## داکر (انگلیسی: Docker‎) یک برنامه رایانه‌ای متن باز است که از شبیه‌سازی سطح سیستم‌عامل برای توسعه و منتشر کردن پکیج ها که به عنوان کانتینر(container) شناخته می‌شوند استفاده می‌کند. داکر استقرار(deployment) نرم‌افزارهای کاربردی را درون کانتینر به وسیلهٔ فراهم کردن لایهٔ انتزاعی اضافه‌ای فراهم می‌کند. نرم افزاری که میزبانی کانتینر های داکر را به عهده دارد موتور داکر(docker engine) نام دارد. داکر در سال ۲۰۱۳ شروع به کار کرد و توسط شرکت داکر توسعه داده می‌شود. این سرویس به دو نوع رایگان و پولی در دسترس است. کانتینرهای داکر قسمتی از نرم‌افزار را در یک سیستم فایل کامل تعبیه می‌کند. به صورتی که شامل هر آنچه جهت اجرا شدن (مانند کد زمان اجرا، ابزارهای سیستم و کتابخانه سیستم) لازم است و هر آنچه که می‌تواند بر روی یک سرور نصب شود. این امر اجرای برنامه را به صورت ثابت در هر نوع محیطی تضمین می‌کند.

به زبانی ساده‌تر داکر برنامه است که کمک میکند برنامه‌ها و پروژه هامون رو بهتر با محیط ها هماهنگ کنیم

https://www.docker.com/

بعضی مواقع ممکن است به ارور (?is the docker daemon running) بر بخورید برای رفع این ارور کافی است از طریق systemctl داکر را start کنید ===========> Sudo systemctl start docker

<https://docs.docker.com/>

===================================================================================

what is docker  
در آدرس زیر موارد استفاده داکر توضیح داده شده است :

https://www.docker.com/use-cases

منظور از container همان محیط ایزوله ای است که برنامه و پروژه تون را در اون اجرا میکنید حالا ما اگر این محیط ایزوله مون را هر کجا که ببریم بازهم اجرا می‌شود

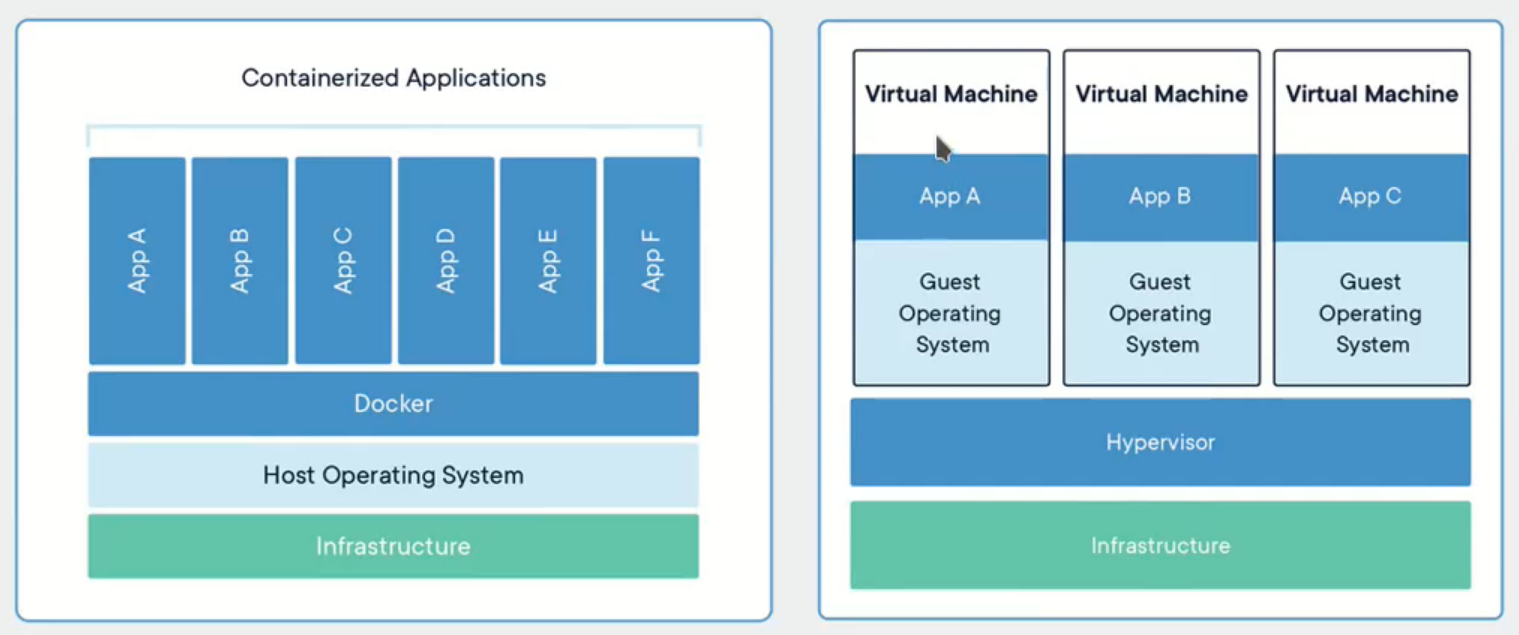


https://www.docker.com/resources/what-container

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

تفاوت میان docker و vm :

