/html
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker build . -f Dockerfile -t nginx-eass
[+] Building 19.4s (9/9) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 146B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/nginx:latest
=> [1/4] FROM docker.io/library/nginx:latest
=> [internal] load build context
=> => transferring context: 476B
=> [2/4] RUN apt-get update
=> [3/4] RUN apt-get -y install vim
=> [4/4] COPY ./html_edit /usr/share/nginx/html
=> exporting to image
=> => exporting layers
=> => writing image sha256:bc57e1f7f36fef0619dfb0fc11cacb6e61546df45a44b8a4c4ee9c0706d5323e
=> => naming to docker.io/library/nginx-eass

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nginx-eass    latest    bc57e1f7f36f   21 seconds ago 196MB
nginx         latest    c919045c4c2b   33 hours ago   142MB
ubuntu        latest    54c9d81cbb44   4 weeks ago    72.8MB
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker run -ti --rm -d -p 8080:80 --name eass-web nginx-eass
d27ad94a6fad0c69fe07a416edf5787877511c666f87941fad8d0167a2f0049d
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                    NAMES
d27ad94a6fad   nginx-eass  "/docker-entrypoint. ..." 35 seconds ago Up 33 seconds  0.0.0.0:8080->80/tcp      eass-web

```

הערה חשובה:

בשורה RUN apt-get -y install vim: המשמעות של -y היא שבזמן ההתקנה כל שאלה מקבלת תשובה yes כדי שההתקנה תרוץ עד הסוף בלי תלות במשתמש.

הנדסת פתרונות תוכנה מתקדמים

הבהרות והערות DOCKER ו-WSL:

- Docker: שם של חברה שמספקת שירות שבו נלקח כלי (כמו מערכת הפעלה, סביבה של שפת תכנות וכו') בתצורת image ואותו "עוטפים" ב-container שמריץ אותו. האימג'ים יכולים להיות לוקליים (מותקנים על המחשב של המשתמש), ב-dockerhub או בשרת פרטי.
- Image - קובץ של הכלי (tool) כמו מערכת הפעלה.
- Container - התהליך של הרצת image.
- דוגמאות לפקודות ב-WSL – docker run (הרצת קונטיינר), echo (הדפסה).
- ניתן לתת לפקודות פרמטרים (options) כך:
 - ❖ ניתן להוסיף קיצור לoption בעזרת dash (מקף -), כלומר לאחר dash תופיעה אות אחת שמייצגת פרמטר מסוים. לדוגמה -v הוא קיצור של --volume. ניתן לרשום גם בצורה הקצרה וגם בצורה הארוכה. את הקיצורים ניתן לשרשר תחת אותו dash כך: tidp - כלומר יש כאן 4 options שונות תחת אותו dash. (option שצריכה לקבל ערך, כמו k שמקבלת פורטים, עדיף לא לשים בשרשור כזה. אם שמים היא צריכה להיות אחרונה)
 - ❖ עבור options שאין להם קיצור או שנרצה לפנות אליהם בשם המלא נצטרך לתת double dash (שני מקפים אחד אחרי השני --). לדוגמה --help, --ip. כדי לדעת איזה options קיימות לפקודה מסוימת נרשום אותה ואחריה --help ותופיעה רשימה. במידה ויש קיצור לפקודה הוא יופיע משמאל לפקודה.

```
natalieafialo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker run --help

Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container

Options:
  --add-host list          Add a custom host-to-IP mapping (host:ip)
  -a, --attach list        Attach to STDIN, STDOUT or STDERR
  --blkio-weight uint16    Block IO (relative weight), between 10 and 1000, or 0 to disable (default 0)
  --blkio-weight-device list Block IO weight (relative device weight) (default [])
  --cap-add list           Add Linux capabilities
  --cap-drop list          Drop Linux capabilities
  --cgroup-parent string   Optional parent cgroup for the container
  --cgroupns string        Cgroup namespace to use (host|private)
                           'host': Run the container in the Docker host's cgroup namespace
                           'private': Run the container in its own private cgroup namespace
                           '': Use the cgroup namespace as configured by the
                              default-cgroupns-mode option on the daemon (default)
  --cidfile string         Write the container ID to the file
  --cpu-period int          Limit CPU CFS (Completely Fair Scheduler) period
  --cpu-quota int           Limit CPU CFS (Completely Fair Scheduler) quota
  --cpu-rt-period int       Limit CPU real-time period in microseconds
  --cpu-rt-runtime int      Limit CPU real-time runtime in microseconds
  -c, --cpu-shares int      CPU shares (relative weight)
  --cpus decimal            Number of CPUs
  --cpuset-cpus string      CPUs in which to allow execution (0-3, 0,1)
  --cpuset-mems string      MEMs in which to allow execution (0-3, 0,1)
  -d, --detach              Run container in background and print container ID
  --detach-keys string      Override the key sequence for detaching a container
  --device list             Add a host device to the container
  --device-cgroup-rule list Add a rule to the cgroup allowed devices list
  --device-read-bps list    Limit read rate (bytes per second) from a device (default [])
```

- משפטים:

- ❖ Image אחד יכול לרוץ במקביל בכמה containers.
- ❖ container אחד יכול לרוץ רק image אחד.

```
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
046895a25506	ubuntu	"bash"	13 seconds ago	Up 11 seconds		clever_nightingale
f2e9f03bff15	ubuntu	"bash"	35 seconds ago	Up 35 seconds		admiring_cerf

- פקודות נוספות ב-WSL:

- ❖ ip : מאפשרת להציג ולשנות נקודות גישה לרשת (network interface)

```
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ ip -4 a
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
6: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    inet 172.28.154.134/20 brd 172.28.159.255 scope global eth0
        valid_lft forever preferred_lft forever
```

- ❖ ifconfig : מאפשרת להציג ולשנות נתונים של network interface במערכת (כמו כתובת IP)

- ❖ hostname : מציג את שם המחשב הלוקלי

```
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ hostname
```

```
LAPTOP-1GN0OVKD
```

- ❖ uname : מחזירה מידע בסיסי על מערכת ההפעלה (linux)

```
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ uname --all
```

```
Linux LAPTOP-1GN0OVKD 5.10.16.3-microsoft-standard-WSL2 #1 SMP Fri Apr 2 22:23:49 UTC 2021
x86_64 x86_64 x86_64 GNU/Linux
```

- ❖ ping : מאבחן ובודק תקשורת לרשת

```
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ ping -c 4 www.github.com
```

```
PING github.com (140.82.121.4) 56(84) bytes of data:
64 bytes from lb-140-82-121-4-fra.github.com (140.82.121.4): icmp_seq=1 ttl=46 time=64.8 ms
64 bytes from lb-140-82-121-4-fra.github.com (140.82.121.4): icmp_seq=2 ttl=46 time=65.5 ms
64 bytes from lb-140-82-121-4-fra.github.com (140.82.121.4): icmp_seq=3 ttl=46 time=68.2 ms
64 bytes from lb-140-82-121-4-fra.github.com (140.82.121.4): icmp_seq=4 ttl=46 time=66.7 ms

--- github.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 64.802/66.303/68.177/1.282 ms
```

❖ find : משמש למציאת תיקיות וקבצים

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ find repos
repos
repos/GitHub
repos/GitHub/.git
repos/GitHub/.git/COMMIT_EDITMSG
repos/GitHub/.git/config
repos/GitHub/.git/description
repos/GitHub/.git/FETCH_HEAD
repos/GitHub/.git/HEAD
repos/GitHub/.git/hooks
repos/GitHub/.git/hooks/applypatch-msg.sample
repos/GitHub/.git/hooks/commit-msg.sample
repos/GitHub/.git/hooks/fsmonitor-watchman.sample
repos/GitHub/.git/hooks/post-update.sample
repos/GitHub/.git/hooks/pre-applypatch.sample
repos/GitHub/.git/hooks/pre-commit.sample
repos/GitHub/.git/hooks/pre-merge-commit.sample
repos/GitHub/.git/hooks/pre-push.sample
repos/GitHub/.git/hooks/pre-rebase.sample
repos/GitHub/.git/hooks/pre-receive.sample
repos/GitHub/.git/hooks/prepare-commit-msg.sample
repos/GitHub/.git/hooks/push-to-checkout.sample
repos/GitHub/.git/hooks/update.sample
repos/GitHub/.git/index
repos/GitHub/.git/info
repos/GitHub/.git/info/exclude
repos/GitHub/.git/logs
repos/GitHub/.git/logs/HEAD
```

❖ wc : פקודה שסופרת שורות וכמות תווים.

```
root@046895a25506:/# wc test.txt
 2  2 28 test.txt
root@046895a25506:/# cat test.txt
1211212112121
1211212112121
```

❖ xxd : מציג נתונים מקובץ בינארי בבסיס הקסדצימלי. כדי לצאת מהמצב הזה יש ללחוץ q

❖ Docker rmi : מחיקה של image.

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker rmi -f kennethreitz/httpbin
Untagged: kennethreitz/httpbin:latest
Untagged: kennethreitz/httpbin@sha256:599fe5e5073102dbb0ee3dbb65f049dab44fa9fc251f6835c9990f8fb196a72b
Deleted: sha256:b138b9264903f46a43e1c750e07dc06f5d2a1bd5d51f37fb185bc608f61090dd
```

❖ / -h du : מציגה ניצול הזיכרון בדיסק של תיקייה או קובץ

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ du -h ./repos
41K    ./repos/GitHub/.git/hooks
0      ./repos/GitHub/.git/info
1.0K   ./repos/GitHub/.git/logs/refs/heads
0      ./repos/GitHub/.git/logs/refs/remotes/origin
0      ./repos/GitHub/.git/logs/refs/remotes
1.0K   ./repos/GitHub/.git/logs/refs
2.0K   ./repos/GitHub/.git/logs
628K   ./repos/GitHub/.git/objects/26
0      ./repos/GitHub/.git/objects/2b
504K   ./repos/GitHub/.git/objects/5f
1.0K   ./repos/GitHub/.git/objects/6c
0      ./repos/GitHub/.git/objects/6e
0      ./repos/GitHub/.git/objects/9d
0      ./repos/GitHub/.git/objects/b7
1.0K   ./repos/GitHub/.git/objects/e4
0      ./repos/GitHub/.git/objects/info
1.5M   ./repos/GitHub/.git/objects/pack
2.6M   ./repos/GitHub/.git/objects
0      ./repos/GitHub/.git/refs/heads
0      ./repos/GitHub/.git/refs/remotes/origin
0      ./repos/GitHub/.git/refs/remotes
0      ./repos/GitHub/.git/refs/tags
0      ./repos/GitHub/.git/refs
2.7M   ./repos/GitHub/.git
336K   ./repos/GitHub/Lec01
```

❖ chown : שינוי של משתמש "בעלים" של תיקייה, קובץ או קישור

❖ chmod : מאפשר לשלוט במי יכול לגשת לקובץ או לחפש בתיקייה

❖ sudo : פקודה שמריצה תהליך נוסף במקביל עם משתמש אחר (לרוב משתמש admin) ולכן גם תדרוש סיסמה.

```
PS C:\Users\natal> wsl sudo apt-get update
[sudo] password for natalieaflalo:
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:5 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [970 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal/main Translation-en [506 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal/main amd64 c-n-f Metadata [29.5 kB]
Get:8 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [8628 kB]
Get:9 http://archive.ubuntu.com/ubuntu focal/universe Translation-en [5124 kB]
Get:10 http://archive.ubuntu.com/ubuntu focal/universe amd64 c-n-f Metadata [265 kB]
Get:11 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [144 kB]
Get:12 http://archive.ubuntu.com/ubuntu focal/multiverse Translation-en [104 kB]
Get:13 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 c-n-f Metadata [9136 B]
Reading package lists... Done
E: Release file for http://archive.ubuntu.com/ubuntu/dists/focal-updates/InRelease is not valid yet (invalid for another 1h 1min 46s). Updates for this repository will not be applied.
E: Release file for http://security.ubuntu.com/ubuntu/dists/focal-security/InRelease is not valid yet (invalid for another 27min 11s). Updates for this repository will not be applied.
E: Release file for http://archive.ubuntu.com/ubuntu/dists/focal-backports/InRelease is not valid yet (invalid for another 28min 45s). Updates for this repository will not be applied.
```

❖ which : מוצאת נתיב של פקודה

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ which echo
/usr/bin/echo
```

❖ grep : מוצא את כל המקומות בנתיב שיש בהם את הערך שניתן לו. אם לא קיים הערך הוא לא מחזיר כלום. אם נוסף -v יציג את כל השורות ללא הערך שניתן לו.

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cat ./a.txt | grep 1
123
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cat ./a.txt | grep 13
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$
```

❖ jobs : מציג תהליכים שרצים ברקע

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ jobs
[1]-  Stopped                  vi
[2]+  Killed                   vi
```

❖ fg : נכנס חזרה למסך של job. ניתן לצאת בעזרת ctrl+z

❖ bg : ה-job ממשיך לרוץ ברקע.

❖ kill : פקודת תקשורת (שליחת סיגנלים) בין תהליכים. בין היתר הורג תהליכים.

