Docker

מבוא:

בעולם הטכנולוגי של היום אנו נתקלים בכל מיני בעיות הנוגעות ליצירת מערכת/אפליקציה, לניהול שלה ולתפקוד שלה:

* קיים צורך בבידוד של ה-services הקיימים - על מנת לבצע הפרדה ביניהם ולשמור על הסדר והארגון
* high availability - קיים צורך ביצירת ה-instances הרצויים מספר פעמים על מנת לשמור על שרידות גבוהה
* יצירת שינויים דינמיים באפליקציה שלנו

קיים פתרון לצרכים הללו והוא יצירת כמות של VMים, אך הפתרון הזה הוא יקר, איטי, כבד ולא יעיל בכללץ

פה נכנס docker, עם פתרון לבעיה הזו.

docker יצרו דרך בה נוכל ליצור מספר services מבודדים על גבי מערכת הפעלה אחת. לכל service כזה יש את הספרות שלו והצרכים שלו, והם מופרדים לחלוטין אחד מהשני ועצמאים לחלוטין ובכל זאת יש להם את היכולת לתקשר אחד עם השני.

מה הפתרון שהם יצרו? **containers.**

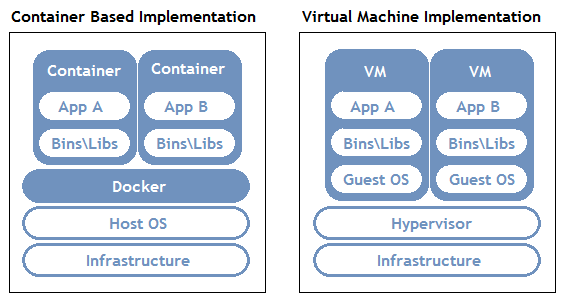
* קונטיינרים חולקים את מערכת ההפעלה של ה-host, ולכן אין להם צורך ביצירת OS משלהם.
* קונטיינר זוהי תוכנה המכילה את כל הדרישות המקדימות והקוד שהוא צריך להכיל על מנת להיות ישות מופרדת.

בגלל שקונטיינר אינו מכיל OS, הוא קל ליצירה, להעברה וקל לבצע בו שינויים. הוא אינו כבד מבחינת כמות המידע, והוא זול בהרבה לעומת יצירה של VM.

**קונטיינרים מציעים בידוד ולא וירטואליזציה**

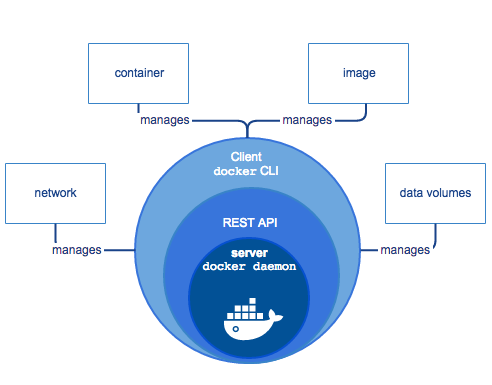
**הם מהווים וירטואליזציה של מערכת ההפעלה**

**לעומת VM שמהווה וירטואליזציה של החומרה**



אז מה זה docker בעצם?

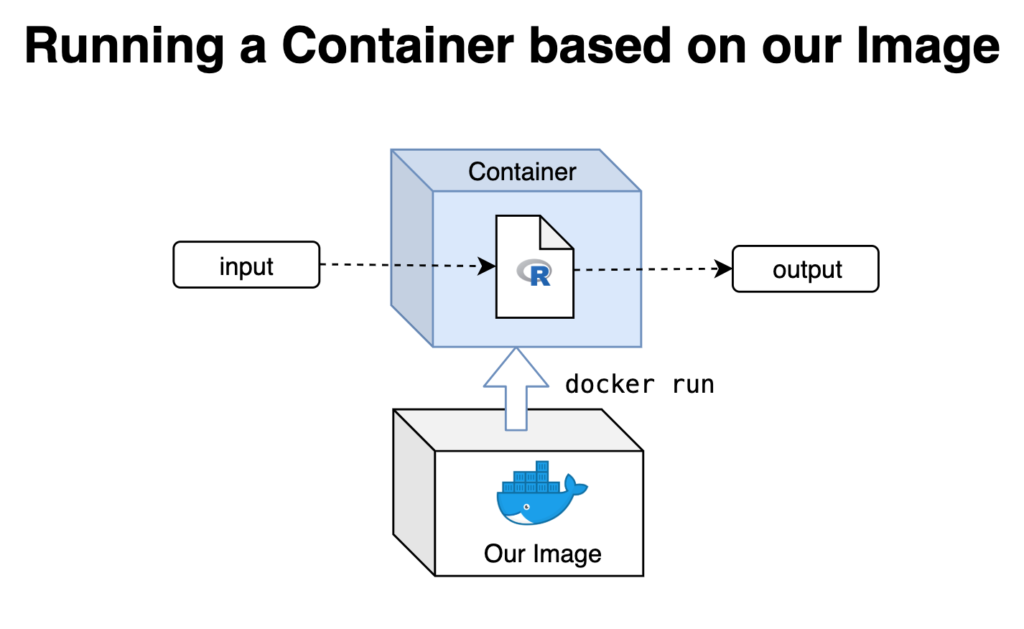
docker הוא **לא** קונטיינר. זוהי חברה שיצרה מוצר לניהול ויצירת קונטיינרים הנקרא: docker engine (קיימים עוד מוצרים כאלה, הוא לא היחיד, אך הוא המוביל בתחום). הם למעשה יצרו פתרון קל ונוח ליצירת קונטיינרים.



