فصل یک: مقدمه

در محیط محاسبات ابری مقیاس بزرگ، مراکز داده ابر و کاربران نهایی از نظر جغرافیایی در سراسر جهان توزیع شده اند. بزرگترین چالش برای مراکز داده ابر این است که چگونه میلیون ها درخواست را که بطور مداوم از کاربران نهایی می رسند، بطور صحیح و موثر رسیدگی و سرویس داده شوند. برای این منظور پلتفرم هایی در سطح paas که سرویس ابری به شکل پلتفرم است؛ توسعه پیدا کرده اند که می‌توانند این تعداد بار زیاد را مدیریت کنند. یکی از این پلتفرم ها کوبرنتیز(Kubernetes) نام دارد که با مدیریت سرویس ها میتواند بار زیادی را تحمل کند و سرویس های ما را به نحو مطلوبی مدیریت و مقیاس پذیر کند. این پلتفرم به تعادل بار برای رسیدن به بیشترین سطح رضایت کاربر و افزایش نرخ استفاده از منابع محاسباتی و حافظه ای به صورت بهینه کمک می کند.

همچنین این پلتفرم به ما کمک میکند که بتوانیم سرویس های خود را به صورت ساختار مایکرو سرویس در بیاوریم و هر کدام را جداگانه در این پلتفرم بارگذاری کنیم. این روش به ما کمک میکند تا پیچیدگی ها در برنامه نویسی کمتر شود و همچنین بتوانیم سرویس های خود را راحت تر و کاربردی تر مانیتور کنیم و در مواقعی که بار زیاد شد سرویس مورد نظر را مقیاس پذیر کنیم و نه همه ی سرویس های موجود را . این قابلیت همچنین باعث می‌شود در مواقعی که سیستم ما دچار اختلال و مشکل میشود راحت تر ایراد و عیب را پیدا کنند چون عملکرد هر سرویس مشخص است و مشکل راحت تر پیدا میشود.

امروزه شرکت های بزرگ برای مدیریت سرویس های خود از این پلتفرم استفاده میکنند تا خدمات خود را در زمان های پیک باری به نحوی که از دسترس خارج نشوند و زمان تاخیر پاسخشان زیاد نشود؛ ارایه کنند. همچنین این پلتفرم برای سازمان ها و شرکت های بزرگ به این دلیل مهم است که کار خودکارسازی را در ابعاد مختلف انجام میدهد و لازم نیست افراد سازمان ها یا شرکت تا کل روز سرویس ها را مانیتور کنند. به طور مثال هنگامی که بار بر روی یک سرویس زیاد شد ، این پلتفرم سرویس های ما را مقیاس پذیر می‌کند و یا سرویسی از دسترس خارج شد یکی دیگر از آن سرویس اجرا میکند و یا سلامت سرویس ها را مستمر چک میکند و در صورت خطا به ما اطلاع داده میشود. پس با این پلتفرم هزینه مدیریت و نگه داری سرویس ها کاهش پیدا می‌کند و همچنین دسترس پذیری سرویس های ما بیشتر می‌شود که این پلتفرم بسیار از لحاظ اقتصادی و مالی برای شرکت ها به صرفه تر و بهینه تر است‌‌.

هدف از انجام این پروژه مقیاس پذیر کردن سرویس ها در زمان های پیک باری با استفاده از پلتفرم کوبرنتیز در مقیاس کوچک تر است. در ابتدا ما سرویس های خود را به صورت ساختار مایکرو سرویس در می‌آوریم و با استفاده سیستم داکر که در فصل دوم توضیح مختصری از این سیستم ارایه میدهیم؛ سرویس های خود را ایمیج میکنیم که بتوانیم در پلتفرم کوبرنتیز بارگذاری؛ اجرا و مدیریت کنیم. در مرحله بعدی ما معیار های که برای مقیاس پذیری مناسب است را بررسی میکنیم و تعدادی از آنها را معرفی میکنیم. برای آنکه این معیار ها را استخراج کنیم باید سرویس های متفاوتی را در کنار پلتفرم کوبرنتیز خود نصب کنیم. بعد از انجام نصب؛ تست های خود را بر اساس معیار ها انجام میدهیم. دو معیاری که برای ما در این پروژه اهمیت دارد معیار های زمان پاسخ سرویس ها و تعداد درخواستی که در ثانیه می‌توانند جواب دهند. این دو معیار رابطه مستقیم با هم دارند چون اگر زمان پاسخ هر سرویس کمتر شود به تبع آن درخواست بیشتری را می‌توانند جواب دهند و بالعکس. ما با استفاده از این دو معیار کار مقیاس پذیری را انجام میدهیم و سپس علمکرد سرویس های خود را بعد از مقیاس پذیری مورد تحلیل و بررسی قرار می‌دهیم و نتایج را به صورت جدول و گراف به نمایش میگذاریم.

**فصل دوم: مفاهیم ماشین مجازی؛ داکر و کوبرنتیز**

در این فصل به توضیح مفاهیم پایه سیستم داکر و پلتفرم کوبرنتیز می پردازیم. به دلیل آنکه ما سرویس های خود را در این دو محیط پیاده و میکنیم؛ لازم است که با این مفاهیم آشنا باشیم.