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ kill -9 %2

[2]+  Stopped                  vi
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ jobs
[1]-  Stopped                  vi
[2]+  Killed                   vi
```

❖ htop : מאפשר בקרה בזמן אמת על תהליכים ומשאבים של המערכת.

❖ top : כמו htop אך פחות נוח לקריאה וללא אפשרות גלילה עם העכבר.

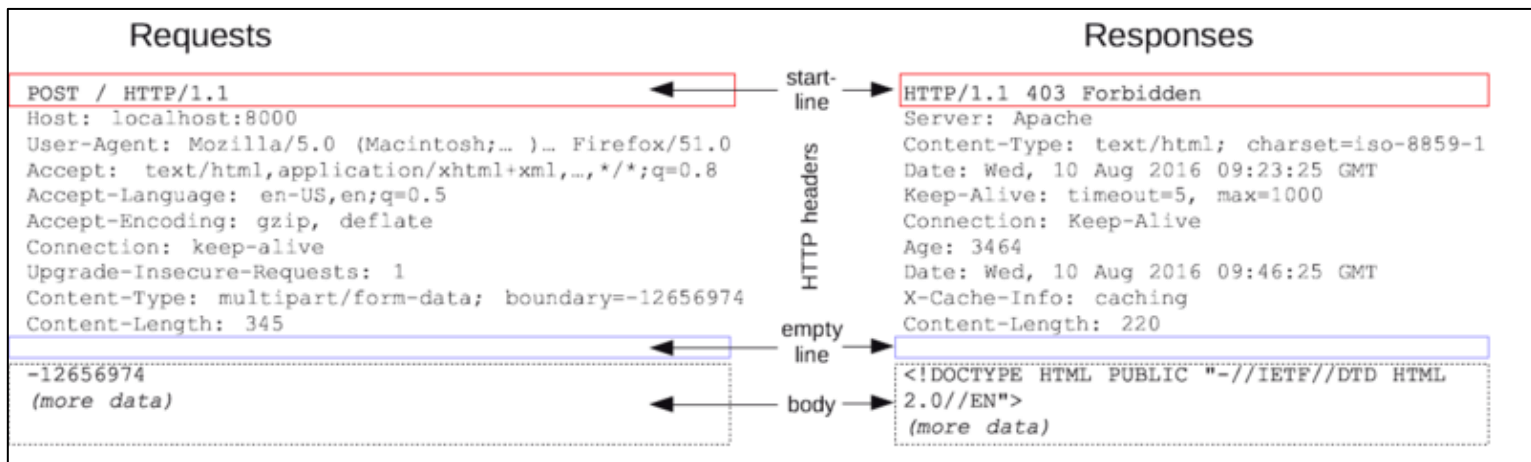
```
1 [          0.0%] 5 [          0.7%]
2 [          0.0%] 6 [          0.0%]
3 [          0.0%] 7 [          0.0%]
4 [          0.0%] 8 [          0.0%]
Mem[|||||] 983M/6.08G Tasks: 10, 27 thr; 1 running
Swp[          ] 0K/2.00G Load average: 0.00 0.00 0.00
Uptime: 06:33:10
```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
140	root	20	0	1640M	30420	13644	S	0.0	0.5	0:03.61	/mnt/wsl/docker-desktop/docker-desktop-proxy --distro-name Ubuntu --docker-desktop-root
153	root	20	0	896	84	20	S	0.0	0.0	0:00.00	/init
155	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:00.56	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
156	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:00.54	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
157	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:00.56	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
158	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:00.59	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
159	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:00.66	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
160	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:00.54	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
161	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:00.56	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
162	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:00.00	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
163	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:00.59	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
164	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:00.00	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
176	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:00.33	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
177	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:00.41	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
537	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:00.59	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
154	natalieaf	20	0	746M	46980	29044	S	0.0	0.7	0:06.09	docker serve --address unix:///home/natalieafalo/.docker/run/docker-cli-api.sock
1483	root	20	0	1012	200	20	S	0.0	0.0	0:00.00	/init
1484	root	20	0	1012	200	20	S	0.0	0.0	0:00.50	/init
1485	natalieaf	20	0	10036	5056	3348	S	0.0	0.1	0:00.35	-bash

F1Help F2Setup F3SearchF4FilterF5Tree F6SortByF7Nice -F8Nice +F9Kill F10Quit

תקשורת בין מיקרוסרוויסים:

- API - ממשק תכנות יישומים. ערכת ספריות קוד, פונקציות, פקודות וכו' של יישום, שניתן לעשות בה שימוש פשוט כדי לנצל את התוכן שלה ליישום שלנו. (לדוגמה, אם נשתמש ב-API של יישום אחר, נוכל להשתמש בפונקציות שלו מבלי לכתוב אותן ביישום שלנו)
 - HTTP API - ממשק תכנות יישומים של שרת רשת או דפדפן, מאפשר שימוש ב-API ללא חשיפת הקוד של היישום. התקשורת מבוססת על בקשה request ותגובה response.
 - REST API - כמו HTTP, בנוסף מציג את state שלו בשקיפות.
<https://www.youtube.com/watch?v=lsMQRaeKNDk>
 - קובץ json- קובץ בו הנתונים מסודרים בתצורת dictionary, כלומר מידע המאוחסן בתצורה של מפתח-ערך.
- תקשורת של מיקרוסרוויסים ביניהם היא over HTTP, כלומר באמצעות API, מעבירים ביניהם json-ים.



:HTTP REQUEST

The operation we want	The HTTP method
Create	POST
Read	GET
Update	PUT
Delete	DELETE

Endpoint: מי שאנחנו פונים אליו בrequest. לדוגמה, <https://www.google.com>. נציין כי אותו endpoint יכול לקבל בקשות GET וגם בקשות POST.

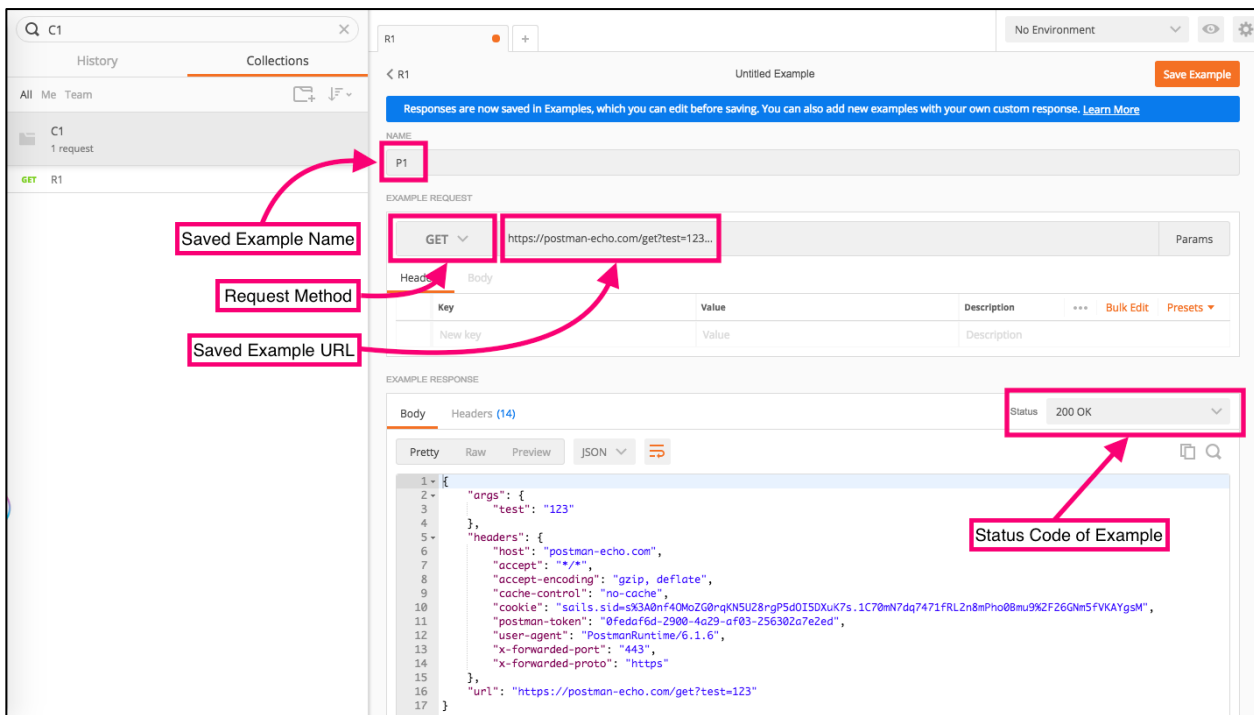
- GET: מבקש להציג מידע מהמקור שאליה אנחנו פונים.
GET /microservice/v1/function?param1=value1¶m2=value2
סימן השאלה הוא הsplitter בין המקור אליו אנו פונים לבין הפרמטרים שלפיהם נרצה לקבל את המידע. כתובת המקור (משמאל לסימן השאלה) היא חד-חד-ערכית.
- POST: מעבירה מידע, קובץ, מחרוזת וכו' שרוצים לעדכן במקור אליו פונים. בבקשה זו ניתן לתת כמות גדולה יותר של פרמטרים ואת כל המידע הזה מעבירים בbody של הבקשה. (דוגמה בתמונה למעלה)

פלטפורמות ליצירת בקשות HTTP: postman, פקודות curl ו-wget ב-command line (מוסבר בסיכום של הרצאה 3).

דוגמה לבקשה בעזרת curl: (ניתן להחליף את GET ב-POST)

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ curl -X GET https://www.google.com
```

דוגמה לבקשה בpostman:



:HTTP RESPONSE

HTTP STATUS CODES

2xx Success

200 Success / OK

3xx Redirection

301 Permanent Redirect

302 Temporary Redirect

304 Not Modified

4xx Client Error

401 Unauthorized Error

403 Forbidden

404 Not Found

405 Method Not Allowed

5xx Server Error

501 Not Implemented

502 Bad Gateway

503 Service Unavailable

504 Gateway Timeout

בקשות HTTP בפייתון-

ספריית requests:

נבצע import requests (או pip install requests אם לא מותקנת)

```
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal$ ipython3
Python 3.8.2 (default, Mar 13 2020, 10:14:16)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.13.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import requests

In [2]: response_get_from_site=requests.get("https://github.com/")

In [3]: type(response_get_from_site)
Out[3]: requests.models.Response

In [4]: response_get_from_site.status_code
Out[4]: 200

In [5]: response_get_from_site.url
Out[5]: 'https://github.com/'

In [6]: response_get_from_site.json
Out[6]: <bound method Response.json of <Response [200]>>
```

ספריית httpx:

נבצע import httpx (או pip install httpx אם לא מותקנת)

```
In [1]: import httpx

In [2]: host = "https://httpbin.org"

In [3]: endpoint = "get"

In [4]: import os

In [6]: address = os.path.join(host, endpoint)

In [7]: address
Out[7]: 'https://httpbin.org/get'
```

כעת נבצע את פקודת ה-GET ונצפה בתוצאה בתצורת json:

```
In [9]: response = httpx.get(address)

In [10]: response
Out[10]: <Response [200 OK]>

In [13]: response.json()
Out[13]:
{'args': {},
 'headers': {'Accept': '*/*',
 'Accept-Encoding': 'gzip, deflate',
 'Host': 'httpbin.org',
 'User-Agent': 'python-httpx/0.22.0',
 'X-Amzn-Trace-Id': 'Root=1-622d2fff-5bff5d6132a4b81645b08396'},
 'origin': '77.137.75.12',
 'url': 'https://httpbin.org/get'}
```

פונקציות נוספות שניתן להריץ על ה-response:

```
In [14]: response.headers
Out[14]: Headers({'date': 'Sat, 12 Mar 2022 23:42:55 GMT', 'content-type': 'application/json', 'content-length': '303', 'connection': 'keep-alive', 'server': 'gunicorn/19.9.0', 'access-control-allow-origin': '*', 'access-control-allow-credentials': 'true'})

In [15]: response.cookies
Out[15]: <Cookies[]>

In [16]: dir(response)
Out[16]:
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattr__',
 '__getstate__',
 '__gt__',
 '__hash__',
```

Cookies - קבצים שנשמרים בדפדפן של הלקוח ומאפשרים התחברות מהירה, שמירת העדפות וכו'.

הנדסת פתרונות תוכנה מתקדמים- הכנה לתרגיל 1

תרגיל ההכנה:

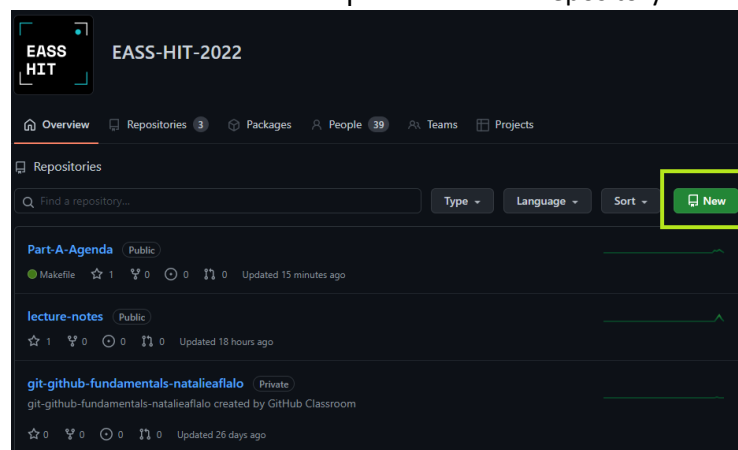
- 1 Create a remote git repo on our organization GitHub <https://github.com/EASS-HIT-2022/> (private/public)
- 2 Name the repo `http-api-demo-<your github name>`
- 3 Include a README, Dockerfile, client.py files
- 4 In client.py include at least two POST/GET requests from httpbin demo HTTP API (<http://httpbin.org/>):
 - POST to any endpoint of your choice (e.g., <http://httpbin.org/post>)
 - GET to any endpoint of your choice (e.g., <http://httpbin.org/get>)
- 5 Make a Dockerfile that execute client.py on startup and prints the status and output from the http requests it performs from step 3. Helpful snippet:

