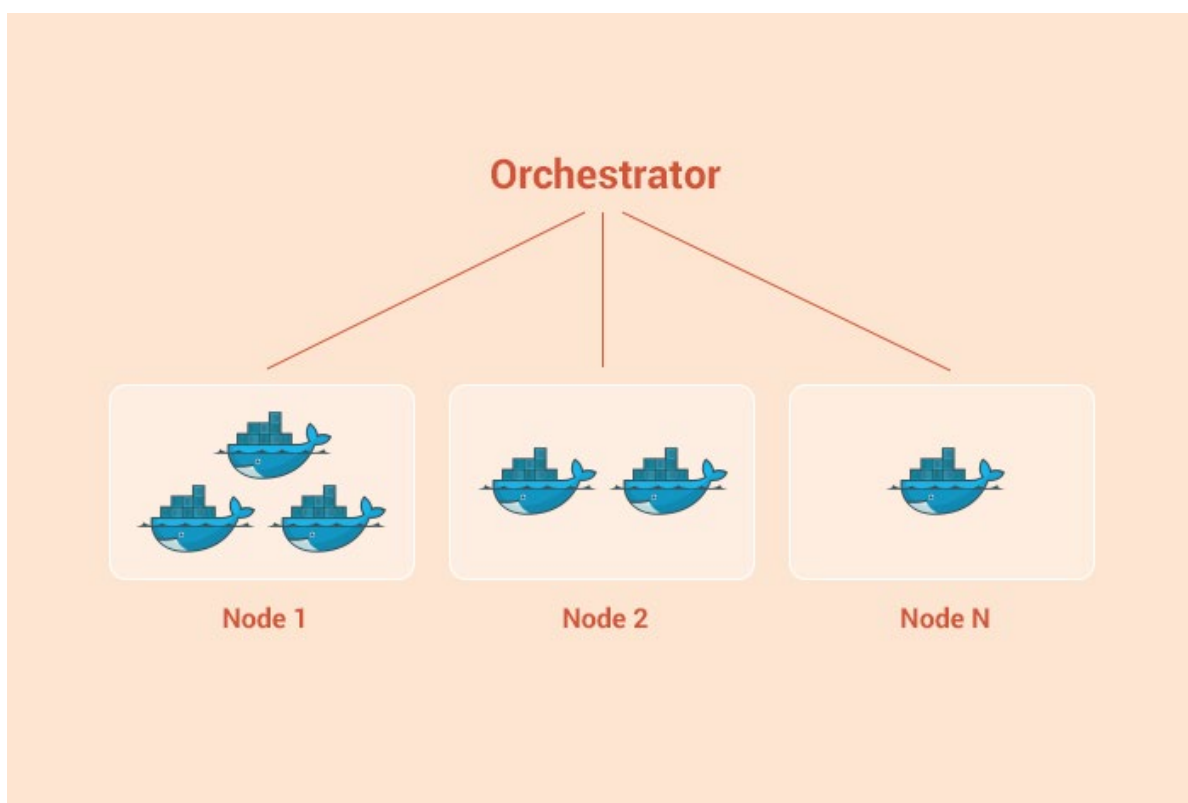


آشنایی با Orchestration

کانتینری شدن سرویس‌ها کمک شایانی به راه‌اندازی پلتفرم‌ها و نرم‌افزارها کرده است. با زیاد شدن container ها به جهت مدیریت و افزوده شدن ویژگیهای جدید ابزارهایی به نام orchestrator به جود آمدند.

در معماری مایکروسرویس ابزارهای مختلف و پیشرفته به جهت Deploy کردن نرم‌افزار و سرویس‌ها پدید آمدند. امکاناتی به وجود آمدند که همه توسعه‌دهندگان را به جهت پایداری نرم‌افزارهای خود به سمت و سوی ابزارهایی همچون Docker و Kubernetes هدایت کردند. یکی از مهمترین مزیت‌های استفاده از چنین ابزارهایی پایداری شدن سرویس‌ها و نرم‌افزارها است و با ایجاد Transparency چه از دیدگاه توسعه دهنده و چه از دیدگاه کاربر اعتماد و تمایل کاربران به استفاده از نرم‌افزارها نیز افزایش یافته است. همان‌طور که در مقالات ارائه شده قبلی اشاره شد ابزارهای container engine یا container run time در معماری مایکروسرویس، نرم‌افزارها و سرویس‌ها را به صورت container base آپ می‌کنند و همین کانتینری شدن سرویس‌ها کمک شایانی به راه‌اندازی پلتفرم‌ها و نرم‌افزارها کرده است. با زیاد شدن container ها به جهت مدیریت و افزوده شدن ویژگیهای جدید ابزارهایی به نام orchestrator به جود آمدند. در حقیقت orchestrator ها ابزارهای پیشرفته‌ای هستند که به جهت ایجاد کلاسترینگ در مدیریت container ها به وجود آمدند.




