# تعاریف

|  |  |  |  |
| --- | --- | --- | --- |
|  | -- | | |
| شماره سند: | شناسه سند | |  |
| طـبقه‌بندی: | عادی/حساس/محرمانه | |  |
| تــاریــخ: | روز/ماه/سال |  | |

v

## انواع سرور

### سرور اختصاصی bare metal

Bare Metal به سرور های فیزیکی گفته می شود، که نقطه مقابل سرور های مجازی و حتی هاست ابری محسوب می گردد. در واقع علت استفاده از این نام گذاری فیزیکی و فلزی بودن نوع سرور و عدم اشتراک گذاری آن بین تعدادی مشتری می باشد.

این سرور ها دارای قدرت پردازشی بالایی هستند، و برای وب سایت های پربازدید متوسط و بزرگ بسیار مناسب هستند. بنابراین سرور اختصاصی Bare Metal همان سرور فیزیکی می باشد، که تنها در اختیار یک مشتری قرار داده می شود، و 100 درصد منابع سرور در اختیار همان مشتری قرار داده می شود.

اصلی ترین تفاوت سرور اختصاصی با سرور مجازی، ساختار فیزیکی یا Bare Metal آن ها می باشد، زیرا سرور های مجازی یا vps ها که از قطعات سخت افزاری مشترک استفاده می کنند.

### ماشین های مجازی

ماشین مجازی یا VM یک شبیه سازی از کامپیوتر واقعی ست که برنامه ها را دقیقا مانند آن اجرا می کند. این شبیه سازی توسط Hypervisor انجام می شود. هایپروایزور(esxi, hyper-v, …) به مجموعه نرم افزار ، میان افزار و سخت افزاری گفته می‌شود که ماشین های مجازی بر روی آن اجرا می شوند. به ماشین های فیزیکی که Hypervisor بر روی آن ها راه اندازی شده است در اصطلاح رایج Host Machine یا Host گفته می شود. در نتیجه هایپروایزور قادر خواهد بود که منابع پردازشی مانند RAM و CPU حقیقی را بین ماشین های مجازی به اشتراک بگذارد. VM ای که بر روی Host اجرا می شود ، Guest Machine نامیده می شود. Guest Machine شامل برنامه مورد نظر و هر آن چه که برای اجرای برنامه به آن نیاز است ( مانندOS ، System Binaries و کتابخانه ها) خواهد بود. در نتیجه واضح است که هر ماشین مجازی Network Adaptor و سایر اجزای مختص به خود را خواهد داشت.

### Container

Container ها مانند ماشین های مجازی منابع پردازشی خاص خود را از هاست دریافت می کنند. می توانند فرمان ها را اجرا کنند. شبکه مجزا از یکدیگر داشته باشند. Route ها و IPTable های مختص به خود را دریافت نمایند. فایل سیستم ها در درون آن ها Mount می شود و بسیاری از ویژگی های دیگر آنها با VM مشترک است.

اما نکته در این است که در VM هر ماشین مجازی Kernel جدا از کرنل ماشین مجازی دیگر دارد. اما در کانتینر ها چنین نیست و کرنل سیستم عامل بین ماشین های مجازی به اشتراک گذاشته شده است.

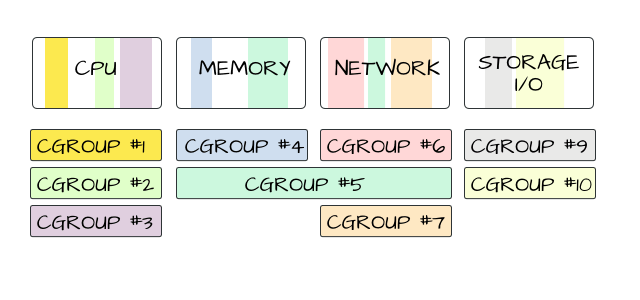
|  |  |
| --- | --- |
| معماری ماشین مجازی | معماری کانتینر در زمان استفاده از Docker Engine |

در معماری کانتینر ، کرنل یک سیستم عامل می تواند بین تمام Application ها به اشتراک گذاشته شود. تنها چیزی که Container ها را منحصر به فرد می کند کتابخوانه ها و باینری های هر Application است که با Application دیگر متفاوت است. این ایده استفاده از Kernel یکسان برای Application های ایزوله متفاوت باعث صرفه جویی بسیار زیادی در مصرف منابع پردازشی می گردد.

# قابلیت‌های کرنل لینوکس

## Cgroups

Cgroups یا Control Groups، مکانیزم کرنل لینوکس برای محدود کردن و اندازه‌گیری کل منابع در حال استفاده، توسط گروهی از پراسس‌های اجرایی روی سیستم است. برای مثال با استفاده از cgroup می‌توانید منابع سیستم‌های لینوکسی خود مثل RAM ،CPU یا IO را کنترل کنید. Cgroups در ابتدا توسط Paul Menage و Rohit Seth در گوگل ساخته شد و اولین قابلیت‌های آن در لینوکس 2.6.24 ادغام شد.



## Namespace

Namespaces مکانیزم دیگری در کرنل برای محدود کردن دید گروهی از پراسس‌ها، نسبت به بقیه سیستم است. برای مثال می‌توان یک یا چند پراسس را محدود کرد تا امکان دیدن و تعامل با بقیه پراسس‌های در حال اجرا روی سیستم یا مثلاً امکان دسترسی به فایل سیستم‌های مانت شده را نداشته باشد.

زمانی که از cgroupها و Namespaceها در کنار یکدیگر استفاده کنیم، در نهایت گروهی از پراسس‌های لینوکس را در یک محیط کاملاً ایزوله شده خواهیم داشت. این دقیقاً همان چیزی است که به آن یک کانتینر لینوکسی می‌گوییم. کانتینرها در لینوکس با مجموعه کاملی از Namespaceها محدود شده‌اند به طوری که آنها فقط می‌توانند دایرکتوری را که از آن بوت شده‌اند، پراسس‌های مربوط به خود، User IDهای خود و هرگونه رابط شبکه ای که اجازه دسترسی به آنها را دارند، مشاهده کنند. به همین ترتیب، کانتینرها برای کنترل استفاده از RAM ،CPU پهنای باند و IO با مجموعه کاملی از cgroupها محدود هستند. بنابراین، با اعمال تمام این محدودیت‌ها پراسس‌های در حال اجرا در یک کانتینر لینوکسی نمی‌توانند هیچ کدام از بخش‌های دیگر سیستم را ببینند و بنابراین طوری رفتار می‌کنند که انگار در یک کامپیوتر یا سرور جداگانه در حال اجرا هستند.

Namespace: یک قابلیت به نام Workspace ایجاد می کند که پروسه ها در آن به صورت ایزوله اجرا می شوند.

در نهایت با استفاده از این دو قابلیت یک Container ایجاد می شود که یک فضای ایزوله با میزان منابع مشخصی است.

## فایل سیستم Union FS

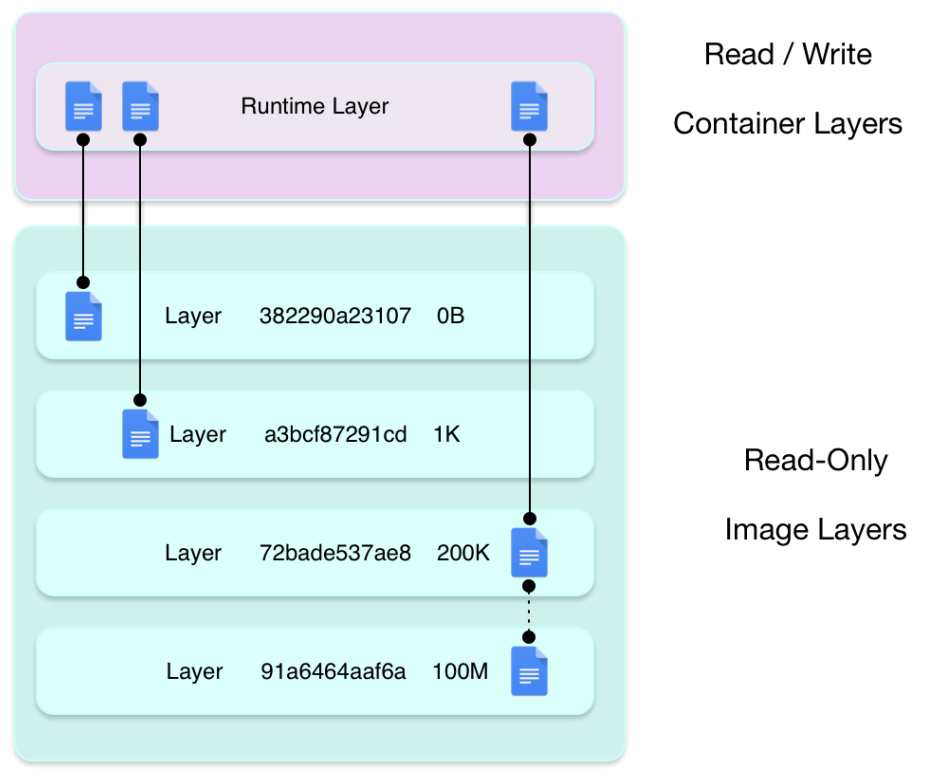
یک فایل سیستم است که در سیستم عامل لینوکس هم از آن استفاده شده است.

از قابلیت های مهم آن می‌توان به موارد زیر اشاره کرد:

* Layering
* Copy-On-Write
* Caching
* Diffing

کانتینرDocker از یک قالب فقط خواندنی به نام docker image ایجاد می شود.

وقتی که یک Container جدید ایجاد می کنیم، یک لایه جدید و نازک قابل نوشتن در بالای پشته لایه های زیرین موجود در Image پایه داکر اضافه می کنیم. تمام تغییرات ایجاد شده در کانتینر در حال اجرا، مانند ایجاد فایل های جدید، اصلاح فایل های موجود یا حذف فایل ها، در این لایه نازک قابل نوشتن، نوشته می شود.



تفاوت عمده بین یک کانتینر و یک Image، لایه قابل نوشتن بالایی است. تمام نوشته‌های موجود در کانتینر که داده‌های جدید را اضافه می‌کنند یا داده‌های موجود را اصلاح می‌کنند در این لایه قابل نوشتن ذخیره می‌شوند. هنگامی که کانتینر حذف می شود، لایه قابل نوشتن نیز حذف می شود و Image پایه بدون تغییر باقی می ماند.