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install python
CMD ["echo", "Hello, EASS 2022"]
```

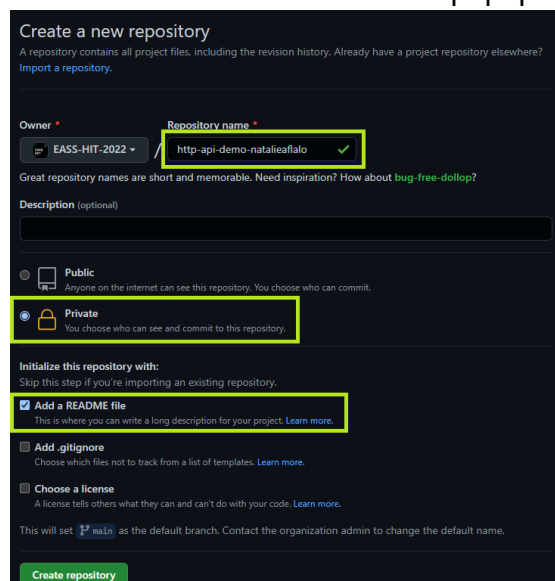
- 6 Now, create a second Dockerfile in your git repo localhost.Dockerfile which call to the local hosted httpbin and call it via `http://localhost: docker run -p 80:80 kennethreitz/httpbin`
- 7 build the second docker image. Useful command:

```
docker build -t tab ./ -f localhost.Dockerfile
```

1. פתיחת repository בגיטהאב של הקורס



2. נגדיר את מאפייני הסר: שם (בין מילים עדיף לשים מקף), מצב תצוגה (פרטי/ציבורי), נוסף קובץ README



כעת נבצע clone של ה-repo לWSL:

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ gh repo clone EASS-HIT-2022/http-api-demo-
Welcome to GitHub CLI!

To authenticate, please run 'gh auth login'.
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: 08F7-F871
Press Enter to open github.com in your browser...
✓ Authentication complete.
- gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as natalieafalo
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ gh repo clone EASS-HIT-2022/http-api-demo-
Cloning into 'http-api-demo-natalieafalo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 615 bytes | 10.00 KiB/s, done.
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$
```

(כדי להתקין את gh צריך תחילה להתקין את brew בWSL לפי המדריך

<https://www.how2shout.com/linux/install-brew-on-wsl-windows-subsystem-for-linux>)

את קובץ README ניתן לערוך בעזרת vi כך-

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cd http-api-demo-
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo- $ git status
On branch main
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo- $
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo- $ ls
README.md
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo- $ vi README.md
```

לאחר שנלחץ על ENTER נכנס למסך תצוגה. כדי לעבור למצב עריכה יש ללחוץ על i במקלדת (תופיע למטה המילה INSERT). לאחר סיום העריכה נלחץ על esc במקלדת, וכדי לשמור ולצאת חזרה לcommandline נרשום wq (כלומר, נקודותיים wq) ואז ENTER.

```
# http-api-demo-

Hello,

In this repo there will be a Dockerfile and client.py file

This is training for EX1

Good luck!

~
:wq
```

לפני שלב 3, ניצור את הקבצים Dockerfile ו-client.py גם כן נשתמש ב-vi כדי לערוך את Dockerfile ואת client.py (אם הקבצים לא קיימים הפקודה יוצרת קובץ ריק):

.4

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

```
import httpx

get_response = httpx.get("https://httpbin.org/get")

post_response = httpx.post("https://httpbin.org/post")

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

print("POST response: ",post_response.json())

~

:wq
```

.5

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo- $ vi Dockerfile
```

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

~

~

~

~

~

~

:wq
```

כעת נחזור לשלב 3 אחרי שיצרנו את הקבצים (ניתן לבצע docker build ודא docker run כדי ליצור אימג' ולראות שהוא רץ באופן תקין)

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo- $ docker build . -t http-api-demo
[+] Building 3.4s (8/8) FINISHED
=> [internal] load build definition from Dockerfile                                0.1s
=> => transferring dockerfile: 131B                                              0.0s
=> [internal] load .dockerignore                                                 0.0s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/python:3.8                   3.0s
=> [1/3] FROM docker.io/library/python:3.8@sha256:53cb5152064a7e7b485ad42704ea63c5155b264c 0.0s
=> [internal] load build context                                                0.0s
=> => transferring context: 253B                                              0.0s
=> CACHED [2/3] RUN pip install httpx                                          0.0s
=> [3/3] COPY ./client.py .                                                    0.1s
=> exporting to image                                                         0.1s
=> => exporting layers                                                         0.1s
=> => writing image sha256:0e50e42b332965bad54fb47b0e3fb5c4813feb889b4c4eb79689c31484b6a0a 0.0s
=> => naming to docker.io/library/http-api-demo                               0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/http-api-demo- $ docker run http-api-demo
GET response: {'args': {}, 'headers': {'Accept': '*/.*', 'Accept-Encoding': 'gzip, deflate', 'Host': 'httpbin.org', 'User-Agent': 'python-httpx/0.22.0', 'X-Amzn-Trace-Id': 'Root=1-62309f50-05f3811968964b123fb19420'}, 'origin': '77.137.75.12', 'url': 'https://httpbin.org/get'}
POST response: {'args': {}, 'data': '', 'files': {}, 'form': {}, 'headers': {'Accept': '*/.*', 'Accept-Encoding': 'gzip, deflate', 'Content-Length': '0', 'Host': 'httpbin.org', 'User-Agent': 'python-httpx/0.22.0', 'X-Amzn-Trace-Id': 'Root=1-62309f51-44f3a52554a0057f782aaff1'}, 'json': None, 'origin': '77.137.75.12', 'url': 'https://httpbin.org/post'}
```

3. נבחן את השינויים שבוצעו ולא מעודכנים בrepo בgithub

```
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/http-api-demo- [redacted] $ git status
On branch main
Your branch is up to date with 'origin/main'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

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

no changes added to commit (use "git add" and/or "git commit -a")
```

נוסיף את כל הקבצים שעברו שינוי או הוספה:

```
$ git add README.md
$ git add Dockerfile
$ git add client.py
```

נעדכן במערכת מי אנחנו כדי שהמערכת תדע מי מבצע את השינוי ונבצע commit לאחר מכן, ואז נדחוף את השינויים לrepo בgithub:

```
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/http-api-demo- [redacted] $ git config --global user.email "[redacted].com"
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/http-api-demo- [redacted] $ git config --global user.name "[redacted]"
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/http-api-demo- [redacted] $ git commit -m "Added Dockerfile and client.py"
[main 0b23373] Added Dockerfile and client.py
 3 files changed, 30 insertions(+), 1 deletion(-)
 create mode 100644 Dockerfile
 create mode 100644 client.py
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/http-api-demo- [redacted] $ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 636 bytes | 26.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To https://github.com/EASS-HIT-2022/http-api-demo-[redacted].git
 aab8f75..0b23373  main -> main
```

לאחר סיום פעולה זו נוכל לראות את הקבצים והשינויים בrepo בgithub דרך הדפדפן.

Dockerfile	Added Dockerfile and client.py	9 minutes ago
README.md	Added Dockerfile and client.py	9 minutes ago
client.py	Added Dockerfile and client.py	9 minutes ago

README.md

http-api-demo-[redacted]

Hello,

In this repo there will be a Dockerfile and client.py file

This is training for EX1

Good luck!

(הסבר לסעיפים 6+7 נמצא בסיכום הרצאה 5)

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-frontend. 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.

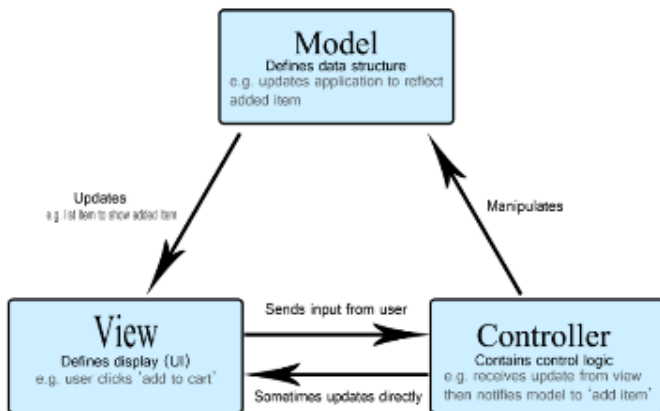
- Uvicorn : מאפשרת הרצת צד שרת server כשעובדים עם fastapi a minimal low-level server/application interface for async frameworks הערות- בפקודה יש להוסיף --reload כדי שהמערכת תעדכן את עצמה אוטומטית (לדוגמה אם נשנה את הקובץ main). בנוסף, הפורט חייב להיות מספר גדול יותר מ-1024 כי אסור לתת מספר קטן יותר ממנו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

```
natalieaf1alo@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) (0.14.2)
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) (3.3.0)
Requirement already satisfied: typing-extensions<3.7.4,>=3.7.4 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from anyio->starlette->fastapi) (3.7.4.3)
Requirement already satisfied: idna<2.8,>=2.3 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from anyio->starlette->fastapi) (2.8)
Requirement already satisfied: sniffio<1.1,>=1.1 in /home/linuxbrew/.linuxbrew/lib/python3.7/site-packages (from anyio->starlette->fastapi) (1.1)
natalieaf1alo@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]) (0.12.0)
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-

```
natalieaf1alo@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-

```
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ mkdir demo-fastapi
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ mv main.py demo-fastapi
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ cd ./demo-fastapi
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-fastapi$ ls
main.py
```

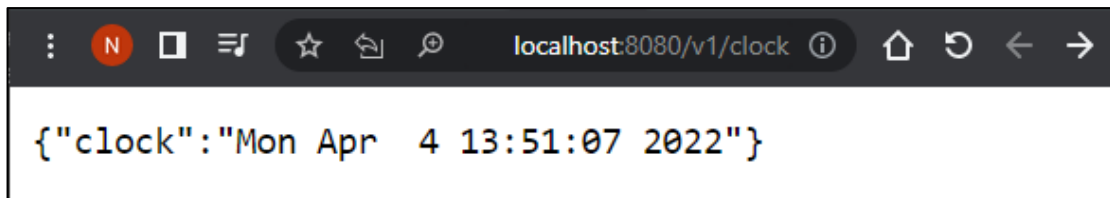
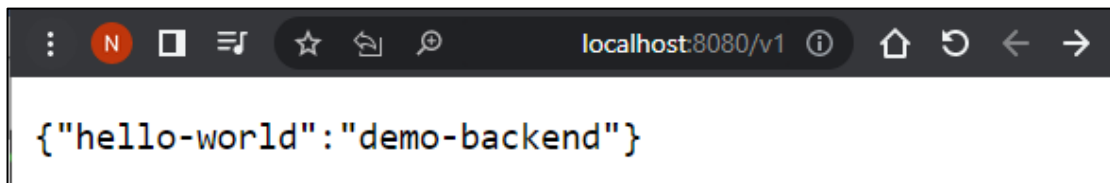
4. נריץ את uvicorn במצב reload עבור קובץ main.py-

```
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"]

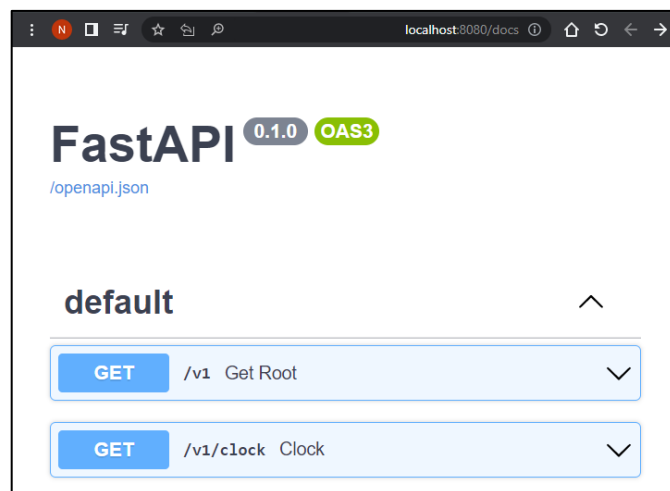
5. נבדוק האם הפקודות פועלות בעזרת הדפדפן. נריץ את localhost שהוא 0.0.0.0 דרך פורט 8080 ונוסיף את הנתוב path שבנו נמצאת הפקודה לפי הקובץ main.py-

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 swgger- 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-

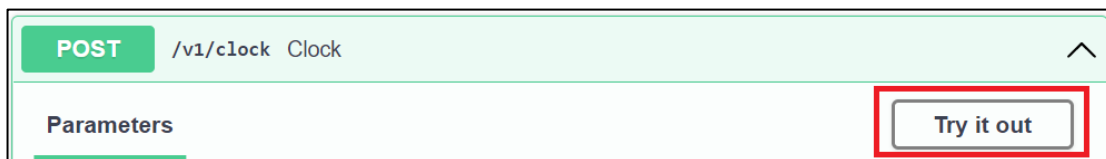
```
from fastapi import FastAPI
import time
from pydantic import BaseModel

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

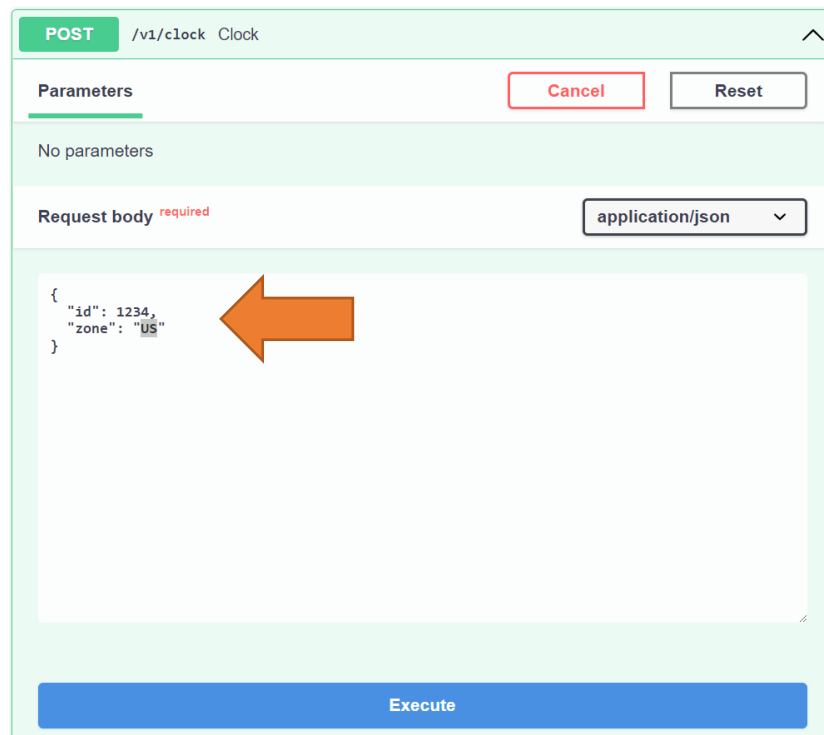
app=FastAPI()

@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	Response body <pre>{ "clock": "Mon Apr 4 14:46:45 2022", "req.zone": "US" }</pre> Response headers <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.