# 1. ماشین‌های مجازی و کانتینرها

قبل از ورود به مباحث اصلی داکر، آشنایی با مشخصات ماشین‌های مجازی و مشکلات آنها می‌تواند دلایلی که باعث بوجود آمدن کانتینرها شدند را بیشتر مشخص کند.

## 1.1. ماشین‌های مجازی

قبل از بوجود آمدن کانتینرها، روش اصلی برای ایجاد محیط ایزوله برای مدیریت نرم‌افزارها استفاده از ماشین‌های مجازی بود. در این روش هر نرم‌افزار و سیستم‌هایی که برای اجرا به آنها نیاز داشت در یک ماشین مجازی مستقل با سیستم عامل مجزا نصب می‌شود. چند ماشین مجازی متفاوت می‌توانند بر روی یک سیستم سخت‌افزاری نصب شده و مشخصات این سیستم (مانند فضای هارد یا رم) را به صورت مجزا استفاده کنند. در این حالت در هنگام تعریف هر ماشین مجازی مقدار منابع سخت‌افزاری که توسط ماشین مجازی قابل استفاده است مشخص شده و در صورت نیاز در هر زمان امکان تغییر این موارد وجود دارد.

## 1.1.1. مشکلات ماشین‌های مجازی

الف) ماشین‌های مجازی با توجه به نیاز به نصب کامل سیستم عامل و امکانات مورد نظر هر نرم‌افزار به صورت مستقل، معمولا حجم بالایی از فضای هارد سرور اصلی را اشغال می‌کنند.

ب) راه‌اندازی و اجرای چند ماشین مجازی همزمان نیاز به استفاده از امکانات سخت‌افزاری بالایی را بر روی سرور اصلی ایجاد می‌کند و در غیر این صورت باعث کندی یا ناپایداری سرور خواهد شد.

ج) راه‌اندازی (boot) هر ماشین مجازی به دلیل آنکه باید سیستم عامل مستقل آن ماشین به صورت کامل راه‌اندازی شود مدت زمان زیادی طول می‌کشد.

د) ماشین‌های مجازی به صورت کلی کمک چندانی به انتقال سیستم‌های نرم‌افزاری از یک سرور به سرور دیگر نمی‌کنند و همچنین در صورتی که سیستم عامل‌ها احتیاج به به روز رسانی داشته باشند این کار برای سیستم عامل هر ماشین مجازی باید به صورت مستقل انجام شود.

## ۲.۱ کانتینرها

کانتینرها بر خلاف ماشین‌های مجازی که ایزوله سازی محیط اجرای نرم‌افزار را در سطح سخت افزار سرور انجام می‌دهند، ایجاد این محیط را به سطح سیستم عامل نصب شده بر روی سرور منتقل می‌کنند. به همین دلیل کانتینرها از نظر استفاده از منابع سرور بسیار کارآمدتر عمل می‌کنند چرا که در این حالت سیستم عامل مجزایی بر روی سرور اصلی نصب نمی‌شود و همینطور منابع نرم‌افزاری مورد نیاز در صورتی که به صورت مشترک در چند کانتینر استفاده می‌شوند می‌توانند بر روی سرور اصلی نصب شده باشند.

### 1.1.2. فواید استفاده از کانتینر

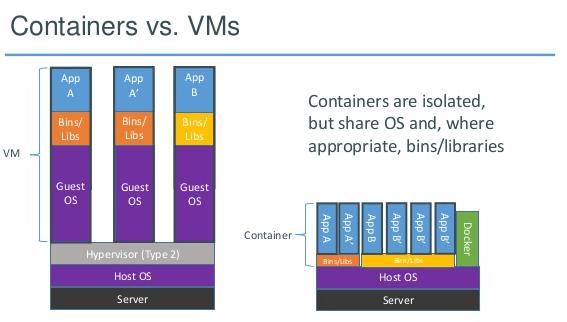
### الف) کانتینرها از سیستم عامل مستقل برای اجرای نرم‌افزارها استفاده نمی‌کنند و می‌توانند منابع نرم‌افزاری مشترک را با هم به اشتراک بگذارند به همین دلیل حجم فضایی که توسط کانتینرها اشغال می‌شود بسیار کمتر از ماشین‌های مجازی است.

### ب) با توجه به اشتراک منابع نرم‌افزار و سخت‌افزاری سرور توسط ماشین‌های مجازی و همچنین عدم نیاز به تخصیص مقدار مشخصی از هر منبع به هر کانتینر، نصب چند کانتینر بر روی یک سرور به امکانات سخت‌افزاری کمتری به نسبت ماشین‌های مجازی احتیاج دارد.

### ج) با توجه به آنکه راه‌اندازی (boot) هر کانتینر فقط نیازمند راه‌اندازی نرم افزارهای مورد نیاز آن کانتینر می‌باشد، سرعت راه ‌اندازی کانتینر بسیار بالاتر از ماشین‌های مجازی است.

### د) انتقال سیستم‌های نرم‌افزار از یک سرور به سرور دیگر به راحتی انجام می‌شود و این سیستم‌ها فقط احتیاج به به‌روز‌رسانی نیازهای نرم‌افزاری مربوط به خود را دارند و به همین دلیل به‌روز‌رسانی کانتینرها به راحتی و تقریبا بدون ایجاد اشکال در کارکرد سیستم امکان‌پذیر است.

