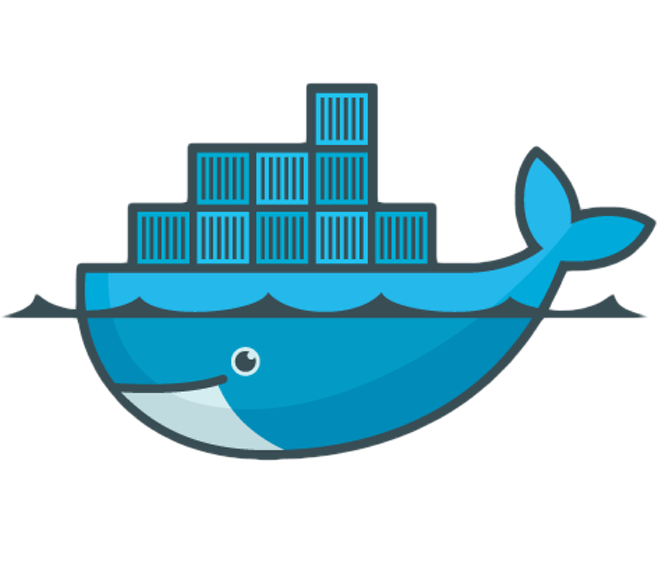
סדנת Docker

השלמה טכנולוגית 003



לפני שמתחילים

**מטרות הסדנה**

* הכרת טכנולוגיית ה-Container ויתרונותיה
* הכרת מושגים בסיסיים
* הכרת פקודות Docker בסיסיות
* מסוגלות ביצירת Image
* ...עוד מטרות

**מבנה הסדנה**

כאן נרשום את סדר הסדנה

* מה זה Docker?
* איך זה עוזר לי?
* מושגים בסיסיים אחרונים
* בואו נתחיל - תרגול פקודות בסיסיות
* שימוש בסיסי ב-Dockerfile ו-Docker Hub

**לפני שמתחילים וודאו שעשיתם את הדברים הבאים:**

* ברשותכם מכונה (וירטואלית או פיזית) עם הפצת Linux כלשהי
* Docker מותקן (apt install docker או snap install docker)
* צרו חשבון Docker Hub ובתוכו Repository (חפשו בגוגל)

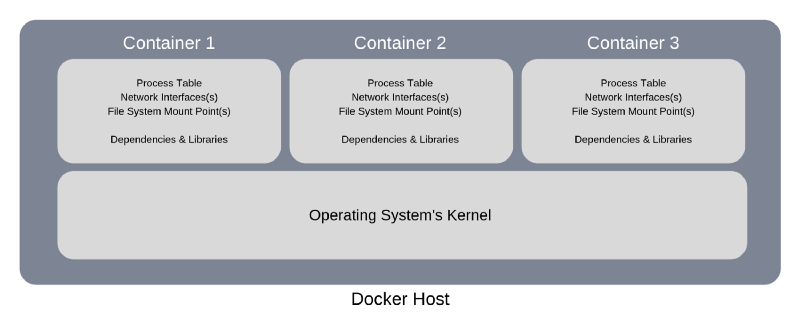
אז עכשיו שהכל סגור – בואו נצא לדרך!

מה זה Docker?

בפשטות, Docker היא פלטפורמה לביצוע וירטואליזציה של מערכת ההפעלה במחשב עליו היא רצה. **בניגוד למכונה וירטואלית** שמדמה את משאבי החומרה, **Docker מדמה את מערכת ההפעלה**. היא עושה זאת על ידי שיתוף ה-Kernel (ליבת מערכת ההפעלה), והרצת Containers על גביו.

**Containers** הם למעשה Processes ש**אינם מכילים את מערכת ההפעלה**, אלא העתקים וירטואליים של חלקים מינימליים ממנה – Process Table, Network Interfaces, File System Mount Points (אין צורך לזכור מה הם, העיקר ההבנה).

בצורה הזו, כל Container מהווה סביבה מבודדת עם ספריות ותלויות משלה, וכולם "חיים" ביחד על אותו Host.



