DevOps – Final Task

מגישים:

כפיר אמויאל 205710411

יונתן שרר 318317682

# Introduction

1. התוצאה שקיבלנו לאחר הקלדת pwd בטרמינל ב UBUNTU:
2. 
3. פקודת pwd – **P**rint **W**orking **D**irectory מציגה את הנתיב האבסולוטי מהתיקייה הנוכחית בה אנו נמצאים.

# 2. Linux

שקף 10:

1. כפי שניתן לראות על פי פקודת pwd ה user name שלי הוא kfiros94



1. ה Machine name שלי הוא ubuntoKfiros94

A screen shot of a computer

AI-generated content may be incorrect.

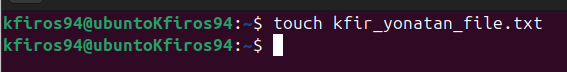
1. לפי צילום המסך בסעיף הקודם ה current directory הוא /home/kfiros94

שקף 26:

בשקף זה אנו מתבקשים ליצור קובץ חדש באמצעות פקודת touch.

יצרנו קובץ טקסט בשם kfir\_yonatan\_file.txt

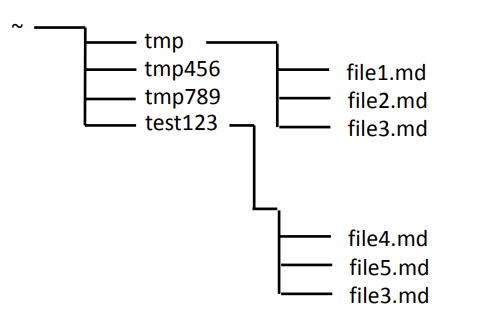
צילום מסך:



A white square with black text

AI-generated content may be incorrect.

לאחר מכן יש class exercise אנו מתבקשים ליצור tree structure לפי התרשים הבא:



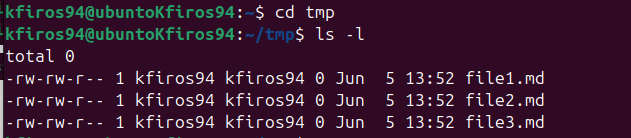
1. יצירת התיקיות באמצעות פקודת mkdir:



1. הוספת קבצים לתיקיית tmp באמצעות פקודת touch:



ואפשר גם לנווט דרך הטרמינל לתיקיית tmp כדי לראות את התוכן (הקבצים החדשים שיצרנו):



השתמשנו בפקודת ls -l כדי לראות את תוכן התיקייה, ואכן הקבצים נוצרו

-l זה דגל כדי לראות את התוכן ב long format כלומר להציג את התוצאה בפורמט מפורט (הרשאות, מספר קישורים, בעלים, קבוצה, גודל קובץ, תאריך שינוי אחרון ושמות הקבצים)

1. נוסיף כעת קבצים לתיקיית test123:



נבדוק הפעם שהקבצים נוספו דרך הUI:

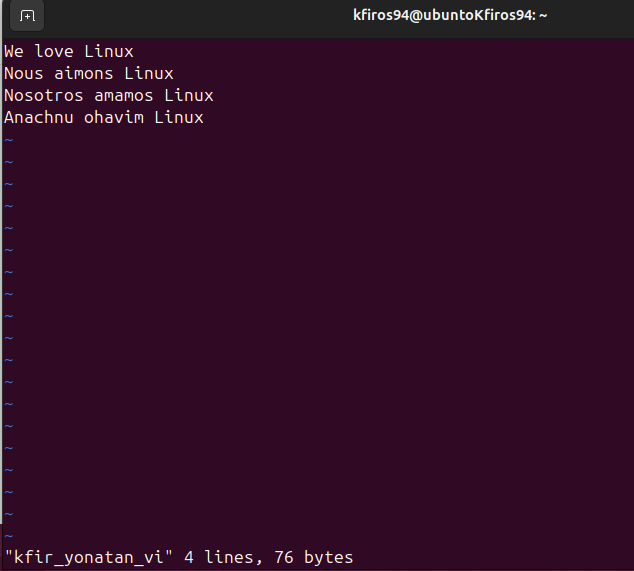
A screenshot of a computer

AI-generated content may be incorrect.

