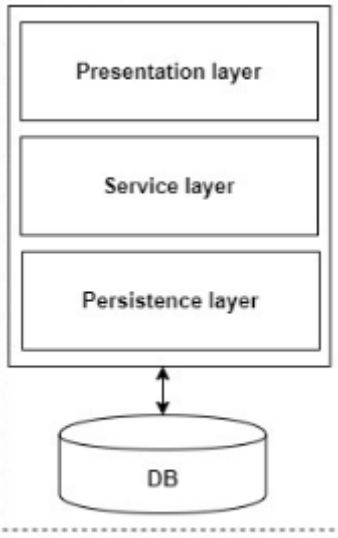
הרצאה 3:

Monolithic vs Microservices architecture

מונוליט: כאשר כל המערכת מפותחת במקום אחד אשר נארזת ומבצעת deploy לקובץ jar\war יחיד.
בעצם מדובר בcodebase אחיד שמכיל את כל הפונקציונליות הנדרשת.



Monolithic architecture

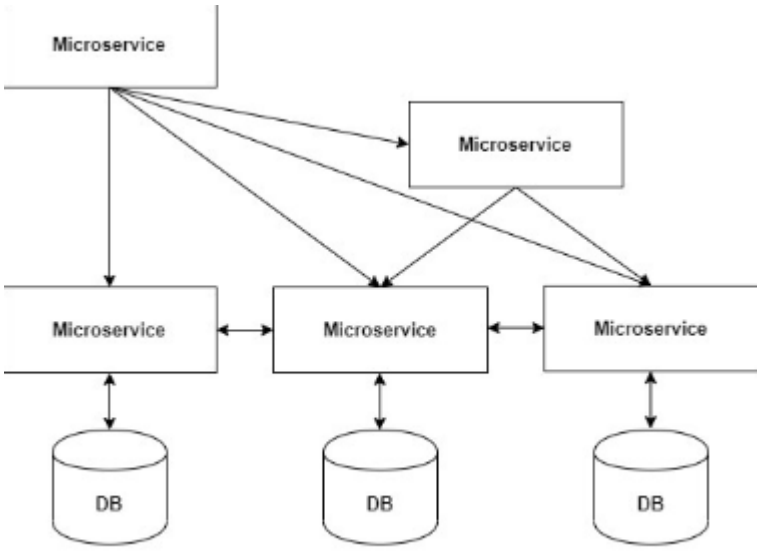
החסרונות של מערכת מונוליטית:

- המערכת נהית גדולה ומסורבלת ככל שעובר הזמן וכך קשה לנהל אותה בצורה מסודרת
- עבור כל שינוי קטן מערכת יש לבצע redeploy למערכת שלמה
- ככל שהמערכת גדולה יותר, כך הזמן שלוקח לה לעלות עולה גם כן
- קשה להבנה עבור מפתחים חדשים

היתרונות של מערכת מונוליטית:

- ביצוע deploy קל יותר כאשר מדובר בjar/war יחיד
- קלה יותר לפיתוח
- מיקוד במערכת יחידה

מיקרוסרוויס: המערכת מחולקת לservices אשר מתקשרים ביניהם באמצעות lightweight protocols כמו הHTTP.



Microservices architecture

לעומת מונוליט שעבור כל המערכת יש database יחיד, לכל מיקרוסרוויס יש database משלו. חיסרון בכך הוא דופליקציה של מידע אך זה הכרחי בשביל להשיג loose coupling.
יתרון נוסף הוא היכולת לממש טכנולוגיות שונות בכל מיקרוסרוויס בגלל החוסר תלות במיקרוסרוויסים האחרים.

עקרונות המיקרוסרוויסים:

- **Single Responsibility** עקרון השייך לעקרונות הSOLID
- **Built around business capabilities** עקרון המעודד שימוש בטכנולוגיות שונות וחדשניות לצורך פתרון בעיות שונות. בשונה ממערכת מונוליטית שבה קשה ליישם טכנולוגיות חדשות.
- **Design for failure** עקרון האומר שיש לבנות מיקרוסרוויסים תוך כדי לקיחה בחשבון של השגיאות שעלולות לקרות. כלומר, יש לוודא שהמערכת יכולה לתפקד גם כאשר המיקרוסרוויס הספציפי לא זמין

Daemons

במערכות הפעלה יוניקס ודמויות-יוניקס, ובמערכות הפעלה אחרות התומכות בריבוי משימות, daemon (דימון) היא תוכנית מחשב שרצה כתהליך רקע, להבדיל מתוכניות הנמצאות תחת שליטתו הישירה של משתמש אינטראקטיבי. בדרך כלל מערכות מפעילות דימונים בזמן האתחול, ולרוב הם משרתים פונקציות כגון תגובה לבקשות רשת, לפעילות חומרה, או לתוכניות אחרות על ידי ביצוע של משימה כלשהי.