איך זה עוזר לי?

* **תמיכה במספר אפליקציות עם תלויות שונות המתנגשות זו עם זו**.  
  תארו לכם שאתם רוצים להריץ על אותו השרת שתי אפליקציות 3 Python, אך כל אחת מהן זקוקה לגרסת Python 3 אחרת. או ששתיהן רצות על אותה גרסת Python, אך משתמשות באותו Package בגרסאות שונות. טכנולוגית Containers תאפשר זאת.
* **יעילות במשאבים.**הזיכרון הפיזי ש-Container צורך הוא MBs בודדים. נוסף על כך, על ה-Container לא מותקנת מערכת הפעלה ולכן אין Overhead גדול בצריכת זיכרון RAM (בניגוד ל-VM). עובדה זו מקטינה את הזמן להרמת הוירטואליזציה לשניות ספורות (בניגוד ל-VM הדורש מספר דקות). כמו כן ישנה גם הקטנת עלויות משום שהוא לא מצריך חומרה חזקה מדיי.
* **הקנית יציבות למערכת.**משום שכל Container מבודד מחבריו, במידה ואחת האפליקציות תגרום לקריסה שלו – שאר ה-Containers לא יפגעו. Docker מאפשר את גישת ה-Micro-Services הדוגלת בפירוק האפליקציה למספר שירותים מינימליים המתקשרים זה עם זה.
* **הרצת אפליקציה כבדה.**הטכנולוגיה מותאמת היטב עבור אפליקציות יחסית קלות. במידה ונדרש להרים שירות כבד עדיף שנעשה זאת מעל VM או לחלופין לפרקו ל-Micro-Services שכל אחד מהם קל יותר.
* **הרצת אפליקציות המצריכות מערכות הפעלה שונות.**משום ש-Docker משתף את ה-Kernel בין כל ה-Containers, אין אפשרות להריץ בהם אפליקציות שדורשות Kernel שונה (לדוגמה – לא נוכל להשתמש ב-Docker כדי להריץ במקביל אפליקציה ל-Windows ואפליקציה ל-Linux).

מושגים בסיסיים אחרונים

* **Image** – תבנית המכילה את האפליקציה וכל התלויות הנדרשות על מנת להריץ אותה. מתבנית זו יוצרים Containers – למעשה, Containers הם אינסטנסיאציה של ה-Image.Image נבנית מקובץ הגדרות מרכזי בשם **.Dockerfile**
* **Registry –** שירות "ספריית Images". מאפשר שליפה, אחסון ותיוג גרסאות של Images.
* **DockerHub –** שירות Registry שמציעה Docker. פועל בדומה לגיט באמצעות Repositories המכילים Images. ניתן לשמור ולשתף את ה-Images של אנשים פרטיים, למשוך Images בנויים מראש של Docker וחברות אחרות, ואפילו תמיכה ב-Webhooks (הרצת שרשרת פעולות בעת שה-Repository מתעדכן, כדי לאחד בין Docker Hub לשירותים אחרים).

סדנה מעשית

**תרגיל #0 – בואו נתחיל – תרגול פקודות בסיסיות**

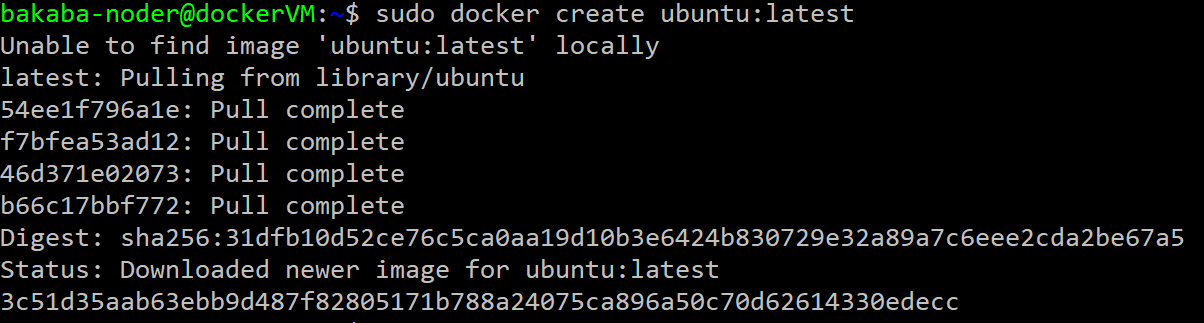
הפקודה הראשונה שנלמד היא docker create שמשמשת אותנו ליצירת Container חדש על בסיס Image. המבנה שלה:

docker create [options] IMAGE [commands] [arguments]