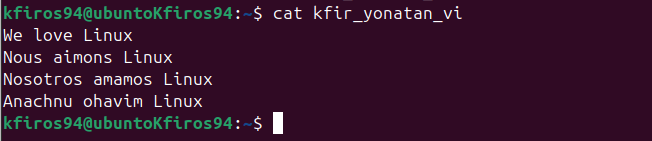
שקף 35:

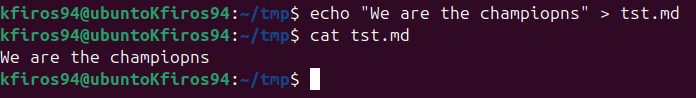
כעת נעשה שימוש בפקודות cat, man ונעשה גם את ה class exercise.

ראשית, ניצור קובץ בשם kfir\_yonatan\_vi ונשתמש ב vi כדי לכתוב תוכן לתוך הקובץ:



כעת נשתמש בפקודת cat כדי לראות את תוכן הקובץ:



נראה גם שימוש בפקודת echo בקובץ test.md שיצרנו שנמצא בתיקיית tmp:  


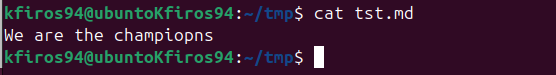
נסביר כעת שימוש בפקודה man עבור הפקודה ls:

הפקודה man = manual פותחת את ספר ההוראות של פקודה מסוימת.

לכן, man ls זה המדריך המלא לשימוש ב ls המציג אילו אפשרויות flags קיימות (למשל -a, -l, -h וכו'..).

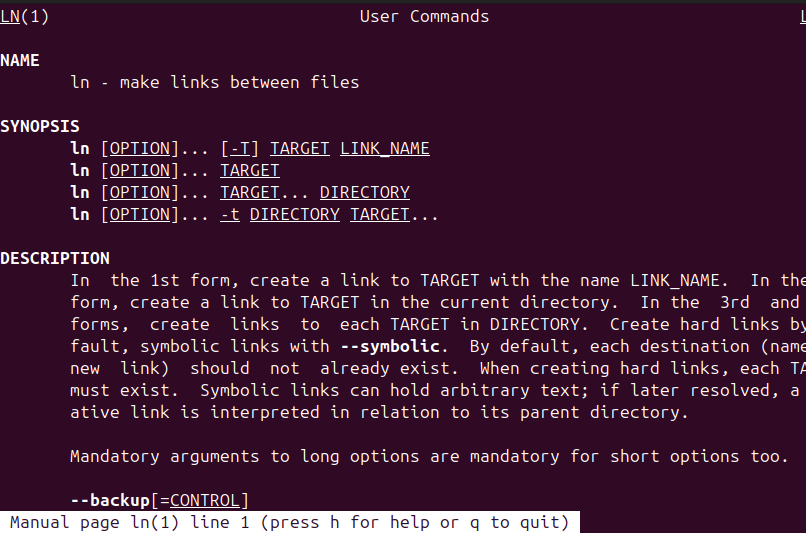
תשובות ל class Exercise:

1. שימוש בפקודה cat:



אנו רואים את התצוגה של tst.md שיצרנו מקודם.

1. הדפסת דוקומנטציה של פקודות ln ו mv (זה ארוך אז נראה רק חלק מהצילום מסך):



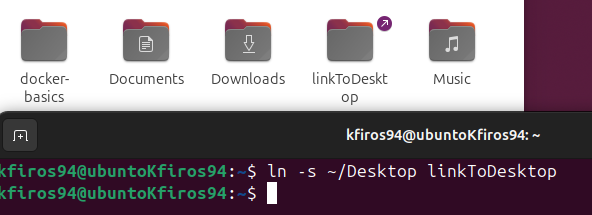
A computer screen with white text

AI-generated content may be incorrect.