بدی vmها این است که بسیار سنگین میباشند و حتی خاموش روشن کردن آن‌ها زمانی حدود ۱۰ دقیقه را هدر میدهد حالا شما فکر کنید که قرار است که به یکی از برنامه هایتان که در یک vm جداگانه میباشد مقدار رم را اضافه کنید شما ابتدا باید vm را خاموش کنید و مقدار رم را تغییر دهید سپس vm را روشن کنید و بدی‌های دیگر که تفاوت اصلی این دو در عکس زیر مشهود است

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

برنامه ای که قرار است نصب شود در داکر به عنوان image شناخته میشود و محیطی که این برنامه ها قرار است در آن ها اجرا شوند به عنوان container شناخته میشوند(توجه شود که نصب iamge در خود docker) و بعد از نصب میتوانیم هر imageی را در هر کانتیری که دوست داشته باشیم اجرا کنیم

=====================================================================================

creating first container

ما در docker hub سه نوع image داریم : ۱.verified publisher : توسعه دهندگان رسمی که docker آن‌ها را تأیید کرده است

۲. official image : این image ها را خوده داکر توسعه داده است

۳. و یکسری image هایی است که برنامه نویسان غیر رسمی آن‌ها را توسعه داده است

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

دستور docker pull name\_imge:tag ایمیج مورد نظر با tag (ورژن) مورد نظر را دانلود کرده و روی کامپیوتر local ما قرار میدهد

اگر tag را وارد نکنید به صورت پیش‌فرض آخرین نسخه را دانلود میکند

دستور docker images تمامی imageهایی که روی کامپیوتر local شما وجود دارد را نمایش میدهد

دستور sudo su در ترمینال لینوکس باعث میشود شما به عنوان کاربر root شناخته شوید

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

docker [option] COMMAND حالت کلی دستورات docker به این صورت میباشد  
اگر نیاز داشتید که درباره ی یک دستور اطلاعات بیشتری به دست آورید میتوانید از دستور docker help COMMAND استفاده کنید

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

دستور docker run name\_image آن image انتخابی شما را داخل یک container اجرا میکند

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

docker ps ====>نمایش containerهایی که بسته نشده اند و هنوز فعال هستند

docker ps -a ====> نمایش containerهایی که بسته شده اند

اسمی که به containerها داده می‌شود به صورت random میباشد اما شما میتوانید این اسم‌ها را overwrite کنید

docker ps -q فقط id های container را نشان میدهد

میتوان آپشن های هر دستور docker را باهم ترکیب کرد مثلا :

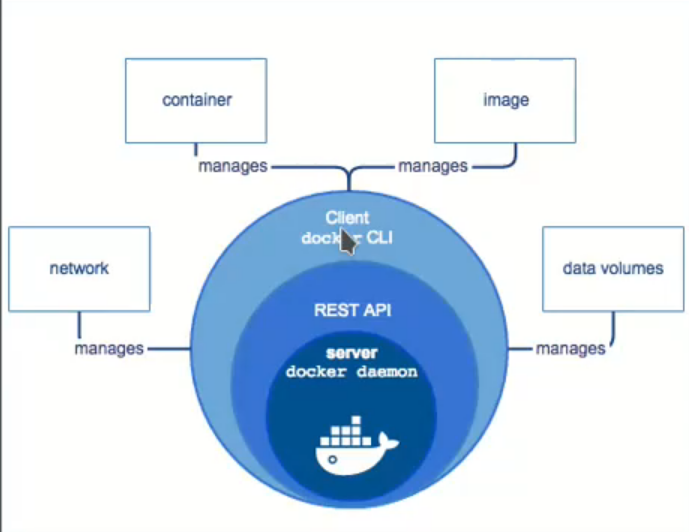
sudo docker ps -aq

=====================================================================================

basic commands

نکته مهمی که باید به آن توجه کرد این است که داکر برای اجرا یک image حتما یک container جدید میسازد حتی اگر آن image تکراری باشد و قبلا ران شده باشد به زبان دیگر اگر بخواهید ۳ بار image hello-world را اجرا کنید در پایان شما ۳ container بسته شده خواهید داشت (اما شما میتوانید container هایی که بسته شده‌اند را اجرا کنید)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



مفهوم کلی داکر در این عکس بیان شده است که ما در docker cli دستورات را وارد میکنیم این دستورات در قسمت rest api ترجمه میشوند و در قسمت docker daemon اجرا میشود و به این ۳ بخش باهم docker engin گفته میشود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

