# 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.

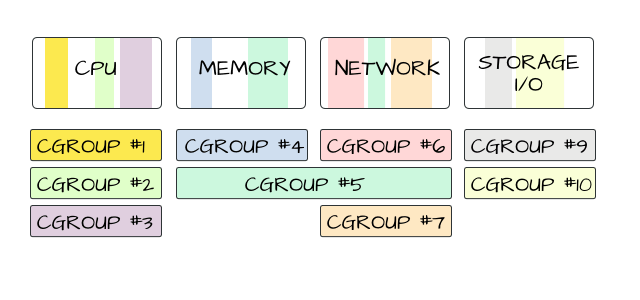
|  |  |
| --- | --- |
| معماری ماشین مجازی | معماری کانتینر در زمان استفاده از Docker Engine |

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

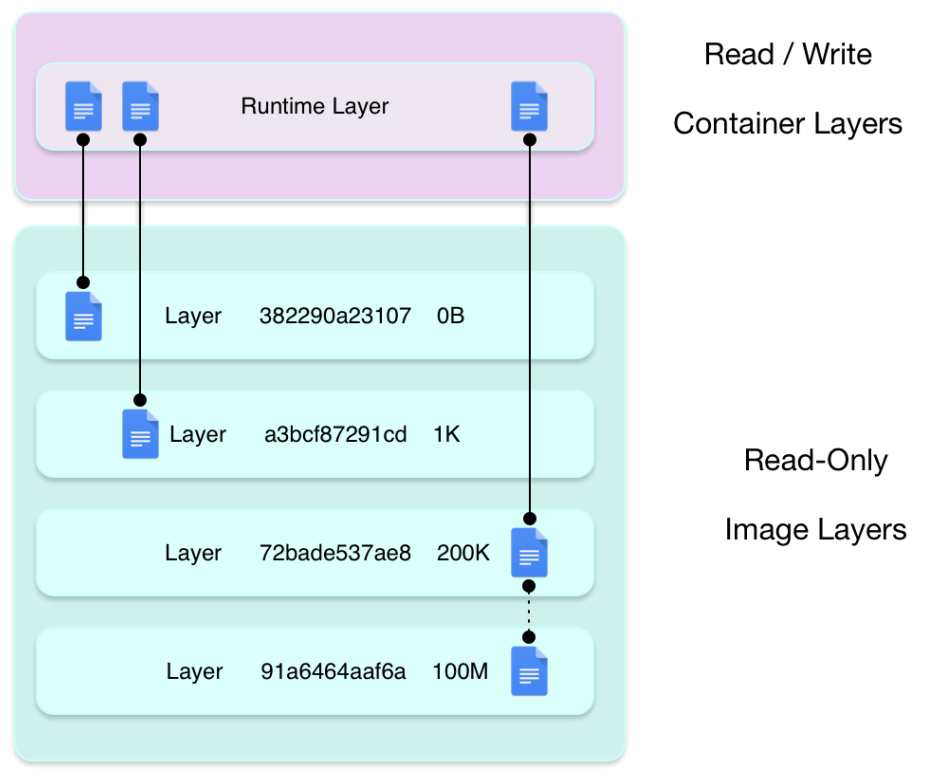
یک فایل سیستم است که در سیستم عامل لینوکس هم از آن استفاده شده است.

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.