در شکل زیر این تفاوت را به شکل واضح تر میتوان مشاهده کرد.



شکل۱ - تفاوت کانتینر و ماشین مجازی

# ۲. داکر(Docker)

داکر ([Docker](https://www.docker.com/)) ابزاری است که برای توسعه، راه‌انداری و اجرای راحت‌تر نرم‌افزارها بوسیله کانتینر (Container) طراحی شده است. کانتینرها به توسعه‌دهنده‌ها اجازه می‌دهند که نرم‌افزارهای خود را به همراه تمام مواردی که برای اجرای آنها احتیاج دارند (کتابخانه‌های نرم‌افزاری و غیره) به صورت یک پکیج آماده کرده و به سرور منتقل کنند. با این روش توسعه‌دهنده می‌تواند مطمئن باشد که نرم‌افزار آماده شده در هر سیستم عاملی که بر روی سرور نصب شده باشد و با هر تنظیماتی که در سیستم عامل ایجاد شده باشد، به درستی کار خواهد کرد و تغییر سیستم عامل یا تنظیمات آن اشکالی در اجرای نرم‌افزار ایجاد نخواهد کرد.

در نگاه اول داکر از نظر کارکرد مشابه ماشین مجازی (virtual machine) به نظر می‌رسد اما بر خلاف ماشین مجازی، بجای راه اندازی یک سیستم عامل کاملا مجزا بر روی سرور، داکر به نرم‌افزارها اجازه می‌دهد که از هسته سیستم عامل اصلی که بر روی سرور نصب شده است استفاده کنند و تنها مواردی که مستقل از سیستم عامل سرور عمل می‌کنند نیازهای اختصاصی نرم‌افزار می‌باشند که بر روی سرور نصب نشده و در کانتینر داکر نصب شده‌اند. این امر افزایش قابل توجهی در عملکرد سیستم ایجاد کرده و حجم کانتینرها را به نسبت ماشین مجازی به مقدار زیادی کاهش می‌دهد.

## 1.2. اجزا اصلی داکر

سه مفهموم اصلی که در هنگام استفاده از داکر باید با آنها کاملا آشنا باشید عبارتند از داکر فایل (Dockerfile)، ایمیج (Image) و کانتینر (Container) مجموعه این مفاهیم، فرایند اصلی داکر و روش استفاده از آن برای مدیریت نرم‌افزارها را مشخص می‌کند.

### **1.1.2. داکر فایل** (Dockerfile)

داکر فایل یک فایل متنی حاوی تمام دستوراتی است که با اجرای آنها تمام نیازمندی‌ها و تنظیمات مربوط به نرم‌افزاری که می‌خواهید توسط داکر اجرا شود در یک بسته داکر به نام داکر ایمیج ایجاد می‌شود. این فایل دستورالعمل مربوط به ایجاد این بسته را در اختیار داکر قرار می‌دهد و داکر با استفاده از این دستورالعمل و با استفاده از دستور docker build این بسته را ایجاد می‌کند.

### **2.1.2. داکر ایمیج** (Docker Image)

داکر ایمیج در تعریف ساده بسته‌ای است که با استفاده از آن می‌توان کانتینرهای داکر را ایجاد کرد. به عبارت دیگر داکر ایمیج بسته‌ای است که پس از ایجاد آن توسط دستورات داکر فایل امکان تغییر آن وجود ندارد و با استفاده از آن می‌توان هر تعداد کانتینر مورد نیاز برای اجرای نرم‌افزار مورد نظر را راه‌اندازی کرد.

داکر ایمیج‌ها در رجیستری داکر ذخیره می‌شوند. این رجیستری می‌تواند یک رجیستری خصوصی باشد که فقط شما به آنها دسترسی دارید یا یک رجیستری عمومی مانند داکر هاب (Docker Hub) که به سایر افراد اجازه می‌دهد از ایمیج ایجاد شده توسط شما برای اجرای نرم‌افزارهای موجود در آن استفاده کنند. استفاده از رجیستری‌های عمومی به صورت متداول در هنگام انتشار نرم‌افزارهای رایگان و سورس باز (Open Source) کاربرد دارند و این امکان را در اختیار دیگران قرار می‌دهند که فقط با دانلود ایمیج و ایجاد یک کانتینر از روی آن به راحتی و بدون نگرانی از نیازمندی‌ها و تنظیمات نرم‌افزار، از آن استفاده کنند.

### **3.1.2. داکر کانتینر** (Docker Container)

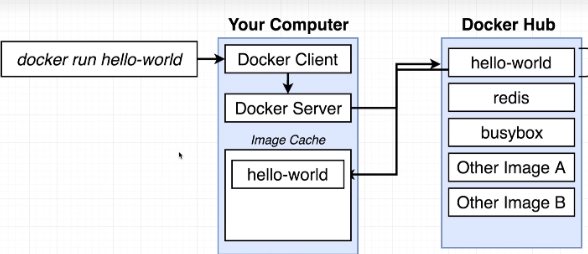
با استفاده از داکر ایمیج و دستور docker run می‌توان یک نمونه‌ی اجرایی از نرم‌افزار مورد نظر را به صورت داکر کانتیتر ایجاد کرد. بنابراین داکر کانتینر به صورت کلی یک نسخه‌ی آماده‌ی اجرا از نرم‌افزار موجود در داکر ایمیج هستند که هدف نهای استفاده از داکر و اجرای نرم‌افزار توسط آنها را محقق می‌کنند.