اختصاص نام برای container:

docker run –name first hello-world اکنون اسم containerی که ایمیج hello-world در آن اجرا می‌شود first میباشد

حذف یک container:

docker rm name\_container\_archive توجه شود که تنها میتوانید containerهایی که stop شده‌اند را حذف کنید

حذف تمام containerها:

docker container prune تمام container هایی که stop شده‌اند را حذف میکند

هنگامی که ما از دستور run استفاده میکنیم اگر image در local ما وجود داشته باشد آن را اجرا میکند اما اگر وجود نداشته باشد ابتدا دانلود کرده سپس اجرا میکند

اجرا در یک container و حذف آن :

docker run –rm name\_\_image ایمیج مورد نظر در یک container اجرا می‌شود و پس از پایان container حذف می‌شود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

همان طور که میدانید busybox یک لینوکس سبک میباشد اگر هنگام اجرا image

busybox از آپشن it استفاده کینم باعث میشود که وارد busybox بشویم

docker run -it –rm busybox

i (interactive):

stdin(standard input) را برای من نگه دار تا زمانی که لازم داشته باشم منظور از stdin جایی است که میتوانم از طریق آن دستورات را وارد کنم(کیبورد)

t (tty) :

کل سیستم شما را به یک محیطی میبرد که بتوانید با آن کار کنید

برای خروج میبایست exit را تایپ کنیم و یا ctrl+D را بزنیم

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

برای حرف یک یا چند image میتوانید از دستور docker rmi name\_image استفاده کنید

=====================================================================================

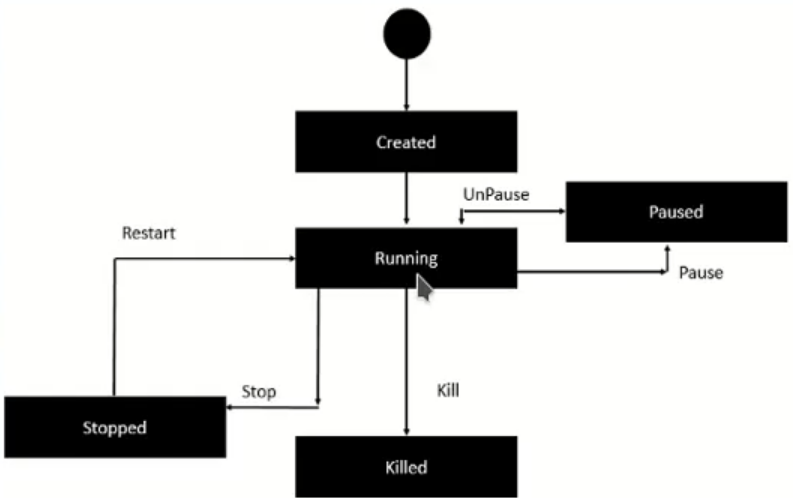
Container Life Cycle

بعضی وقت ها لازم است که container ما تا زمانی که ما بخواهیم در حال اجرا باشد برای این منظور هنگام run از آپشن detach استفاده میکنیم که container را در background اجرا میکند و در حالت اجرا باقی میگذارد

docker run [-d or –detach ] redis

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

container life cycle:



- با استفاده از دستور docker create میتوانیم یک container جدید را ایجاد کنیم

- containerی که جدید ساخته شد را میتوانید با دستور start اجرا کنید و به حالت running ببرید

- containerهایی که در حالت stopped قرار دارند را میتوانید با دستورهای start و restart دوباره به حالت running منتقل کنید

- تفاوت pause و stop در این است که در حالت pause کانتینر هنوز در حالت اجرا میباشد اما processی انجام نمیدهد

- تفاوت بین kill و stop در این است که اگر شما از دستور stop استفاده کنید به container اجازه میدهد که processهایش را انجام دهد و بعد stop شود اما kill این‌گونه نیست و سریعاً container را از حالت running خارج میکند

بعد از تمام این دستورات اسم container مورد نظر شما قرار میگیرد =====> docker commands name\_container

نکته ای که در اینجا باید دقت شود این است که اگر containerی با دستور kill هم متوقف شده باشد هم میتوان دوباره با دستورات start و restart آنرا فعال کرد

=====================================================================================

image layers

jتمام image هایی را که ما استفاده میکنیم بدون استثناء لایه لایه هستند به این صورت که از لایه پایین شروع کرده‌اند و یکسری تنظیمات را روی هم اضافه کردند که الان شده اون imageی که شما دارید ازش استفاده میکنید