הגיע הרגע – הפקודה הראשונה שלך עם Docker. הפקודה הבאה תיצור Container ubuntu ריק:

docker create ubuntu:latest

ברכות – יצרת את ה-Container הראשון שלך! ה-Shell אמור להראות כך:



ביקשנו ליצור Container עם Image בשם ubuntu(בגרסה האחרונה שזמינה). מכיוון והוא לא היה קיים לוקאלית, Docker פנה ל-Docker Hub והוריד את ה-Image. בשורה האחרונה קיבלנו פלט ארוך – זהו ה-ID (מזהה) של ה-Container שיצרנו.

* הערה – יתכן ותצטרכו להשתמש ב-sudo.

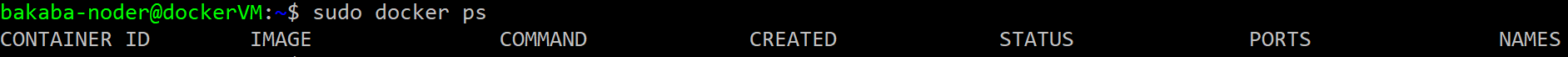
איך נדע שבאמת יצרנו Container? נשתמש בפקודה הבאה:

docker ps

פקודה זו מראה את כל ה-Containers שרצים כרגע ומציינת עבורם:

* ID: המזהה הייחודי של כל Container.
* Image: מאיזה Image נוצר ה- Container.
* Command: אילו פקודות רצות בעת הרמת ה-Container.
* Created: הזמן שעבר מאז שה-Container נוצר לראשונה.
* Status: המצב הנוכחי של ה-Container בצירוף הזמן שהוא מצא בו. במידה וה-Container רץ, יוצג כאן לדוגמה Up About an hour. אם הוא נעצר, יוצג Exited, קוד סטאטוס היציאה. לדוגמה Exited(0) 3 weeks ago.
* Ports: על איזה פורטים ה-Container יושב.
* Names: כדי לא להשתמש ב-ID ארוך, אפשר להשתמש בשם, שגם הוא צריך להיות ייחודי לכל Container.

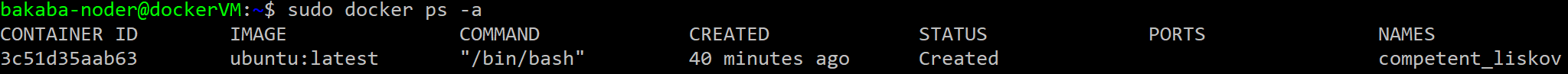
כרגע נקבל את הפלט:



הסיבה לכך היא שלא הרצנו את ה-Container שלנו. כדי לראות את כלל ה-Containers (גם אלה שלא רצים) נשתמש ב-

docker ps -a

עכשיו נראה את ה-Container:



כדי להריץ Container נשתמש ב-

docker start [options] CONTAINER ID/NAME [CONTAINER ID/NAME…]

לדוגמה:

docker start competent\_liskov

באופן דומה נוכל לעצור את פעולת ה-Container עם פקודת stop:

docker stop [options] CONTAINER ID/NAME [CONTAINER ID/NAME…]

כרגע אין טעם להריץ אותם על ה-Container שלנו כי הוא ריק – לא יהיו לו פקודות לבצע ולכן הוא יצא מיד. אתם מוזמנים לנסות את פקודת Start ולראות כיצד הסטאטוס של ה-Container משתנה מ-Created ל-Exited.