شکل۲ - نحوه کارکرد سیستم داکر

## 2.2. کارکرد کلی پلتفرم داکر

داکر از دو بخش داکر کلاینت و داکر سرور تشکیل شده است که بخش کلاینت دستورات را از ما میگیرد و به داکر سرور برای اجرای آن دستور می فرستد. داکر کلاینت به CLI سرور ما متصل شده است که از آن طریق دستورات را دریافت میکند. با استفاده از داکر سرور ما میتوانیم ایمیج های خود را با استفاده داکر فایل ها درست کنیم و به داکر هاب منتقل کنیم. این کار باعث میشود در هر سرور دیگری بتوانیم این ایمیج ها را به راحتی اجرا کنیم. حال در شکل زیر این ارتباط را مشاهده میکنیم.

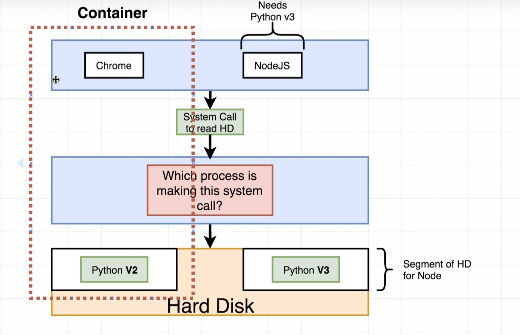


شکل۳ - نحوه اجرای یک ایمیج از داکر هاب

همان طور که در شکل بالا مشاهده میشود با اجرای دستور داکر ما به داکر کلاینت متصل میشویم و سپس اگر ایمیج قبلا از داکر هاب دانلود شده بود؛ دیگر نیازی به متصل شدن به داکر هاب نیست و داکر سرور این ایمیج را اجرا خواهد کرد. در غیر این صورت باید متصل شود و ایمیج را دانلود و سپس اجرا کند.

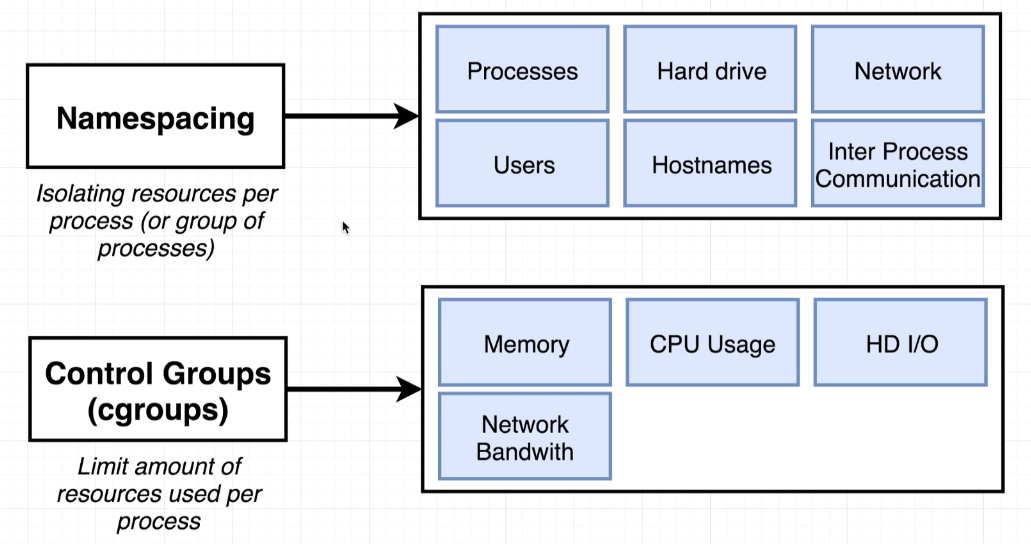
## 3.2. نحوه اجرا شدن کانتینر در سرور

همان طور که گفته شد کانتینر ما یک نرم افزار است که با تمام مواردی لازم دارد اجرا شود ایزوله میشود و توسط container runtime اجرا میشود. حال میخواهیم ببینیم این عملیات چگونه صورت میپذیرد.



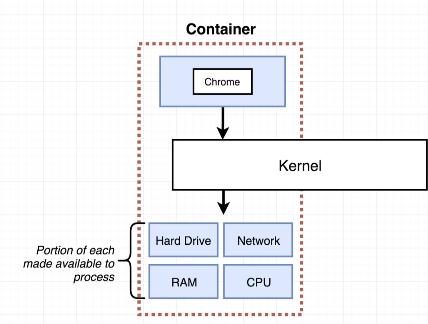
شکل۴ - فضای یک کانتینر در سرور

در شکل زیر مشاهده میکنیم برای اینکه این برنامه اجرا شود به Python V2 نیاز دارد. پس قسمتی از دیسک به این منظور برای این برنامه جدا میشود و در آن این برنامه ذخیره میشود. پس کانتینر به این صورت عمل میکند که یک برنامه ای است که نیازمندی های نرم افزاری و سخت افزاری اش توسط سیستم عامل مدیریت میشود و مفهومی به اسم namespace وجود دارد که برای هر کانتینر یک فضای مختص از لحاظ حافظه؛ شبکه ای و موارد دیگر در نظر گرفته میشود که هر موقع این برنامه درخواستی از سیستم عامل داشت؛ سیستم عامل تشخیص بدهد کدام برنامه است و هنگامی که این برنامه نیاز به این برنامه ها برای اجرا شدن داشت باید به کدام قسمت حافظه باید درخواست بدهد.