1. ניצור soft link ל desktop:

ב Linux קוראים לזה symbolic link

והפקודה שנשתמש בה היא ln -s



Ln היא הפקודה ליצירת לינק

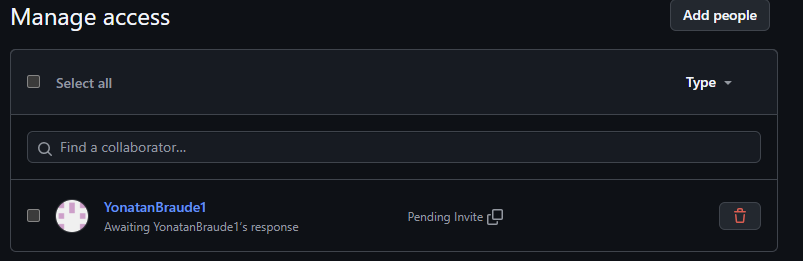
-s אומר לעשות לינק סימבולי

וכל מה שבא אחר כך זה היעד והשם של הקובץ.

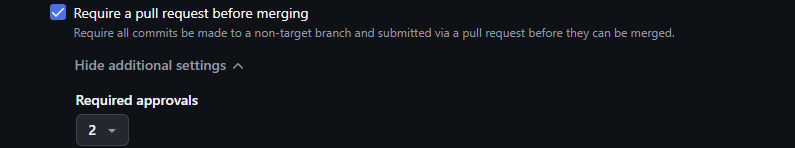
# Git

נבצע את המשימה על ידי כך שנבנה אתר html פשוט, ונשתמש ב repository של המטלה שלנו.

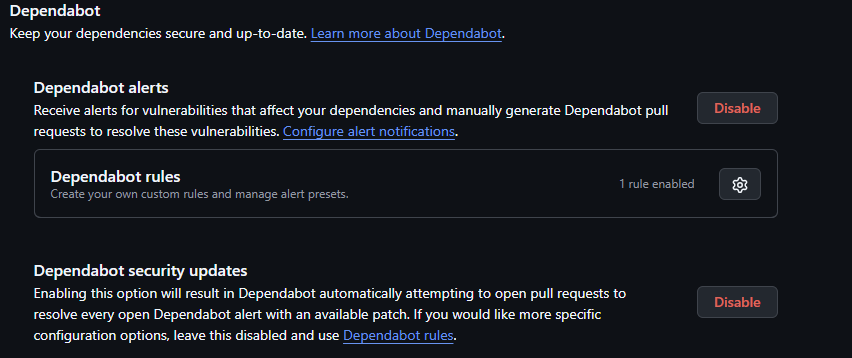
הוספתי את יונתן כ collaborator לריפוזיטורי שלנו:



תחת branches אני יוצר ruleset חדש על מנת לבקש pull request ולקבל לפחות 2 אישורים לפני merging ל main branch.



כעת, תחת security נאפשר את **Dependabot:**

****

הוא סורק באופן אוטומטי את התלויות שלך ויוצר בקשות משיכה (pull requests) כאשר מתגלה גרסה פגיעה או מיושנת, ובכך שומר על הפרויקט מאובטח ומעודכן ללא צורך במאמץ ידני.

1. למה אנחנו צריכים אותו?

* בפרויקטים מודרניים יש המון תלויות (dependencies) — ספריות קוד פתוח, פריימוורקים, כלים.
* לעיתים קרובות, ספריות אלו מתעדכנות: מתקנים בהן פרצות אבטחה, באגים, או מוסיפים שיפורים.
* אם אנחנו לא עוקבים אחרי העדכונים, הפרויקט שלנו עלול לכלול פרצות או לעבוד עם קוד לא יציב.
* Dependabot חוסך לנו את העבודה הידנית של לעקוב אחרי עדכונים. הוא בודק את ה-repository שלנו באופן קבוע ומעדכן אותנו.