از آنجایی که هر کانتینر لایه نازک قابل نوشتن خود را دارد و همه تغییرات در این لایه کانتینر ذخیره می‌شوند، این بدان معنی است که چندین کانتینر می‌توانند از یک Image پایه استفاده کنند و در عین حال داده خود را داشته باشند.

Docker storage driver مسئول فعال کردن و مدیریت لایه‌های Image و لایه قابل نوشتن است.

دو فناوری کلیدی مدیریت Image و کانتینر، Layering و Copy-On-Write هستند.

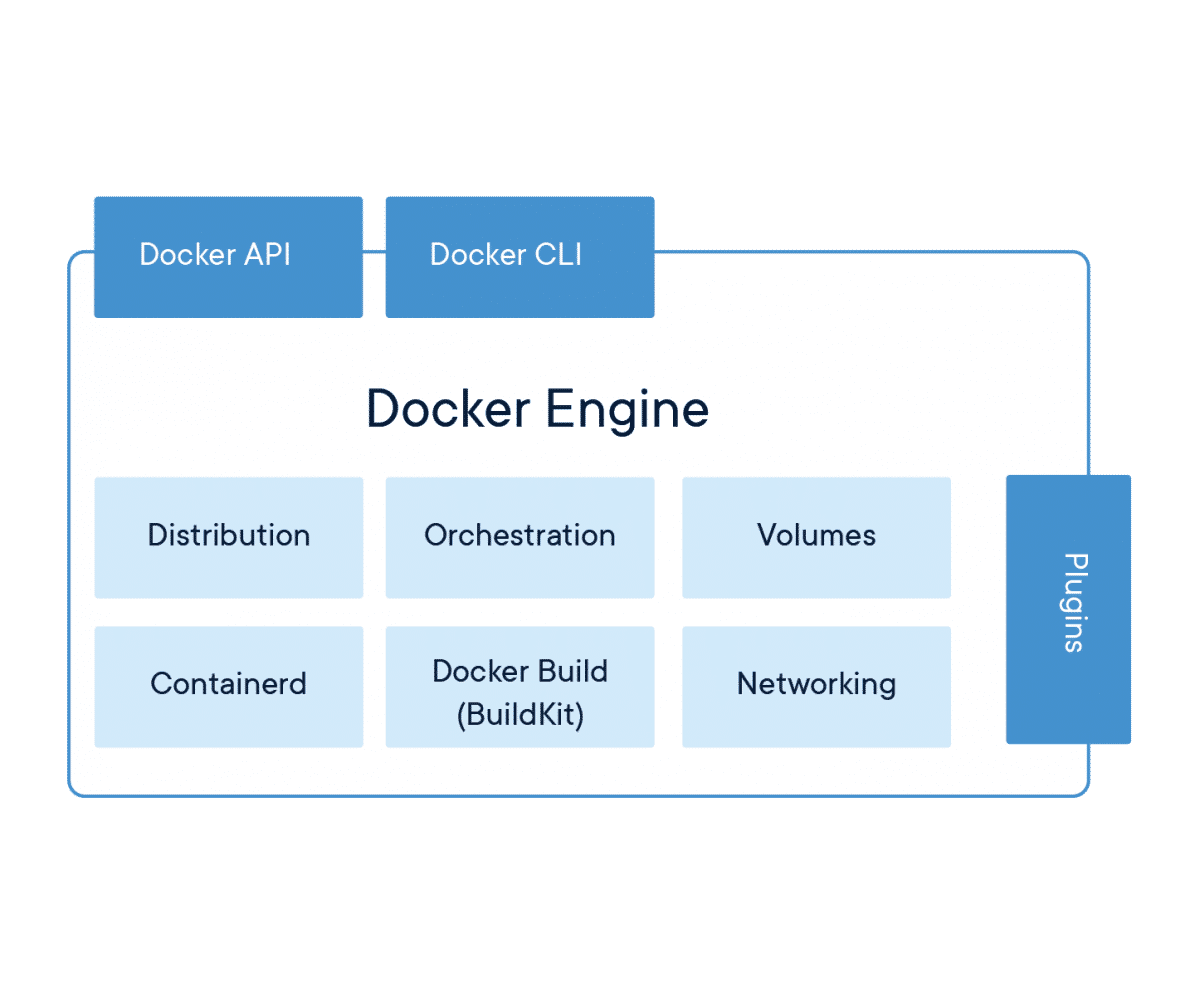
استراتژی Copy-On-Write نه تنها میزان فضای مصرفی کانتینرها را کاهش می دهد، بلکه زمان لازم برای راه اندازی یک کانتینر را نیز کاهش می دهد. در زمان شروع، داکر فقط باید یک لایه نازک قابل نوشتن برای هر کانتینر ایجاد کند.

# Docker Engine

بالاترین لایه ، که داکر بر روی آن Run می شود را Docker Engine می گویند. داکر انجین ابزار مدیریت کانتینرها، Image ها، Build ها و بسیاری موارد دیگر است. Docker Engine بر روی Linux اجرا می شود و شامل سه جز زیر است :

* یک Docker Daemon که بر روی Host راه اندازی شده است.
* یک کلاینت داکر که از طریق ارتباط با Docker Daemon دستورات را اجرا می نماید.
* یک REST API که ارتباط با Docker Daemon را به صورت Remote امکان پذیر می نماید.

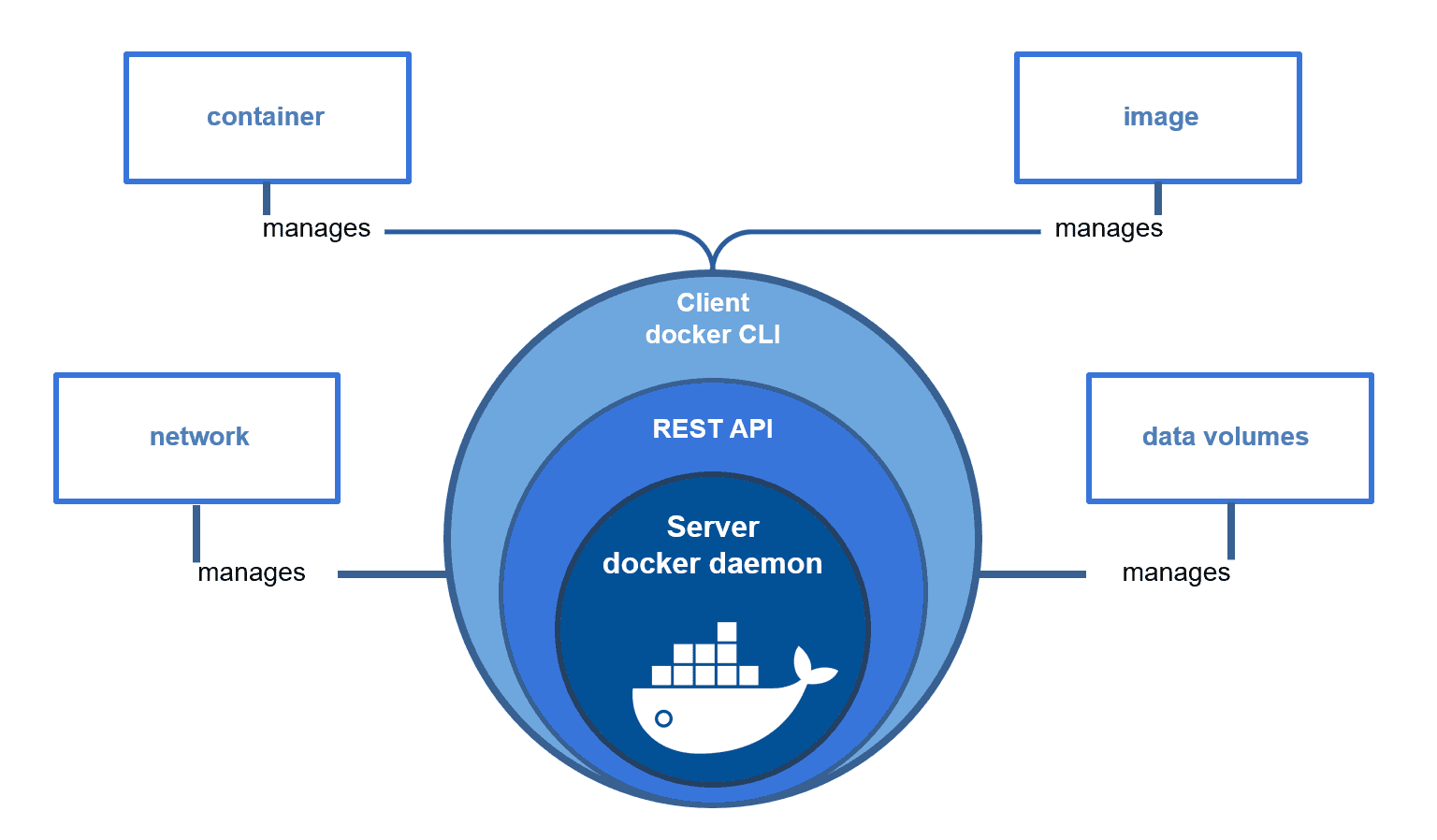
CLI از داکر APIها برای کنترل یا تعامل با Docker Daemon از طریق اسکریپت نویسی یا دستورات مستقیم CLI استفاده می‌کند. بسیاری از برنامه‌های دیگر Docker از API و CLI زیرین استفاده می‌کنند. دیمون اشیاء Docker مانند تصاویر، کانتینرها، شبکه‌ها و حجم‌ها را ایجاد و مدیریت می‌کند.



Containerd به عنوان Container Runtime اصلی Docker Engine است.

# کامپوننت های داکر

وقتی از داکر استفاده می‌شود، در واقع از Imageها، Containerها، شبکه‌ها، Volumeها، افزونه‌ها و سایر «اشیای داکر» استفاده شده است.