EASS שיעור 6- תכנון הפרוייקט, חידוד על Dockerfile ובדיקות

- Draw.io - אתר שאיתו ניתן ליצור דיאגרמות בלוקים
- Load balancer – מציג ללקוח נקודת חיבור (כתובת Virtual IP -VIP) אחת למערכת ואז כשהלקוח ניגש לנקודת החיבור הזו ה-load balancer קובע לאיזו אפליקציה 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. בהצטרף 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)

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	μ s	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 Zip	10 μ s	
Send 1 KB bytes over 1 Gbps network	10 μ s	
Read 4 KB randomly from SSD*	150 μ s	~1GB/sec SSD
Read 1 MB sequentially from memory	250 μ s	
Round trip within same datacenter	500 μ s	
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	

חידודים בנושא Dockerfile:

- פקודות כמו RUN, COPY, WORKDIR 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 מראות. 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
```

- `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'
>>>
```

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:

```
natalieafalo@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/python3.0.7/site-packages (from pytest) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /home/linuxbrew/.linuxbrew/lib/python3.0.7/site-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-7.1.1
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ mkdir demo-testing
natalieafalo@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:

```
natalieafalo@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
```

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.

הדגמה של profiling:

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}
```

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 declares 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 ביניהם. בדוגמה הבאה נפריד בין קובץ main לקובץ הבודק אותו ונראה כיצד הם מתקשרים.

דוגמה לשימוש ב-pytest עם fastapi:

```
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing$ mkdir fastapi-test
natalieafalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal/demo-testing$ cd fastapi-test
natalieafalo@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"}
```

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

בקובץ הבדיקה נקרא לקראת app 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

```
natalieafalo@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 =====
```

: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

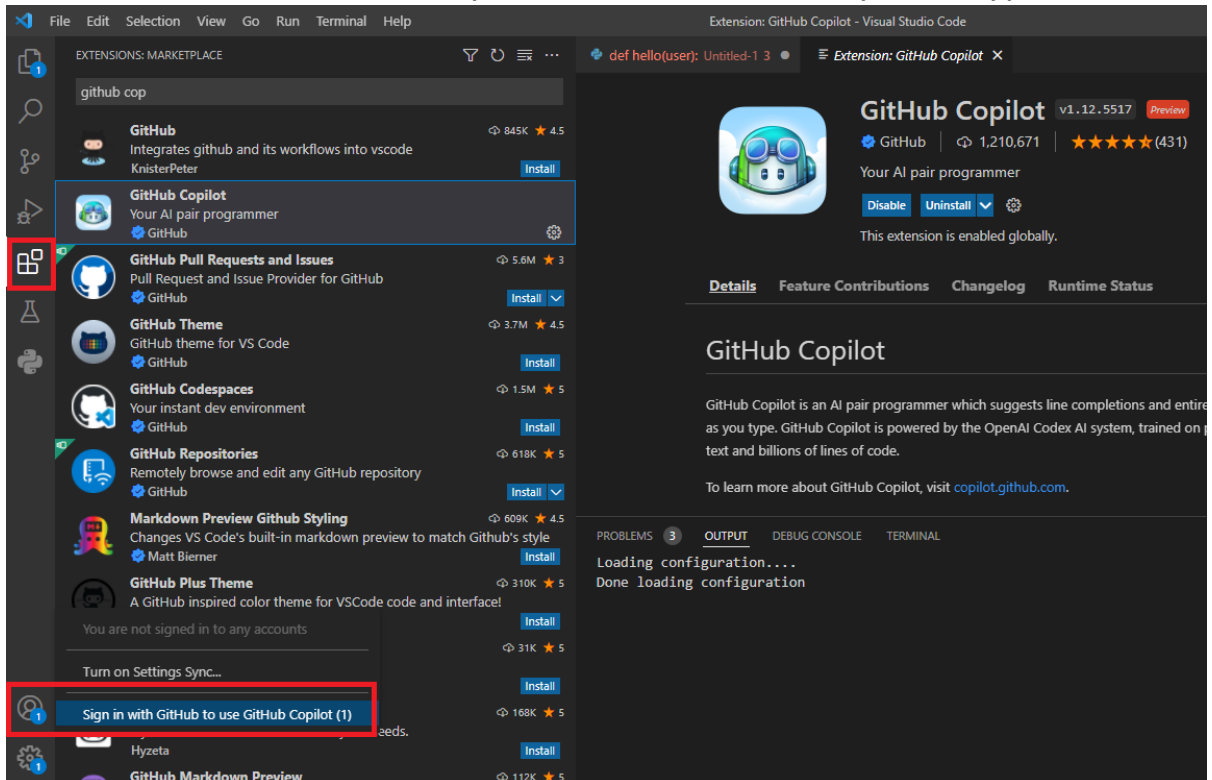
תזכורת- עבודה אסינכרונית משמעה לבצע פעולה, ובמקום לחכות עד שהיא תסתיים להמשיך לפעולה הבאה (שלא תלויה בקודמת) ולבאה אחריה. לאורך הזמן מבצעים "דגימות" כדי לראות אם פעולות קודמות החזירו תשובה (הסתיימו) וכך ממשיכים. כלומר, אין תלות בסיום של פעולה כדי להמשיך להתקדם.

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

:Github copilot

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

תחילה יש להתקין את התוסף ולבצע התחברות לגיטהב דרך VSCode



לאחר הסנקרון 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>

EASS שיעור 7- חזרה ודגשים, המשך Redis, המשך Async IO

חזרה והערות כלליות לקראת הגשת המטלה הראשונה:

- ההגשה נדחתה לתאריך 26.04.2022
- המטלה תוגש באמצעות github וצריכה להכיל את כל הרכיבים הבסיסיים הנדרשים וביניהם:
 - ✓ README מפורט ומסודר
 - ✓ סידור של כל הקבצים בתיקיות שונות בהתאם לתפקידם
 - ✓ קובץ דרישות requirements.txt - מה צריך המשתמש להתקין כדי שיוכל להריץ את המערכת
 - ✓ קובץ בדיקות unit וסקריפט בדיקת אינטגרציה/ הסבר כיצד לבצע בדיקת אינטגרציה למערכת ב-README
 - ✓ שימוש בfastapi שמגדיר את התשובות לבקשות HTTP שונות ו-pydantic להמרה של JSON class פייתוני ולהפך.
- בדיקות אינטגרציה-המלצת המרצה לבנות shell_script.sh שמבצע את ההרצה של כל ה-dockerfile ושולח בקשות HTTP כדי לבחון שאכן מוחזרת התשובה הנכונה. הסקריפט בוחן שהמערכת באופן כללי עובדת כמו שצריך (לדוגמה אם אחד מה-microservices הוא בסיס נתונים אז הבקשה צריכה לגרום ל-backend לגשת לבסיס הנתונים, כדי לבחון ששניהם עובדים כראוי ביחד)
- OCR- מבצע המרה של תמונה לטקסט (deep learning)
- כיצד להשתמש בקובץ requirements.txt :
- בקובץ טקסט requirements.txt נרשום את השמות של הספריות אותן נרצה שהמשתמש יתקין עם pip install. ניתן לרשום עם גרסאות או בלי. לדוגמה-

```

requirements.txt
1 ##### Requirements without Version Specifiers #####
2 fastapi
3 uvicorn[standard]
4
5 ##### Requirements with Version Specifiers #####
6 numpy==2.2.2
7 pandas>= 1.4.0

```

הפקודה בdockerfile שמריצה התקנה לכל מה שיש בקובץ requirements.txt-

RUN pip install -r requirements.txt

RUN pip install -r requirements.txt

הערה- לכל microservice יהיה קובץ requirements.txt משלו.

הדגמה למבנה מערכת בעזרת דיאגרמת בלוקים:

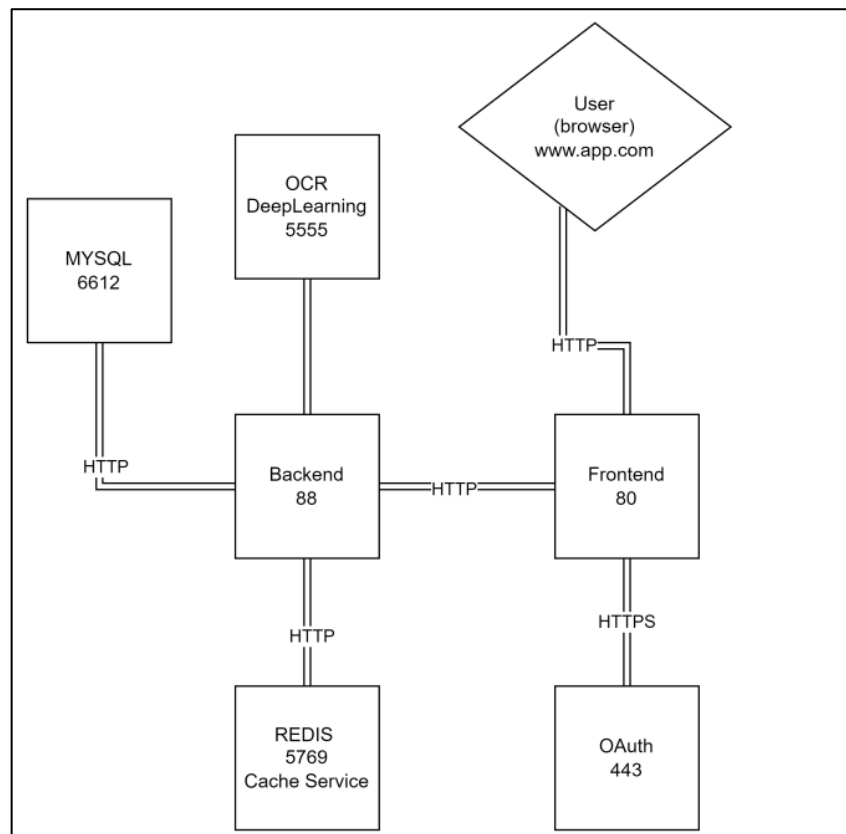
כל ריבוע מהווה microservice שעבורו נייצור Dockerfile נפרד שעובד עם port שונה. נציין שרק ה-port של frontend הוא חשוף ללוקוח, כל השאר הם פנימיים למערכת.

ל- Dockerfile לא בהכרח יש קובץ קוד.

אם ב- Dockerfile יש קטע קוד fastapi , צריך שיהיה לו גם תיקייה משלו עם main, app, requirements.txt לפחות.

ה-microservices מתקשרים אחד עם השני באמצעות פרוטוקול HTTP, כלומר שולחים JSON בבקשות ובתשובות.

בפרוייקט של קורס זה עלינו לממש לפחות 3 microservices, כש- backend,frontend הם הכרחיים.



Redis המשך:

1. כדי שהמידע מהcontainer של redis יישמר בדיסק שלנו (host) ולא יימחק, נשתמש בvolume (-v) ונקשר תיקייה לוקלית לתיקיית data בקונטיינר (בתייעוד documentation של אימג' redis אמרו לנו מהי התיקייה שאליה אנחנו צריכים לפנות):

```
docker run --rm --name redis-container -v data_local:/data -p1234:6379 -d redis
```

2. בשלב הבא ניצור את התקשורת עבור redis:

```
docker network create --subnet 172.20.0.0/16 --ip-range 172.20.240.0/20 redis-demo-network
```

3. נחבר בין הקונטיינר של redis לתקשורת שיצרנו:

```
docker network connect --ip=172.20.0.1 redis-demo-network redis-container
```

4. נריץ קונטיינר פייתון שמחובר לתקשורת של redis, וכל מידע שנגדיר יישמר גם אם הקונטיינר יסתיים:

```
docker run -it --network redis-demo-network --rm python:3.9 bash
```

לדוגמה, אם בטרמינל פייתון שייפתח נרשום:

```
>>> import redis
>>> r = redis.Redis(host='172.20.0.1', port=6379, db=0)
>>> r.set('foo', 'bar')
```

(כלומר, אם נבצע r.get('foo') נקבל b'bar')

אז גם אם הקונטיינר יסתיים ואז נפתח אותו שוב ונבצע רק r.get('foo') נקבל תשובה b'bar' מבלי שעשינו שוב set, כי המידע שהוגדר בset לפניין כבר שמור.

Async IO המשך:

- Event loop – מבנה שדוגם את האירועים שממתינים לתשובה מהם. בfastapi יש event loop מובנה אוטומטית.

אנו נתייחס למושגים GIL, process, thread לפי המשמעות שלהם בסביבת פייתון.

