יעד ראשון עם דוקר

23/04/2019 נושאים: יומי, docker

דוקר (Docker) הוא אחד הכלים השימושיים ביותר למפתחים של פרויקטים מכל הגדלים בשלבי הפיתוח, הבדיקות וה Production. בואו ניקח צעד ראשון יחד כדי להכיר את הכלי ולראות אם שווה לשלב אותו גם בפרויקט שלכם.

תוכן עניינים

1. מה זה דוקר
2. התקנה והפעלת קונטיינרים
3. פיתוח קונטיינר ראשון שלי
4. לאן עכשיו?

1. מה זה דוקר

דוקר הוא פלטפורמת תוכנה שמאפשרת לבנות, לשתף ולהריץ יישומים בתוך מיכלים (Containers).

בשביל להבין טוב יותר למה צריך פלטפורמה כזאת בואו נניח שיש לכם מערכת שמורכבת ממספר רכיבים: בסיס נתונים, שרת ווב, שרת יישום, סרביסים קטנים שעוזרים ליישום לרוץ ועוד.

בלי דוקר אתם עלולים להיתקל במספר בעיות בעבודה עם מערכת כזאת. קודם כל בסביבת הפיתוח:

1. יכול להיות שתרצו לנסות לפתח גירסא חדשה של המערכת מול גירסא חדשה יותר של בסיס הנתונים, אבל עדיין לשמור על בסיס הנתונים הישן מותקן על המחשב כדי לתקן באגים בגירסת הפרודקשן.

2. ובעצם אותה הבעיה קיימת עם כל רכיב תשתית שבחרתם - שידרוג של השרת, או של גירסת שפת התכנות או ספריות מסביב. כל אלה יכולים ליצור התנגשויות כשמנסים לשלב מספר גירסאות על אותה מכונת פיתוח.

3. כשנכנס מפתח חדש לצוות הוא עלול למצוא את עצמו מבלה יום שלם בהתקנות. התקנת כל החלקים והתשתיות של המערכת על מכונה חדשה נהיית מורכבת יותר ככל שיש יותר רכיבים.

ובאותו האופן בשלב המעבר לייצור אתם עשויים להיתקל במספר בעיות:

1. קושי להוסיף או להוריד "שרתים" מהרשת - מאחר וכל שרת צריך להחזיק את כל רכיבי התשתית כדי להריץ את המערכת.

2. בעיית ניצול של השרתים הקיימים בגלל התקנות גלובליות - יכול להיות שהשרת שלכם מספיק "פנוי" רוב הזמן כדי להריץ עוד רכיב במערכת, אבל הרכיב הזה דורש גירסא אחרת של בסיס הנתונים ואתם לא רוצים ליצור התנגשות.

דוקר יודע לפתור חלק גדול מהבעיות האלה באמצעות שימוש במיכלים. כל מיכל מחזיק בתוכו אפליקציה אחת או יותר ודוקר עצמו יודע "להפעיל" מיכלים כאלה בתוך סביבה חצי מבודדת. זה לא מכונה וירטואלית נפרדת אלא משהו הרבה יותר קליל. התהליכים והקבצים נשמרים ישירות על המכונה הנוכחית, אבל בנפרד מהמערכת הראשית ובעצם אנחנו נותנים ליישום הרגשה שהוא רץ לבד, למרות שהוא בעצם רץ עם עוד הרבה יישומים אחרים.

חשוב לציין שבגלל שאין פה הפרדה מוחלטת בין המיכלים אם מיכל מסוים ירצה לשבור לכם את השרת או להאזין למיכלים אחרים הוא יוכל לעשות את זה. לכן חשוב לזכור שדוקר אינו מכונה וירטואלית וחשוב להריץ רק מיכלים שאתם סומכים עליהם.

2. התקנה והפעלת קונטיינרים

לפני שנתחיל לדבר על קונטיינרים צריך לדבר על Image-ים. אימג' היא חבילה שכוללת את כל מה שיישום צריך כדי לרוץ: הקוד, ספריות בהן הוא תלוי, הגדרות, משתני סביבה וכן הלאה. האימג' נשמר לכם על המחשב ואתם יכולים ליצור ממנו קונטיינר.

קונטיינר הוא הפעלה של אימג' מסוים.

אפשר לחשוב על הקשר בין השניים כמו הקשר בין "קובץ" ל"תהליך". האימג' מייצג קובץ על זק, וכשמפעילים את האימג' מקבלים "תהליך" בזיכרון. בגלל זה הפקודה שמראה את כל ווקונטיינרים שכרגע רצים לכם על המחשב היא:

```
$ docker ps
```

בדיוק כמו שהפקודה ps מראה את כל התהליכים.