docker hub:

זהו אתר המפנה לכתובת הבאה: <https://hub.docker.com/> ונקרא גם docker registry ומכיל בתוכו אוסף של images לבניית קונטיינרים. ניתן גם ליצור יוזר לאתר הנ"ל ולהוסיף לשם images משלי.

קונטיינרים רצים ישירות מה-image, לא ניתן למחוק image כאשר קונטיינר רץ, ולא ניתן לערוך image לאחר שהוא נוצר. ניתן ליצור חדש כגרסא אחרת שלו כ-layer נוסף.



הפקודה docker run מריצה קונטיינר מ-image שנבחר לנו, לדוגמא:

docker run --name myweb -p 7090:80 -d nginx

**name** - בחירת שם לקונטיינר

**p** - הפורט שנרצה לייחצן החוצה, כיוון שהקונטיינר נמצא ברשת פנימית שלא ניתן לגשת אליה, לפי הדוגמא הזו פורט 80 של הקונטיינר ממופה לפורט 7090

**d -** הרצת הקונטיינר ברקע, ללא צפייה ב-shell

docker hub Organization:

אופציה זו מיועדת בעיקר לחברות וארגונים על מנת לבצע סדר בארגון ה-IMAGES בהם נעשה שימוש. אופציה זו אינה חינמית.

מה מכיל organization?

הוא מכיל משתתפים, אותם ניתן לנהל לפי צוותים ולשייך אותם ל-repositories.

כל repo יכיל את ה-imageים הרלוונטים.

בנוסף, ניתן לשייך קישור ל-github/bitbucket ולשלוף משם imageים אל תוך ה-REPO שלנו.

יתרון נוסף הוא שאפשר לשלוח התראות למשתתפים על buildים שנכשלו וכו'

Container Volumes:

קיימות שתי דרכים עיקריות לאחסון מידע בקונטיינרים, ולביצוע שינויים דינמיים במידע המוכל בהם:

* **volumes** - מנוהלים על ידי docker ונמצאים בתיקיה: /var/lib/docker/volumes
* **bind mount** - מאונט הממופה ישירות מה-host אל הקונטיינר

bind mount - בעת הרצת הקונטיינר נשתמש בפרמטר v באופן הבא:

-v <host\_directory>:<container\_directory>

כאשר נריץ את הקונטיינר, המידע שנמצא ב-host directory ימופה אל המידע שנמצא ב-container directory

זוהי דרך טובה בעיקר לשם העברת מידע מה-host.

גם לאחר עצירת הקונטיינר ומחיקתו, המידע שנמצא ב-host directory יישאר.

volumes - על מנת ליצור volume נשתמש בפקודה הבאה:

docker volume create <volume\_name>

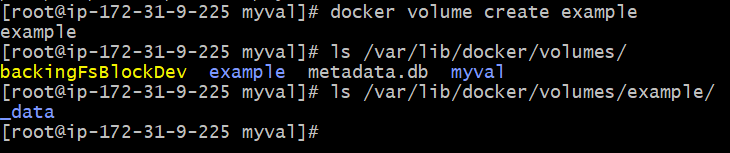
וכדי למפות את ה-volume שיצרנו אל קונטיינר, בעת הרצת הפקודה run נוסיף את הפרמטר הבא:

-v <volume\_name>:<container\_directory>

כלל ה-volumes של docker מאוחסנים בתיקיה הבאה בשרת:

/var/lib/docker/volumes

תיקיה זו תכיל מספר תיקיות כאשר כל אחת מהן היא volume נפרד, לכל תיקיה של volume קיימת עוד תיקיה בתוכה בשם: \_data - שם מאוחסן המידע של הקונטיינר.



רוב המקרים בהם נרצה להשתמש ב-volumes היא אם נרצה להציג מידע מהקונטיינר אל ה-host.

Dockerfile:

זהו קובץ באמצעותו ניתן לבנות docker images.

כל Dockerfile חייב להתחיל עם הפקודה: FROM

פקודה זו מהווה כ-base image, בסיס לבניית כלל ה-image שלנו, אפשר להשתמש לדוגמא ב-ubuntu, ועליו להריץ פקודות נוספות.