## Docker Client

کلاینت داکر جزئی ست که کاربر نهایی با آن در ارتباط خواهد بود. برای درک بهتر می توانید آن را یک UI برای داکر بنامید. هر چند که فاصله آن از واژه User Friendly بسیار زیاد است. شما به عنوان کاربر نهایی با Docker Client ارتباط خواهید داشت و کلاینت داکر دستورات شما را به Docker Daemon منتقل می نماید.

## Docker Daemon

داکر دیمون جزئی ست که دستورات ارسال شده به کلاینت Docker را اجرا می نماید. دستوراتی مانند Building یا ساختن ، Running یا راه اندازی و Distributing یا توزیع Container ها.

Docker Daemon ، خود ، بر روی هاست اجرا می شود. اما به عنوان کاربر شما هیچ گاه به صورت مستقیم با آن در ارتباط نخواهید بود. همان طور که در قسمت قبل در توضیح Docker Client گفته شد ، کلاینت داکر انتقال دهنده دستورات شما به Docker Daemon خواهد بود. نکته جالب اینکه تفاوتی ندارد که Docker Client بر روی همان هاستی باشد که Docker Daemon قرار دارد یا از روی هاست دیگری این مهم را به انجام برساند.

## Image

داکر ایمج یک فایل غیر قابل تغییر است که شامل کد منبع، libraries، وابستگی ها (dependencies) ، ابزارها و سایر فایل های مورد نیاز برای اجرای یک برنامه است. به دلیل وجود کیفیت read-only، از این ایمج ها یا تصاویر گاهی به عنوان اسنپ شات نام برده می شود. آنها نمایانگر یک برنامه کاربردی و محیط مجازی آن در یک زمان خاص هستند. وجود این ثبات یکی از ویژگی های عالی داکر است که به توسعه دهندگان اجازه می دهد تا نرم افزار را در شرایط یکنواخت و ثابت آزمایش کنند.

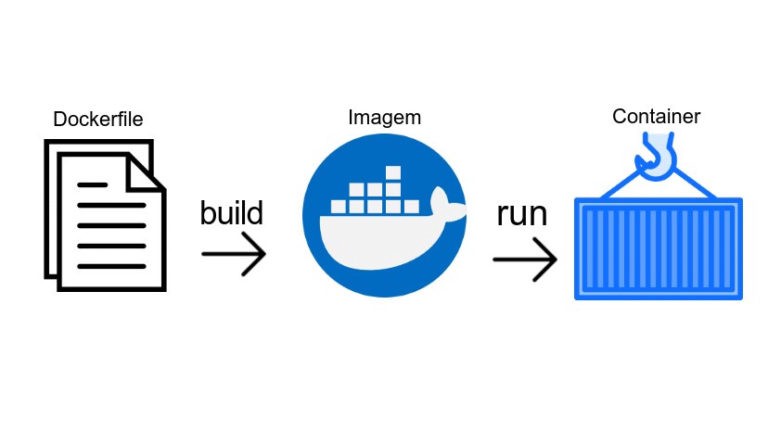
از آنجا که ایمج ها به نوعی فقط الگو یا template هستند، نمی توانید آن ها را استارت یا اجرا کنید. کاری که می توانید انجام دهید این است که از آن الگو به عنوان پایه برای ساخت کانتینر استفاده کنید. کانتینر در نهایت فقط یک ایمج در حال اجرا است. هنگامی که یک کانتینر ایجاد می کنید، یک لایه قابل نوشتن در بالای آن ایمج تغییرناپذیر اضافه می کند، به این معنی که اکنون می توانید آن را اصلاح کنید.

داکر رجیستری (Docker Registry) انبار (repository) ایمیج های مختلف با ورژن های مختلف است.

## Container

کانتینر Docker یک محیط مجازی run-time است که در آن کاربران می توانند برنامه ها را از سیستم اصلی جدا کنند. این کانتینر ها واحد های قابل حمل و فشرده ای هستند که می توانید در آن ها به راحتی یک برنامه را راه اندازی کنید.

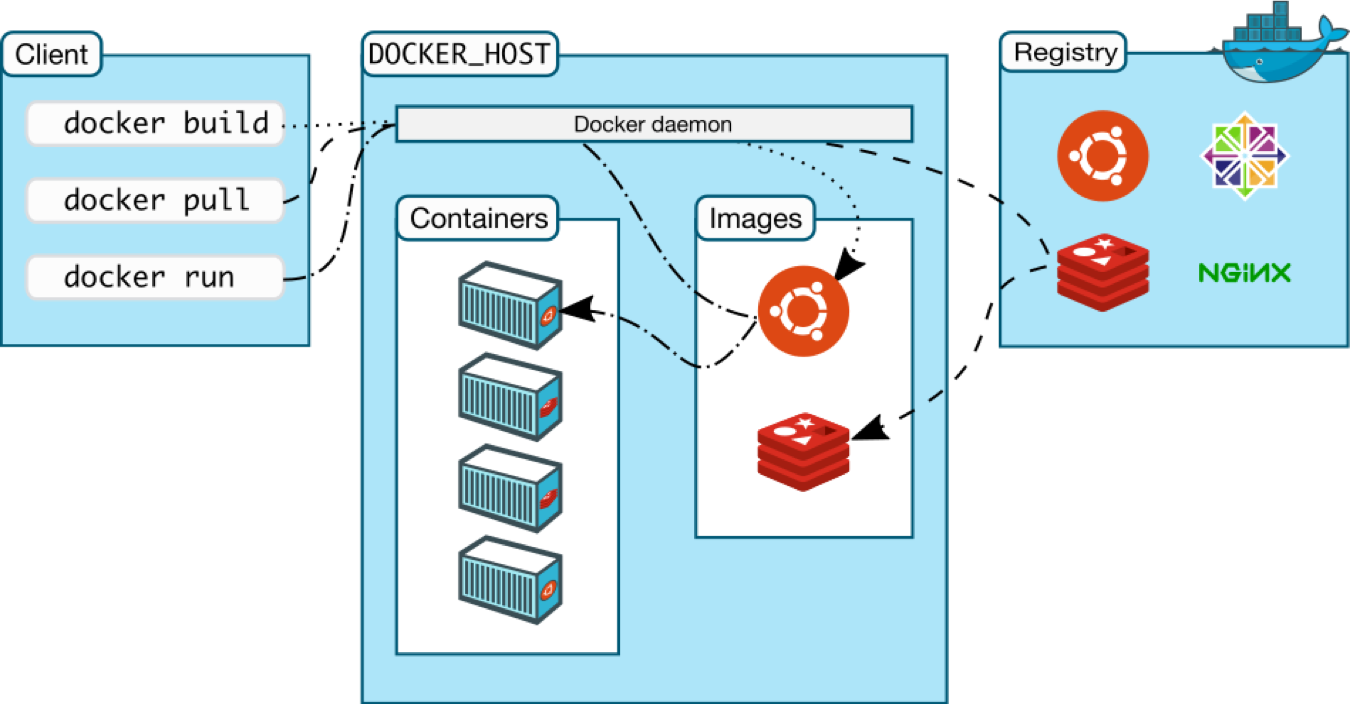
برخلاف ماشین های مجازی (VM) که مجازی سازی در سطح سخت افزاری انجام می شود، کانتینرها در لایه برنامه مجازی سازی می شوند. آن ها می توانند از یک ماشین استفاده کنند. هسته آن را به اشتراک بگذارند و سیستم عامل را مجازی کنند تا فرایندهای جداگانه را اجرا کند. این باعث می شود کانتینر ها به میزان زیادی سبک شوند و امکان حفظ منابع مهم برای شما وجود داشته باشد.



# گردش کار Docker

اکوسیستم داکر از اجزء اصلی زیر تشکیل شده است:

* Daemon Docker
* Docker Client
* Docker Images
* Docker Registries
* Docker Containers



# نصب داکر

با توجه به نسخه سیستم عامل از طریق راهنمای سایت داکر اقدام به نصب کنید.

[نصب داکر](https://docs.docker.com/engine/install/)

# مشاهده اطلاعات داکر

جهت مشاهده اطلاعات داکر از دستور زیر استفاده کنید.

|  |
| --- |
| # docker info |

# مشاهده لاگ کانتینر ها

لاگ هر کانتینر در مسیر root داکر، فولدر containers، در یک فولدر هم نام id کانتینر و در یک فایل هم نام id کانتینر قرار دارد.

برای فهمیدن مسیر root داکر می توانید از دستور مربوط به [مشاهده اطلاعات داکر](#_مشاهده_اطلاعات_داکر) استفاده کنید.

به منظور جلوگیری از حجیم شدن فایل لاگ می باید تنظیمات مروبط به Rotate لاگ فایل را انجام دهید.

# پیکربندی داکر

دو راه برای پیکربندی داکر وجود دارد راه اول تغییر در فایل systemd و راه دوم تغییر در فایل daemon.josn است.

## پیکربندی با daemon.josn

در مسیر root داکر([مشاهده اطلاعات داکر](#_مشاهده_اطلاعات_داکر))، با اجرای دستور زیر اگر فایل daemon.josn وجود داشته باشد آن را ویرایش می کنیم و اگر وجود نداشته باشد آن را ایجاد می کنیم.

|  |
| --- |
| # vim /var/lib/docker/daemon.json |

از طریق راهنمای زیر می توانید اطلاعات زیادی در خصوص نحوه پیکربندی داکر از جمله موارد زیر را کسب کنید.

* تغییر IP داکر
* تغییر root داکر
* ثبت registry mirrors
* تنظیمات لاگ کانتینر (تعداد و حجم هر فایل)
* و ...

[راهنمای پیکربندی داکر](https://github.com/AhmadRafiee/Docker_training_with_DockerMe/blob/master/configuration/docker_daemon_config.md)

# والیوم‌(Volumes)

والیوم ها برای ذخیره و نگهداری اطلاعات کانتینرها روی host و یا روی یک storage دیگر استفاده می شوند.

هیچ داده ای نباید در کانتینر‌ها نگهداری شود، داده‌های تولید شده و مورد استفاده توسط کانتینرها پس از راه‌اندازی مجدد(Restart) یا حذف کانتینرها به صورت کامل حذف خواهند شد. بنابراین می توانیم از Docker volumes برای مدیریت داده‌ها استفاده کنیم.