1. Kubernetes

کوبرنتیز که اغلب به عنوان K8s هم شناخته می شود، محیطی را فراهم می کند که اپلیکیشن های کانتینری بر روی سیستم های میزبان مختلف دیپلوی، اجرا و مقیاس بندی شوند.


این مجموعه قوی و پیچیده برای خودکار سازی بسیاری از اقدامات مربوط به چرخه عمر اپلیکیشن طراحی شده است و مثل بازی تتریس، کانتینرها را برای یک گردهمایی با منابع محاسباتی بهینه انتخاب می کند تا این که حجم کار به پایان برسد. قابلیت های دیگری مثل تعمیر خودکار و راه اندازی مجدد کانتینرها در صورت خرابی هم جزو وظایف این پلتفرم است.

هدف اصلی این ابزار، ساده سازی کار تیم فنی است چون بسیاری از کارهای مربوط به دیپلوی برنامه ها که قبلاً به صورت دستی انجام می شود، با Kubernetes به صورت خودکار انجام می شود.

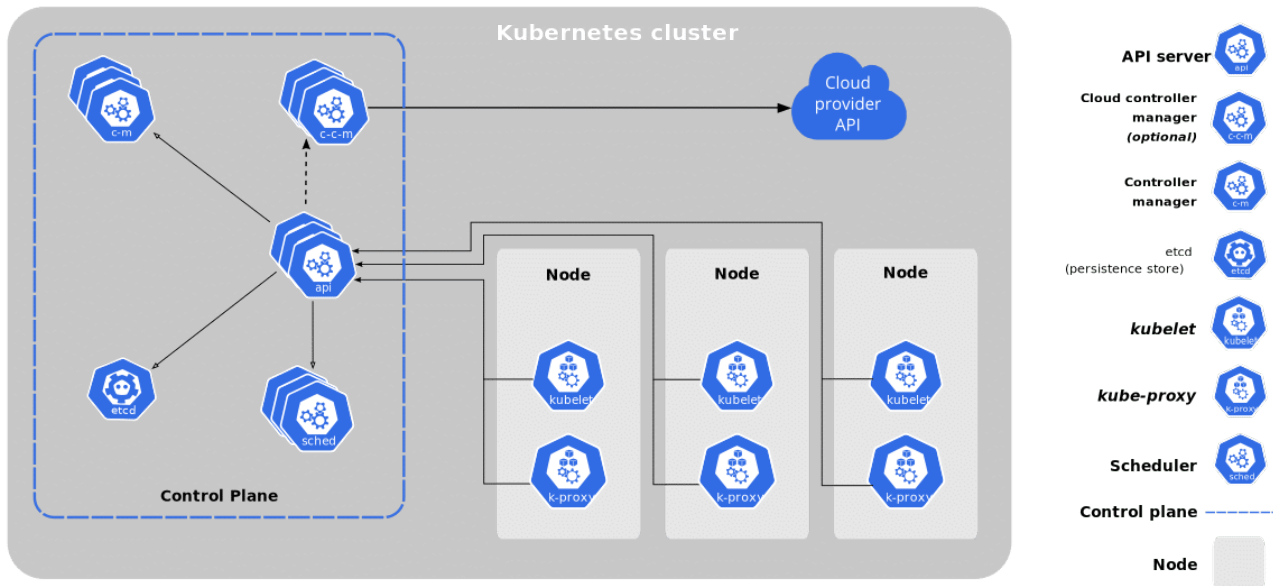
Kubernetes Advantages



- 1 Automatic Bin packing
- 2 Load Balancing & Service Discovery
- 3 Storage Orchestration
- 4 Self-Healing
- 5 Secret & Configuration Management
- 6 Batch Execution
- 7 Horizontal Scaling
- 8 Automatic Rollbacks & Rollouts