דוגמאות לפקודות ב-Dockerfile:

LABELS → add metadata to image

RUN → execute commands **in new layer**

ADD/COPY → add files and folders into image

CMD → run command on “docker run”

ENTRYPOINT → allow you to configure a container that will run as an executable

VOLUME → creates a mountpoint

EXPOSE → container listen on the specified port at runtime

ENV → set environment variable

USER → set user name (or UID)

WORKDIR → sets the working directory

ARG → defines a variable that users can pass at build time

ONBUILD → add to the image a trigger instruction to be executed at a later time

CMD:

מבנה הפקודה יראה באופן הבא:

CMD [<shell\_command>, <argument>]

ENTRYPOINT:

מבנה הפקודה יראה באופן הבא:

ENTRYPOINT [<shell\_command>]

במצב בו לא מצויין argument, מצופה מהמשתמש להכניס אותו בעת הרצת הקונטיינר.

לדוגמא יש לנו את ה-image הבא:

FROM ubuntu:latest

ENTRYPOINT [“echo”]

אם נריץ את הקונטיינר באופן הבא; תתקבל לנו שורה ריקה:

docker run <image\_name>

אך, אם נריץ את הקונטיינר עם ארגומנט לבחירתינו, יתקבל הפלט שלו:

docker run <image\_name> hello

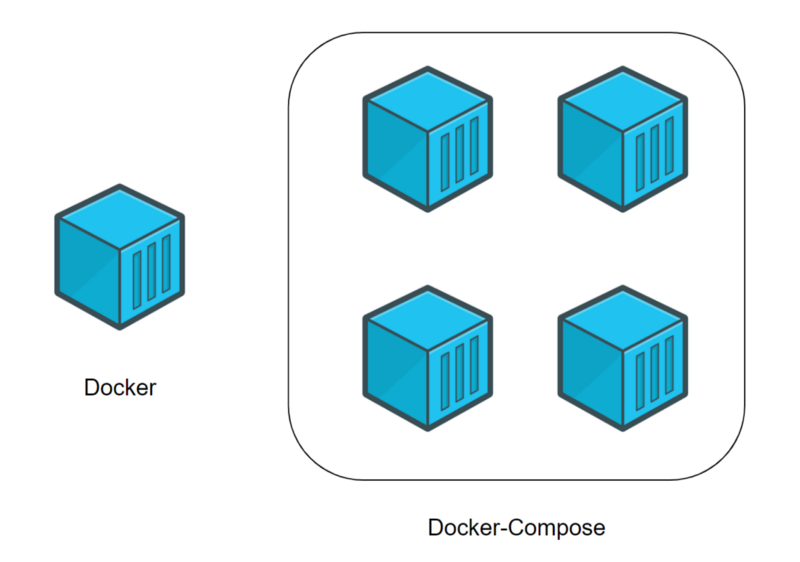
הפלט שיתקבל יהיה: hello

* ניתן לשלב את שניהם ביחד - כאשר ENTRYPOINT מהווה כ-command ו-CMD מהווה כ-argument  
  למה זה שימושי בכלל?
  + CMD משמש כ-default argument - במצב בו המשתמש לא מכניס אחד משלו בעת הרצת הקונטיינר.
  + למקרים בהם נרצה להריץ סקריפט כלשהו, ונרצה להשתמש בפקודה אחרת בכלל להרצת הקונטיינר עצמו. אז לסקריפט נשתמש ב-ENTRYPOINT ולהרצת הפקודה שמפעילה את הקונטיינר נשתמש ב-CMD.

Docker compose:

זהו כלי נוסף של docker שמטרתו היא הרצה של מספר קונטיינרים יחד. באמצעות כלי זה, ניתן להגדיר קובץ YAML המכיל את כלל ה-servies הדרושים להרצה, בפקודה אחת.

על מנת שנוכל להשתמש בו יש להתקין docker-compose על ה-host



Multi-Stage Dockerfile:

זוהי שיטה בה נוכל לבנות image על בסיס image אחר, לדוגמא:

FROM openjdk:8 AS BUILD\_IMAGE

RUN apt update && apt install maven -y

RUN git clone -b vp-docker https://github.com/imranvisualpath/vprofile-repo.git

RUN cd vprofile-repo && mvn install

FROM tomcat:8-jre11

RUN rm -rf /usr/local/tomcat/webapps/\*

COPY --from=BUILD\_IMAGE vprofile-repo/target/vprofile-v2.war /usr/local/tomcat/webapps/ROOT.war

EXPOSE 8080

CMD ["catalina.sh", "run"]

בדוגמא זו ניתן לראות כי בקטע הראשון של הקוד אנו לוקחים image (שיש לנו צורך בו) ובונים באמצעותו image אחר.

למה זה שימושי? בצורה זו, אנחנו מצמצמים את הגודל של ה-image, וככה הבנייה שלו יכולה להיווצר בקלות, והיא לא כבדה כלל.

בעת הרצה של build image יווצרו לנו שני images.

האחד בשם none - שהוא למעשה ה-base image שלנו  
והשני זה ה-image הראשי שלנו (בשם שנבחר לו) - שגודלו יהיה קטן יותר מה-base image