سه روش برای ایجاد والیوم وجود دارد(دو روش اول متداول تر هستند):

* docker volume
* bind mount
* tmpfs mount

## روش docker volumes

در این روش با استفاده از docker volume اطلاعات داخل کانتینر بر روی host و یا بر روی یک storage دیگری ذخیره می‌شود البته این امکان نیز وجود دارد که این اطلاعات و یا کانفیگ‌ها از روی کانتینر دیگری فراخوانی و یا در آن ذخیره شود. بر روی سیستم‌عامل لینوکس وقتی از درایور لوکال استفاده شود اطلاعات به صورت پیش‌فرض در این مسیر /var/lib/docker/volumes/ قرار می‌گیرد. البته این مسیر به صورت پیش‌فرض می‌باشد و شما در هنگام نصب و یا بعد از آن می‌توانید آن را تغییر دهید.

در این روش با استفاده از دستور مدیریتی docker volume می‌توان والیوم ایجاد کرده و دیتای کانتینر را در آن قرار داد. با استفاده از روش والیوم، والیومی که ایجاد شده است داخل کانتینر mount می‌شود. این روش بسیار ساده بوده و استفاده از آن خیلی مرسوم است. با استفاده از این روش می‌توان یک والیوم را برای چند کانتینر نیز استفاده کرد. کانتینر و والیوم به صورت جداگانه مدیریت می‌شود و اگر کانتینر پاک و یا از کار بیافتد والیوم به صورت خودکار تغییری نمی‌کند و پابرجا خواهند بود. همواره باید برای والیوم‌ها یک نام انتخاب کنید که اگر این کار در زمان ایجاد والیوم انجام نشود خود سرو+یس داکر یک اسم رندم برای آن انتخاب می‌کند.

به عنوان مثال در دستور زیر یک کانتینر nginx ساخته می شود که یک والیوم با نام demo-earthly را به دایرکتوری /usr/share/nginx/html در کانتینر وصل می کند.

|  |
| --- |
| # docker run -it --rm --name nginx -p 8080:80 -v demo-earthly:/usr/share/nginx/html nginx |

## روش bind mounts

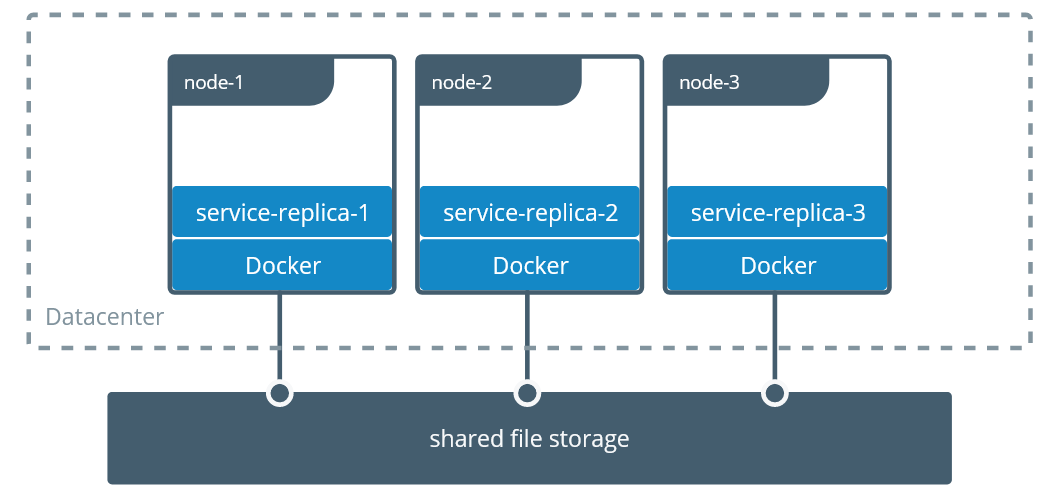
با استفاده از این روش داده‌های داخل یک کانتینر بر روی دایرکتوری‌های سیستم هاست قرار داده و این اطلاعات بر روی سیستم هاست نگهداری می‌شود و هر زمان که برای کانتینر مشکل پیش‌ بیاید این داده‌ها بر روی هاست باقی خواهند ماند.

به عنوان مثال در دستور زیر یک کانتینر nginx ساخته می شود که دایرکتوری هاست $(pwd)"/target به دایرکتوری /usr/share/nginx/html در کانتینر وصل می کند.

|  |
| --- |
| # docker run -it --rm --name nginx -p 8080:80 -v "$(pwd)"/target:/usr/share/nginx/html nginx |

## اشتراک گذاری داده ها بین ماشین ها

هنگام ساخت برنامه های کاربردی، ممکن است لازم باشد چندین نسخه مشابه از یک سرویس را پیکربندی کنید تا به فایل های مشابه دسترسی داشته باشید.



# شبکه داکر

شبکه دارکر شامل ارتباطات بین کانتینرها در یک هاست، ارتباطات بین کانتیرها با هاست خود و ارتباطات بین کانتینرها با چندین هاست و کانتینرهای آن می‌باشد.

داکر به صورت پیش‌فرض بعد از نصب ۳ شبکه‌ برای خود ایجاد می کند.

* Bridge
* None
* Host

هنگام راه اندازی کانتینر با آپشن network– می‌توانید کانتینر خود را به هر کدام از این شبکه‌ها متصل نمایید.

## شبکه‌ی Bridge

داکر به صورت پیش‌فرض شبکه‌ی مورد نیاز برای ارتباط کانتینر با هاست را بر روی درایور bridge پیکربندی می‌کند و در دسترس قرار می‌دهد. اگر برای کانتینر خود شبکه‌ای پیکربندی نکنید به صورت پیش‌فرض به شبکه‌ای به نام docker0 متصل می‌شود. این شبکه همواره بعد از نصب داکر بر روی سیستم‌عامل شما ایجاد می‌شود که مربوط به سرویس‌دهنده‌ی داکر می‌باشد و معمولا هم subnet آن 172.17.0.0 می‌باشد. البته این موارد به صورت کامل قابل پیکربندی است و شما می‌توانید سرویس داکر خود را به گونه‌ای راه‌اندازی کنید که شبکه‌ی bridge آن دارای تنظیمات دیگری باشد.

به صورت پیش‌فرض این bridge بر روی سیستم‌عامل شما دارای ip با شماره‌ی 172.17.0.1 می‌باشد که کانتینرهای شما در همین رنج ip خواهند گرفت.

## شبکه‌ی none

این شبکه برای زمانی است که شما نیاز ندارید داخل کانتینر خود کارت شبکه‌ای داشته باشید. این موضوع بسیار کم کاربرد می‌باشد اما این امکان فراهم است که شما کانتینری راه‌اندازی کنید که هیچ‌گونه ارتباط شبکه‌ای حتی با هاست خودش هم نداشته باشد.

## شبکه‌ی host

در این حالت کانتینر از شبکه‌ی هاست استفاده می‌کند و در این حالت هیچ ایزولیشنی بین هاست و کانتینر وجود نخواهد داشت. به عنوان مثال اگر پورت ۸۰ یا ۳۳۰۶ از کانتینر در دسترس باشد به صورت کامل همان پورت هاست خواهد بود و تفاوتی بین این دو نیست و به گونه‌ای به صورت کامل کانتینر تمام مواردش بر روی هاست قرار خواهد گرفت.

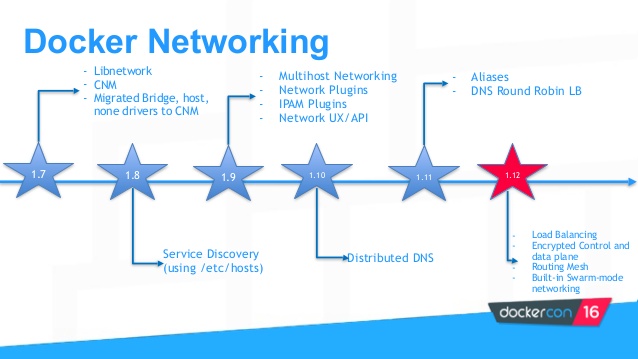
به صورت پیش‌فرض شبکه‌ی bridge به کانتینرها متصل می‌گردد و اگر بخواهید از دو شبکه‌ی دیگر استفاده کنید حتما باید آنها را در راه‌اندازی کانتینر به صورت آپشن وارد کنید. این موضوع به صورت کامل قابل پیکربندی است و می‌توانید همانند تمام موارد دیگر پیش‌فرض شبکه‌ را نیز تغییر دهید.

جهت نمایش درایورهای شبکه داکر می توانید از دستور زیر استفاده کنید.

|  |
| --- |
| # docker network ls |

# DNS سرور داخلی داکر

در تصویر زیر سیر تکامل شبکه‌ی داکر را به ازای نسخه‌های مختلف سرویس داکر مشاهده می‌کنید.



از نسخه‌ی ۱.۱۰ داکر به بعد این امکان فراهم شد که dns server به صورت داخلی برای داکر وجود داشته باشد که با استفاده از آن برخی از موارد تسهیل می‌شود. بعد از این نسخه این امکان فراهم بود که به صورت داخلی service discovery برای تمامی کانتینرها با توجه به اسم کانتینر و یا net-alias و یا لینک‌های آن وجود داشته باشد.

سرویس DNS داخلی داکر با ip به شماره‌ی 127.0.0.11 در دسترس می‌باشد و همواره داخل فایل resolv.conf کانتینر‌ها این nameserver پیکربندی شده است. اگر داخل یک کانتینر بخواهید کانتینر دیگه‌ای که در همان شبکه هست را صدا کنید تنها نیاز است که نام یا ID آن را استفاده کنید این DNS Server ip آن را پیدا کرده و در اختیار شما قرار می‌دهد. در کل این Dns Server اطلاعات تمامی کانتینرهای داخل یک شبکه را دارد و به راحتی می‌توانید از آن استفاده کنید.