شکل۵ - Namespacing و cgroups در داکر

در شکل بالا مشاهده می شود که با استفاده از namespacing داکر سرور ما محیط ایزوله برای منابع کانتینر ما فراهم میکند و دسترسی کانتینر به آن namespace محدود میشود. Cgroup ها هم مشخص میکنند که هر برنامه ما که به صورت کانتینر در آمده است چه مقدار اجازه دارد که از منابع مختلفی که در سرور وجود دارد و در شکل بالا هم برخی از این منابع آمده است استفاده کند. شکل زیر به صورت ملموس تر نشان میدهد که یک کانتینر در سیستم ما چگونه است. با استفاده از درخواست هایی که برنامه ما که همان یک پروسس در سیستم عامل ما هست به کرنل میدهد؛ سیستم عامل با استفاده از منابعی که برای این پروسس در نظر گرفته است جواب میدهد و منابع را در اختیار این پروسس قرار میدهد.



شکل۶ - نحوه ایزوله شدن یک کانتینر در داکر

## 4.2. میکرو سرویس ها

بعد از اینکه سیستم داکر شهرت گرفت؛ این پلتفرم برنامه نویسان را به این سمت سوق داد که برنامه های خود را به صورت مایکروسرویس بنویسند به این معنی که سرویس خود را به بخش های کوچکتر تقسیم میکنیم و هر کدام وظیفه انجام کاری را دارند. این نوع ساختار برنامه نویسی باعث شد تا برنامه هایی که پیچیدگی زیاد دارند را با تقسیم به برنامه های کوچکتر این پیچیدگی کمتر شود و راحت تر و سریع تر بتوانند کار عیب یابی و رفع اشکالات را انجام دهند. همچنین با استفاده از این پلتفرم میتوان از ارکستریتور ها هم استفاده کرد به این منظور که هنگامی که بار بر روی یک کانتینر زیاد شد تعداد بیشتری از این کانتینر ها ساخته شود و سرویس دهی بهبود یابد.

در این پروژه ما از این ساختار استفاده خواهیم کرد و سرویس های خود را به مایکرو سرویس ها در خواهیم آورد تا بتوان بر اساس پلتفرم کوبرنتیز که در بخش بعدی به آن خواهیم پرداخت؛ این مایکرو سرویس ها را مدیریت کنیم.

# ۳. پلتفرم کوبرنتیز (Kubernetes)

## 1.3. توضیح پلتفرم کوبرنتیز و مزیت های استفاده از آن

کوبرنتیز ( Kubernetes) (که به شکل k8s نیز ارجاع می‌شود) سامانه‌ای متن‌باز برای خودکارسازی دیپلوی(deployment)؛ مقیاس و مدیریت برنامه‌های کانتینرسازی شده در سراسر زیرساخت است که در ابتدا توسط گوگل توسعه داده شد و سپس در سال ۲۰۱۵ به بنیاد CNCF اهدا شد. کوبرنتیز پیاده سازی جدیدی از بیش از یک دهه تجربه گوگل در اجرای نرم افزارهای سمت سرور در مقیاس بسیار بالاست که به صورت متن باز (open source) در اختیار همه قرار گرفته است.

این پلتفرم وظیفه اجرا و مدیریت کانتینرها را بر روی گروهی از سرورهای موجود در یک یا چند مرکز داده ها (data center) به عهده دارد. کوبرنتیز در واقع نسل سوم از این فناوریست که در شرکت گوگل از ابتدا به زبان گو (Go) پیاده سازی شده است. دو نسل قبلی آن برگ (Borg) نام داشته که پیاده سازی آن به زبان سی پلاس پلاس بوده است و گوگل همچنان از آن در محیط عملیاتی استفاده میکند.

مزیت کلیدی کوبرنتیز در این است که بدون نیاز به یک تیم بزرگ برای راه اندازی و نگهداری، میتوان آن را در مقیاس وسیع برای اجرای تعداد زیادی برنامه کاربردی به کار گرفت. از مزایای دیگر آن قابلیت اجرا بر روی بسترهای متفاوت است؛ از سرورهای یک مرکز داده های خصوصی گرفته تا سرویسهای ابری عمومی، یا حتی ترکیبی از هر دو. به طور کلی هر شرکتی که یک یا چند سرویس نرم افزاری اجرا می کند به طور بالقوه در مرحله اول به کانتینرها و سپس به سیستمی مانند کوبرنتیز نیاز دارد. دلیل اصلی نیاز به کانتینرها امکان جداسازی برنامه ها (isolation) از یکدیگر در بهترین سطح ممکن است تا فرآیند تولید، تست و در نهایت اجرا بر روی یک زیرساخت مشترک تسهیل شود.