בשביל שנוכל להתחיל לייצר קונטיינרים צריך תחילה להתקין את דוקר או ליתר דיוק את המנוע Docker Engine. מנוע זה הוא שמריץ את הקונטיינרים. אפשר לבחור התקנה למערכת ההפעלה שלכם מהדף הזה:

https://hub.docker.com/search/?type=edition&offering=community

((https://hub.docker.com/search/?type=edition&offering=community

אני משתמש במק אז בחרתי להוריד משם את Docker Desktop for Mac ולהתקין אותו. אתם יכולים לבחור את הגירסא שמתאימה לכם.

אחרי ההתקנה תרצו לוודא שדוקר עובד לכם כמו שצריך. כנסו ל CMD או ל Terminal וכתבו:

```
$ docker --version
Docker version 18.09.2, build 6247962
```

אפשרות נוספת היא להפעיל את הפקודה info ולקבל מידע הרבה יותר מפורט על הגירסא:

```
$ docker info
Containers: 13
 Running: 0
 Paused: 0
 Stopped: 13
Images: 8
Server Version: 18.09.2
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 9754871865f7fe2f4e74d43e2fc7ccd237edcbce
runc version: 09c8266bf2fcf9519a651b04ae54c967b9ab86ec
init version: fec3683
Security Options:
 seccomp
  Profile: default
Kernel Version: 4.9.125-linuxkit
Operating System: Docker for Mac
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 1.952GiB
Name: linuxkit-025000000001
ID: I5AI:DBZD:JRZJ:V5IJ:7ONX:IRKU:S32C:KRPM:AI3U:3A7A:3JYY:5EYQ
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): true
 File Descriptors: 24
 Goroutines: 50
 System Time: 2019-02-19T08:03:02.877296Z
 EventsListeners: 2
HTTP Proxy: gateway.docker.internal:3128
HTTPS Proxy: gateway.docker.internal:3129
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
 127.0.0.0/8
```

```
Live Restore Enabled: false
Product License: Community Engine
```

הפקודה `docker run` מחפשת אימג' ומריצה אותה. אני יודע שרק הרגע התקנתם את דוקר אז כם עדיין אף אימג' על המחשב, אבל תופתעו לשמוע שדוקר יודע לחפש ולהוריד אימג'ים באופן אוטומטי מתוך מאגר עצום של אימג'ים קהילתיים (עליו נדבר באחד הפוסטים הבאים בסידרה).

אנחנו נתחיל בהתקנה של האימג' מתוך הריפוזיטורי הזה:

https://github.com/docker-library/hello-world (https://github.com/docker-library/hello-world)

האימג' עצמו נבנה וזמין במאגר של דוקר שנקרא Dockerhub וזה הקישור אליו שם:

https://hub.docker.com/_/hello-world (https://hub.docker.com/_/hello-world)

יש בו תוכנית קטנה בשם hello שמדפיסה הודעת ברכה על המסך. הפקודה הבאה מורידה את האימג' מדוקר האב, יוצרת ממנו קונטיינר ומפעילה את הקונטיינר:

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:2557e3c07ed1e38f26e389462d03ed943586f744621577a99efb77324b0fe535
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

בואו נראה עכשיו איזה קונטיינרים ואיזה אימג'ים יש לנו. הפקודה הבאה מציגה את כל האימג'ים שהתקנתם על המחשב:

```
$ docker image  ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              fce289e99eb9        7 weeks ago         1.84kB
```

והפקודה הבאה מציגה את כל הקונטיינרים שיש לכם על המחשב:

```
$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
4cb8496372d3        hello-world         "/hello"            7 minutes ago       Exited (0) 3 min
73177b43ff6d        hello-world         "/hello"            7 minutes ago       Exited (0) 7 min
```

ה `-a` אומר לפקודה להוסיף גם את הקונטיינרים שעכשיו סגורים. שימו לב שאצלי יש שני קונטיינרים לאותו אימג' בגלל שהפעלתי את `docker run` פעמיים.