1. מתי משתמשים בו?

* בפרויקטים תוכנה שמכילים קבצי ניהול תלות.
* כאשר אנחנו רוצים לשמור על עדכניות ואבטחה בלי צורך לבדוק הכל ידנית.
* מתאים מאוד לפרויקטים ב**-**GitHub, שם הוא משולב מעולה.

1. למה כל כך חשוב להשתמש בו?

* **אבטחה** - מזהה ספריות עם פגיעויות ומציע תיקונים אוטומטיים.
* **תחזוקה** - שומר את הפרויקט מעודכן, מונע "ריקבון תוכנה" (software rot)
* **חיסכון בזמן** - לא צריך לבדוק ידנית אם יצאו עדכונים
* **אוטומציה -** פותח Pull Requests מוכנים עם עדכונים - נשאר רק לבדוק ולמזג.

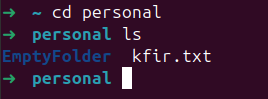
# Ansible

במשימה זו התבקשנו לכתוב סקריפט בלינוקס אשר יעתיק את תוכן תיקיית personal לתיקיית גיבוי ששמה backup\_archive<date> כלומר השם של התיקייה יכיל גם את חותמת הזמן הנוכחית.

**קובץ הקוד מצורף לגיט.**

צילומי מסך:

תוכן התיקייה אותו אני רוצה לגבות:



הרצת הסקריפט:

A computer screen with white text

AI-generated content may be incorrect.

בדיקה שזה אכן גובה:

A screenshot of a computer code

AI-generated content may be incorrect.

# Docker

נבצע את המשימה הראשונה המופיעה בשקף 29:

אנו משתמשים במשימה זו ב play-with-Docker, יצרנו instance חדש והקלדנו בטרמינל:

docker run -dp 80:80 docker/getting-started:pwd

נפרק את הפקודה הזו ונסביר שלב שלב:

Docker run – זוהי הפקודה שמבקשת להריץ קונטיינר חדש מתוך image.

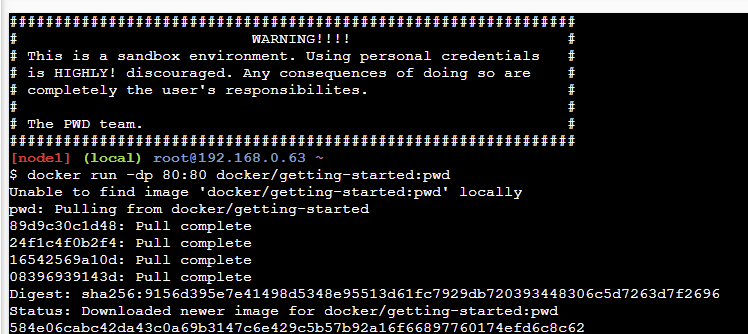
-d זה דגל קיצור של –detach כלומר מריץ את הקונטיינר במצב "מנותק", כלומר ברקע.

P 80:80 קיצור של publish – עושה מיפוי פורטים, הפורט השמאלי הוא הפורט במחשב שלנו (ה host)   
והחלק הימני הוא הפורט בתוך הקונטיינר.

כלומר, כל פנייה ל <http://localhost:80> תעבור לקונטיינר, לפורט 80 שלו.

Docker/getting-started הוא השם של ה image הרשמי כלומר דוקר למתחילים

:pwd זה הגרסה של אותו image.



זוהי תמונה מתוך ה play with docker

כעת, נריץ את הפקודה docker ps על מנת לראות את רשימת הקונטיינרים שרצים כרגע במערכת.