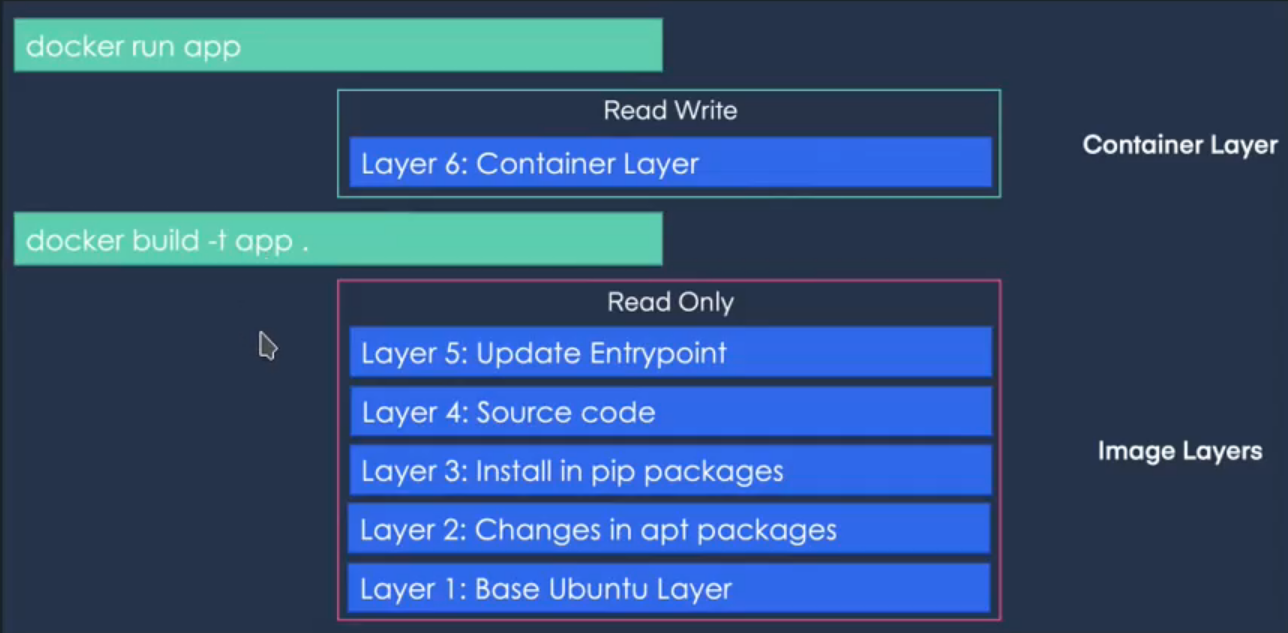
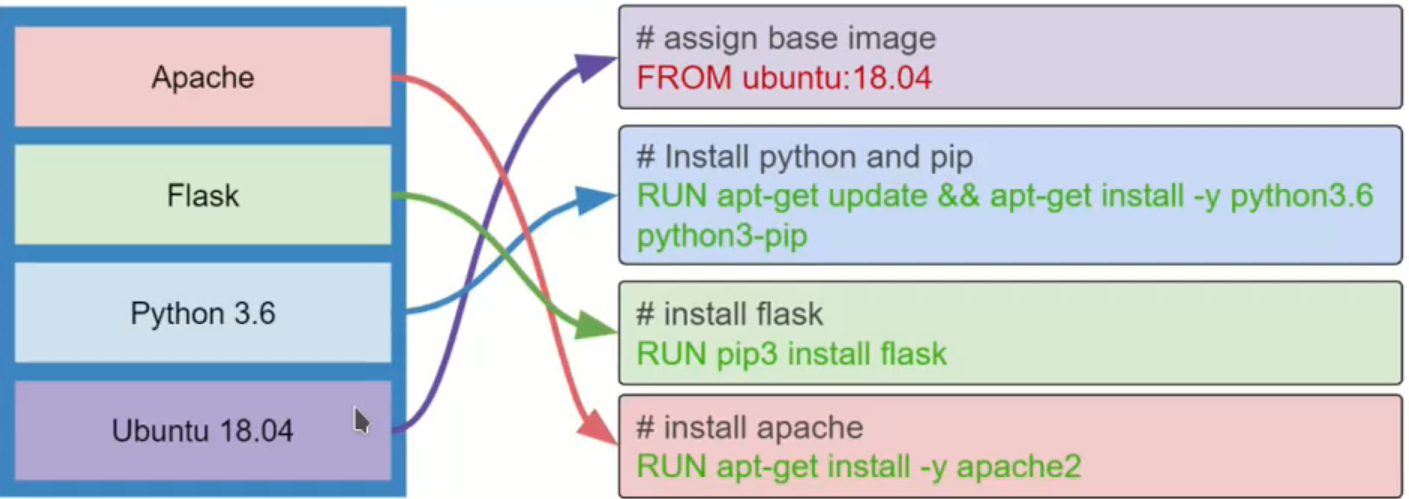
برای مثال image پایتون را در نظر بگیرید image پایتون درست است که باید پایتون باشد اما برای اینکه پایتون اجرا شود به یک سیستم عاملی نیاز دارد پس اگر بخواهید پایتون را اجرا کنید به یک سیتم عامل نیاز خواهید داشت اگر توجه کنید سایز image پایتون حتی از سایز image اوبونتو هم بیشتر میباشد