راه‌های مختلفی برای ارتباط با این DNS‌ سرور داخلی داکر وجود دارد و می‌‌توان آن را کانفیگ کرد که در ادامه به برخی از این آپشن‌ها اشاره می‌شود:

نام کانتینر: با استفاده از این آپشن شما برای کانتینر خود یک نام انتخاب می‌کنید که به صورت کامل می‌توانید با همان نام در شبکه‌ی خود کانتینر آن را صدا کنید. البته به صورت خودکار نام برای هر کانتینر ایجاد می‌شود که آن نام نیز معتبر است. با ID هر کانتینر هم می‌توان آن را صدا کرد و بررسی نمود.

|  |
| --- |
| --name=CONTAINER-NAME |

نام alias: این نام در شبکه‌ی کانتینر معتبر بوده و آن کانتینر علاوه بر نام خود با این نام نیز شناخته می‌شود. شما با اضافه کردن این آپشن می‌توانید برای کانتینر خود یک نام دیگر در DNS قرار دهید.

|  |
| --- |
| --network-alias=ALIAS or --alias=ALIAS |

برای کسب اطلاعات بیشتر از لینک زیر استفاده کنید.

[استفاده از DNS سرور داخلی داکر](https://virgool.io/DockerMe/28docker-internal-dns-server-uim7yogffpue)

# آشنایی با Command Line داکر

با استفاده از دستور زیر می توانید راهنمای دستورات داکر را مشاهده کنید.

|  |
| --- |
| # docker --help |

در بخش Commands دستورات قدیمی تر داکر دیده می شوند که هنوز هم قابل استفاده می باشند(با توجه به زیاد شدن دستورات ادامه پیدا نکرد) و در بخش Management Commands دستورات داکر به صورت دسته بندی شده قرار دارند به عنوان مثال برای دیدن دستورات مربوط به کانتینر از دستور زیر استفاده کنید.

|  |
| --- |
| # docker container --help |

## معرفی مجموعه دستورات Container

با دستور زیر می توانید راهنمای کلیه دستورات این مجموعه را ببینید.

|  |
| --- |
| # docker container --help |

start: کانتینری که قبلا متوقف شده است را مجددا استارت می کند.

stop: کانتینری که در حال اجرا را متوقف می کند.

restart: کانتینر را ریست می کند.

pause: کانتینر pause می شود(تفاوت اصلی بین حالت های pause و stop این است که حافظه در حالت stop یک کانتینر پاک می شود، در حالی که در حالت pause، حافظه آن دست نخورده می ماند).

unpause: کانتینر unpause می شود.

ls: جهت نمایش لیست کانتینرها استفاده می شود. در حالت عادی فقط کانتینرهای در حال اجرا نمایش داده می‌شوند برای نمایش لیست همه کانتینرها می توانید از پرچم -a استفاده کنید.

run: برای ایجاد یک کانتینر استفاده می شود.

برای مشاهده تمام آپشن های docker container run از دستور زیر استفاده می کنیم.

|  |
| --- |
| # docker container run --help |

بعضی از آپشن های مهم run:

* -i (--interactive) به معنی این است که container به صورت تعاملی ایجاد شود یعنی امکان ارتباط با آن را داشته باشید.
* -t به این معناست که یک tty به container اختصاص داده شود. که به صورت خلاصه به معنی آن است که شما امکان لاگین بر روی container را داشته باشید.
* -d کانتینر را در background را‌ه‌اندازی می‌کند و تنها id آن را نمایش می‌دهد(درصورتیکه استفاده نشود container اجرا می‌شود و شما به داخل container می‌روید، یعنی مثلا اگر کانتیر centos را ساخته باشید به صورت اتوماتیک وارد فضای centos که کانتینر شما است خواهید شد و برای خروج از container باید دستور exit را وارد کنید تا به فضای سرور اصلی خود بازگردید).
* --name نام کانتینر را مشخص می کند.
* --hostname تعریف hostname کانتینر
* -p پورت کانتینر را مشخص می کند(به طور مثال –p 8080:80 پورت 8080 هاست را به پورت 80 کانتینر وصل(Map) می کند).
* -v برای ساخت والیوم استفاده می شود.
* -e متغیرهای محلی به کانتینر اضافه می‌کند.

create: همان کار run را انجام می دهد با این تفاوت که بعد از ایجاد کانتینر آن را اجرا نمی کند.

cp: با استفاده از این دستور می‌توان فایل یا دایرکتوری رو بین کانتینر و هاست جابه‌جا کرد. هم می‌توان از هاست به کانتینر و هم از کانتینر به هاست منتقل کرد.

|  |
| --- |
| # docker cp example.txt my-container:/example.txt |

diff: توضیحی از تغییرات داخل فایل یا دایرکتوری‌های داخل کانتینر در اختیار ما قرار می‌دهد. این دستور نیز هیچ‌گونه آپشنی ندارد. در واقع تفاوت بین Image و کانتینر را نشان می دهد.

* سمبل A: یعنی اینکه یک فایل یا دایرکتوری اضافه شده است.
* سمبل D: یعنی اینکه فایل یا دایرکتوری پاک شده است.
* سمبل C: یعنی اینکه فایل یا دایرکتوری تغییر کرده است.

|  |
| --- |
| # docker diff my-container  C /tmp  D /tmp/hsperfdata\_jenkins  A /tmp/hsperfdata\_jenkins/8  D /tmp/jetty-0.0.0.0-8080-war-\_-any-4465523861556992604.dir  A /tmp/jffi8867713240318490067.tmp  D /tmp/jna--1712433994  A /tmp/winstone1616795870189931273.jar |

exec: با استفاده از این دستور می‌توانید در کانتینرهای در حال کار دستور اجرا کنید.

* آپشن detach: با استفاده از این آپشن دستور شما در background کانتینر اجرا می‌شود.
* آپشن detach-keys: با این آپشن کلید‌‌های مربوط به جدا شدن از کانتینر را نادیده می‌گیرد.
* آپشن env: می‌توان متغییرهای محیطی (environment variables) برای آن تنظیم کرد.
* آپشن interactive: با استفاده از این دستور می‌توانید به صورت تعاملی با کانتینر ارتباط داشته باشید.
* آپشن privileged: دستور را با دسترسی بالا اجرا می‌کند.
* آپشن tty: با این دستور می‌توانید از کانتینر خود یک tty داشته باشید.
* آپشن user: می‌توان user یا uid و group یا gid برای آن تنظیم کرد.
* آپشن workdir: می‌توانید دستور مد نظر خود را در دایرکتوری که مشخص می‌کنید اجرا شود.

|  |
| --- |
| # docker exec -it my\_container bash  # docker exec -it -e VAR=1 my\_container bash |

Inspect: با استفاده از این دستور شما می‌توانید اطلاعات زیادی در مورد هر کدام از آبجکت‌های داکر داشته باشید. به عنوان مثال می‌توانید اطلاعات در مورد کانتینر و ایمیج و والیوم و یا نتورک‌های که ایجاد شده است داشته باشید.

Kill: با استفاده از این دستور یک تا تعدادی از کانتینرها را می‌توان kill کرد. تنها آپشن این دستور s- می‌باشد که با استفاده از آن می‌توان سیگنال مد نظر خود را به کانتینر ارسال کرد.

|  |
| --- |
| # docker kill my\_container |

با استفاده از دستور kill -l در لینوکس می‌توانید لیست سیگنال‌های دستور kill را مشاهده کنید. شما هم می‌توانید عدد سیگنال مد نظرتون یا اختصار آن را قرار بدید. دقیقا همانند مثال زیر که SIGHUP همان HUP بوده که عدد سیگنال آن ۱ می‌باشد. هر سه دستور پایین دقیقا یک کار را انجام می‌دهند.

|  |
| --- |
| # docker kill --signal=SIGHUP my\_container  # docker docker kill --signal=HUP my\_container  # docker kill --signal=1 my\_container |

Logs: برای مشاهده log های کانتینر استفاده می شود.

|  |
| --- |
| # docker logs my-container |

آپشن‌ها

--deails : نمایش جزئیات اضافی.

--follow / -f : نمایش خروجی گزارش به صورت دنبال کردن.

--since : نمایش گزارش‌های از این زمان به بعد (مثلاً 2013-01-02T13:23:37Z) یا نسبی (مثلاً 42m برای 42 دقیقه پیش به بعد).

--until : نمایش گزارش‌های قبل از زمان خاص (مثلاً 2013-01-02T13:23:37Z) یا نسبی (مثلاً 42m برای 42 دقیقه پیش).

|  |
| --- |
| # docker logs -f --until=2s test |

--tail / -n تعداد خطوطی که باید از انتهای گزارش‌ها نشان داده شود.

|  |
| --- |
| # docker logs -f –tail 10 test |

port: لیست پورت‌های که از کانتینر به سرور مپ شده است را نمایش می‌دهد. هیچ‌گونه آپشنی هم ندارد(خروجی این دستور نشان می دهد چه پورتی از کانتینر به چه پورتی از هاست مپ شده است).

|  |
| --- |
| # docker container port test |

rm: کانتینرهای متوقف شده را پاک می‌کند.

برای پاک کردن کانتینرهای در حال اجرا می توانید از پرچم –f استفاده کنید.

|  |
| --- |
| # docker container rm -f test |

stats: این دستور state کانتینرها را نشان می‌دهد(میزان استفاده از منابع نظیر رم و cpu)

|  |
| --- |
| # docker container stats test |

## معرفی مجموعه دستورات Image

دستورات ls, inspect, rm همانند دستورات کانتینر عمل می کند.

pull: با این دستور شما image مد نظر خود را از ریجیستری‌های مختلف دانلود و دریافت می‌کنید. این دستور ساده بوده و آپشن‌های زیادی ندارد. بر این اساس شما نام ایمیج به همراه تگ مد نظر خود را در جلوی دستور قرار می‌دید تا برای شما اون ایمیج را دریافت کند. نکته‌ اینکه اگر تگ را انتخاب نکنید به صورت پیش‌فرض تگ latest را برای ایمیج قرار می‌دهد.

|  |
| --- |
| # docker pull ubuntu:latest  docker pull -a busybox |

* بدون آپشن: ایمیج را با همان تگی که جلوی آن قرار داده شده است دانلود می‌کند.
* آپشن -a : این آپشن تمام تگ‌های آن ایمیج در ریپوزیتوری را برای شما دانلود می‌کند.
* آپشن disable-content-trust- : فرآیند چک کردن ایمیج را انجام نمی‌دهد.

اگر بخواهیم از یک ریجیستری به خصوص ایمیج را دریافت کنیم ابتدا نام ریجیستری را وارد کرده سپس نام ایمیج و بعد تگ آن را وارد می‌کنیم. به مثال زیر توجه کنید.

|  |
| --- |
| # docker pull docker.dockerme.ir/ubuntu:latest |

push: با استفاده از این دستور می‌توان ایمیج‌ها را داخل ریجیستری قرار داد.