לשם נוחות, יש לנו פקודה נוספת:

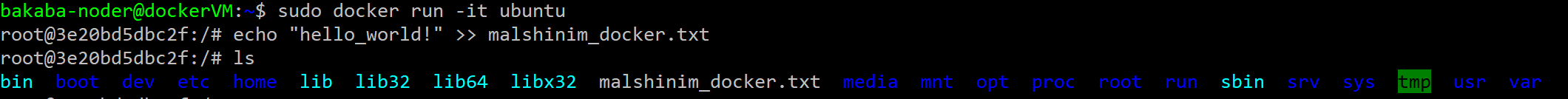
docker run [options] IMAGE [commands] [arguments]

הפקודה יוצרת ומיד מריצה את ה-Container (בעצם שילוב של create ו-start)

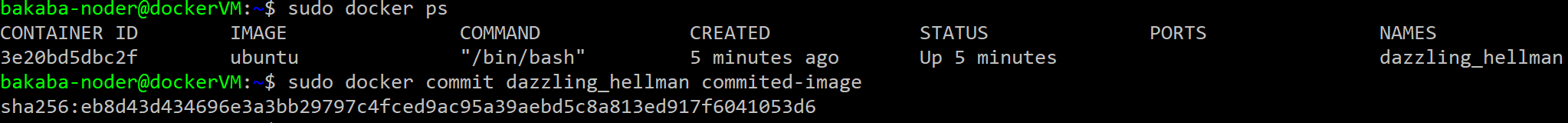
נסו את הפקודה הבאה –

docker run -it ubuntu

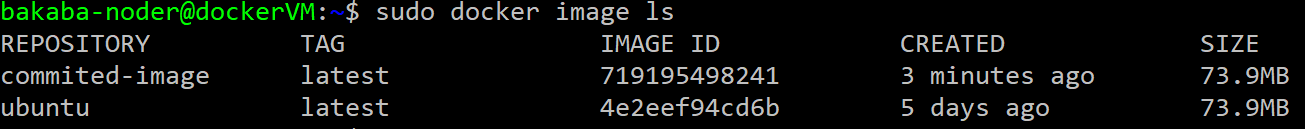
-it יפתח לנו טרמינל מיד כשנריץ את ה-Container וכך נוכל להריץ עליו פקודות במקום שיצא לנו מיד. בדוגמה שלפניכם יצרנו קובץ אך באותה המידה זו יכולה להיות התקנה של אפליקציה כלשהי שנרצה להריץ על גבי ה-Container.



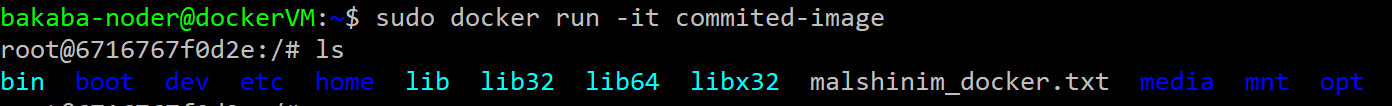
עד כאן זה די פשוט אבל גם מעפן – ברור שלא נרים את סביבת האפליקציה שלנו כל פעם ידנית. הדרך הכי פשוטה (ולא מומלצת – על כך בהמשך) היא באמצעות docker commit. פקודה זו "מקליטה" את כל השינויים שנעשו ב-Container מה-Image המקורי, ויוצרת מהם Image חדש. כך נקבל Image שמיד בעת הפעלתו יהיה מוכן עם הסביבה להרצת האפליקציה. עקבו אחר הפעולות הבאות ב**טרמינל חדש** (אל תצאו מה-Container שהרצנו):



נוודא שאכן יש לנו את ה-Image החדש:



ולבסוף נשתמש ב-run:



אפשר לראות שהקובץ שיצרנו ב-Container הקודם נוצר אוטומטית ב-Container החדש!

**תרגיל #1 – שימוש ב-Dockerfile ו-Docker Hub**

טוב, עד עכשיו מעולה, אבל לא ככה עובדים! מה אם נרצה לשכתב את ה-Image החדש שעשינו עם מיליון התקנות?