אנחנו יכולים להפעיל מחדש קונטיינר שנסגר עם הפקודה:

```
$ docker start -a 4cb8496372d3
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

שימו לב להחליף את 4cb8496372d3 במזהה הקונטיינר שמופיע אצלכם. הפעם ה `-a` אומר לפקודה שאנחנו רוצים שהפלט של הקונטיינר יופיע אצלנו על המסך.

הצגה חוזרת של רשימת הקונטיינרים מראה שפעולה זאת לא עשתה שום שינוי ולא יצרה קונטיינר חדש. בסך הכל הפעילה מחדש קונטיינר שנסגר:

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
4cb8496372d3        hello-world         "/hello"            9 minutes ago       Exited (0) 18 se
73177b43ff6d        hello-world         "/hello"            9 minutes ago       Exited (0) 9 min
```

לעומת זאת אם תפעילו שוב את `docker run` תיצרו קונטיינר חדש עבור האימג':

```
$ docker run hello-world
...
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
ff88c7e43750        hello-world         "/hello"            7 seconds ago       Exited (0) 6 s
onds ago                        zealous_khayyam
4cb8496372d3        hello-world         "/hello"            11 minutes ago      Exited (0) 2 min
utes ago                        reverent_sanderson
73177b43ff6d        hello-world         "/hello"            11 minutes ago      Exited (0) 11 mi
nutes ago                       focused_bose
```

לכן אפשר לחשוב על קונטיינר בתור משהו בין מכונה וירטואלית לתהליך - מצד אחד הוא רץ על המחשב שלכם כמו תהליך רגיל בלי תיווך המכונה הוירטואלית, אבל מצד שני הוא מנותק מהמחשב שלכם ויכול להכיל תוכנות ולעבוד על קבצים שיישמרו כולם בתוך הקונטיינר.

בצורה כזאת נוכל לפתור את הבעיות שדיברנו עליהן קודם:

1. אפשר להחזיק כל מספר של גירסאות או תוכנות על המכונה ולא תהיה התנגשות ביניהן כי כל תשתית רצה בתוך קונטיינר משלה.

2. קל להרים עוד מכונה או להוסיף מתכנת חדש לצוות - פשוט תנו להם את הקונטיינרים המתאימים ושם הם כבר ימצאו את כל מה שצריך.

3. קל לנתק את הקשר בין התוכנות שרצות לבין מספר וסוג השרתים שיש לנו, וכך לקבל ניצול טוב יותר של התשתית.

3. פיתוח קונטיינר ראשון שלי

בואו נכתוב קונטיינר ראשון משלנו שיפעיל יישום ווב ב Python. בשביל זה תצטרכו לפתוח תיקיה חדשה על המחשב (לא משנה איפה) ובתוכה ליצור את הקבצים הבאים:

הקובץ `Dockerfile` מגדיר את תוכן האימג' ואת ההוראות ליצירת קונטיינר חדש מתוכה. צרו קובץ בשם `Dockerfile` עם התוכן הבא:

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

קובץ ה Dockerfile מתיחס לקובץ בשם requirements.txt (הוא מריץ pip כדי להתקין את התלויות שנמצאות בקובץ זה). לכן ניצור את הקובץ `requirements.txt` שוב באותה תיקיה ועם התוכן הבא:

```
Flask
Redis
```

לסיום הקובץ `Dockerfile` מתיחס גם לקובץ בשם `app.py` (בשורה האחרונה שלו - יש הרצה של הקובץ `app.py`). לכן ניצור קובץ בשם `app.py` עם התוכן הבא:

```
from flask import Flask
```

```
from flask import Flask
from redis import Redis, RedisError
import os
import socket

# Connect to Redis
redis = Redis(host="redis", db=0, socket_connect_timeout=2, socket_timeout=2)

app = Flask(__name__)

@app.route("/")
def hello():
    try:
        visits = redis.incr("counter")
    except RedisError:
        visits = "<i>cannot connect to Redis, counter disabled</i>"

    html = "<h3>Hello {name}!</h3>" \
           "<b>Hostname:</b> {hostname}<br/>" \
           "<b>Visits:</b> {visits}"
    return html.format(name=os.getenv("NAME", "world"), hostname=socket.gethostname(), visits=visits)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80)
```

עכשיו שיש לנו את שלושת הקבצים בואו נבנה מהם אימג' באמצעות הפעלת הפקודה הבאה מאותה התיקיה:

```
$ docker build --tag=friendlyhello .
```

ונציג את האימג' החדש שלנו:

```
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
friendlyhello       latest              89995b3d4e62        10 seconds ago      131MB
python              2.7-slim            99079b24ed51        6 days ago          120MB
hello-world         latest              fce289e99eb9        7 weeks ago         1.84kB
```

דוקר הוריד את האימג' python בתור בסיס לאימג' שלי (כי כך היה כתוב בדוקרפייל), ונתן לאימג' שלי את השם friendlyhello שזה השם שבחרתי בעת הפעלת פקודת הבניה.