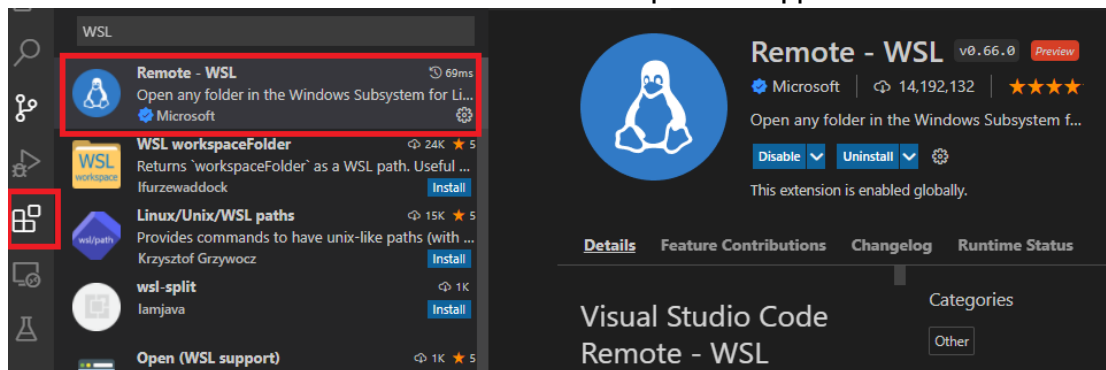
מאמר שמבצע השוואה והסבר על דרכי הפעולה של process, multithreading, asyncio: <https://medium.com/analytics-vidhya/asyncio-threading-and-multiprocessing-in-python-4f5ff6ca75e8>

- GIL - Global Interpreter Lock, מערכת שמאפשרת רק ל-thread אחד לרוץ בזמן מסוים (כלומר, לא מאפשר פעילות מקבילה של כמה threads). מערכת זו משפיעה כשמריצים קוד שהוא multi-threaded.
- Process - תהליך, קטע קוד שרץ. בסביבת פייתון ייתכן מצב של multi-processing.
- Thread - תהליכון, חלק מ-process מסוים. אם יש כמה threads בתהליך אחד, יש לכל אחד מהם זיכרון פנימי אישי וזיכרון משותף לכולם. בסביבת פייתון לא ניתן שיותר מ-thread אחד ירוץ במקביל.

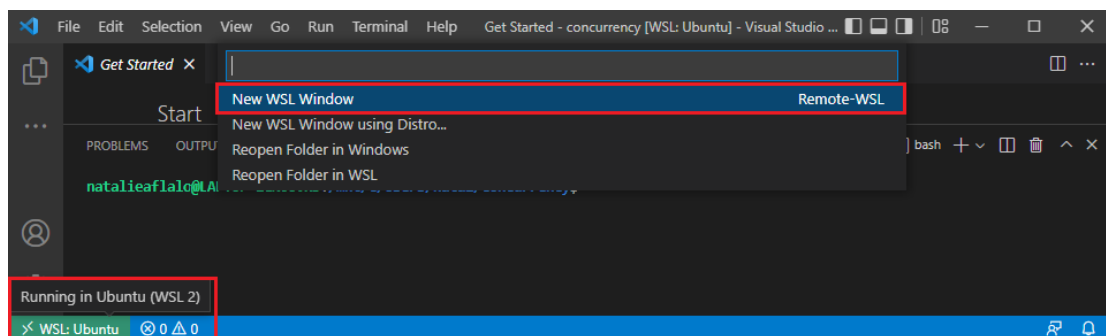
EASS שיעור 8 - VSCode, המשר, Async IO, Classes, Encoding binary data

עבודה עם VSCode בסביבת WSL:

בתוכנת VSCode יש להתקין את התוסף WSL-Remote.



לאחר ההתקנה יש לפתוח חלון של WSL בעזרת לחיצה בצד שמאל למטה ואז בחירה ב-
new WSL window



ברגע שרשום למטה WSL:ubuntu ניתן לעבוד עם VSCode בסביבה הלינוקסית.

הרצת פעולות במקביל - 3 השיטות:

1. Multithreading
2. Processes
3. Async IO

הדגמת Multithreading:

ניצור תיקייה וקובץ שבהם נעבוד. נפתח את הקובץ ב-VSCode:

```
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal$ mkdir concurrency
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal$ cd concurrency
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/concurrency$ touch concurrency.py
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/concurrency$ ls
concurrency.py
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/concurrency$ code .
```

נריץ פונקציה שמריצה דיפולטיבית 5 threads שכל אחד מהם מבצע print ונבדוק כמה זמן לקח לקוד לרוץ: (יש לוודא שמבצעים pip install numpy לספריות שלא מותקנות כבר בסביבה הלוקלית)

The screenshot shows the VS Code interface with a Python file named `concurrency.py` open. The script defines a function `run_threading(n_threads=5)` that creates 5 threads, each printing its ID. The main function calls `run_threading()`. The terminal output shows the execution of the script, displaying the thread IDs and the time taken to complete the execution.

```
concurrency.py > ...
1  import threading
2  import concurrent.futures
3  import time
4  from tqdm.asyncio import trange, tqdm
5  import asyncio
6  import numpy as np
7
8
9  def run_threading(n_threads=5):
10     threads = []
11     print("Starting...")
12     start = time.time()
13     for i in range(n_threads):
14         thread = threading.Thread(target=print, args=[f"I am thread {i}."])
15         thread.start()
16         threads.append(thread)
17
18     for thread in threads:
19         thread.join()
20     end = time.time()
21     print(f"Time to complete: {end - start}")
22
23 if __name__ == "__main__":
24     run_threading()
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/concurrency$ cd /mnt/c/Users/natal/concurrency
-pyhton.python-2022.4.1/pythonFiles/lib/python/debugpy/launcher 42195 -- /mnt/c/Users/natal/conc
Starting...
I am thread 0.
I am thread 1.
I am thread 2.
I am thread 3.
I am thread 4.
Time to complete: 0.02778911590576172
```

- תחילה נמדוד ונתעד את זמן ההתחלה של הריצה.
- בלולאה הראשונה נגדיר כל `thread`, נפעיל אותו עם `start` ונוסיף אותו ל-`pool` שהיא רשימת `threads`.
- בלולאה השנייה נריץ את ה-`threads` בעזרת פונקציית `join` – הפונקציה דורשת שהמערכת תמתין עד סיום הריצה של ה-`thread` שזימן אותה (מבצעת BLOCK)
- לאחר שהיא מסתיימת נמדוד את זמן הסיום. כעת נחשב את ההפרש בין זמן ההתחלה לסיום.

נדגים מצב שבו לא משתמשים בפונקציית join:

```

9  def run_threading(n_threads=5):
10      threads = []
11      print("Starting...")
12      start = time.time()
13      for i in range(n_threads):
14          thread = threading.Thread(target=print, args=[f"I am thread {i}."])
15          thread.start()
16          threads.append(thread)
17
18      #for thread in threads:
19      |   #thread.join()
20      end = time.time()
21      print(f"Time to complete: {end - start}")
22
23  if __name__ == "__main__":
24      run_threading()

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```

natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/concurrency$
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/concurrency$
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/concurrency$
natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/concurrency$ cd /mnt/c/Users/natal/concur
-python.python-2022.4.1/pythonFiles/lib/python/debugpy/launcher 37177 -- /mnt/c/Users/natal
Starting...
I am thread 0.
I am thread 1.
I am thread 2.
I am thread 3.
Time to complete: 0.017986536026000977I am thread 4.

```

ניתן לראות שה-thread האחרון סיים לרוץ (הדפיס) אחרי שנמדד זמן הסיום והודפס הזמן הסופי.

```

1  import threading
2  import concurrent.futures
3  import time
4  from tqdm.asyncio import trange, tqdm
5  import asyncio
6  import numpy as np
7
8
9  def do_concurrent(n=32):
10     start = 10_000_000
11     print_list = [i for i in range(start, start + n)]
12     print("Starting...")
13     start = time.time()
14     with concurrent.futures.ProcessPoolExecutor(max_workers=2) as executor:
15         futures = {executor.submit(print, i): i for i in print_list}
16     for f in concurrent.futures.as_completed(futures):
17         print(f"done {futures[f]} {f.result()}")
18     end = time.time()
19     print(f"Time to complete: {end - start}")
20
21 if __name__ == "__main__":
22     do_concurrent(5)
23
24
25

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```

natalieaflalo@LAPTOP-1GN00VKD:/mnt/c/Users/natal/concurrency$ /home/linuxbrew/.linuxbre
w/opt/python@3.10/bin/python3 /mnt/c/Users/natal/concurrency/concurrency.py
Starting...
10000000
10000002
10000001
10000003
10000004
done 10000003 None
done 10000000 None
done 10000002 None
done 10000004 None
done 10000001 None
Time to complete: 0.024893999099731445

```

פונקציית `as_completed` בשורה 16 מחכה עד להשלמת הפעולה של כל `process`, וכך תוצאת הזמן הסופית מתבצעת אחרי סיום כל התהליכים.

```

1  import threading
2  import concurrent.futures
3  import time
4  from tqdm.asyncio import trange, tqdm
5  import asyncio
6  import numpy as np
7
8  async def wait_and_print(t, n):
9      await asyncio.sleep(t)
10     print(f"coroutine {n} slept for {t} seconds")
11
12
13  async def do_tqdm_asyncio(n=10):
14      coroutines = []
15      async for i in trange(n):
16          print(f"asyncio {i}")
17          coroutines.append(wait_and_print(np.random.randint(1, 5), i))
18      await asyncio.gather(*coroutines)
19
20
21  if __name__ == "__main__":
22      asyncio.run(do_tqdm_asyncio(6))
23
24
25

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```

natalieaf1alc@LAPTOP-1GN00VKD:/mnt/c/Users/natal/concurrency$ /home/linuxbrew/.linuxbre
w/opt/python@3.10/bin/python3 /mnt/c/Users/natal/concurrency/concurrency.py
0%|          | 0/6 [00:00<?, ?it/s]
asyncio 0
asyncio 1
asyncio 2
asyncio 3
asyncio 4
asyncio 5
100%|██████████| 6/6 [00:00<00:00, 15096.48it/s]
coroutine 5 slept for 1 seconds
coroutine 4 slept for 2 seconds
coroutine 0 slept for 3 seconds
coroutine 1 slept for 3 seconds
coroutine 2 slept for 3 seconds
coroutine 3 slept for 3 seconds

```

פונקציית gather בשורה 18 מחכה לכל ה-coroutines במערך שיסתיימו (בלעדיה לא היינו רואים את ההדפסות "coroutine {n} slept for {t} seconds").

השימוש בכוכבית בשורה 18 הוא כדי לבצע flatten. כלומר אם המערך הוא בצורה ([3,3,2,2,3,3]), אז הכוכבית הופכת אותו לצורה (3,3,2,2,3,3).

Class בפייתון:

- אינקפסולציה- מאפיין חשוב בתכנות מונחה עצמים המתייחס לאריזה של מידע (משתנים) ופעולות על המידע (מתודות) כיחידה אחת.
- אובייקטים- אינקפסולציה של משתנים ומתודות.
- Class מחלקה - תבנית ליצירת אובייקטים.

דוגמה:

Lst=list() ליצירת אובייקט רשימה ריקה

Lst=list([1,2,3]) ליצירת אובייקט רשימה עם ערכים התחלתיים

– class יצירת

❖ Self: כמו this בשפות אחרות, מתאר את האובייקט עצמו.

❖ __init__: פונקציית בנאי

❖ Super: המחלקה ממנה אנו יורשים. בדוגמה- super של dog הוא animal.

class Animal:

def __init__(self, name, weight):

self.name = name

self.weight = weight

def eat(self):

print("%s is eating." % self.name)

def make_sound(self):

print("%s is making a sound." % self.name)

class Dog(Animal):

def __init__(self, name, weight, size):

self._size = size

super().__init__(name, weight)

def make_sound(self):

print("Barking")

Encoding binary data

לא כל מידע ניתן לתרגם ל-string, כמו תמונה. אך ניתן להמיר כל דבר לנתון בינארי, וכך ניתן להעביר אותו.

- ❖ Encode - המרת קובץ/תמונה/מחרוזת וכו' למידע בינארי
- ❖ Decode - המרת מידע בינארי לקובץ/תמונה/מחרוזת וכו'. יכול להיות שיהיה שוני בין הקובץ המקורי לקובץ שמוחזר מפעולת ה-decode.

```
>>> import base64
>>> encoded_string = base64.b64encode(b'binary\x00string')
'YmluYXJ5AHN0cmZw=='
>>> decoded_bytes = base64.b64decode(encoded_string)
b'binary\x00string'
```

נדגים כיצד להשתמש ב-encoding על תמונה:

שלב ראשון- בצד שרוצה לשלוח את התמונה: (המרה מקובץ png לבינארי)

```
with open('image.png', 'rb') as f:
    data = f.read()
encoded_data = base64.b64encode(data)
```

שלב שני- בצד שרוצה לקבל את התמונה: (המרה חזרה של בינארי לקובץ png)

```
decoded_data = base64.b64decode(encoded_data)
with open('image_copy.png', 'wb') as f: # note the 'wb' mode!
    f.write(decoded_data)
```

הדגמה של פקודת GET עם FASTAPI שמחזירה תמונה כמידע בינארי ב-JSON:

```
import base64
app=FastAPI()
img="./image.jpg"
@app.get("/v1/get-image")
async def main():
    return {"image": base64.b64encode(open(img, "rb").read())}
```

אם נרצה להחזיר מערך של תמונות כמידע בינארי ב-JSON:

```
img_arr=["a.jpg","b.jpg","c.jpg"]
@app.get("/v1/get-image-arr")
async def main():
    return {"image": [base64.b64encode(open(single_img, "rb").read())
                      for single_img in img_arr]}
```

הערה- ספרייה נוספת שניתן להשתמש בה- imageio.