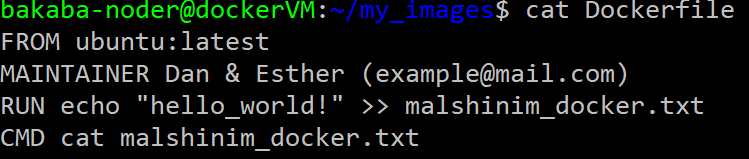
הדרך הנכונה היא לתחזק Dockerfile – קובץ המכיל רצף של פקודות כיצד לבנות את ה-Image שאתם צריכים.

צרו תיקיה בשם my\_images, ובתוכה כתבו קובץ בשם Dockerfile עם התוכן הבא:

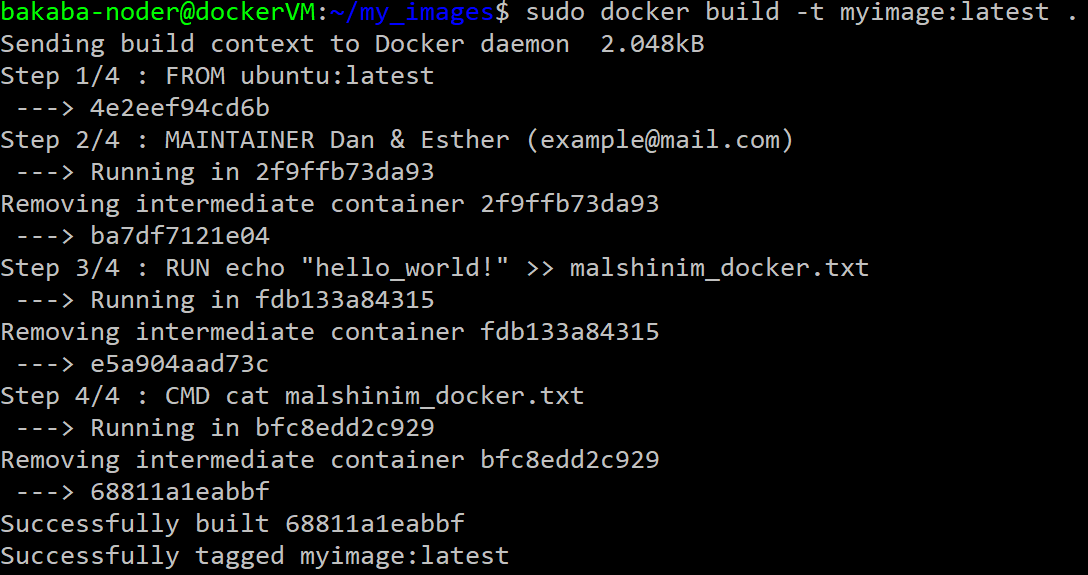
FROM ubuntu:latest  
MAINTAINER Dan & Esther (example@mail.com)  
RUN echo "hello\_world!" >> malshinim\_docker.txt  
CMD cat malshinim\_docker.txt

משמעות הפקודות שורה אחר שורה:

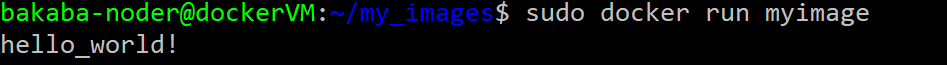
FROM – מה ה-Image הבסיסי על גביו נעשה שינויים.  
MAINTAINER – מי כתב את ה-Dockerfile. Good Practice.  
RUN – הרצת פקודות shell. משמש בעיקר להתקנות.  
CMD – פקודות שירוצו בעת הרמת ה-Container (אך ידרסו אם נשתמש באופציה -it).