تنها آپشنی که این دستور دارد disable-content-trust می‌باشد که مربوط به چک و بررسی ایمیج می‌باشد.

نکته‌ی مهم اینکه قبلا باید حتما داخل یک ریجیستری وارد شده باشید تا بتوانید که ایمیج‌های خود را داخل آن قرار دهید.

save: با کمک این دستور می‌توانید یک یا چند ایمیج را در یک فایل tar ذخیره کنید.

|  |
| --- |
| # docker save -o NAME wordpress:latest mysql:5.7 |

در این دستور دو ایمیج wordpress:latest mysql:5.7 در یک فایل با نام Name دخیره می شود.

load: با کمک این دستور می توانید ایمیج های ذخیره شده را در داکر بارگذاری کنید.

|  |
| --- |
| # docker load -i Name |

tag: با استفاده از این دستور می‌توانید تگ (tag یا برچسب) مربوط به ایمیج را تغییر نام می‌دهید و می‌توانید برای ایمیج خود نام و تگ جدید قرار بدید. این دستور آپشن اضافی ندارد و به صورت زیر می‌توانید تگ هر ایمیج را تغییر دهید و از آن یک ایمیج جدید با تگ جدید داشته باشید. این آپشن‌ها بسیار کاربردی بوده و استفاده می‌شود.

|  |
| --- |
| # docker tag alpine:latest localhost:5000/alpine:latest |

Login: با استفاده از این دستور می‌توانید داخل ریجیستری لاگین کنید. اطلاعات مربوط به لاگین شما به صورت رمزنگاری شده در داکر کلاینت شما ذخیره می‌شود و تا زمانی که از آن ریجیستری logout نکنید این اطلاعات برقرار می‌باشد. نکته‌ اینکه می‌توانید به چندین ریجیستری به صورت همزمان نیز لاگین کنید. می‌توانید از روش‌های مختلفی نیز برای قرار دادن پسورد و اطلاعات مربوط به لاگین خود نیز استفاده کنید. این موضوع شامل ریجیستری‌های عمومی و خصوصی می‌شود.

آپشن p-: با استفاده از آن می‌توانید پسورد را در دستور قرار بدهید. این موضوع از نظر امنیتی مشکل داشته و پسورد در history سرور باقی خواهد ماند.

آپشن password-stdin: پسورد را از روی یک فایل دریافت می‌کند.

آپشن u-: آپشن مربوط به حساب کاربری می‌باشد.

برای مشخص‌تر شدن موضوع به مثال‌های زیر توجه کنید. دقت کنید اگر اطلاعات کاربری را در دستور وارد شده قرار ندهید بعد از اعمال دستور این موارد که شامل user و pass می‌باشد را از شما سوال می‌کند.

|  |
| --- |
| # docker login https://docker.dockerme.ir  # docker login --username admin https://docker.dockerme.ir  # cat test.txt | docker login --username admin --password-stdin https://docker.dockerme.ir |

اگر لینک ریجیستری را نیز وارد نکنید به صورت پیش‌فرض به https://hub.docker.com وصل خواهد شد که یک ریجیستری عمومی است و برای لاگین به آن و قرار دادن ایمیج‌های خود در آن از طریق این لینک باید ثبت‌نام انجام دهید.

logout: با این دستور از ریجیستری که لاگین کرده بودیم قبلا خارج می‌شود و اطلاعات ارتباط با آن برداشته می‌شود.

# Dockerfile

Dockerfile یک فایل متنی است که توصیف می‌کند که چگونه Docker container باید ساخته شود. با استفاده از Dockerfile، می‌توانید تمام مراحل مورد نیاز برای ساختن یک ایمیج Docker را تعریف کنید، از شروع کانتینر با ایمیج پایه (base image) گرفته تا اضافه کردن فایل‌ها، تنظیمات، نرم‌افزارها و پیکربندی‌های مورد نیاز برنامه‌های شما.

به طور معمول، Dockerfile حاوی دستورها و دستورالعمل‌هایی است که Docker می‌تواند آن‌ها را اجرا کند تا محیط داخل کانتینر شما را تعریف کند. این دستورالعمل‌ها می‌توانند شامل اجرای دستورات در محیط کانتینر، تنظیم متغیرها، کپی فایل‌ها از محیط میزبان (host) به کانتینر، تعریف محیط متغیرها و غیره باشند.

با اجرای دستور `docker build` و ارائه مسیر پوشه‌ی حاوی Dockerfile، Docker اقدام به خواندن Dockerfile و ساخت تصویر Docker از آن می‌کند. این تصویر سپس می‌تواند با استفاده از دستور `docker run` به یک کانتینر تبدیل شود و برنامه‌های شما درون آن اجرا خواهند شد.

## Dockerfile best practice

در ادامه، توضیحاتی از Dockerfile Best Practices یا بهترین روش‌های استفاده از Dockerfile ارائه خواهد شد:

1. استفاده از تصویر پایه (Base Image):

- انتخاب تصویر پایه‌ای که نیازهای برنامه‌ی شما را پوشش دهد.

- استفاده از تصویر‌های رسمی و معتبر از Docker Hub یا منابع اصلی Docker.

- از تصویر‌های کوچک‌تر و به‌روزتر استفاده کنید تا اندازه کانتینر کاهش یابد.

2. استفاده از .dockerignore:

- از فایل .dockerignore برای انتخاب فایل‌ها و پوشه‌هایی که نباید در کانتینر کپی شوند، استفاده کنید.

- جلوگیری از کپی فایل‌های بی‌اهمیت و مورد نیاز نشده به داخل کانتینر.

3. کشیدن Layer‌ها:

- دستورات که زمان زیادی برای اجرا می‌شوند را در قسمت‌های بالای Dockerfile قرار دهید تا از کش برای لایه‌ها بهره ببرید.

- تغییرات کم‌حجم را در زمان زودتر از دستورات بزرگ‌تر انجام دهید تا زمان ساخت تصویر کاهش یابد.

4. کمترین تعداد لایه‌ها:

- تلاش کنید کمترین تعداد لایه‌ها را در تصویر نهایی داشته باشید.

- استفاده از دستورات چندخطی با \ (backslash) به‌جای دستورهای جداگانه کمک به کاهش تعداد لایه‌ها می‌کند.

5. استفاده از Alpine و Scratch:

- استفاده از تصویر Alpine برای کاهش اندازه کانتینر و افزایش سرعت دانلود.

- استفاده از تصویر Scratch برای ساخت کانتینرهای کوچک و کاهش حجم نهایی.

6. ترتیب دستورات:

- دستورات با تغییر کمتر را قبل از دستورات با تغییر بیشتر قرار دهید.

- بهینه‌سازی دستورات کپی و نصب وابستگی‌ها با توجه به فایل‌ها و پوشه‌های پرتکرار در طول زمان.

7. استفاده از Docker Multi-Stage Build:

- برای تولید تصویر‌های کم حجم و حذف وابستگی‌های توسعه‌ای، از Docker Multi-Stage Build استفاده کنید.

8. پاک‌سازی بعد از نصب وابستگی‌ها:

- پس از نصب وابستگی‌ها، اجرای دستورات پاک‌سازی و حذف فایل‌های غیرضروری را انجام دهید.

9. تنظیم متغیرها با ENV:

- از ENV برای تنظیم متغیرهای محیطی استفاده کنید، تا انعطاف‌پذیری و قابلیت پیکربندی برنامه‌ها را افزایش دهید.

10. استفاده از Entrypoint:

- استفاده از دستور Entrypoint به جای CMD برای اجرای دستورهای ورودی به کانتینر.

- ممکن است این کار به مدیریت بهتر اجرای برنامه‌ها و آسیب‌پذیری‌های امنیتی کمک کند.

11. اجرای غیر root:

- اجرای کانتینر به عنوان کاربر غیر root برای بهبود امنیت.

همچنین، توصیه می‌شود که همواره Dockerfile خود را با دقت کنترل کنید، از تکرار دستورات جلوگیری کنید و از ابزارهای مانند `dockerfile\_lint` برای بررسی اشکالات محتمل در Dockerfile استفاده کنید. با رعایت این بهترین روش‌ها، می‌توانید تصاویر Docker کم حجم، امن و کارآمد تری ایجاد کنید.

## نوشتن DockeFile

### Syntax

* Dockerignore

فایل .dockerignore یک فایل مشابه فایل .gitignore است که به Docker اطلاع می‌دهد که چه فایل‌ها و پوشه‌ها را نباید در فرآیند ساخت تصویر Docker استفاده کند. با استفاده از این فایل، شما می‌توانید فایل‌ها و پوشه‌هایی که به عنوان اجزای مزاحم و غیرضروری در تصویر نهایی Docker شما هستند، را از ساخت تصویر حذف کنید و باعث کاهش اندازه نهایی تصویر می‌شوید.

به عبارت دیگر، فایل .dockerignore به Docker می‌گوید که این فایل‌ها و پوشه‌ها در محیط Docker کپی نشوند و نادیده گرفته شوند. این فایل به صورت پوشه‌محور کار می‌کند و می‌توانید الگوهای مناسب برای نادیده گرفتن فایل‌ها و پوشه‌ها را در آن قرار دهید.

برخی از موارد که می‌توانید در فایل .dockerignore قرار دهید:

1. فایل‌های کش:

معمولاً فایل‌های کش و فولدر‌های موقتی که توسط برنامه‌های شما ایجاد می‌شوند، نیازی به وجود در داخل کانتینر ندارند.

2. فایل‌های لازم‌نیست:

فایل‌های کانفیگ، فایل‌های مستندات، فایل‌های لازم‌نیست که به اجرای برنامه شما نیاز ندارند و در کانتینر نیازی به وجود ندارند.

3. فولدرهای تولید شده:

فولدرهایی که توسط فرآیند ساخت یا اجرای برنامه‌های شما ایجاد می‌شوند و به اجرای برنامه نیاز ندارند.

4. فایل‌های مخفی:

فایل‌هایی که با نقطه شروع می‌شوند و به طور معمول فایل‌های مخفی در سیستم‌عامل هستند و نباید در کانتینر نیاز به وجود داشته باشند.