Mount

פעולת mount (עיגון) במחשוב מתבצעת לפני גישת המחשב אל התקנים שונים (כגון החסנים ניידים, מחיצות נוספות, וכונני CD-ROM). משמעות הmount היא הצמדת ההתקן אל מערכת הקבצים הבסיסית של מערכת ההפעלה, והקצאת מספר סידורי שיהיה מזהה ייחודי להתקן, ובכך לאפשר למחשב והמשתמש גישה אליו.

בלינוקס, mount היא פקודה המשמשת לעיגון מערכות קבצים. על מנת לגשת לקובץ, מערכת הקבצים המכילה אותו חייבת להיות מעוגנת בעזרת הפקודה mount. הפקודה מנחה את מערכת ההפעלה לשימוש במערכת הקבצים החדשה, מעגנת אותה בנקודה מסוימת בהיררכיית מערכת הקבצים הגלובאלית (mount point), ומגדירה אפשרויות הנוגעות לגישה אליה.

הרחבה על Docker

Docker Hub

בעצם זהו repository ענק המכיל בתוכו מאגר שלם של images שניתן למשוך אלינו ולבצע בהן שימוש.
כמו כן ניתן לעלות אימג'ים משלנו (ממש באופן דומה לGithub).

Dockerfile

קובץ טקסט המכיל בתוכו פקודות אשר הרצתן יגרום ליצירת Image.

cheat sheet של מבנה הDockerfile

Command	Overview
FROM	Specify base image
RUN	Execute specified command
ENTRYPOINT	Specify the command to execute the container
CMD	Specify the command at the time of container execution (can be overwritten)
COPY	Simple copy of files / directories from host machine to container image
ADD	COPY + unzip / download from URL (**not recommended**)
ENV	Add environment variables
EXPOSE	Open designated port
WORKDIR	Change current directory

דוגמה לDockerfile

```
FROM python:3.8

# set a directory for the app
WORKDIR /usr/src/app

# copy all the files to the container
COPY . .

# install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# define the port number the container should expose
EXPOSE 5000

# run the command
CMD ["python", "./app.py"]
```

Docker volumes

Docker volumes הם מערכות קבצים המותקנות על קונטיינרים של Docker כדי לשמר נתונים שנוצרו על ידי הקונטיינר הפועל.
הנתונים שנוצרים על ידי הקונטיינרים אינם נשמרים כאשר הקונטיינר נמחק, דבר המקשה במצב שבו נרצה לשמור את הנתונים במצב בו ישנו תהליך שצריך נתונים אלו.

דוגמה ממחישה שימוש בDocker

https://www.docker.com/blog/how-to-use-the-official-nginx-docker-image/

1. `docker run -it --rm -d -p 8080:80 --name web nginx`
2. `curl http://localhost:8080`
3. `docker stop web`
4. Add `index.html` to local site-content and map it to `/usr/share/nginx/html` (https://gist.github.com/chrisvfritz/bc010e6ed25b802da7eb)
5. `docker run -it --rm -d -p 8080:80 --name web -v ~/site-content:/usr/share/nginx/html nginx`
6. Doing stuff via Dockerfile (`docker build -t webserver`):

```
FROM nginx:latest
COPY ./index.html /usr/share/nginx/html/index.html
```

7. `docker run -it --rm -d -p 8080:80 --name web webserver`

תיאור הflow:

1- באמצעות פקודת **docker run** אנחנו מריצים image של nginx.
תיאור הoptions בתוך הפקודה:
-it מאפשרת לקונטיינר להיות אינטראקטיבי
--rm מחיקה של הקונטיינר אוטומטית לאחר שיוצאים ממנו
-d הרצת הקונטיינר ברקע והדפסת הID שלו
-p חשיפת הפורטים לשימוש הקונטיינר. צד שמאל זה הפורט שאנו פותחים עבור הקונטיינר במחשב הhost וצד ימין זה הפורט שהקונטיינר חושף לשימוש
name-- יצירת שם לקונטיינר, כאן קראנו לו web
2- הפקודה **curl** מראה את התוכן של מה שיש בתוך הlocalhost:8080
3- באמצעות פקודת **docker stop** עצרנו את הקונטיינר

השלבים 4-7 מהווים אפשרויות שונות לעריכת הindex.html שהקונטיינר מכיל.
בעצם מה שרצינו לעשות בflow זה, זה להריץ קונטיינר של nginx ולערוך את הindex.html שמוצג בתוך הlocalhost:8080 למשהו אחר.