כעת נשתמש ב- docker build בכדי ליצור Image מה-Dockerfile:



תוכלו להשתמש ב-docker image ls כדי לוודא שה-image החדש אכן ברשימה.  
כעת אם נריץ docker run:

  
הקובץ אכן נוצר, משום שפקודת cat הצליחה לקרוא ממנו. הצלחנו ליצור את אותו Image עם הרבה פחות עבודה! מעבר לכך, אם נרצה להוסיף תלות נוספת לסביבה שלנו – נוכל להוסיף שורה ל-Dockerfile שיצרנו ויצור את ה-Image החדש בקלות! 😊

* ישנן פקודות נוספות שכדאי לחקור – ENTRYPOINT, ENV. בגדול ENV מאפשרת הגדרת משתני סביבה אוטומטית, ו-ENTRYPOINT משמשת להרצת Container כקובץ הרצה (זאת אומרת שמאפשרת קבלת ארגומנטים בעת הרצת ה-Container, **ולא** נדרסת על ידי -it).

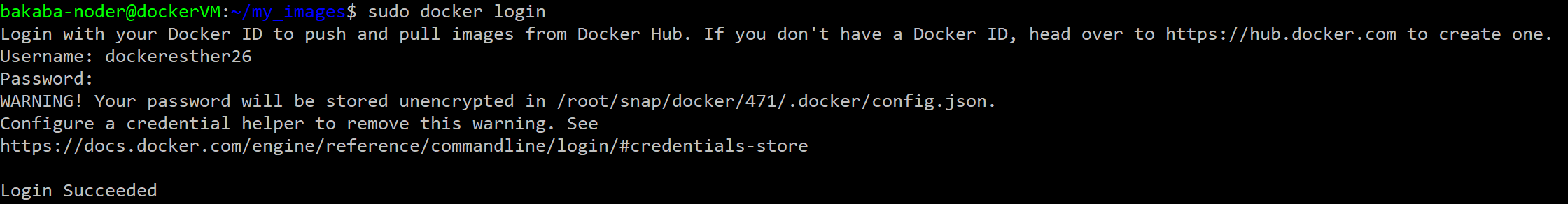
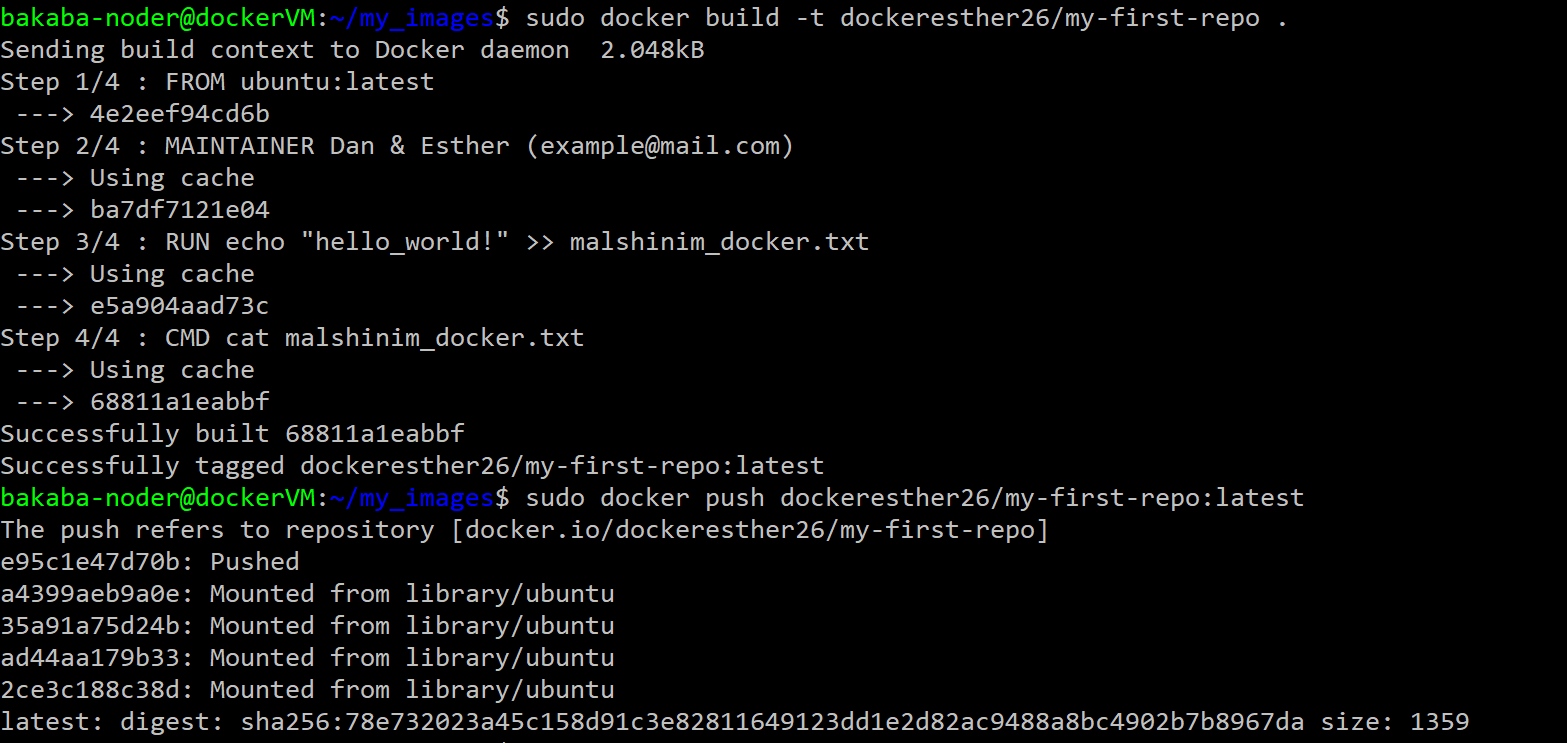
אתגרו את עצמכם על ידי עריכת ה-Dockerfile כך ש:

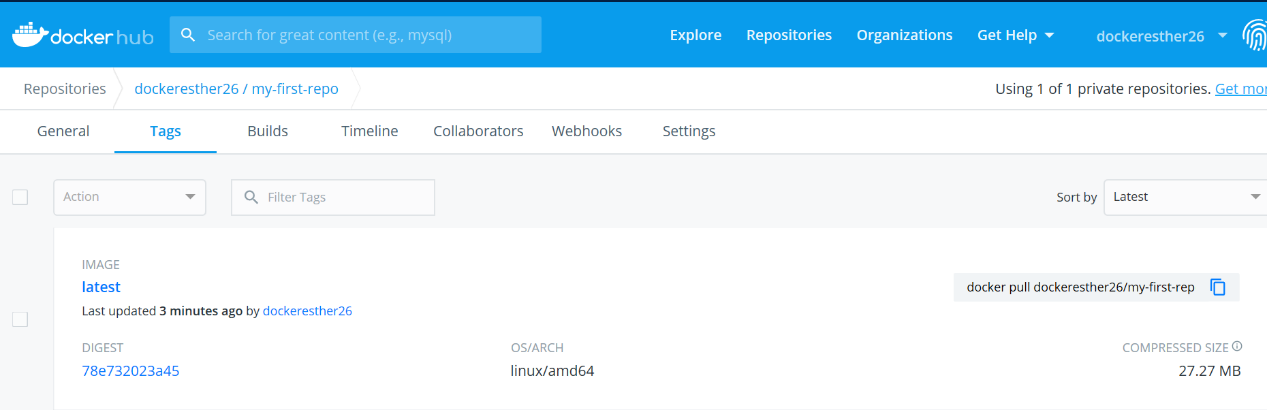
1. יגדיר משתנה סביבה וידפיס אותו.
2. יקבל כקלט שם של קובץ להדפסה בעת הרמת ה-Container וידפיס את תוכן הקובץ.