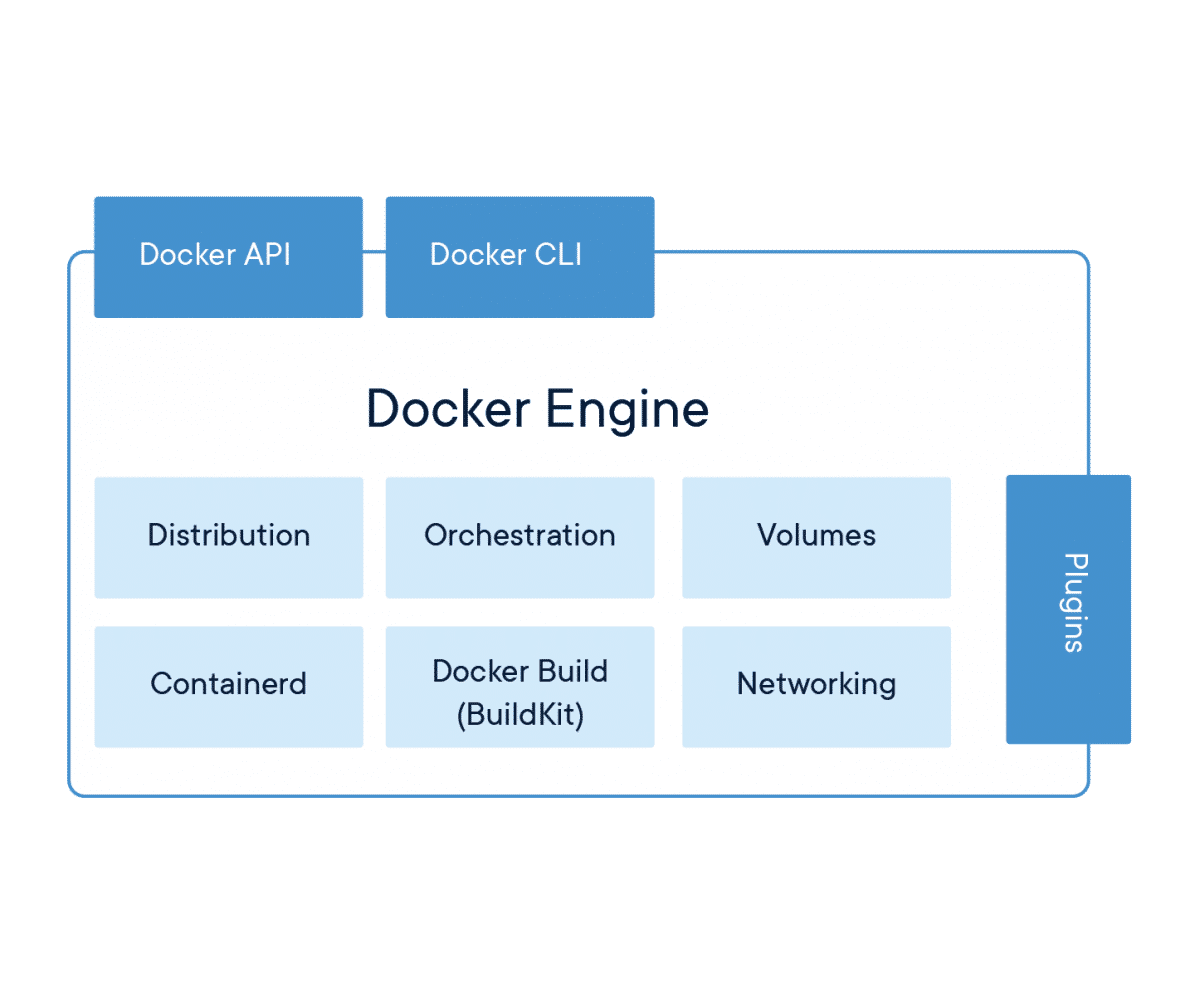
استراتژی Copy-On-Write نه تنها میزان فضای مصرفی کانتینرها را کاهش می دهد، بلکه زمان لازم برای راه اندازی یک کانتینر را نیز کاهش می دهد. در زمان شروع، داکر فقط باید یک لایه نازک قابل نوشتن برای هر کانتینر ایجاد کند.