با دستور exec میتوانیم یک دستور را به یک container در حال اجرا ارسال کنیم و خروجی دریافت کنیم:

docker exec name\_container name\_command

docker exec -it rd bath

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ



مفهوم عکس دوم به این صورت است که لایه‌های image هایی که دانلود میکنید readonly میباشند و شما نمیتوانید در اون لایه ها تغییری ایجاد کنید اما میتوانید در container layer یک لایه به این image اضافه کنید

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

برای اینکه بتوانید به یک containerی که در حالت detach اجرا می‌شود وارد شوید میتوانید دستوری که به وسیله exec ارسال میکنید bash باشد ======> docker exec -it container\_name bash

=====================================================================================

Dockerfile

Docker می تواند با خواندن دستورالعمل های Dockerfile به طور خودکار imageها را بسازد. Dockerfile یک سند متنی است که شامل تمام دستوراتی است که کاربر می تواند در خط فرمان برای جمع آوری یک image فراخوانی کند. با دستور

docker build کاربران می توانند یک برنامه قابل اجرا خودکار ایجاد کنند که چندین دستورالعمل خط فرمان را به طور متوالی اجرا کند.

https://docs.docker.com/engine/reference/builder/

- در اولین خط در Dockerfile باید base ایمیج خود را مشخص کنید که با دستور FROM اینکار انجام می‌شود

- خط بعدی WORKDIR میباشد که محل اصلی که فایل‌های برنامه باید ذخیره و اجرا شود را مشخص میکند به این صورت که مثلاًWORKDIR /sajjad: به این معنی میباشد که در دایرکتوری که هستی پوشه sajjad اگه وجود داشت برو داخلش و محل اصلی از این به بعد این پوشه میباشد و اگر وجود نداشت بساز و داخلش برو و به عنوان پوشه اصلی در نظر بگیر

- دستور بعدی COPY میباشد که از فایلی که مشخص شده است یک کپی میگیرد و به محلی که آدرس آن را مینویسید میبرد ( منظور از . همان WORKDIR میباشد )

- دستور بعدی RUN میباشد که معمولاً برای نصب پکیج ها از آن استفاده می‌شود

- دستور بعدی CMD میباشد که برای دستورات unixی استفاده می‌شود

در قسمت Dockerfile بعدی توضیحات بیشتری وجود دارد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

برای ساخت image از Dockerfile از دستور build باید استفاده شود به این صورت که به بعد از دستور build باید آدرس Dockerfile قرار گیرد

حالت کلی دستور build

docker build [OPTIONS] PATH | URL | -

docker build -t name:tag address

docker buld -t req:1.10 .

=====================================================================================

push to registery

docker login ======> این دستور برای ست کردن username و password داکر شما در ترمینال میباشد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ساخت یک image جدید از image موجود :

docker tag old\_name new\_name

docker tage req:1.10 ahmadalim/req:one

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

آپلود کرد در dockerhub:

docker push ahmadalim/req:one

===================================================================================

Port Forwarding:

منظور از port forwarding این است که به docker بفهمانیم برای این container یک پورت را خالی کن تا بتوانیم خارج از اون container بتوانیم بهش وصل شویم

<https://docs.docker.com/config/containers/container-networking/>

این کار به صورت‌های مختلفی انجام می‌شود و از آپشن p- استفاده می‌شود :

8080:80 ===> در اینجا این منظور است که پورت 8080 لکال من را به پورت 80 کانتینر متصل کن

توجه داشته باشید که میتوانید host خود را مشخص کنید اگر host خود را مشخص نکنید به طور پیش‌فرض 0.0.0.0 درنظر گرفته می‌شود

192.168.1.100:8080:80

===================================================================================

load and save:

گاهی وقت‌ها لازم است که imageی را بصورت فایلی ذخیره کرده تا بتوانید از آن در جای دیگر استفاده کنید برای این منظور کافی از دستورات save و load استفاده کنید:

docker save -o alpine.tar alpine

آپشن o نام فایل خروجی را مشخص میکند و همیشه باید پسوندش tar باشد

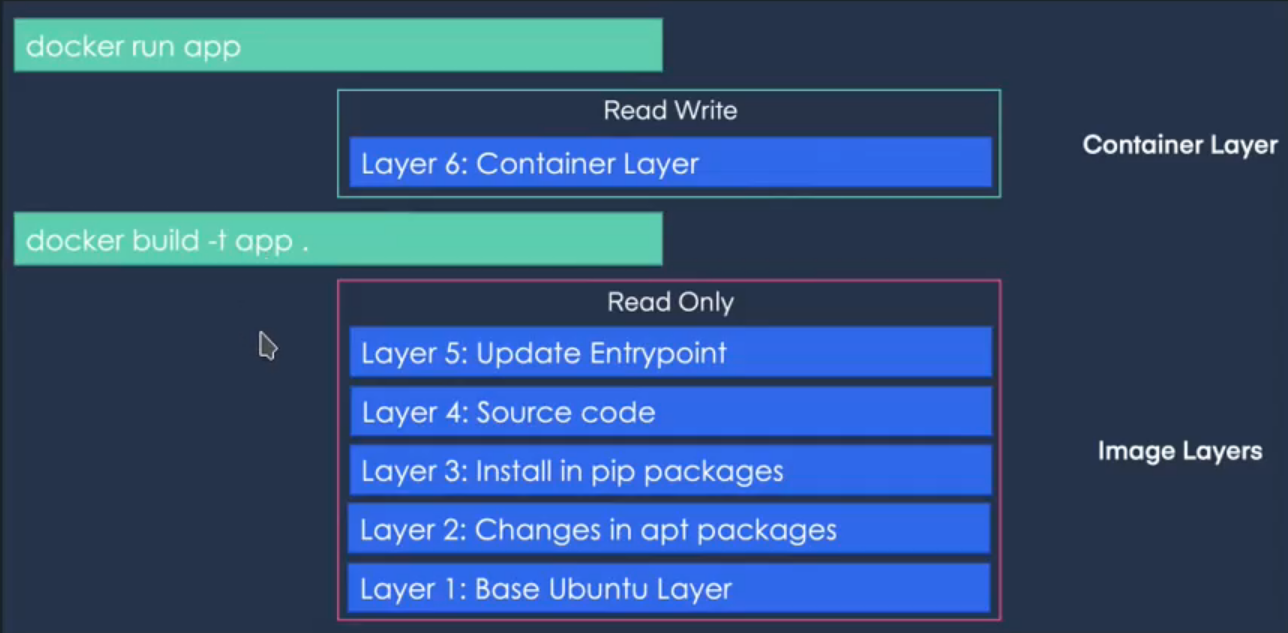
docker load -i alpine.tar

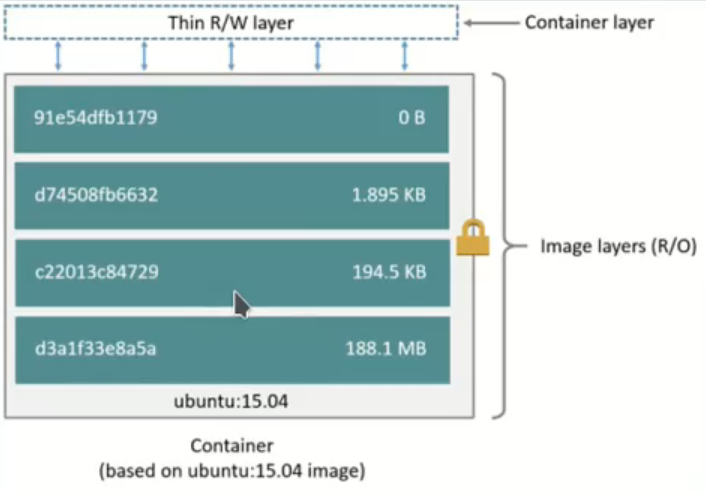
آپشن I نام فایل ورودی را مشخص میکند

===================================================================================

commit container layers

همان‌طور که از عکسهای زیر مشخص است لایه‌هایی که در image اصلی وجود دارند read only هستند و ما نمیتوانیم آن هارا تغییر دهیم اما میتوانیم لایه‌های دیگری از طریق container layer اضافه کنیم برای این منظور باید از image مورد نظر یک container ساخته شود و تغییراتی که مد نظرمان میباشد اعمال شود سپس با استفاده از دستور زیر یک image جدید همراه با تغییرات ما ساخته می‌شود :





docker container commit name\_container name\_new\_image

docker container commit py python:first

با درستور history میتوانیم تاریخچه دستوراتی که در ترمینال زده ایم را ببینیم

=====================================================================================

commit attributes

قبل از اینکه یک container را commit کنیم میتوانیم با دستور diff ببینم چه تغییراتی در این container به وجود آمده است :

docker diff name\_container

docker diff py

هنگام مشاهده تغییرات حروفی وجود دارد که A به معنی add , حرف C به معنی change و D به معنی delete میباشد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

هنگام commit کردن میتوانیم با آپشن های a و m نویسنده این تغییرات و توضیحی درباره این تغییرات مشخص کنیم :

docker container commit -a ‘sajjad’ -m ‘add test.txt’ py python:first

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

با استفاده از دستور inspect میتوانیم اطلاعاتی درباره ی یک image ببینیم :

docker inspect python:first

در اینجا نویسنده و متن پیغام مشخص می‌شود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

با استفاده از دستور info میتوانیم اطلاعاتی درباره ی خود docker به دست آوریم:

docker info

=====================================================================================