קיבלנו:

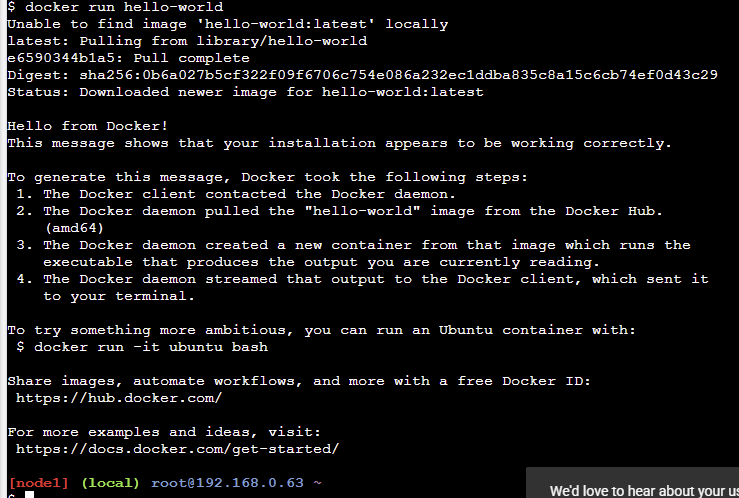


כעת נריץ את הפקודה docker run hello-world:

פקודה זו למעשה היא אחת מהפקודות הכי בסיסיות לבדיקה שה-docker מותקן ועובד כמו שצריך

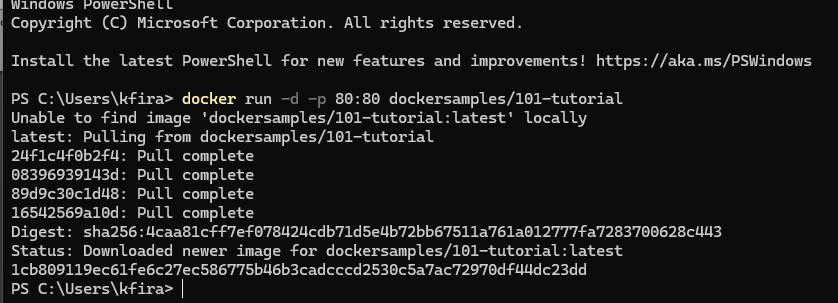
היא מבקשת להריץ קונטיינר מ image בשם hello-world.

הפלט שקיבלנו:

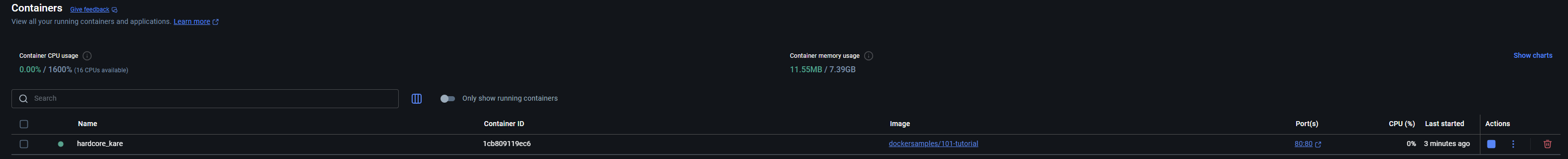


בשקף 33 התבקשנו לעשות דבר דומה עם docker Desktop.