# 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.

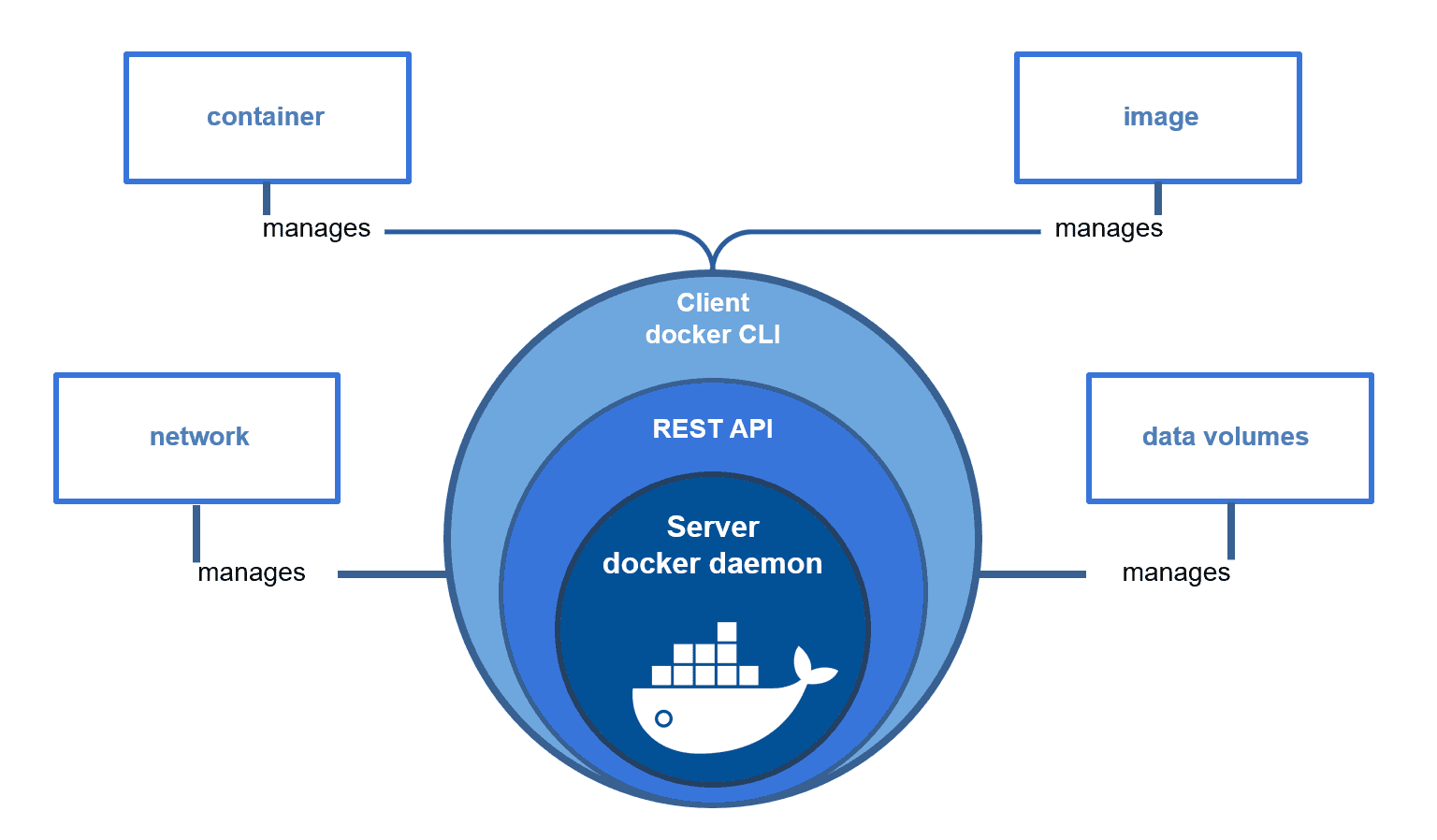
CLI از داکر APIها برای کنترل یا تعامل با Docker Daemon از طریق اسکریپت نویسی یا دستورات مستقیم CLI استفاده می‌کند. بسیاری از برنامه‌های دیگر Docker از API و CLI زیرین استفاده می‌کنند. دیمون اشیاء Docker مانند تصاویر، کانتینرها، شبکه‌ها و حجم‌ها را ایجاد و مدیریت می‌کند.



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.