עכשיו אנחנו כמעט מוכנים להריץ את הקונטיינר - אתם זוכרים שקודם הרצנו את הקונטיינר עם:

```
$ docker run hello-world
```

וזה עבד. הפעם המצב קצת יותר מסובך. בנוסף להפעלת תוכנית אנחנו גם צריכים להעביר לקונטיינר מיפוי של הפורטים. שימו לב לשורה הבאה שהופיעה ב Dockerfile:

```
EXPOSE 80
```

זה אומר שפורט 80 של הקונטיינר יהיה זמין לעולם החיצון. אבל ברור שמי שכותב קונטיינר לא יודע איפה הקונטיינר הזה אמור לרוץ, ומה יהיו הקונטיינרים האחרים שירוצו על אותה מכונה. לכן כשאנחנו מריצים קונטיינר אנחנו צריכים לחבר בין אותו פורט 80 שיוצא מהקונטיינר לפורט לבחירתנו על המכונה שמריצה את הקונטיינר (במקרה הזה המחשב שלי). אני בוחר בפורט 4123 כי הוא פנוי.

בנוסף הקונטיינר מריץ יישום פייתון שימשיך לרוץ כל עוד הקונטיינר חי. מאחר ומדובר כאן ב Web Application זה אומר שהיישום ימשיך לרוץ לאורך זמן. בשביל שלא יתפוס לי את הטרמינל אני מעדיף להריץ את הקונטיינר ברקע - ובשביל זה אנחנו מוסיפים `-d` .

סך הכל פקודת הרצת הקונטיינר ברקע עם מיפוי הפורטים נראית כך:

```
$ docker run -p 4123:80 -d friendlyhello
5c203b090f3dda9a6bfd0b176ce52ae4e16957a8358ceed6f10786c328814618
```

אני יכול לראות שהקונטיינר עדיין רץ עם ps:

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
5c203b090f3d        friendlyhello       "python app.py"     13 seconds ago      Up 12 seconds
```

ואני יכול להתחבר אליו מהדפדפן על ידי גלישה לכתובת `http://localhost:4123/` . אגב כשנמאס לכם מהקונטיינר תוכלו לסגור אותו עם:

```
$ docker stop 5c203b090f3d
```

רק תחליפו את מזהה הקונטיינר בזה שמופיע אצלכם.

אתם יכולים גם למחוק קונטיינר מהמחשב שלכם לחלוטין עם:

```
$ docker rm 5c203b090f3d
```

ולמחוק אימג' לחלוטין עם:

```
$ docker rmi friendlyhello
```

```
$ docker run -it friendlyhello
```

לאן עכשיו?

ראינו איך בונים אימג', איך מפעילים אותו ואיך יוצרים ממנו קונטיינר. ראינו גם איך לחבר את הפורטים של הקונטיינר כדי שיעשה משהו מועיל על המחשב שלנו. עדיין נשאר לנו להבין איך לשלב מספר קונטיינרים ולהפעיל אותם בשיתוף פעולה כדי לקבל מערכת אחת מלאה, שמחולקת בין מספר שרתים. בנוסף נרצה להתעמק במבנה ה Dockerfile ולראות איזה עוד דברים אפשר יהיה לעשות עם קונטיינרים. בפוסטים הבאים בסידרה נחקור לעומק את שני הנושאים האלה.

מעדיפים לקרוא מהטלגרם? בקרו אותנו ב: @tocodeil (https://t.me/tocodeil)

או הזינו את כתובת המייל וקבלו את הפוסט היומי בכל בוקר אליכם לתיבה:

כתובת:

me@email.com

צרפו אותי

נהניתם מהפוסט? מוזמנים לשתף ולהגיב

Facebook Twitter +Google

What do you think?

34 Responses

Upvote Funny Love Surprised Angry Sad

2 Comments **ToCode** **Disqus' Privacy Policy** **Login**

Favorite 1 Tweet Share **Sort by Oldest**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Lior • 3 years ago

היי יינון. תודה על מדריך מעולה ושימושי כמו כל המדריכים שלך!
ישנה בעיה בחישוב ה-visitors. ככל הנראה בעיית התחברות ל-redis.

.תודה

• Reply • Share ›

Ynon Perek Mod Lior • 3 years ago

מדויק. מחר פוסט המשך שמראה איך לשלב מספר קונטיינרים למערכת אחת ואז נוכל לקבל גם את רדיס

• Reply • Share ›

Subscribe **Add Disqus to your siteAdd DisqusAdd** **Do Not Sell My Data**