Dockerfile 2

هنگام ساخت Dockerfile در اولین خط هنگامیکه میخواهیم base image را مشخص کنید میتوانیم از scratch استفاده شود که باعث می‌شود یک image خالی برای شما ساخته شود و خود شما باید تمام configها را مشخص کنید که پیشنهاد نمیشود چون بسیار مشکل میباشد:

FROM scratch

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

در Dockerfile دستور MAINTAINER وجود دارد که میتوانید اطلاعاتی درباره کسی که این Dockerfile را میسازد بنویسید:

MAINTAINER ‘ahmadalim mr.ahmadalimoshtaghi@gmail.com’s

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

در Dockerfile برای کامنت گذاری میتوانیم از علامت # استفاده کنیم

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

دستور RUN برای اجرا کردن هر دستوری داخل ترمینال استفاده می‌شود (در alpine با دستور apk میتوانیم برنامه‌ها رو نصب کنیم):

RUN apk update

Run apk add nano

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

دستور CMD هم ماننده RUN برای اجرا کردن استفاده می‌شود اما با این تفاوت‌ها که از دستور CMD برای اجرا کردن برنامه‌ها استفاده می‌شود (مثلاً برنامه‌های پایتونی) و از دستور RUN برای اجرا کردن دستورات سیستمی استفاده می‌شود تفاوت دوم این است که از دستور RUN به هر تعدادی میتوانیم داخل Dockerfile استفاده کنید اما از دستور CMD تنها یکبار میتوانید استفاده کنید و اگر به تعدادی از دستور CMD استفاده کرده باشید فقط آخرین CMD اجرا می‌شود

CMD python ./main.py

تفاوت سوم این است دستور RUN هنگام ساخت image صدا زده می‌شود اما دستور CMD هنگامیکه image اجرا می‌شود صدا زده می‌شود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

هنگام ساخت image از Dockerfile به این صورت عمل می‌شود که هر دستور به عنوان یک لایه در نظر گرفته می‌شود و هنگام اجرای هر دستور یک image جدید ساخته می‌شود به این صورت که برای ساخت image جدید برای دستور جدید از image قبلی استفاده می‌شود و بعد از ساخت image جدید image قبلی حذف می‌شود پس به زبان دیگر هنگام بیلد Dockerfile به تعداد دستورات موجود در Dockerfile ایمیج ساخته می‌شود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

دستور docker images -a تمام imageها را نمایش میدهد (ماننده ps ) در این دستور میتوانید imageهای کش شده را مشاهده کنید ( image هایی که imageهای جدید برای ساخته شدن از آن‌ها استفاده کرده‌اند )   
دلیل وجود این image ها این است که اگر شما یکبار دیگر از دستور build استفاده کردید از این image ها استفاده کند

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

اگر از قبل یک image ساخته باشید و اکنون یک image جدید با همان نام قبلی ایجاد کنید docker آن image قبلی را به عنوان dangling image در نظر میگیرد و اسم آن را None میگذارد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

برای حذف dangling image میتوانیم از دستور rmi همراه با image id استفاده کنیم اما اگر dangling image هایمان زیاد بود میتوانیم از روش زیر استفاده کنیم:

docker rmi $(docker images -q –filter “dangling=true”)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

دستور EXPOSE یک پورت را خالی میکند که بتوانیم به آن وصل شویم

دستور ENV باعث set شدن environment variable می‌شود

دستور ENTRYPOINT به طور خلاصه نقطه شروع image را مشخص میکند برای مثال :

FROM alpine:latest

RUN apk add mysql-client

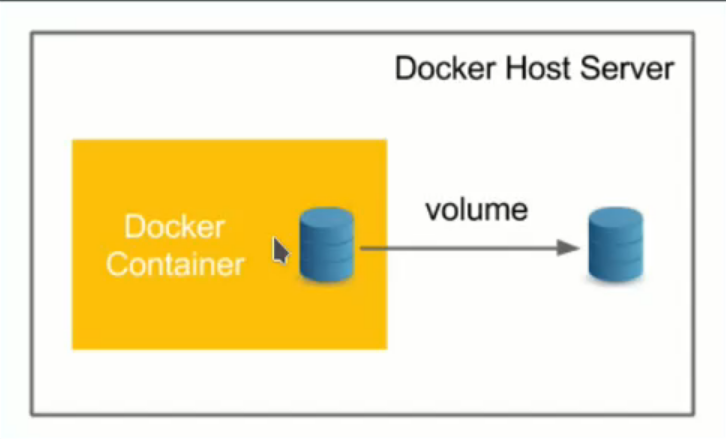
ENTRYPOINT [“mysql”]

بعد از نصب mysql-client خوده mysql را اجرا میکند ( یک جورایی میشه گفت جایگزین CMD میباشد با این تفاوت که CMD یک دستور را اجرا میکند اما ENTRYPOINT یک برنامه را اجرا میکند

=====================================================================================