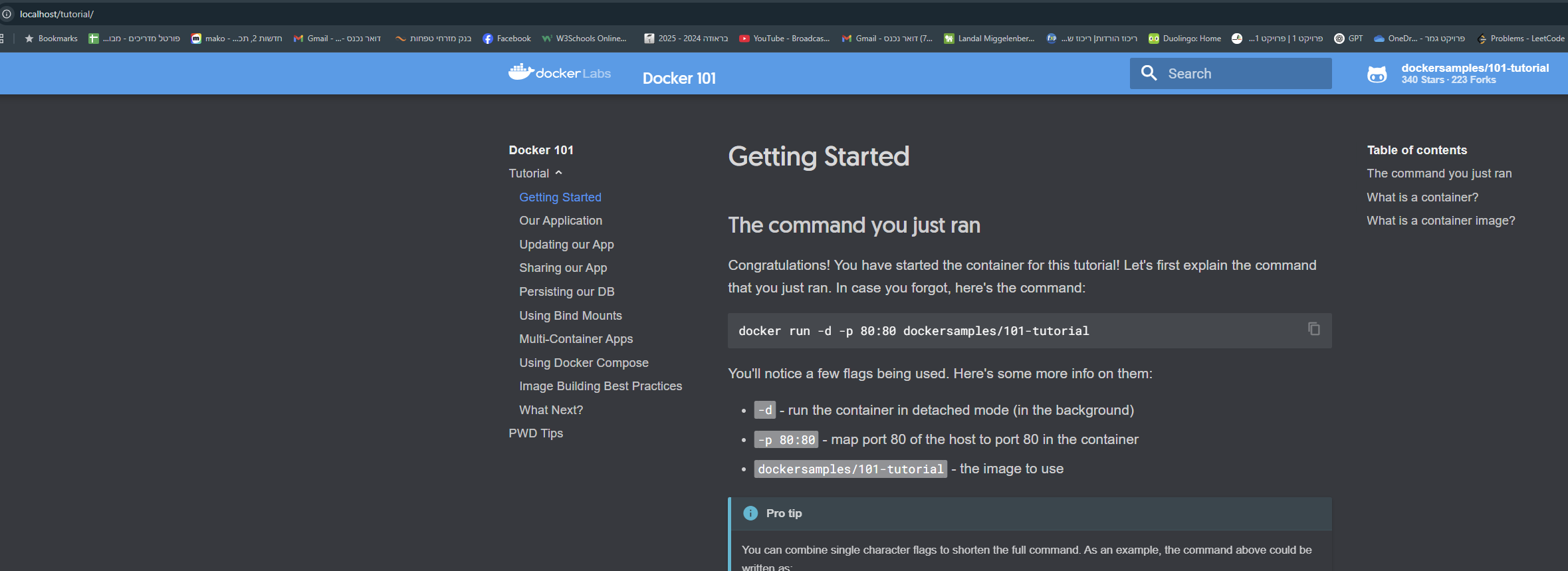
כתבנו ב PowerShell:



ואכן ניתן לראות ב Docker Desktop שהקונטיינר רץ:



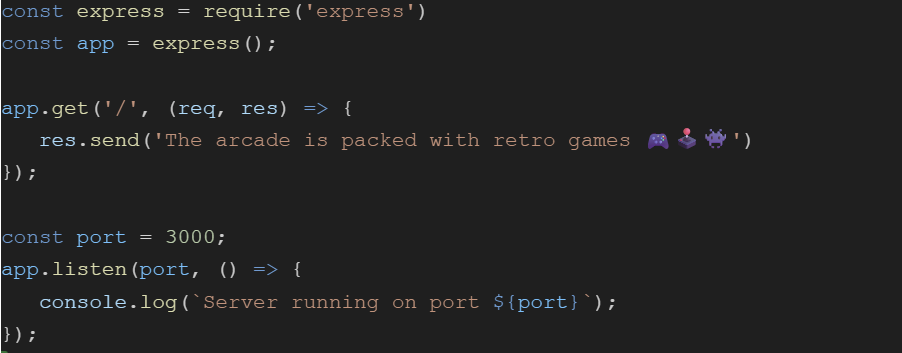
כעת, כאשר ה container רץ ניתן לכתוב בדפדפן <http://localhost/> ולראות:



שקף 45:

תיקיית task5\_docker-basics מצורפת ל repository שלנו.

הסברים + צילומי מסך:

בנינו אפליקציית שרת node והקוד שלו נראה כך:  


נרצה לקחת את האפליקציה הקטנה הזאת ולדחוף אותה לתוך קונטיינר.

נעבור שלושה שלבים:

1. נבנה Dockerfile זה תמיד יהיה השם של הקובץ והוא יתחיל בפקודה FROM.

נכתוב FROM node:latest כדי לקבל את הגרסה האחרונה של node.

Node הוא למעשה base image כלשהו שאנו מביאים בתור תלות, ממש כמו יבוא ספרייה.