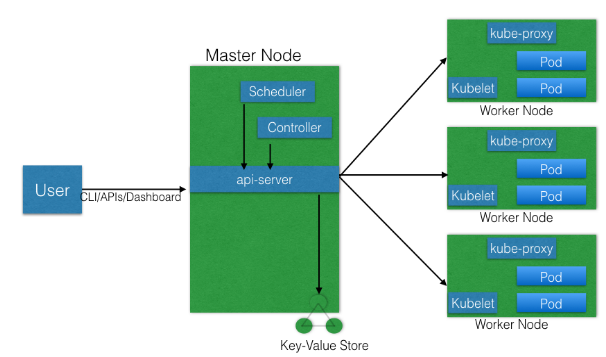
در مرحله بعد نیاز به کوبرنتیز پیدا میشود تا اجرای این کانتینرها بر روی دسته ای (cluster) از ماشینها را تا حد زیادی اتوماتیک کند. در واقع کوبرنتیز مانند سیستم عاملیست که بر روی تمام سرورهای شما به صورت یکپارچه اجرا میشود و به شما این امکان را میدهد که دیگر نگران هیچ ماشینی به طور خاص نباشید. اگر ظرفیت کافی در زیرساخت شما وجود داشته باشد، این سیستم به راحتی میتواند از دست دادن یک یا چند ماشین را برای شما به گونه ای مدیریت کند که کاربران هیچ تغییری در سرویسهای در حال اجرا بر روی این بستر احساس نکنند.

این سیستم امکاناتی مانند بررسی سلامت (health check) و تکثیر (replication) برنامه ها را به راحتی بر روی مجموعه سرورهای شما فراهم میکند. از دیگر قابلیتهای آن نیز ویژگیهای مناسب و سطح بالا، مانند کشف سرویسها (service discovery)، توزیع بار (load balancing) و مدیریت پیکربندی (configuration management) است که برای ساخت سیستمهایی با معماری مایکروسرویسی (micro-service architecture) حیاتیست و برای تیمهای شما امکان تولید، تغییر و مقیاس پذیری (scaling) بخشهای مختلف هر سرویس را بر اساس شرایط مورد نیاز فراهم میکند. همچنین این سیستم به صورت خودکار کانتینر هایی که خراب شده اند را از بین میبرند و دوباره تولید میکنند و اجازه اینکه ترافیک به این کانتینر خراب فرستاده شود؛ جلوگیری میکند. مزیت دیگری که میتوانیم برای این سیستم نام ببریم؛ مدیریت منابع حافظه ای است که میتواند حافظه برای کانتینر ها را از طرق مختلف تامین کند. به طور مثال از طریق حافظه داخلی سرور یا حافظه فراهم کنندگان ابری و یا از طریق Network Storage System فراهم کند. در کنار برخی از مزایایی که برای این پلتفرم گفته شد؛ قابلیت کاربردی و مهمی که این سیستم دارد این است که میتوان این پلتفرم را توسعه داد و بخش هی متفاوتی اضافه کرد. این به این دلیل است که ساختار کوبرنتیز هم ماجولار (Modular) است و توسعه دهندگان می توانند بخش های مختلفی را به این پلتفرم اضافه کنند.

اگر چه بسیاری از نرم افزارها سعی میکنند این قابلیت ها را در سطح برنامه کاربردی پیاده کنند ولی تجربه نشان داده است که این کار با وجود صرف زمان و انرژی زیاد در اکثر موارد منجر به یک راه حل شکننده و غیر قابل نگهداری میشود که برای برنامه های کاربردی بعدی باید از نو تکرار شود. کوبرنتیز با انتقال این دغدغه ها به لایه مناسب و آزاد کردن برنامه کاربردی از قید و بند آنها به شما کمک میکند که وقت و انرژی تیم را در جای مناسب و برای تولید ویژگیهای خاص برنامه کاربردی خودتان صرف کنید.

## 2.3. اجزا و ساختار پلتفرم و گره رهبر در کوبرنتیز

حال در این بخش به ساختار و اجزای کوبرنتیز می پردازیم و بخش های مختلف این پلتفرم را توضیح میدهیم تا دید بهتری نسبت به این پلتفرم پیدا کنیم.

شکل۷ - ساختار کلی پلتفرم کوبرنتیز

همان طور که در شکل بالا مشاهده می کنیم؛ پلتفرم کوبرنتیز شامل چند بخش مهم است که اصلی ترین بخش آن گره رهبر (Master node) است. این بخش مسئول مدیریت و هماهنگی بخش های مختلف کلاستر ما است و به نوعی همه عملیات های کلاستر ما از طریق این بخش انجام میشود. این گره شامل بخش های متفاوتی است که در بخش زیر آن ها را توضیح خواهیم داد. همه کاربران برای آنکه بتوانند به کلاستر دسترسی پیدا کنند و عملیات های مختلف انجام دهند باید از طریق این گره درخواست های خود را بفرستند. این درخواست ها هم از طریق ترمینال یا داشبورد و یا API های مختلف امکان پذیر است.