Volume

تا الان ما هر containerی که ایجاد میکردیم , اطلاعاتی که داخل اون containerها بودند به صورت non persistent بودند به این معنی که بعد از حذف container اون اطلاعات هم از بین میرفت اما بعضی مواقع لازم است این اطلاعات نگهداری شود اینجاست که volumeها به کمک ما می‌آیند



مفهوم عکس بالا به این صورت است که خارج از container قسمتی از حافظه را در نظر میگیریم و اطلاعاتی که لازم است از container ذخیره شوند را داخل volume قرار میدهیم

https://docs.docker.com/storage/volumes/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

دستورات volumeها:

برای ساخت یک volume از دستور زیر استفاده می‌شود

docker volume create name\_volume ======> اگر برای volume مورد نظر اسم قرار ندهید docker برای آن یک hash در نظر میگیرد

برای لیست کردن volumeها از دستور زیر استفاده می‌شود

docker volume ls

برای دیدن اطلاعات مخصوص یک volume از دستور زیر استفاده می‌شود

docker volume inspect my\_volume

برای حذف کردن volume از دستور زیر استفاده می‌شود

docker volume rm my\_volume

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

وصل کردن container به یک volume:

$ docker run -d \

--name devtest \

-v myvol2:/app \

nginx:latest

یا

$ docker run -d \

--name devtest \

--mount source=myvol2,target=/app \

nginx:latest

آپشن v در اینجا به این معنی است که volume که به این container متصل شده است myoluv میباشد و در container هر چیزی که در دایرکتوری home ذخیره می‌شود را به volume v1 منتقل کن

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

هر اتفاقی که در container بیافتد در داخل volume وجود دارد و بر عکس این قضیه هم وجود دارد

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

محل ذخیره volume ها به صورت زیر میباشد:

/var/lib/docker/volumes/name\_volume/\_data

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

متصل کردن یک container به volume از یک container دیگر:

docker run -it -d –name ap2 –volumes-from ap alpine

در این مثال container ap2 به volumeی که برای container ap بود متصل شد

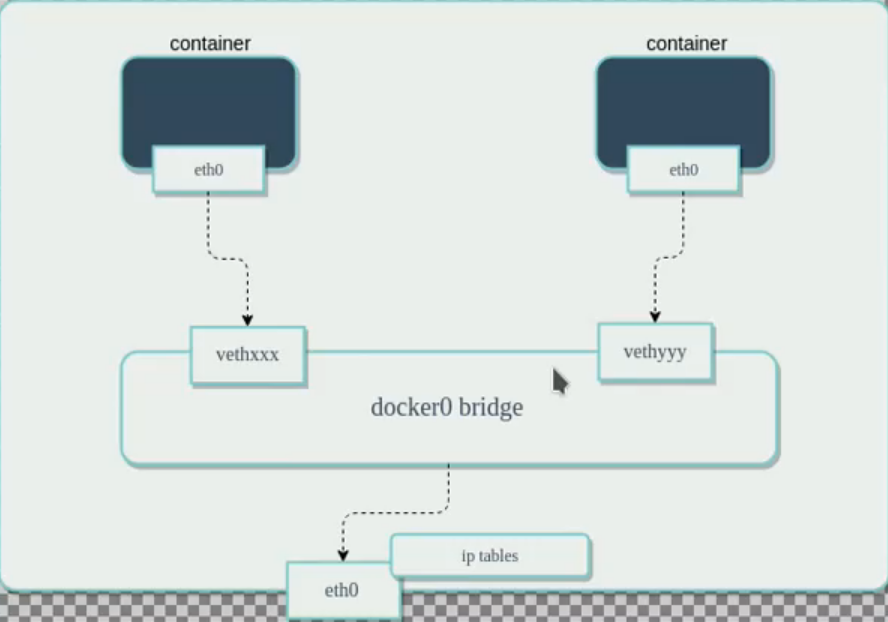
ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

حال میخواهیم که container ap3 فقط بتواند اطلاعات را بخواند و نتواند اطلاعات را تغییر دهد:

docker run -it -d –name ap3 –volumes-from ap:ro alpine

===================================================================================

Network:



دو نوع ارتباط بین container میتوان بوجود آورد یکی bridge و دیگری host

که بصورت پیش‌فرض bridge میباشد bridge برای زمانی مناسب است که containerهایتان در یک سیستم میباشند

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

دستورات network:

docker network create one

docker network inspect one

docker network connect one py

دقت داشته باشید که وقتی چندین container را به یک network متصل کردید برای آدرس دهی کافیست که فقط اسم آن container را صدا بزنید

===================================================================================

docker compose:

ابزاری است که میتوانیم چندین container را همزمان باهم در یک host اجرا کنیم به این صورت میباشد که شما یک file yaml ایجاد میکنیم و configهای خود را به آن اضافه میکنیم

اولین دستوری که باید نوشته شود version که باعث هماهنگ کننده compose و docker engin می‌شود