כעת נפרט כל אחת מהאפשרויות:

דרך ראשונה: באמצעות Docker volume (פקודות 4-5)

4- יצירת תיקיה בhost הנקראת site-content ובתוכו ליצור קובץ index.html חדש שנרצה למפות באמצעות **Docker volume**.
רשימת פקודות עבור פעולה זו:
mkdir site-content <-
cd site-content <-
touch index.html <-
vi index.html <-

בתוך עריכת הקובץ, נדביק קובץ HTML כלשהו ונשמור את הקובץ על ידי לחיצה על esc ואז כתיבת ":wq" ולחיצה על enter

cd .. <-
5- נריץ שוב את הפקודה כמו ב(1) רק ההבדל היחידי זה הוספת הvolume באמצעות **-v**
המיפוי מתבצע כך שבצד שמאל הנתיב הוא של הhost והימני זה הנתיב בתוך הקונטיינר

דרך שניה: באמצעות Dockerfile (פקודות 6-7)

6- ניצור בתוך הhost קובץ הקרוי בשם Dockerfile באמצעות הפקודות:
touch Dockerfile <-
vi Dockerfile <-

נדביק את התוכן הבא:
FROM nginx:latest
COPY ./index.html /usr/share/nginx/html/index.html

פירוט הפקודות:
ה**FROM** זה בעצם האימג' שלנו
ה**COPY** דורס את הindex.html בתוך הקונטיינר עם הindex.html שבתוך הhost (היצירה של הקובץ זהה לדרך שבה עשינו זאת בשלב 4).

נשמור את הקובץ ונריץ את הפקודה הבאה:
docker build -t webserver
הפקודה הזו תבנה את האימג' שלנו על ידי הרצת הפקודות שכתבנו בDockerfile ותקרא לו בשם webserver (בזכות השימוש באופציה t-)

7- נריץ את אותה הפקודה כמו בשלב 1 אך הפעם השם של הimage הוא לא nginx אלא webserver שזה האימג' החדש שאנחנו יצרנו בשלב 6.

הערה: בשיעור שיחקנו קצת עם הbash של הקונטיינר וניסינו לערוך קבצים ישירות בתוך הקונטיינר.
שמנו לב שvim לא היה מותקן והתקנו אותו ישירות בתוך הbash של הקונטיינר.
כדי "לחסוך" את ההתקנה הידנית, כאשר הרצנו את הדוגמה של nginx בדרך השנייה, הוספנו לDockerfile את הפקודות הבאות:
RUN apt-get update
RUN apt-get install vim -y
וכך כאשר הרמנו קונטיינר עם האימג' שלנו, הvim היה כבר מותקן בזכות הפקודות החדשות.

איך מיקרוסרוויסים מתקשרים ביניהם – API

Application Programming Interface - API

הוא ערכה של ספריות קוד, פקודות, פונקציות ופרוצדורות מן המוכן, בהן יכולים המתכנתים לעשות שימוש פשוט, בלי להידרש לכתוב אותן בעצמם כדי שיוכלו להשתמש במידע של היישום שממנו הם רוצים להשתמש לטובת היישום שלהם.

Web API

זהו בעצם API עבור הWEB שהגישה אליו היא באמצעות HTTP

Hypertext Transfer Protocol - HTTP

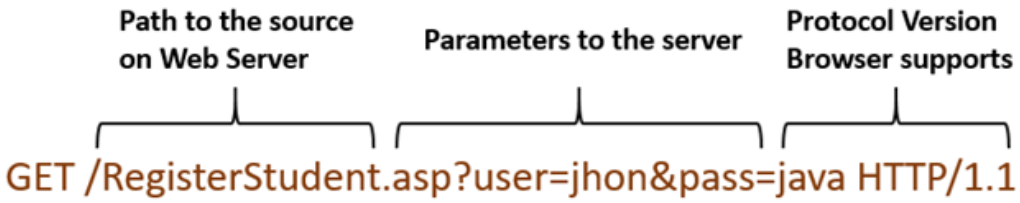
פרוטוקול המאפשר תקשורת בין clients והservers

HTTP methods

- GET
- POST
- PUT
- HEAD
- DELETE
- PATCH
- OPTIONS
- CONNECT
- TRACE

מתודת GET

תפקידה לשלוח בקשה לשרת לצורך השגת נתונים.



מתודת POST

תפקידה לשלוח מידע לשרת לצורך הוספה/עדכון של מידע קיים.
בשונה ממתודת GET, המידע שנשלח לשרת נשלח בתוך BODY ולא בתוך הURL.