به دلیل آنکه این گره بسیار نقش اساسی در کلاستر ما دارد؛ باید همیشه در دسترس باشد و اگر دچار خطا شود؛ هزینه زیادی را به شرکت یا سازمان تحمیل خواهد کرد. برای رفع این مشکل؛ معمولا چند گره رهبر به کلاستر اضافه میکنند و به عبارتی کلاستر در حالت دسترس پذیری بالا (HA)قرار میدهند و اگر یکی از این گره ها دچار مشکل شد؛ گره های رهبر بتوانند سرویس دهی را انجام دهند. حال سراغ بخش های مختلف این گره می رویم و توضیح مختصری برای هر کدام ارائه میکنیم.

۱ API server .

این نقطه‎ی اصلی مدیریت و ورود به Kubernetes است که به کاربر اجازه می‎دهد Kubernetes را پیکربندی کند. در واقع kube-apiserver پلی بین اجزای مختلف با هدف نگهداری و حفظ سلامت کلاستر و انتشار اطلاعات و اجرای دستور عمل‌ها است. api server یک رابط RESTfull ایجاد می‎کند که به این معنی است که بسیاری از ابزارها و کتابخانه‎ها به راحتی می‎توانند با آن ارتباط برقرار کنند. api server درخواست ها را دریافت میکند؛ سپس آنها را تایید و بررسی میکند. بعد از آنکه حالت فعلی کلاستر را از یک پایگاه داده توزیع شده به اسم etcd که در بخش بعد توضیح خواهیم داد؛ خواند و با نتیجه درخواستی که الان آمده است مقایسه کرد؛ حالت جدید تولید شده از نتیجه درخواست فعلی را در این پایگاه داده ذخیره می کند. api server تنها بخشی است که اجازه دارد به این پایگاه داده دسترسی پیدا کند و اطلاعات کلاستر ما را بخواند یا بنویسد. همچنین در پلتفرم کوبرنتیز این امکان وجود دارد که چند api server وجود داشته باشد و api server اصلی ما درخواست ها را به بقیه api server ها بفرستد و آن ها را مدیریت کند.

۲ Scheduler .

این بخش مسئولیت این را دارد که آبجکت ها جدید مانند پاد ها را به گره ها اختصاص بدهد. برای آنکه scheduler ما بتواند تصمیم بگیرد که کدام آبجکت را به کدام گره اختصاص دهد؛ ابتدا باید نیازمندی ها و محدودیت های هر آبجکت را که در کانفیگ فایلش نوشته میشود بررسی کند و همچنین ظرفیت گره های کارگر را بررسی کند. scheduler این اطلاعات را باید از پایگاه داده etcd از طریق api server دریافت کند و سپس پس از بررسی این اطلاعات کار اختصاص دادن این آبجکت ها به گره ها را انجام میدهد و حالت فعلی کلاستر را از طریق api server در etcd ذخیره میکند. این بخش هم همچنین قابل توسعه است و توسعه دهندگان میتوانند scheduler های خود را به کوبرنتیز اضافه کنند. برای آنکه آبجکت ما توسط یک scheduler خاص زمانبندی شود باید در کانفیگ فایل آن آبجکت اسم scheduler را گذاشته باشیم در غیر این صورت توسط scheduler پیش فرض زمانبندی میشود.

۳ Controller managers .

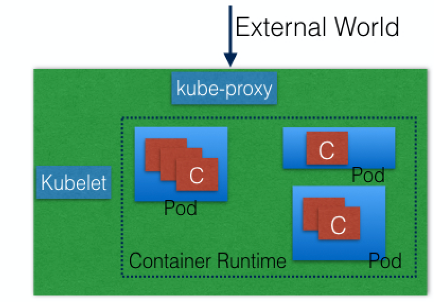
این کنترلر ها مسئولیت این را دارند که وضعیت فعلی کلاستر مطابق وضعیت مطلوب باشد. به طور مثال اگر در کانفیگ فایل نوشته شده است که باید ۳ پاد اجرا شود و در حال حاضر ۱ پاد برای یک آبجکت اجرا میشود؛ باید تعداد پاد ها را زیاد کند تا وضعیت مطلوب حاصل شود. این کنترلر وضعیت فعلی را از پایگاه داده etcd از طریق api server دریافت میکند و به طور مستمر این مقایسه را انجام میدهد که کلاستر ما در وضعیت مطلوب نگه داشته شود.

۴ etcd .

این بخش یک پایگاه داده توزیع شده است که داده ها را به صورت key-value نگه داری میکند. اطلاعات کانفیگ فایل های آبجکت های مختلف در این پایگاه داده ذخیره میشود که تنها api server میتواند به این پایگاه داده دسترسی پیدا کند و اطلاعات را بخواند و بنویسد. این پایگاه داده هم میتواند در گره رهبر ما قرار بگیرد و یا آنکه جدا از این گره باشد و در گره های دیگری قرار بگیرد به منظور آنکه اگر گره رهبر ما دچار مشکل شد یا مورد حمله قرار گرفت؛ اطلاعات کلاستر ما از بین نرود. همچنین کوبرنتیز قابلیت های backup ؛ snapshot و بازیابی اطلاعات را برای این پایگاه داد قرار داده است تا اطلاعات ما به راحتی از بین نرود.

## 3.۳. معماری و اجزا تشکیل دهنده گره کارگر

بعد از اتمام بخش گره رهبر ما سراغ گره کارگر میرویم و بخش های مختلف این گره را بررسی میکنیم.