EASS שיעור 9-SQL, DATACLASS, אלמנטים נוספים בפייתון, מבוא

docker compose

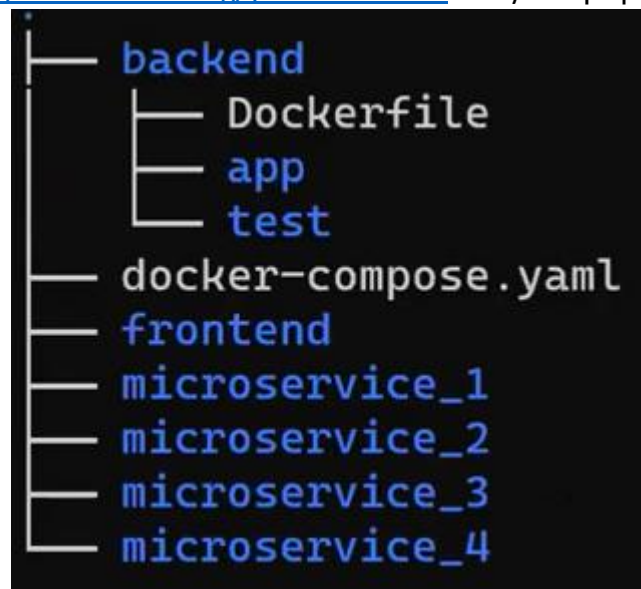
SQL (Structured Query Language) - שפה לניהול בסיסי נתונים יחסיים.
 מנועים שממשים את השפה - PostgreSQL, MySQL, SQLite.
 כדי לפנות לבסיס הנתונים נשלח למנוע בקשות HTTP שיכילו queries.
SQLAlchemy - ספרייה (toolkit) בפייתון שמתרגמת פקודות בשפת פייתון לשאילתות ב-SQL, שאותן שולחת למנוע SQL ואת התשובה מתרגמת לפייתון. מתקשר עם כל אחד מהמנועים של SQL (שחלקם הוזכרו מעלה). דוגמה:

```
SQL :
SELECT * FROM census
WHERE sex = F

SQLAlchemy :
db.select([census]).where(census.columns.sex == 'F')
```

SQLite3 - ספרייה בפייתון שמאפשרת עבודה עם קובץ DB ללא שרת נפרד (microservice במקרה שלנו). <https://docs.python.org/3/library/sqlite3.html>

תצורת פרוייקט - כל מיקרוסרוויס הוא תיקייה, כמו גם ה-frontend וה-backend. לכל אחד מהם צריך ליצור קונטיינר משלו לכן בכל אחת מהתיקיות יהיה Dockerfile וגם טסטים. את הכל יחבר docker-compose שהוא מסוג yaml. (ניתן להיעזר באתרים לביצוע format לקובץ yaml כמו <https://jsonformatter.org/yaml-formatter>)



בREADME יש להוסיף תמונה של דיארגמת בלוקים המתארת את כל חלקי הפרוייקט וכיצד הם מתקשרים (הייתה דוגמה בשיעור וגם בשיעור 7)

:Dataclass

Dataclass הוא class שמכוון לאחסון מידע, כלומר יותר פשוט ונוח למתכנת ליצור בו משתנים.

יצירת Dataclass דורשת ייבוא מהספרייה dataclasses. כל משתנה שניצור בו יהיה לו בנאי (__init__) אוטומטי:

```
from dataclasses import dataclass

@dataclass
class Employee:
    employeeID : int
    name: str
    designation: str
```

לעומת זאת, ב-class רגיל צריך מתודת בנאי __init__:

```
class Employee:

    def __init__(self, employeeID, name, designation):
        self.employeeID = employeeID
        self.name = name
        self.designation = designation
```

Pydantic מזכיר בצורתו Dataclass.

:Decorator

מתודה שמקבלת מתודה אחרת, "עוטפת" אותה, מוסיפה לה פונקציונליות ומחזירה אותה. נשתמש בסימן @ ואחריו שם ה-decorator כדי לגרום למתודה להיות עטופה על ידיו.

דוגמה:

```
>>> def make_pretty(func): #This is the decorator
>>>     def inner():
>>>         print("I got decorated")
>>>         func()
>>>     return inner

>>> @make_pretty
>>> def ordinary(): #This is a regular function that's going to be decorated
>>>     print("I am ordinary")

>>> ordinary()
I got decorated
I am ordinary
```

כל הפונקציות שהשתמשנו בהן ב-fastAPI עטופות ע"י decorators (@app.get וכו')

ניתן גם ליצור decorator מסוג class.

מתודת __call__: מתודה ב-class שמגדירה מה יקרה כשיקראו לו, לדוגמה MyClass()

:Generator

פונקציה שמחזירה אובייקט שניתן לבצע עליו איטרציות, כלומר אובייקט שניתן לקחת כל פעם ערך אחד שלו ולבצע עליו פעולה כלשהי. לדוגמה- לולאה על רשימה של 10 מספרים.

כל פונקציה שמכילה את הפעולה yield לפחות פעם אחת היא generator. yield היא פעולה כמו return, רק שהיא לא מסיימת את פעילות הפונקציה (כמו return) אלא רק עוצרת אותו ושומרת את המצב הקיים עד שניתן להמשיך.

דוגמה:

```
def my_gen():
    n = 1
    print('This is printed first')
    # Generator function contains yield statements
    yield n

    n += 1
    print('This is printed second')
    yield n

    n += 1
    print('This is printed at last')
    yield n

for item in my_gen():
    print(item)
```

ומה שמוחזר:

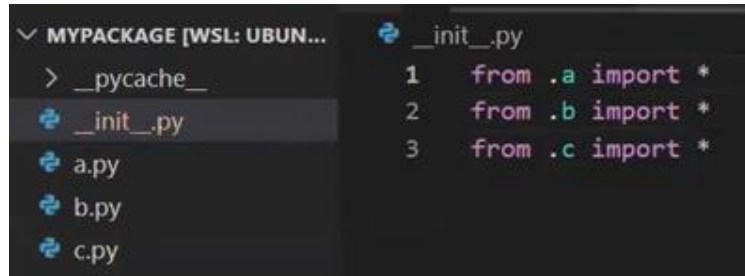
```
This is printed first
1
This is printed second
2
This is printed at last
3
```

ניתן לראות את פעולת השמירה של my_gen בין הדפסה להדפסה גם בעזרת פונקציית next בדוגמה הבאה:

```
>>> a = my_gen()
>>> next(a)
This is printed first
1
>>> next(a) #Value 1 is saved while generator is paused.
This is printed second
2
```

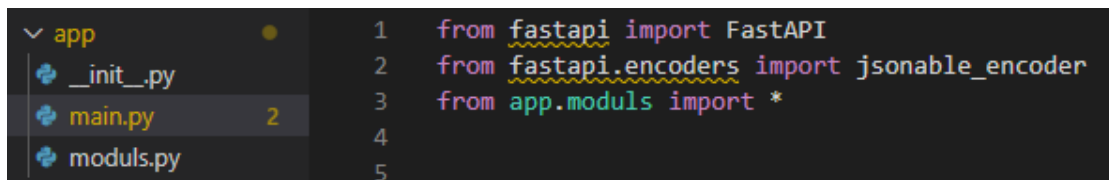
Package - מודל פייתוני שיכול להכיל מודלים אחרים ואף packages אחרים. את ה-package צריך להתקין לוקלית (עם pip install) ואז לייבא בעזרת import בקובץ פייתון שבו נרצה אותו.

על מנת ליצור package עליו להכיל קובץ `__init__.py`, קובץ זה יכול להיות ריק או להכיל קוד. בכל תיקייה נוספת שניצור בתוך ה-package צריך להיות עוד קובץ `__init__`. ניתן לייבא את פונקציונליות שיש בקבצי `py` אחרים שכתבנו בpackages לקובץ ה-`__init__` (ניתן לעשות את זה באופן כללי- לייבא פונקציונליות מקובץ `py` אחד לאחר) בצורה הבאה:



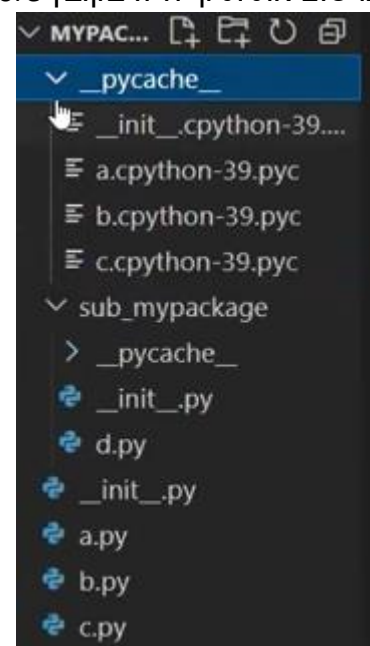
הכוכבית מסמנת שאנו רוצים לייבא את כל הפונקציונליות של הקובץ. ניתן לבחור גם משתנים, פונקציות או class ספציפיים במקום כוכבית.

דוגמה נוספת:



בקובץ modules יצרנו את ה-classes של pydantic. לאחר מכן ייבאנו אותן לקובץ main כדי שנוכל להשתמש בהם. (ניתן גם היה לרשום modules. במקום app.modules מכיוון שהmain הוא גם בתיקיית app)

__pycache__ - תיקייה שנוצרת שבה נשמרים דברים שנמצאים בcache. לדוגמה- אם נבצע import למשהו הוא יישמר שם כחלק משימוש אוטומטי יעיל בקוד. נרשום את תיקייה זו בקובץ gitignore כשנעלה את הקוד ל-github.



Context manager - אמצעי לניהול משאבים כמו קבצים ו-DB בתוך הקוד. לדוגמה- ניתן באמצעותו לקרוא קובץ txt. בתוך הקוד. כאשר נממש את class FileManager חייב לממש בה את שתי הפונקציות: `__enter__` ואת `__exit__`.
נשתמש במילה השמורה with בקריאה לclass הזה פונקציה `__enter__` תופעל אוטומטית ובסיום השימוש פונקציה `__exit__` תופעל אוטומטית.

דוגמה למימוש פשוט-

```
class FileManager:
    def __init__(self, filename):
        self.filename = filename

    def __enter__(self):
        self.file = open(self.filename)
        return self.file

    def __exit__(self, type, value, traceback):
        self.file.close()

with FileManager('sample.txt') as f:
    for line in f:
        print(line)
```

תכנות פונקציונלי- סגנון כתיבת קוד בדומה לצורת כתיבת ביטוי מתמטי, כלומר הקוד מוצג כהרכבה של פונקציות. כדי ליישם זאת נשתמש בפונקציות high-order ו-Lambda.
פונקציות high-order הן פונקציות שמקבלות כקלט או מחזירות כפלט פונקציה אחרת (או פונקציות אחרות).
פונקציית Lambda היא פונקציה אנונימית שמועברת כביטוי לפונקציה אחרת.
דוגמה לבניית מערך של ערכים ריבועיים ללא תכנות פונקציונלי-

```
numbers = [1, 2, 3, 4]
squared_numbers = []
for n in numbers:
    squared_numbers.append(n**2)
print(squared_numbers)
```

ומימוש של הפעולה באמצעות תכנות פונקציונלי-

```
numbers = [1, 2, 3, 4]
squared_numbers = map(lambda x: x**2, numbers)
print(list(squared_numbers))
```

Map- פונקציה שמורה שמקבלת פונקציה ומבנה איטרבילי (כמו רשימה, dictionary, מערך וכו'), ואז מבצעת את הפונקציה על כל אחד מהערכים במבנה האיטרבילי.

```
def myfunc(n):
    return len(n)
```

```
x = map(myfunc, ('apple', 'banana', 'cherry'))
```

ערכו של x בסוף הפעולה [5, 6, 6].

Pandas - ספרייה שנועדה לעיבוד וניתוח נתונים. מאפשרת עבודה עם מבני נתונים כמו dataframe (טבלה) ו-series (סדרה).

Docker compose - מאפשר חיבור של כמה microservices באמצעות קובץ אחד מסוג .yaml.

נבצע את הבנייה של imagen באמצעות פקודה docker-compose build
(במקום docker build שעשינו עד כה)

נבצע את יצירת הקונטיינר באמצעות פקודה docker-compose up
(במקום docker run שעשינו עד כה)

דוגמה לקובץ .yaml :

```
version: "3.9" #The docker-compose version
services:
  backend:
    build: . # File location. One dot means the location is current file.
    ports:
      - "8888:80" #8888 is the port in localhost.
  redis:
    image: "redis:alpine"
```

EASS שיעור 10 - docker compose, Streamlit

חלק שלישי של המטלה-

- Docker-compose
- סרטון קצר שמציג כיצד להריץ את האתר (docker-compose up וכו')
- לפחות 3 מיקרוסרוויסים (backend, frontend הם שני מיקרוסרוויסים שחייב לעשות)

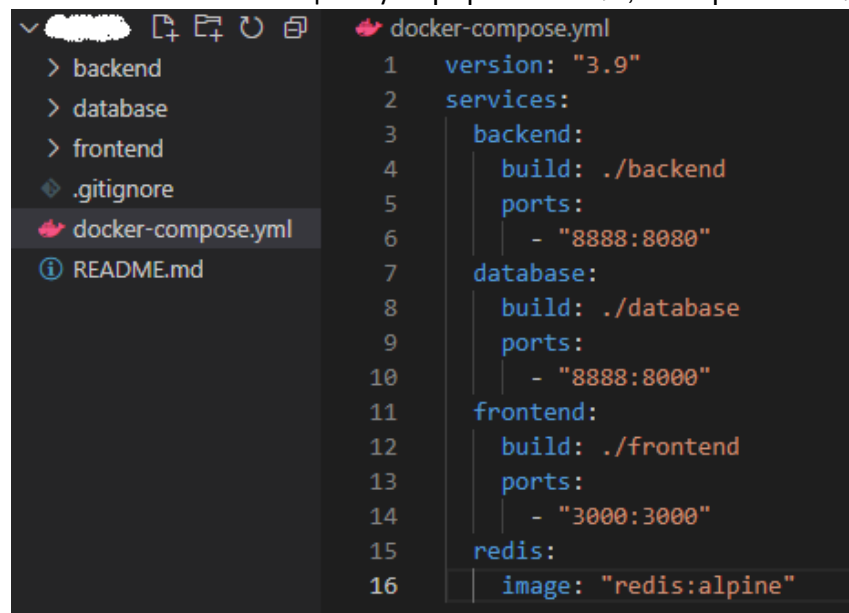
:Docker-compose

מאפשר חיבור של כמה microservices באמצעות קובץ אחד מסוג yml.