5. وابستگی‌های توسعه‌ای:

در زمان ساخت تصویر نهایی، اجرای وابستگی‌های توسعه‌ای به کانتینر نیاز ندارد و می‌توانند در فایل .dockerignore قرار گیرند.

مطمئن شوید که هنگام تعیین محتوای فایل .dockerignore، فایل‌ها و پوشه‌های مهم را از دست ندهید و برنامه‌ی شما بدون مشکل اجرا می‌شود. با تنظیم دقیق فایل .dockerignore می‌توانید از کاهش حجم تصویر و بهبود زمان ساخت آن بهره‌برده و تصویر کم‌حجم‌تر و کارآمد‌تری ایجاد کنید.

* FROM

دستور FROM یکی از اصلی‌ترین دستورات در Dockerfile است و برای تعیین تصویر پایه (base image) برای ساخت کانتینر Docker استفاده می‌شود. هنگامی که یک تصویر Docker ایجاد می‌شود، باید از یک تصویر پایه شروع شود که شامل سیستم‌عامل و پکیج‌های مورد نیاز برای اجرای برنامه‌هاست.

ساختار دستور FROM به صورت زیر است:

|  |
| --- |
| # FROM [نام\_تصویر]:[تگ] |

[نام\_تصویر]: نام تصویر پایه که می‌خواهید از آن استفاده کنید. این می‌تواند تصویر رسمی از Docker Hub یا یک تصویر سفارشی باشد که خود شما ساخته‌اید.

[تگ]: تگ (Tag) تصویر پایه که مشخص می‌کند از کدام نسخه از تصویر استفاده می‌شود. این بخش اختیاری است و اگر نام‌گذاری تگ در تصویر پایه انجام نشده باشد، از تگ پیش‌فرض (معمولاً latest) استفاده خواهد شد.

نمونه‌ای از استفاده از دستور FROM در Dockerfile:

|  |
| --- |
| # تصویر پایه ما از تصویر رسمی اوبونتو با تگ 20.04 استفاده می‌کند  FROM ubuntu:20.04  # دستورات دیگر برای ساخت کانتینر... |

در این مثال، Dockerfile با استفاده از تصویر رسمی اوبونتو با تگ 20.04 ساخته می‌شود. این تصویر پایه حاوی سیستم‌عامل اوبونتو با نسخه 20.04 و پکیج‌های مورد نیاز برای اجرای برنامه‌ها است که برای ساخت کانتینر از آن استفاده می‌شود.

* MAINTAINER

در نسخه‌های قدیمی‌تر Docker، دستور MAINTAINER برای تعیین نام و اطلاعات تماس مالک یا مدیر Dockerfile استفاده می‌شد. با این دستور، شما می‌توانستید نام و ایمیل یا اطلاعات تماس دیگر مالک یا مدیر Dockerfile را مشخص کنید.

ساختار دستور MAINTAINER به صورت زیر است:

|  |
| --- |
| MAINTAINER [نام مالک]:[ایمیل یا اطلاعات تماس] |

اما از نسخه ۱.۱۳ Docker به بعد، توصیه می‌شود از دستور MAINTAINER استفاده نکنید و به جای آن از برچسب (Label) LABEL استفاده کنید. دستور LABEL امکان اضافه کردن برچسب‌ها و متادیتا به تصویر Docker را فراهم می‌کند که از طریق دستور docker inspect قابل مشاهده هستند و اطلاعات مورد نیاز را برای مدیریت و نگهداری تصویر فراهم می‌کنند.

نمونه‌ای از استفاده از دستور LABEL به جای MAINTAINER در Dockerfile:

|  |
| --- |
| # تعیین نام مالک و ایمیل با استفاده از LABEL  LABEL maintainer="John Doe <john.doe@example.com>"  # دستورات دیگر برای ساخت کانتینر... |

به این ترتیب، به‌جای دستور MAINTAINER، اطلاعات مالکیت و تماس با استفاده از برچسب LABEL در Dockerfile تعیین می‌شود. این تغییر در توصیه‌های جدید Docker صورت گرفته است تا با استانداردهای به‌روز و بهتر در زمینه‌ی مدیریت تصاویر Docker سازگار شود.

* RUN

دستور RUN یکی دیگر از دستورات اساسی در Dockerfile است که برای اجرای دستورات و دستورالعمل‌ها در محیط کانتینر استفاده می‌شود. با استفاده از این دستور، می‌توانید دستورات مورد نیاز را برای نصب وابستگی‌ها، تنظیم محیط، کپی فایل‌ها و هرگونه دستور دیگری که نیاز است در هنگام ساخت کانتینر اجرا شود، تعریف کنید.

ساختار دستور RUN به صورت زیر است:

|  |
| --- |
| RUN [دستور] |

نمونه‌ای از استفاده از دستور RUN در Dockerfile:

|  |
| --- |
| # تصویر پایه ما از تصویر رسمی اوبونتو با تگ 20.04 استفاده می‌کند  FROM ubuntu:20.04  # اجرای دستور برای به‌روزرسانی منابع نرم‌افزاری در اوبونتو  RUN apt-get update  # نصب یک پکیج با استفاده از دستور apt-get install  RUN apt-get install -y package-name  # دستورات دیگر برای ساخت کانتینر... |

در این مثال، ما از دستور RUN برای اجرای دو دستور apt-get update و apt-get install استفاده کردیم. این دستورات به ترتیب منابع نرم‌افزاری را به‌روزرسانی و یک پکیج را نصب می‌کنند. این دستورات در زمان ساخت تصویر، درون کانتینر اجرا می‌شوند و اجرای آن‌ها نتیجه‌ای مشخص (نصب پکیج) را در تصویر نهایی داشته و در زمان اجرای کانتینر بر اساس تغییرات اعمال شده در آن انجام می‌شود.

* EXPOSE

دستور EXPOSE در Dockerfile برای تعریف پورت‌هایی استفاده می‌شود که برنامه‌های درون کانتینر می‌توانند به آن‌ها listen کنند. این دستور، Docker را مطلع می‌کند که کدام پورت‌ها باید در کانتینر اجرا شده برای ارتباط با برنامه‌های بیرون از کانتینر مورد استفاده قرار بگیرند.

ساختار دستور EXPOSE به صورت زیر است:

|  |
| --- |
| EXPOSE [پورت\_شماره1] [پورت\_شماره2] ... |

اصولاً دستور EXPOSE فقط تعیین پورت‌هایی که برنامه درون کانتینر به آن‌ها نیاز دارد ولی برای اتصال برنامه‌های خارجی به کانتینر، باید از اپشن -p یا --publish هنگام اجرای دستور docker run استفاده کنید.

نمونه‌ای از استفاده از دستور EXPOSE در Dockerfile:

|  |
| --- |
| # انتخاب تصویر پایه  FROM python:3.8  # کپی فایل‌های برنامه  COPY app.py .  # تعریف پورت که برنامه به آن listen می‌کند  EXPOSE 8000  # تعریف دستوری که باید اجرا شود وقتی کانتینر اجرا می‌شود  CMD ["python", "app.py"] |

در این مثال، کانتینر از تصویر پایه‌ای که Python نسخه 3.8 را دارد، ایجاد می‌شود. برنامه‌ی Python که فایل app.py نام دارد به کانتینر کپی می‌شود و سپس دستور EXPOSE 8000 تعیین می‌کند که برنامه به پورت 8000 مشغول شود. و در نهایت با دستور CMD، برنامه‌ی Python اجرا می‌شود.

* ENV

دستور ENV در Dockerfile برای تنظیم متغیرهای محیطی (Environment Variables) در کانتینر استفاده می‌شود. متغیرهای محیطی اطلاعاتی هستند که مقادیر مشخصی را برای اجرای برنامه‌ها درون کانتینر تعیین می‌کنند. با استفاده از متغیرهای محیطی، می‌توانید پیکربندی‌ها و تنظیمات مختلف را در کانتینرها اعمال کنید و کاربردهای مختلفی را برای آن‌ها داشته باشید.

ساختار دستور ENV به صورت زیر است:

|  |
| --- |
| ENV [نام\_متغیر] [مقدار] |

نمونه‌ای از استفاده از دستور ENV در Dockerfile:

|  |
| --- |
| # تنظیم متغیر محیطی با نام VERSION و مقدار 1.0  ENV VERSION=1.0 |

در این مثال، متغیر محیطی با نام VERSION و مقدار 1.0 تعیین شده است. در صورتی که این Dockerfile را برای ساخت کانتینر استفاده کنید، متغیر VERSION با مقدار 1.0 در داخل کانتینر ایجاد شده و می‌توانید از آن برای پیکربندی و یا استفاده در برنامه‌ها و اسکریپت‌های خود استفاده کنید.

مزیت استفاده از ENV این است که شما می‌توانید متغیرها را به‌صورت داینامیک و قابل پیکربندی تعیین کنید. به این ترتیب، هنگام ساخت کانتینر، می‌توانید مقادیر متغیرها را به عنوان پارامترهای ورودی از خارج از Dockerfile تعیین کنید و تغییرات در برنامه‌ها و تنظیمات کانتینر را بدون نیاز به اصلاح Dockerfile اعمال کنید.

* ARG

دستور ARG یکی دیگر از دستورات مهم در Dockerfile است که برای تعریف متغیرها (Variables) در محیط ساخت تصویر Docker استفاده می‌شود. متغیرهای تعریف شده با ARG می‌توانند در طول مراحل ساخت تصویر تغییر کنند و به‌عنوان پارامترها برای دستورات مختلف مورد استفاده قرار گیرند.

ساختار دستور ARG به صورت زیر است:

|  |
| --- |
| ARG [مقدار\_پیش‌فرض]=[نام متغییر] |

[نام\_متغیر]: نام متغیری که می‌خواهید تعریف کنید.

[مقدار\_پیش‌فرض]: مقدار پیش‌فرض متغیر (اختیاری) که در صورتی که این متغیر به صورت مستقیم از سطر دستور ساخت تصویر تعیین نشود، از آن استفاده می‌شود.

استفاده از متغیرهای ARG برای پارامتری‌سازی Dockerfile می‌تواند مفید باشد، به‌ویژه زمانی که می‌خواهید تصویر Docker را با تنظیمات مختلف ساخته و پارامترهای مختلفی مانند نسخه‌ها، مسیرها و تنظیمات متغیر را در زمان ساخت تصویر تغییر دهید.