شکل۸ - شمای کلی گره کارگر در پلتفرم کوبرنتیز

سرویس های ما در این گره های کارگر اجرا میشوند و سرویس های ما در آبجکت پاد که کوچکترین آبجکت پلتفرم کوبرنتیز است؛ قرار دارند که هر پاد میتواند چند کانتینری که عملکردشان بسیار مرتبط با هم است قرار بگیرند که اکثرا در هر پاد یک کانتینر اجرا میشود. کاربران با دسترسی به این گره ها میتوانند به سرویس های ما دسترسی پیدا کنند و عملیات مورد نظر را انجام دهند. در فصل ۳ انواع آبجکت های کوبرنتیز را مورد بررسی قرار میدهیم.

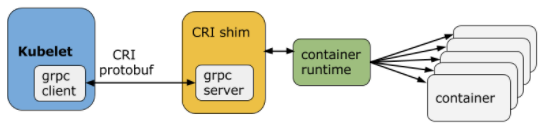
همانطور که در شکل بالا مشاهده می کنیم؛ گره کارگر ما دارای سه بخش Container Runtime ؛ Kube-proxy و Kubelet است. در بخش زیر توضیح مختصری برای هر کدام ارائه میدهیم.

۱. Container Runtime

اگرچه همانطور که گفتیم کوبرنتیز یک پلتفرم برای مدیریت و اجرای کانتینر ها است؛ ولی این پلتفرم خود به طور مستقیم این کانتینر ها را نمیتواند مدیریت و اجرا کند. برای اینکار باید از یک Container Runtime برای هر گره کارگر استفاده کند تا کوبرنتیز بتواند پاد ها را در آن گره اجرا و مدیریت کند. Container Runtime های متفاوتی مانند Docker؛ Containerd و CRI-O وجود دارد که ما در این پروژه از Docker برای اجرای کانتینر ها استفاده میکنیم.

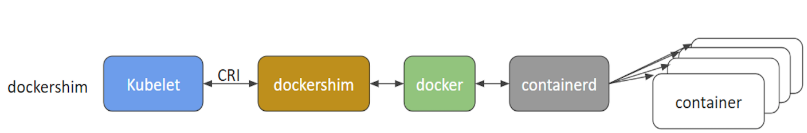
۲ Kubelet .

این عامل در گره کارگر مسئولیت ارتباط با بخش های مختلف گره رهبر را دارد و کانفیگ فایل پاد را از طریق api server دریافت میکند سپس به container runtime وصل میشود و کانتینر ها مشخص شده در کانفیگ فایل را از طریق container runtime اجرا میکند. این عامل همچنین سلامت کانتینر هایی که در حال اجرا هستند را مانیتور میکند.



شکل۹ - نحوه ارتباط kubelet با container runtime

همانطور که در شکل بالا مشاهده میکنیم؛ kubelet از طریق Container Runtime Interface توانسته است به Container Runtime متصل شود. این CRI شامل پروتکل هایی است که Kubelet از طریق آن ها میتواند به Container Runtime متصل شود. در اینجا Kubelet به عنوان grpc client عمل میکند و به CRI shim که به عنوان grpc server عمل میکند؛ متصل میشود و عملیات های مربوط به کانتینر ها را انجام میدهد. CRI دو سرویس اصلی دارد. اولی به عملیات های مربوط به ایمیج ها میشود که ImageService نام دارد و دومی عملیات های مربوط به کانتینر ها و پاد ها میشود که RuntimeSerivce نام دارد. همانطور که قبلا هم اشاره کردیم؛ کوبرنتیز با container runtime های مختلفی میتواند کار کند به شرط آنکه CRI را داشته باشد.



شکل۱۰ - نحوه ارتباط kubelet با محیط اجرای کانتینر برای docker

به دلیل آنکه ما از داکر برای اجرای کانتینر های خود استفاده میکنیم؛ ارتباط kubelet با داکر به صورت شکل بالا است و از طریق CRI مخصوص داکر به داکر متصل میشود و داکر هم از طریق containerd کانتینر ها را اجرا میکند.

۳. kube-proxy

این عامل ارتباطاتی در گره کارگر است که IP های پاد ها در IPtables نگه داری میکند و هنگامی که این IP ها تغییر کرد باید جدول را بروزرسانی کند. همچنین درخواست هایی که از طرف کاربران به این گره وارد میشود را به پاد ها می فرستد. در فصل ۳ بیشتر به نحوه ارتباطات پاد ها با یکدیگر و محیط بیرون میپردازیم.

# ۴. جمع بندی

در این فصل؛ مزیت های کانتینر ها نسبت به ماشین های مجازی گفته شد و همچنین محیط داکر به منظور اجرا شدن کانتینر ها معرفی و توضیح داده شد. در بخش آخر ما پلتفرم کوبرنتیز و اجزای مختلف آن را توضیح دادیم و نحوه بارگذاری و اجرای داکر بر روی این پلتفرم بیان گردید. دلیل توجه ما به این موضوع این است که ما قصد داریم سرویس های خود را به صورت ایمیج درآوریم و از طریق داکر آن ها را اجرا کنیم و با استفاده از پلتفرم کوبرنتیز مدیریت خودکار بر روی این کانتینر ها داشته باشیم. عملکرد گره های رهبر و کارگر نیز در محیط کوبرنتیز در این رابطه تشریح گردید.