נסו להסתכל כאן אם אתם מסתבכים: <https://rominirani.com/docker-tutorial-series-writing-a-dockerfile-ce5746617cd>

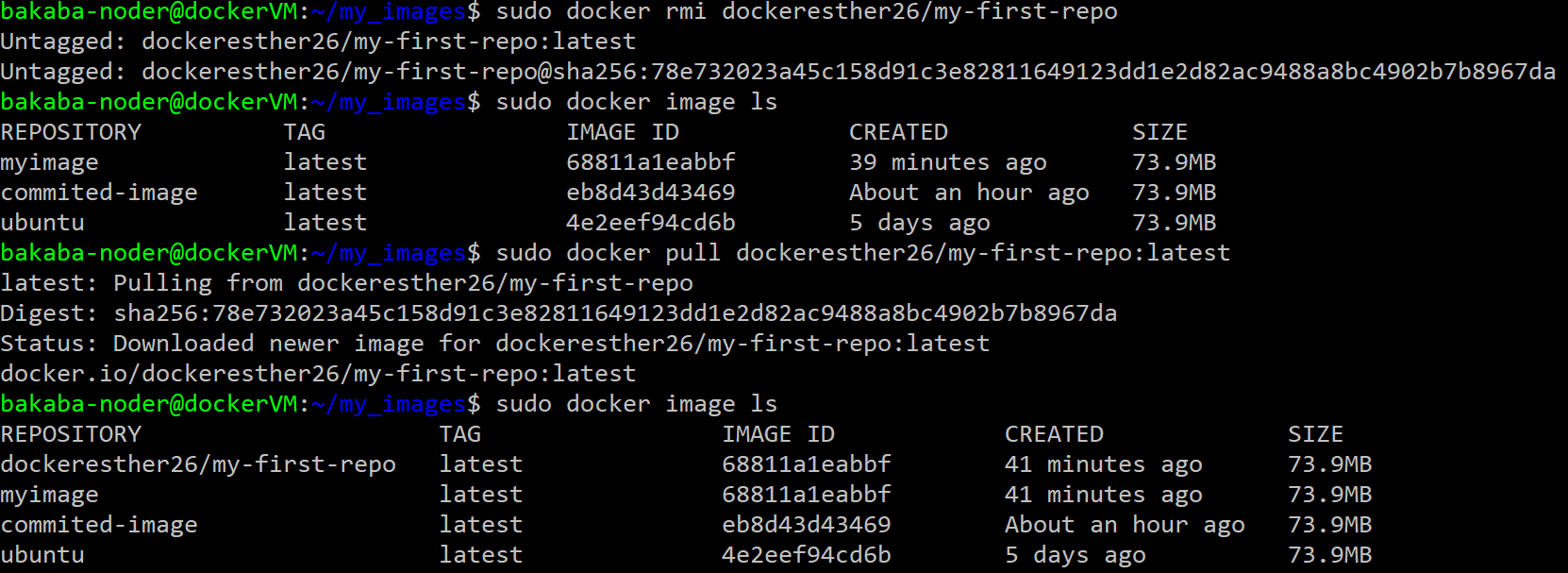
* **שווה מאוד** להציץ במדריך להרצת שרת Node.js (קישור בסוף המסמך). יש שם שימוש פרקטי ב-Container ומעבר לכך שימוש בפקודות חשובות שלא הזכרנו כמו COPY להעתקת קבצים ל-Image, ו-EXPOSE כדיי לפתוח port ב-Container.

הנושא בוא נסיים הוא Docker Hub. על מנת לשתף, לשמור ולערוך בקרת תצורה על ה-Image שלנו, נעלה אותו ל-Repository שלנו ב-Docker Hub.

נתחבר לחשבון ה-Docker Hub שלנו:  
  
לאחר מכן נבנה את ה-Image ונדחוף אותו ל-repo שלנו:  


אם נסתכל בחשבון שלנו נראה שה-Image אכן נדחף!  


לסיום, נלמד כיצד למשוך את ה-Image שלנו מ-Docker Hub:

  
בדוגמה שלפניכם מחקנו את ה-image שיצרנו, בדקנו זאת על ידי image ls ולבסוף משכנו אותו חזרה מ-Docker Hub.

מקורות מידע נוספים

כאן תבוא רשימה של חומרי עזר נוספים ללמידת הטכנולוגיה

* מדריך מפורט למתחילים ב-Docker: <https://rominirani.com/docker-tutorial-series-a7e6ff90a023>
* חשוב – הרמת אפליקציית Node.js על Container - <https://nodejs.org/de/docs/guides/nodejs-docker-webapp/>