نمونه‌ای از استفاده از دستور ARG در Dockerfile:

|  |
| --- |
| # تعریف متغیر نام کاربری با مقدار پیش‌فرض "appuser"  ARG USERNAME=appuser  # استفاده از متغیر در ایجاد کاربر در کانتینر  RUN useradd -m -s /bin/bash $USERNAME  # دستورات دیگر برای ساخت کانتینر... |

در این مثال، با استفاده از ARG متغیر USERNAME تعریف شده است و مقدار پیش‌فرض آن به "appuser" تنظیم شده است. سپس در دستور RUN از این متغیر برای ایجاد یک کاربر با نام مقدار دلخواه استفاده شده است. اگر مقداری برای USERNAME به عنوان آرگومان از سطر دستور ساخت تصویر تعیین شود، مقدار پیش‌فرض جایگزین می‌شود و کاربر با نام تعیین‌شده در آرگومان ایجاد خواهد شد.

* ADD و COPY

در Dockerfile، دو دستور ADD و COPY برای کپی فایل‌ها و پوشه‌ها از میزبان (host) به داخل کانتینر استفاده می‌شوند. با این دستورها، می‌توانید فایل‌ها، سندباک‌ها، برنامه‌ها، اسکریپت‌ها و سایر منابع مورد نیاز برای اجرای برنامه‌ها را به کانتینر انتقال دهید. اما دو دستور ADD و COPY با چند تفاوت اساسی همراه هستند که در زیر توضیح داده می‌شوند:

COPY:

دستور COPY فقط برای کپی کردن فایل‌ها و پوشه‌ها از میزبان به کانتینر استفاده می‌شود. این دستور به شکل زیر استفاده می‌شود:

|  |
| --- |
| COPY app.py /app/  COPY data/ /app/data/ |

توجه داشته باشید که آدرس مقصد در کانتینر باید یک مسیر معتبر درون کانتینر باشد.

ADD:

دستور ADD همچنین برای کپی کردن فایل‌ها و پوشه‌ها از میزبان به کانتینر استفاده می‌شود، اما از این دستور می‌توان برای انجام عملیات‌های بیشتری مانند خودکار استخراج فایل‌های زیپ شده (auto-extraction)، کپی فایل‌ها از URL و ... نیز استفاده کرد.

|  |
| --- |
| ADD app.tar.gz /app/  ADD http://example.com/data.txt /app/ |

توجه داشته باشید که اگر منبع یک فایل زیپ شده باشد، با استفاده از دستور ADD، Docker به‌طور خودکار آن را استخراج می‌کند.

اگر برای کپی کردن فایل‌ها به کانتینر از میزبان استفاده می‌کنید و نیاز به اجرای عملیات خاصی ندارید، بهتر است از دستور COPY استفاده کنید. اگر نیاز به عملیات خاصی دارید مانند خودکار استخراج فایل‌ها یا کپی از URL، می‌توانید از دستور ADD استفاده کنید.

* VOLUME

در Dockerfile، دستور VOLUME برای تعریف یک مسیر (path) در داخل container استفاده می‌شود که به عنوان یک Volume (حافظه دائمی) مورد استفاده قرار می‌گیرد. Volume ها به container اجازه می‌دهند تا داده‌ها را بین host و container ها به اشتراک بگذارند یا داده‌ها را در دستگاه میزبان (host) ذخیره کنند. از ویژگی اصلی Volume ها این است که تغییرات در داده‌ها به صورت پایدار باقی می‌مانند حتی پس از اتمام یا حذف container.

توضیح دستور VOLUME در Dockerfile به صورت زیر است:

تعریف یک VOLUME:

برای تعریف یک VOLUME در Dockerfile، از دستور VOLUME استفاده می‌شود. معمولاً این دستور به صورت زیر به‌کار می‌رود:

|  |
| --- |
| VOLUME [مسیر] |

در اینجا، [مسیر] مسیری است که می‌خواهید به عنوان Volume در container تعریف شود. این مسیر می‌تواند یک مسیر مطلق یا نسبی باشد. به طور معمول، مسیرهای مطلق (مبنا شده از ریشه فضای فایل) معمولاً ترجیح داده می‌شوند تا اطمینان حاصل شود که مسیر در تمام اجراها یکسان باقی بماند.

به عنوان مثال:

|  |
| --- |
| VOLUME /var/data |

در این مثال، مسیر /var/data به عنوان یک Volume در container تعریف شده است.

توضیح مسیر های چندگانه:

می‌توانید چندین مسیر را با استفاده از چندین دستور VOLUME تعریف کنید. به عنوان مثال:

|  |
| --- |
| VOLUME /var/data  VOLUME /var/logs |

در این مثال، دو مسیر /var/data و /var/logs به عنوان دو Volume مجزا در container تعریف شده‌اند.

ارث بری Volume:

اگر یک تصویر از یک تصویر دیگری به ارث برده شده باشد و در تصویر والد دستور VOLUME تعریف شده باشد، تصویر فرزند همچنین این Volume را ارث بری می‌کند. اما می‌توان در تصویر فرزند این VOLUME را بازنویسی کرد یا به کلی حذف کرد.

برای مثال:

|  |
| --- |
| FROM base\_image  VOLUME /var/data |

در این حالت، تصویر base\_image یک Volume به نام /var/data تعریف کرده است که توسط تصویر Dockerfile فرزند به ارث برده می‌شود.

Volume های استفاده شده در تصویرهای مختلف:

اگر در تصویرهای مختلف از یک Volume با همان مسیر استفاده شده باشد، تغییرات در هر یک از container ها بر روی همان Volume تأثیر خواهند گذاشت.

به عنوان مثال، اگر دو تصویر با استفاده از Dockerfile های زیر تعریف شوند:

Dockerfile تصویر اول:

|  |
| --- |
| FROM base\_image  VOLUME /var/data |

Dockerfile تصویر دوم:

|  |
| --- |
| FROM base\_image  VOLUME /var/data |

در این حالت، هر دو تصویر از یک Volume به نام /var/data استفاده کرده‌اند و تغییرات در هر یک از container ها به همان Volume اعمال می‌شود.

توجه داشته باشید که تنها اطلاعات درون Volume ها ذخیره می‌شوند و نه سایر اطلاعات container نظیر فایل‌های اجرایی یا کانفیگ‌ها.

* USER

با استفاده از این آپشن می‌توانید کاربر پیش‌فرضی که فرآیند‌ها در کانتینر اجرا می‌شوند، را مشخص کنید.

معمولاً برای امنیت بهتر و کاهش ریسک‌های مربوط به اجرای فرآیند‌ها با اکانت root درون کانتینر، توصیه می‌شود از این آپشن استفاده کنید و کاربر غیر root را به عنوان کاربر اجرایی مشخص کنید.

سینتکس استفاده از آپشن USER در Dockerfile به شکل زیر است:

|  |
| --- |
| USER <user>[:<group>] |

<user>: نام کاربر اجرایی که می‌خواهید فرآیند‌ها با آن اجرا شوند.

<group> (اختیاری): گروه مربوط به کاربر اجرایی.

توجه داشته باشید که نام کاربر و گروه مشخص شده باید از قبل در داخل کانتینر موجود باشد. برای تعریف کاربر و گروه‌های مورد نیاز، می‌توانید از دستورات RUN مرتبط با سیستم‌عامل مبدا توسط Dockerfile استفاده کنید.

نمونه‌ای از استفاده از آپشن USER در Dockerfile به شکل زیر است:

|  |
| --- |
| # استفاده از تصویر اصلی بر پایه سیستم‌عامل اوبونتو  FROM ubuntu:latest  # اضافه کردن کاربر غیر root به نام "myuser"  RUN useradd -ms /bin/bash myuser  # تعیین کاربر اجرایی به عنوان کاربر "myuser"  USER myuser  # اجرای دستورات بعدی به عنوان کاربر "myuser"  CMD ["command"] |

در این مثال، یک کاربر به نام "myuser" ایجاد می‌شود و سپس با استفاده از آپشن USER، کاربر اجرایی به "myuser" تغییر می‌کند. همچنین، دستور CMD نیز به عنوان کاربر "myuser" اجرا خواهد شد.

* CMD

در Dockerfile، آپشن CMD یکی از دستورهایی است که برای تعیین دستور اجرایی پیشفرض در زمان راه‌اندازی یک container مشخص می‌کند. CMD دستور اجرایی است که زمانی که container راه‌اندازی می‌شود، به صورت پیش‌فرض اجرا می‌شود.

توجه داشته باشید که تنها یک دستور CMD در Dockerfile مجاز است. اگر چندین دستور CMD در Dockerfile وجود داشته باشد، فقط آخرین آن‌ها اعمال خواهد شد.

در Dockerfile، می‌توانید CMD را به دو صورت مختلف تعریف کنید:

CMD به صورت آرایه‌ای:

|  |
| --- |
| CMD ["executable","param1","param2"] |

این شکل CMD معمولاً برای اجرای دستورهای داخلی سیستم عامل container استفاده می‌شود. برای مثال، اگر شما می‌خواهید یک web server مانند Apache را اجرا کنید، CMD ممکن است به این صورت باشد:

|  |
| --- |
| CMD ["apache2ctl", "-D", "FOREGROUND"] |

CMD به صورت رشته‌ای:

|  |
| --- |
| CMD command param1 param2 |

این شکل CMD برای اجرای دستورهایی استفاده می‌شود که می‌توانند از shell در container اجرا شوند. برای مثال:

|  |
| --- |
| CMD npm startMD command param1 param2 |

در این حالت، دستور npm start با استفاده از شل در container اجرا می‌شود.

مثال دیگر:

|  |
| --- |
| CMD python app.py |

این دستور، فایل app.py را با استفاده از پایتون در container اجرا می‌کند.

توجه داشته باشید که توصیه می‌شود برای اجرای دستورهای طولانی‌تر، بهتر است از آرایه‌ای استفاده کنید تا بتوانید پارامترها را به صورت جداگانه مشخص کنید و مشکلات محتمل فاصله‌گذاری (escaping) را حل کنید.

ویدیو 4