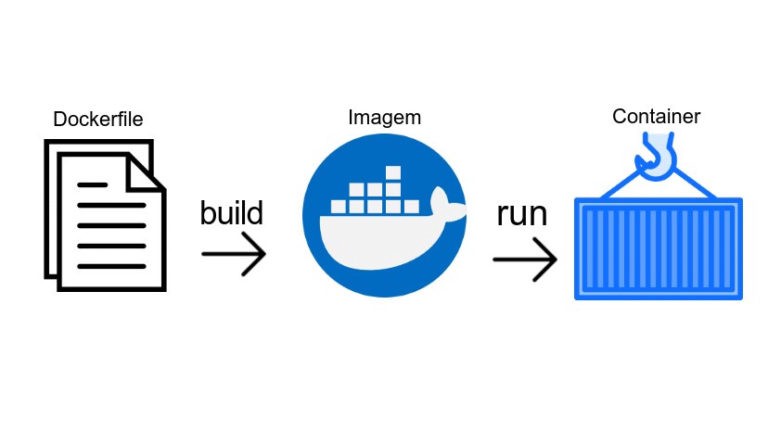
از آنجا که ایمج ها به نوعی فقط الگو یا template هستند، نمی توانید آن ها را استارت یا اجرا کنید. کاری که می توانید انجام دهید این است که از آن الگو به عنوان پایه برای ساخت کانتینر استفاده کنید. کانتینر در نهایت فقط یک ایمج در حال اجرا است. هنگامی که یک کانتینر ایجاد می کنید، یک لایه قابل نوشتن در بالای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

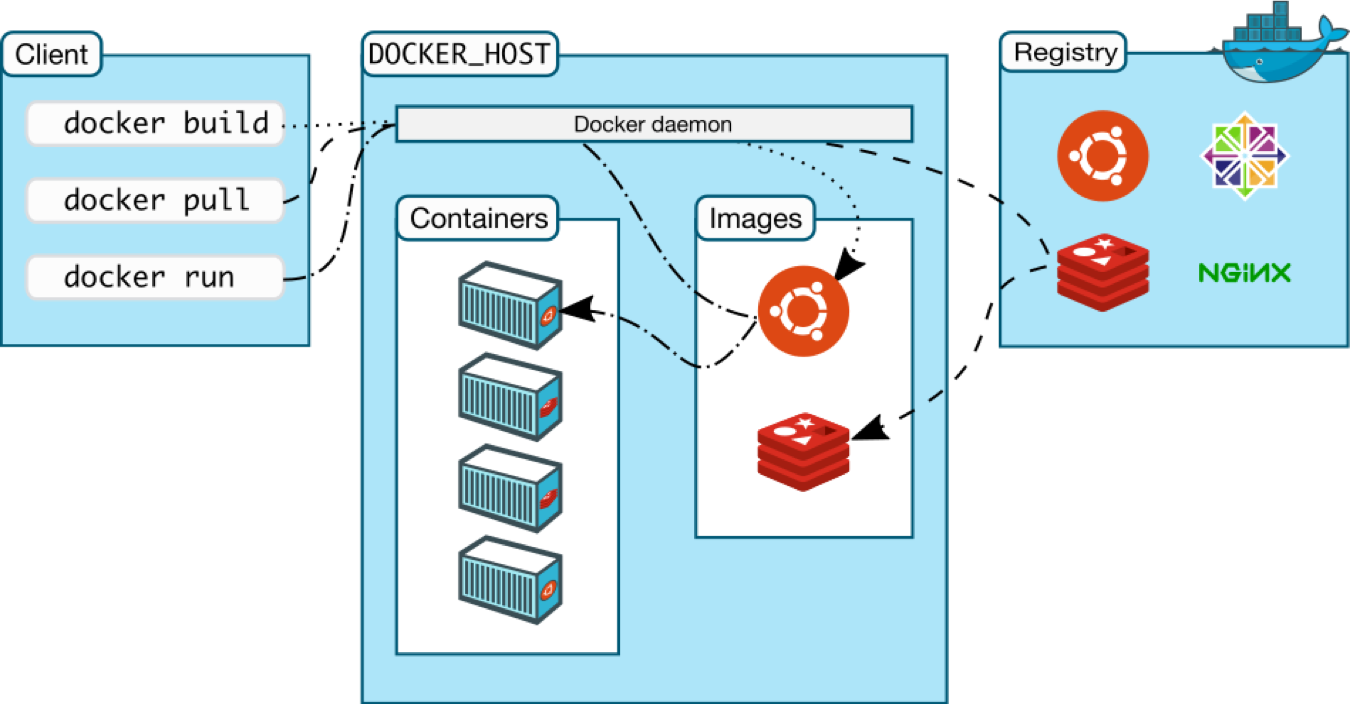
برخلاف ماشین های مجازی (VM) که مجازی سازی در سطح سخت افزاری انجام می شود، کانتینرها در لایه برنامه مجازی سازی می شوند. آن ها می توانند از یک ماشین استفاده کنند. هسته آن را به اشتراک بگذارند و سیستم عامل را مجازی کنند تا فرایندهای جداگانه را اجرا کند. این باعث می شود کانتینر ها به میزان زیادی سبک شوند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