נבצע את הבנייה של image באמצעות פקודה docker-compose build (במקום docker build שעשינו עד כה)

נבצע את יצירת הקונטיינר באמצעות פקודה docker-compose up (במקום docker run שעשינו עד כה)

עבור הפרוייקט שלנו, עלינו לבנות קובץ docker-compose.yml בצורה הבאה:



```

1  version: "3.9"
2  services:
3    backend:
4      build: ./backend
5      ports:
6        - "8888:8080"
7    database:
8      build: ./database
9      ports:
10       - "8888:8000"
11   frontend:
12     build: ./frontend
13     ports:
14       - "3000:3000"
15   redis:
16     image: "redis:alpine"

```

נשים לב למילים השמורות:

- Version- הגרסה של docker-compose
- service- הצגת המיקרוסרוויסים שנרצה לחבר באמצעות docker-compose
- build- מיקום התיקיה של המיקרוסרוויס
- ports- הפורטים שדרכם המיקרוסרוויס יעבוד. הפורט השמאלי הוא הפורט החיצוני (זה שנרשום בדפדפן לצד: localhost) והפורט הימני הוא הפורט הפנימי של הקונטיינר.
- Image- קריאה לimage שקיים בdocker hub ונרצה להשתמש בו (במקום ליצור image עם תיקייה נוספת שיש בה Dockerfile ושאר הרכיבים שבונים אותו). נרשום את שם ה-image והגרסה שלו כמו שקיים ב-docker hub. במידה ונשתמש בזה ניישם את השימוש בסרוויס זה בתוך אחד מקבצי py. בעזרת ספריות client של פייתון.
- Volumes- העתקת קובצים לוקליים לקונטיינר (מתאים לשימוש בסרוויס database). משמאל לנקודותיים הנתיב הלוקלי, מימין לנקודותיים הנתיב בקונטיינר.

```

volumes:
  - ./database:/src

```

:Streamlit

ספרייה בפייתון שיוצרת UI (user interface).

שימוש: תחילה יש להריץ `pip install streamlit`.

הערה: אם נבחר להשתמש ב-streamlit ב-frontend שלנו, נשים לב שכאשר ניצור את ה-dockerfile של ה-frontend נכתוב בקובץ requirements.txt את streamlit וכך פעולת ההתקנה תתבצע גם בקונטיינר.

ניצור תיקייה וקובץ

```
/natal$ mkdir streamlit-test
/natal$ cd streamlit-test
/natal/streamlit-test$ vi api.py
```

דוגמה לתוכן של קובץ (מהמצגת שיעור)

```
import streamlit as st
st.title("Hello, Streamlit!")
st.write("Here's our first attempt at using Streamlit")
st.markdown("""
# Streamlit is **awesome**.

That's why we use it for all our projects.
""")

st.latex(r'''
a + ar + a r^2 + a r^3 + \cdots + a r^{n-1} =
\sum_{k=0}^{n-1} ar^k =
a \left(\frac{1-r^n}{1-r}\right)
''')
```

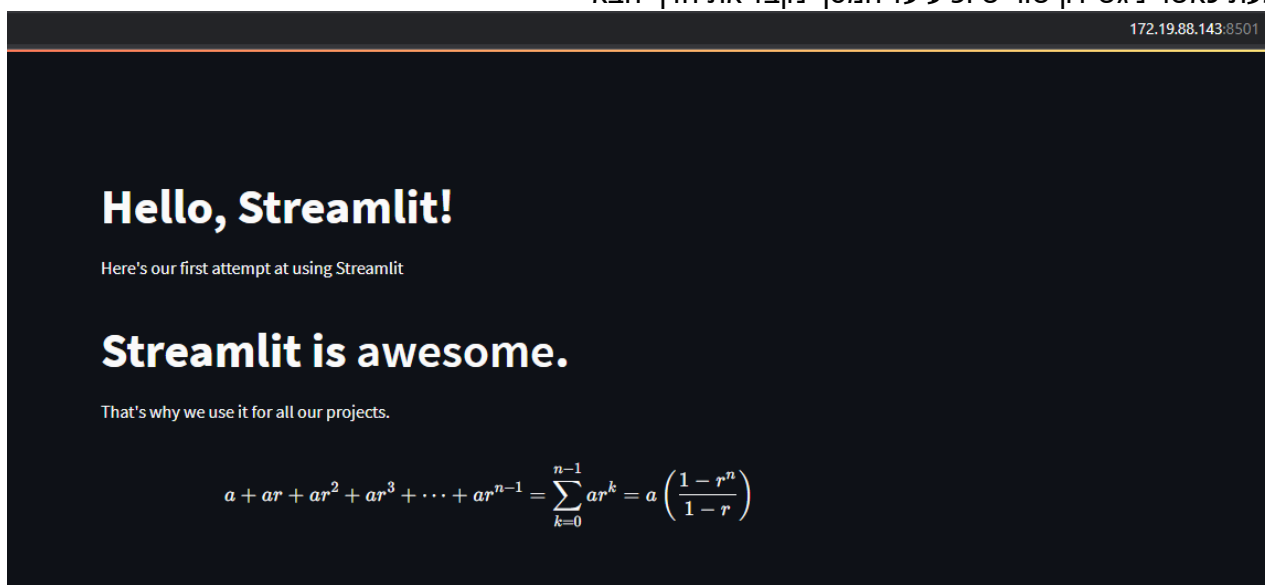
ונריץ את הפקודה

```
natalieaflalo@LAPTOP-1GN0VKD:/mnt/c/Users/natal/streamlit-test$ streamlit run api.py --server.port 8501

You can now view your Streamlit app in your browser.

Network URL: http://172.19.88.143:8501
External URL: http://77.137.65.14:8501
```

כעת כאשר ניגש לקישור שיופיע על המסך נקבל את הדף הבא-



EASS שיעור 11- עבודה עם GITHUB, המשך STREAMLIT

CMD VS ENTRYPOINT

הערות לעבודה עם github:

1. קבצים שלא נרצה שיוצגו בגיטהאב יש לשים בקובץ gitignore (כמו .DS_Store , תיקיית .vscode , cache וכו').
כדי לכלול בקובץ את כל הקבצים שיש להם סיומת זהה ניתן להשתמש בכוכבית, לדוגמה: *.cache
- ניצור קובץ gitignore. ונרשום בו בכל שורה את השמות של הקבצים/תיקיות המיותרים.
ניתן לחפש באינטרנט תבניות מוכנות template של קבצי gitignore להשראה.
קובץ readme איכותי. קישור להסברים והדגמות:
2. <https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax>
3. בכל העלאת קבצים לגיטהאב ניצור commit לכל שינוי עם שם משמעותי.
4. עבור כל מטלה/פרוייקט ניצור repository נפרד.
5. ישנה תוכנה בשם github desktop שבה ניתן לערוך ולהעתיק repositories.
6. עבודה עם branch: ניצור branch שבו נעלה שינויים, כשנאחד את השינויים עם branch הראשי נמחק את הbranch המשני. מצב שבו לא נמחק branches זה כשיש גרסאות שונות אז לכל גרסה יהיה branch.

- בפקודת write ניתן ליצור ולהדפיס טבלה בעזרת pandas dataframe

```
import streamlit as st
st.title("Hello, Streamlit!")
st.write("Here's our first attempt at using Streamlit")

st.markdown(
    """
    # Streamlit is **awesome**.

    That's why we use it for all our projects.

    $ \mathbf{F} = m \mathbf{a}$

    """)
st.latex(r'''
    a + ar + a r^2 + a r^3 + \cdots + a r^{n-1} =
    \sum_{k=0}^{n-1} ar^k =
    a \left(\frac{1-r^n}{1-r}\right)
    ''')

import pandas as pd

st.write("Here's our first attempt at using data to create a table:")
st.write(pd.DataFrame({
    'first column': [1, 2, 3, 4],
    'second column': [10, 20, 30, 40]
}))
```

← → ↺ ⌂ 🌐 localhost:2222 🔍 📄

$k=0$

Here's our first attempt at using data to create a table:

	first column	second column
0	1	10
1	2	20
2	3	30
3	4	40

- שימוש ב-widget כדי לקבל קלט מהמשתמש. דוגמה:

```
import streamlit as st
x = st.slider('x') # this is the widget
st.write(x, 'squared is', x * x)
```

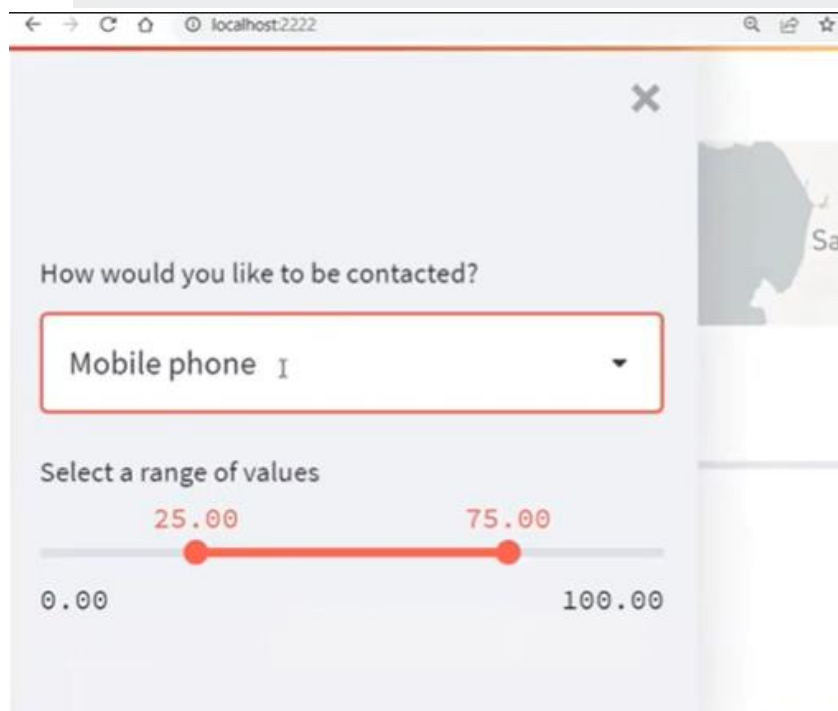


- Layout – הוספת אלמנטים לפריסת המסך כמו תפריט צד, תא שניתן להרחיב ולצמצם אותו. קישור לדוגמאות נוספות: <https://docs.streamlit.io/library/api-reference/layout>

```
import streamlit as st

# Add a selectbox to the sidebar:
add_selectbox = st.sidebar.selectbox(
    'How would you like to be contacted?',
    ('Email', 'Home phone', 'Mobile phone')
)

# Add a slider to the sidebar:
add_slider = st.sidebar.slider(
    'Select a range of values',
    0.0, 100.0, (25.0, 75.0)
)
```

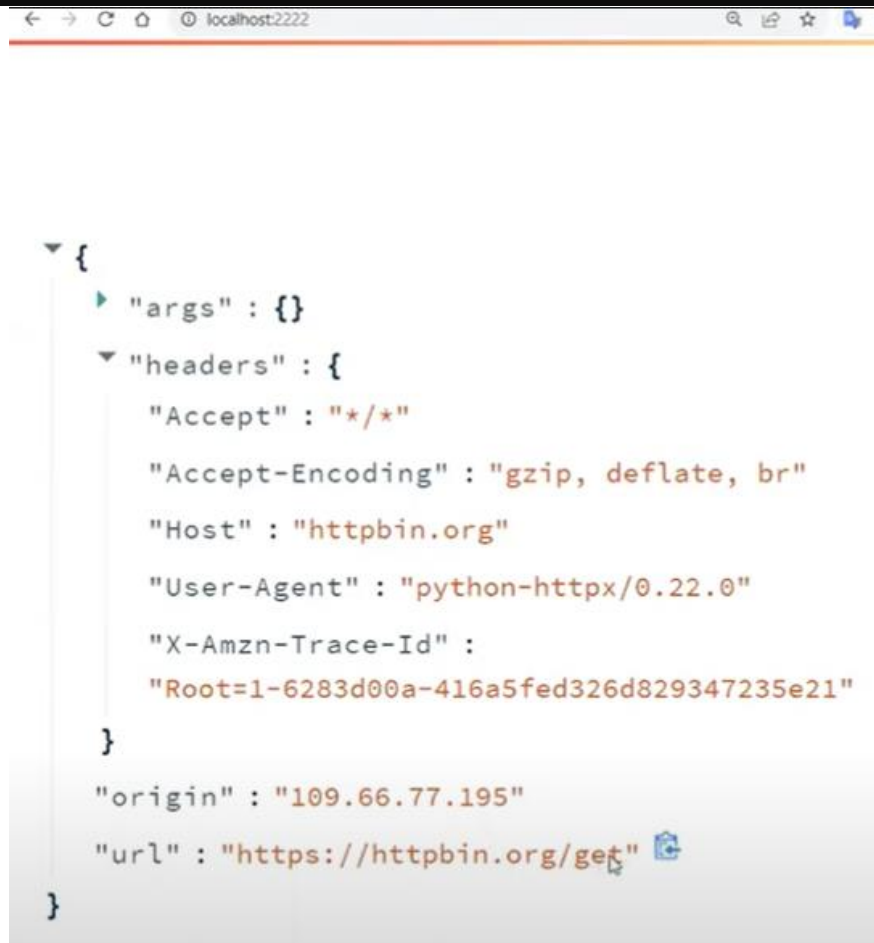


- תקשורת עם ה-backend : נשתמש בבקשות HTTP בעזרת ספריית httpx

```
import streamlit as st
import httpx

backend_endpoint = "https://httpbin.org/get"
r = httpx.get(backend_endpoint)

st.json(r.json())
```



```
{
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate, br",
    "Host": "httpbin.org",
    "User-Agent": "python-httpx/0.22.0",
    "X-Amzn-Trace-Id": "Root=1-6283d00a-416a5fed326d829347235e21"
  },
  "origin": "109.66.77.195",
  "url": "https://httpbin.org/get"
}
```

- שימוש ב-Streamlit באמצעות Dockerfile:

```
FROM python:3.8
RUN pip install streamlit
COPY . /app
WORKDIR /app
ENTRYPOINT ["streamlit", "run", "app.py"]
```

הבדל בין ENTRYPOINT ל-CMD:

שתי הפקודות הללו מגדירות מה ירוץ כשהקונטינר יתחיל.