זה אומר שהקונטיינר הולך להריץ אפליקציה שמבוססת על node ובמקרה שלנו את הגרסה האחרונה של node. אנו מביאים את ה base image של node מהמאגר הרשמי של docker images שנקרא Docker Hub.

אנחנו רוצים גם להגדיר איזשהו working directory בתוך ה image לכן נכתוב:

WORKDIR /app

ואז כדי להעתיק את הקבצים שלי לתוך ה image נכתוב COPY . . .

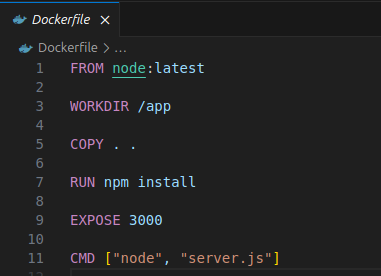
כדי לא להעתיק את כל ה dependencies נמחק את node modules מתוך התקייה שלנו או להוסיף קובץ .dockerignore ולהוסיף שם node\_modules.

ונוסיף ב DOCKER FILE :

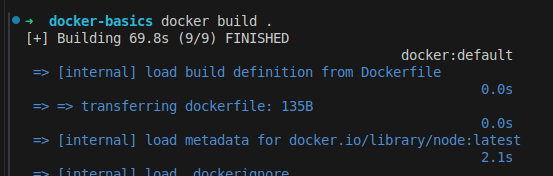
RUN npm install.

כדי להתחיל את השרת נכתוב CMD [“node”, “server.js”] הפקודה הזו למעשה תרוץ בתוך הקונטיינר.

להלן צילום מסך של dockerfile:



1. נבנה את ה image על ידי פקודה בטרמינל: docker build .



אם נכתוב עכשיו בטרמינל docker images נוכל לראות את ה image :



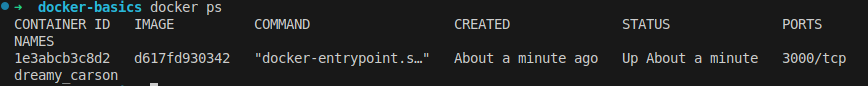
לכל image יש IMAGE ID שנשתמש בו כדי להרים את הקונטיינר.

1. כעת נרים את הקונטיינר, נרים את האפליקציה מתוך בועה שכבר מותקן עליה node.

נחזור לטרמינל ונכתוב docker run <IMAGE ID> כאשר המזהה של ה image הוא מה שראינו מקודם.



על ידי כתיבת פקודה docker ps בטרמינל כדי לראות את כל הקונטיינרים שרצים עכשיו:



הקונטיינר באוויר ומה שנשאר לנו זה לחשוף פורט לקונטיינר.



ואם נלך ל local host 3001 נראה שהאפליקציה באוויר:



# Jenkins

נעשה כעת את Class Exercise 1 שבשקף 20:

נריץ את Jenkins בפורט 8080:

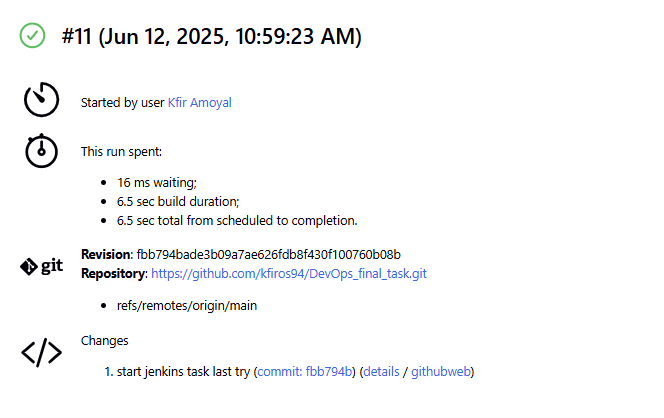
A screenshot of a computer

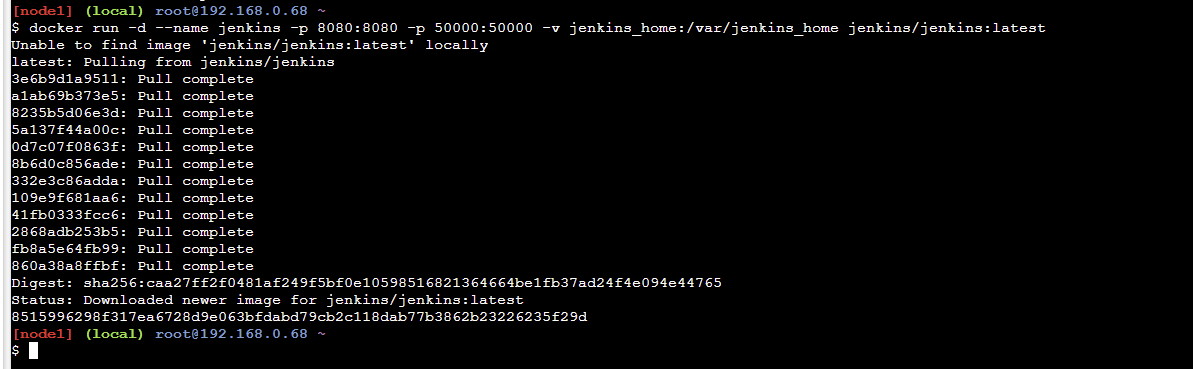
AI-generated content may be incorrect.

נכנס ל New item -> pipeline ונקרא לו CI-Exercise1:

נקשר אותו ל main branch שלנו ונדחוף Jenkinsfile עם הסקריפט המתאים (הקובץ מצורף בזיפ וגם בריפוזיטורי)

עשינו build וקיבלנו הצלחה



ב PLAY WITH DOCKER כתבנו כדי להריץ ג'נקינס בקונטיינר :  


# Kubernetes

נבצע כעת את המשימה בשקף 31:

ניצור Master Node:

על ידי הפקודה

kubeadm init --apiserver-advertise-address $(hostname -i) \

--pod-network-cidr 10.5.0.0/16

לאחר מכן מתי שזה יסתיים נעתיק את מה שזה הדפיס:

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

נריץ את kubectl

kubectl apply -f <https://raw.githubusercontent.com/cloudnativelabs/kube-router/master/daemonset/kubeadm-kuberouter.yaml>

מה output של המאסטר, נעתיק את כל kubeadm join..

לכל worker:

kubeadm join <master-IP>:6443 --token <token> \

--discovery-token-ca-cert-hash sha256:<hash>

נוודא שנוצרו לנו מאסטר ו-3 workers על ידי פקודה

kubectl get nodes

שקופית 50-51

נריץ קונטיינר ב pod

# 0. Nothing running yet

kubectl get pods

# 1. Create an nginx deployment

kubectl create deployment nginx --image=nginx

# 2. Scale to 5 replicas

kubectl scale deployment nginx --replicas=5

# 3. Check the pods

kubectl get pods

``` :contentReference[oaicite:13]{index=13}

You now have five identical \*\*nginx\*\* Pods spread across your three workers – proof the cluster is functioning.

> \*\*Optional – expose and visit it\*\*

> ```bash

> kubectl expose deployment nginx --type=NodePort --port=80

> kubectl get svc nginx

> ```

> Copy the `NODE-PORT` (e.g. `30123`) and open `http://<any-worker-IP>:<NODE-PORT>` in your browser.

---

### Recap

1. \*\*PWK master\*\* → `kubeadm init` → copy kubeconfig → install CNI.

2. \*\*Each PWK worker\*\* → paste `kubeadm join` once.

3. `kubectl get nodes` shows 4 nodes.

4. Deploy & scale nginx to 5 Pods.

That completes the \*cluster creation\* exercise \*\*and\*\* slides 50-51.

Ping me when you’re ready for the next slide set (Services, Ingress, etc.) or if anything in PWK misbehaves!

::contentReference[oaicite:7]{index=7}