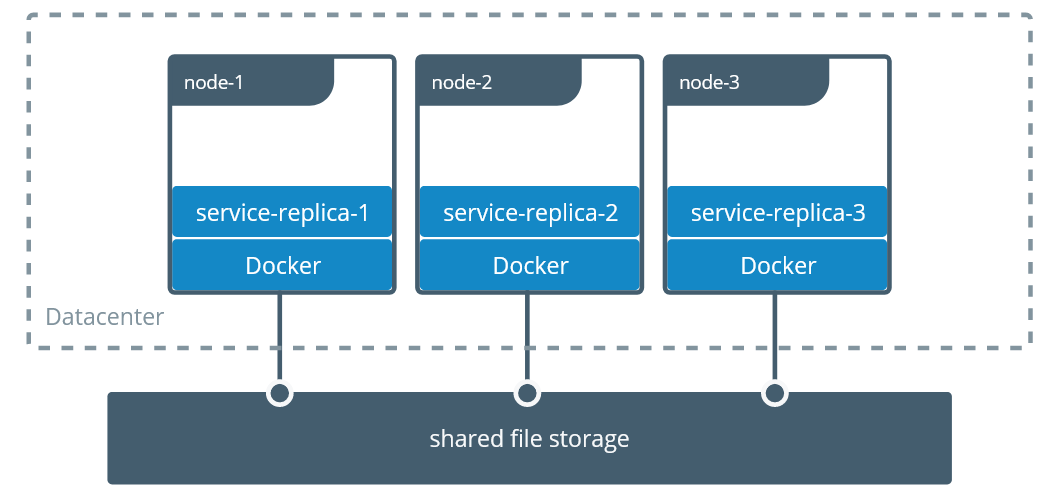
با استفاده از این روش داده‌های داخل یک کانتینر بر روی دایرکتوری‌های سیستم هاست قرار داده و این اطلاعات بر روی سیستم هاست نگهداری می‌شود و هر زمان که برای کانتینر مشکل پیش‌ بیاید این داده‌ها بر روی هاست باقی خواهند ماند.

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.

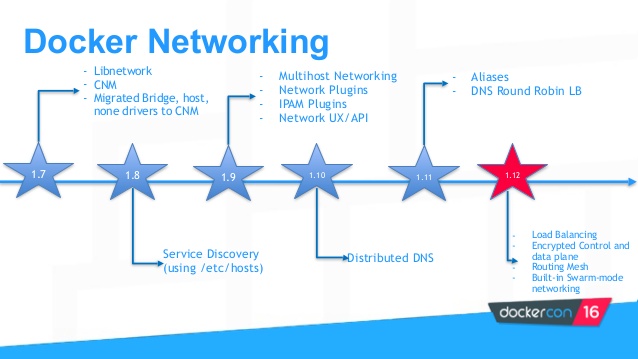
به صورت پیش‌فرض شبکه‌ی bridge به کانتینرها متصل می‌گردد و اگر بخواهید از دو شبکه‌ی دیگر استفاده کنید حتما باید آنها را در راه‌اندازی کانتینر به صورت آپشن وارد کنید. این موضوع به صورت کامل قابل پیکربندی است و می‌توانید همانند تمام موارد دیگر پیش‌فرض شبکه‌ را نیز تغییر دهید.

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.

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.

1:11:45