ההבדל ביניהן הוא ש-ENTRYPOINT נותן פרמטרים שלא "נדרסים" אוטומטית (אלא "נדרסים" רק אם נשים --entrypoint עם פרמטר חדש בשורת ה-run), לעומת זאת CMD נדרס אוטומטית אם נשים פרמטר (ללא צורך בקידומת).

```
from ubuntu:latest
RUN apt-get update
RUN apt-get install -y iputils-ping

ENTRYPOINT ["/bin/ping"]
CMD ["localhost"]
```

נוסיף את פרמטר google.com שיחליף את localhost אוטומטית:

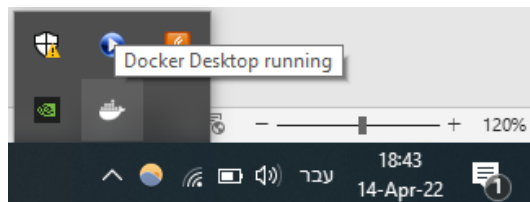
```
→ html-demo docker run -it aaa google.com
PING google.com (142.251.37.174) 56(84) bytes of data.
64 bytes from 142.251.37.174 (142.251.37.174): icmp_seq=1 ttl=37 time
=43.2 ms
64 bytes from 142.251.37.174 (142.251.37.174): icmp_seq=2 ttl=37 time
=43.6 ms
```

נוסיף את קידומת --entrypoint עם הפרמטר החדש שלה כדי לדרוס את /bin/ping :

```
→ html-demo docker run --entrypoint bash -it aaa
root@4d0d6d9e9f2f:/# exit
exit
```

ריכוז פעולות נפוצות ב-Docker

הערה- יש לעבוד בסביבת WSL ותוכנת DOCKER DESKTOP צריכה לרוץ



- docker build : בונה image מה-dockerfile. פרמטר -f בוחר את ה-dockerfile. פרמטר -t נותן שם ל-image.

```
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker build . -f Dockerfile -t nginx-eass
[+] Building 19.4s (9/9) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 146B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/nginx:latest
=> [1/4] FROM docker.io/library/nginx:latest
=> [internal] load build context
=> => transferring context: 476B
=> [2/4] RUN apt-get update
=> [3/4] RUN apt-get -y install vim
=> [4/4] COPY ./html_edit /usr/share/nginx/html
=> exporting to image
=> => exporting layers
=> => writing image sha256:bc57e1f7f36fef0619dfb0fc11cacb6e61546df45a44b8a4c4ee9c0706d5323e
=> => naming to docker.io/library/nginx-eass
```

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

- docker pull : המשתמש נותן לפקודה כלי (וגרסה) שהוא רוצה לעבוד עם האימג' שלו. במידה והאימג' לא קיים לוקלית מתבצעת הורדה של האימג' מ-dockerhub. דוגמה- ubuntu:20.04
אם לא רשמים גרסה אז מתבצעת הורדה של הגרסה האחרונה latest.

```
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker pull python
Using default tag: latest
latest: Pulling from library/python
dbba69284b27: Pull complete
9baf437a1bad: Pull complete
6ade5c59e324: Pull complete
b19a994f6d4c: Pull complete
8fc2294f89de: Pull complete
79232cc264be: Pull complete
5f345e9e59ca: Pull complete
016285d99bc7: Pull complete
636646f6a8cd: Pull complete
Digest: sha256:ea83294c1fd52c5772958947e323d827af3bd6f0c95537fd24b7faba4c425117
Status: Downloaded newer image for python:latest
docker.io/library/python:latest
```

- docker run : יוצר container ל-image, מריץ את האימג'. אם האימג' לא קיים מבצע pull אוטומטית.
אם מוסיפים לפקודה -ti זה הופך את הטרמינל לאינטרקטיבי (פותח שורה חדשה שבה ניתן לתת פקודות ל-container).
אם מוסיפים בסוף הפקודה /bin/sh הרצה בshell (command שונה).
אם מוסיפים בסוף הפקודה /bin/top מוצג איזה services רצים.

```
natalieafalalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker run -ti python:latest
Python 3.10.4 (main, Apr 14 2022, 04:16:58) [GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

- docker ps : מציג את כל הcontainers שרצים כרגע. פרמטר -a מציג גם קונטיינרים שעצרו או "נהרגו" (kill)

```
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d41a4f139250	redis	"docker-entrypoint.s..."	10 days ago	Exited (137) 10 days ago		some-redis
08a9d8542d54	local-http-api-demo	"python ./local_clie..."	3 weeks ago	Exited (0) 3 weeks ago		thirsty_dhawan
21bf22e6d825	b138b9264903	"gunicorn -b 0.0.0.0..."	3 weeks ago	Exited (137) 3 weeks ago		pensive_blackwell
9c9cae99ff72	http-api-demo	"python ./client.py"	3 weeks ago	Exited (0) 3 weeks ago		focused_proskuriakova
87f1a16cc458	b138b9264903	"gunicorn -b 0.0.0.0..."	3 weeks ago	Exited (137) 3 weeks ago		modest_payne
e93b26458dc3	local-http-api-demo	"python ./local_clie..."	3 weeks ago	Exited (0) 3 weeks ago		clever_spence
b9356324c792	b138b9264903	"gunicorn -b 0.0.0.0..."	3 weeks ago	Exited (0) 3 weeks ago		nervous_gates
7445ae59856f	b138b9264903	"gunicorn -b 0.0.0.0..."	4 weeks ago	Exited (137) 4 weeks ago		confident_faraday
cd53a0a517bf	http-api-demo	"python ./client.py"	4 weeks ago	Exited (0) 4 weeks ago		heuristic_hypatia
46cd0ecf5870	ubuntu	"bash"	4 weeks ago	Exited (0) 4 weeks ago		mystifying_kare
a284bfa955c7	python	"python3"	4 weeks ago	Exited (0) 4 weeks ago		elastic_kalam
25aba0e109fb	python	"python3"	4 weeks ago	Exited (137) 4 weeks ago		boring_herschel
046895a25506	ubuntu	"bash"	5 weeks ago	Exited (129) 5 weeks ago		clever_nightingale
f2e9f03bff15	ubuntu	"bash"	5 weeks ago	Exited (0) 5 weeks ago		admiring_cerf
b22925a96740	ubuntu	"bash"	6 weeks ago	Exited (0) 6 weeks ago		vigorous_benz
7c47bc2cf9be	ubuntu	"bash"	6 weeks ago	Exited (0) 6 weeks ago		clever_mclaren
76ebffad3fbf	nginx	"/docker-entrypoint...."	6 weeks ago	Created		newWeb
d7b38d596674	ubuntu	"bash"	6 weeks ago	Exited (129) 6 weeks ago		determined_driscoll
8bcc5cbb5e7a	ubuntu	"bash"	6 weeks ago	Exited (137) 6 weeks ago		stoic_hopper
a1d8e1031643	ubuntu	"bash"	6 weeks ago	Exited (137) 6 weeks ago		magical_leavitt
90ff6493d8eb	ubuntu	"bash"	6 weeks ago	Exited (0) 6 weeks ago		tender_dubinsky
a31e181f08fe	ubuntu	"bash"	6 weeks ago	Exited (0) 6 weeks ago		strange_lalande
47f6b4a4fb6f	ubuntu	"bash"	6 weeks ago	Exited (129) 6 weeks ago		festive_noyce
9c6550a2faeb	ubuntu	"bash"	6 weeks ago	Exited (0) 6 weeks ago		musling_keldysh

- docker rm \$(docker ps -a -q) : פקודה שמוחקת את כל הקונטיינרים (גם אלה שלא רצים כרגע). כלומר, אחרי הרצת הפקודה הזו, docker ps יהיה ריק לחלוטין.

```
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker rm $(docker ps -a -q)
```

d41a4f139250
08a9d8542d54
21bf22e6d825
9c9cae99ff72
87f1a16cc458
e93b26458dc3
b9356324c792
7445ae59856f
cd53a0a517bf
46cd0ecf5870
a284bfa955c7
25aba0e109fb
046895a25506
f2e9f03bff15
b22925a96740
7c47bc2cf9be
76ebffad3fbf

- docker images : מציג את כל הimages שקיימים לוקלית, כל גרסה מקבלת שורה נפרדת.

```
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
redis	latest	bba24acba395	2 weeks ago	113MB
local-http-api-demo	latest	300689d94d34	3 weeks ago	922MB
http-api-demo	latest	0e50e42b3329	4 weeks ago	919MB
python	latest	178dcaa62b39	5 weeks ago	917MB
ubuntu	20.04	2b4cba85892a	5 weeks ago	72.8MB
nginx-eass	latest	bc57e1f7f36f	6 weeks ago	196MB
ubuntu	latest	54c9d81cbb44	2 months ago	72.8MB

- docker rmi : מחיקת image לוקלית.

```
python <none> 178dcaa62b39 5 weeks ago 917MB
ubuntu latest 54c9d81cbb44 2 months ago 72.8MB
natalieaf1alo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker rmi 178dcaa62b39
Untagged: python@sha256:3204faabc2f0b5e0939b8b29079a2a330c38dee92a22482a9ed449c5649a55
Deleted: sha256:178dcaa62b393b539abc8b866c39be81e8ade01786880dc5d17ce3fe02426dbb
Deleted: sha256:400d99adaab999aea8c559754ba5978abb831ddab8043c02f1d16ae8b58b6c58
Deleted: sha256:25698fb762b3458e2c3d7ef87084fffd596d14a59499f4a7bd53d0bc744b75ba5
Deleted: sha256:0e457ae19e1984adb75f45e4882b75c7aace9765835b9205161cb07731b86775
```

- docker network ls : מייצר לכל מודול network בפני עצמו, ls מציג את סוגי הnetworks.

```
natalieaflo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
e001825e57c3        bridge             bridge              local
6251abd4e1a6        host               host                local
da7811d86645        none              null                local
```

- Docker kill : עוצר את הcontainer שהID שלו רשום בסוף הפקודה.

```
natalieaflo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
6e286312ab83       python:latest      "python3"          18 seconds ago     Up 15 seconds      0.0.0.0:80->0.0.0.0:80   charming_blackwell
natalieaflo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker kill 6e286312ab83
6e286312ab83
natalieaflo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
natalieaflo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$
```

- Docker exec -ti : יוצר טרמינל אינטרקטיבי לcontainer שכבר רץ ברקע. יש לתת לפקודה זו ID או שם של הcontainer.

```
natalieaflo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ docker exec -ti web_edit bash
root@56689d70a4cc:/# ls
bin boot dev docker-entrypoint.d docker-entrypoint.sh etc home lib lib64
root@56689d70a4cc:/# ls /user/share/nginx/html
index.html
```

יצירת Image חדש:

באמצעות Dockerfile אנו יכולים לכתוב image. הוא יכול להתבסס על image קיים, ניתן להתקין בו תוכנות נוספות, להוסיף לו קבצים ועוד.

נתחיל בעריכת dockerfile באמצעות הפקודה vi Dockerfile. לאחר לחיצה על enter נכנס למסך תצוגה. יש ללחוץ על i במקלדת כדי להיכנס למצב עריכה. כעת נכניס את תוכן האימג' :

- FROM נספק לdockerfile אימג' קיים שממנו הוא יורש. חייב כאן לרשום את הגרסה של האימג'.
- COPY העתקה מה-host (מחשב לוקלי) לתיקייה מסוימת בdockerfile
- RUN התקנת תוכנה. לפני ההתקנות חייב לרשום שורה RUN apt-get update
- פקודות (instruction) נוספות והסברים-

<https://www.learnitguide.net/2018/06/dockerfile-explained-with-examples.html>

בסוף העריכה נלחץ esc במקלדת וכדי לשמור את השינויים ולצאת חזרה לWLS נרשום wq: (נקודותיים ואז wq) ונלחץ enter.

ליצירת האימג' dockerfile נשתמש בפקודת docker build

דוגמה:

```
natalieaflalo@LAPTOP-1GN0OVKD:/mnt/c/Users/natal$ vi Dockerfile
```

```
FROM nginx:latest
RUN apt-get update
RUN apt-get -y install vim
COPY ./html_edit /usr/share/nginx/html
:~
:~
:~
:~
:~
:wq
```

ניתן ליצור ולערוך את קובץ Dockerfile גם באמצעות Visual Studio Code.