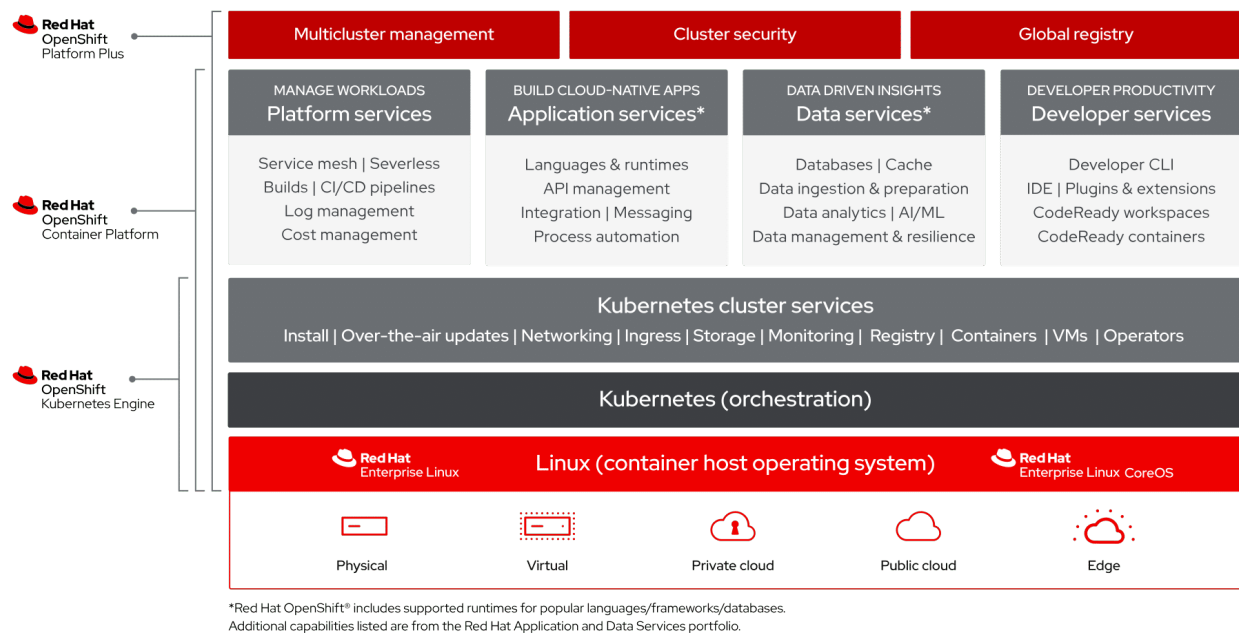
معماری Kubernetes



2. Openshift

شرکت Redhat نیز راهکاری را ارایه نموده است به نام **OpenShift**. نسخه Enterprise آن **Redhat OpenShift** نام دارد که هم به صورت **On-Premise** و هم روی **Private/Public Cloud** پیاده سازی می شود. نسخه Community آن در ابتدا **OpenShift** نام داشت اما اخیراً به نام **OKD** منتشر می شود.

برخی از قابلیت ها در **Kubernetes** به شکل دیگری در **OpenShift** ارایه می شوند مثلاً در **Kubernetes** از **Helm Charts** برای نصب پکیج استفاده می شود و یا از **Ingress** در تنظیمات شبکه برای ارایه سرویس استفاده می شود که نوعی **reverse proxy** بر اساس **nginx** است اما در **OpenShift** از **Template** های به عنوان پکیج استفاده می شود و یا برای شبکه سرویس ها از **Route** استفاده می شود که بر اساس **F5-Big IP** ساخته شده است.



OpenShift که توسط Red Hat توسعه داده شده است با زبان های Go و AngularJS نوشته شده است. این برنامه از Node.js، Go، Java، Python، PHP و Ruby پشتیبانی می کند، اما می توان آن را برای پشتیبانی از زبان های برنامه نویسی دیگر گسترش داد. OpenShift به راحتی با سایر ابزارهای DevOps ادغام می شود و با Open Container Initiative (OCI) برای میزبانی کانتینر و زمان اجرا سازگار است. می تواند از کانتینرهای Docker استفاده کند و از آنجایی که مبتنی بر Kubernetes است، برای توسعه دهندگانی که از آن پلتفرم ها می آیند، آشنا خواهد بود.

شرکت هایی که با OpenShift کار می کنند به دنبال یک پلتفرم همه کاره با سیاست های امنیتی سختگیرانه، استقرار سریعتر برنامه ها و پشتیبانی اختصاصی هستند. این ویژگی ها آن را به یک راه حل بسیار جذاب برای پروژه های در مقیاس بزرگ یا شرکت های کوچکتر که فاقد منابع اختصاصی برای مدیریت، ایمن سازی و نظارت بر برنامه های خود هستند تبدیل می کند.

مقایسه OpenShift با Kubernetes

مقایسه تفاوت های این دو ابزار با یکدیگر

تجاری در مقابل رایگان

بزرگترین تفاوت بین آنها این است که OpenShift یک محصول تجاری مبتنی بر اشتراک است و Kubernetes برای استفاده به عنوان یک پروژه منبع باز رایگان است.

اشتراک OpenShift شامل مجموعه کاملی از ابزارها و پشتیبانی اختصاصی است. Kubernetes دارای پشتیبانی جامعه است و با سایر ابزارهای شخص ثالث برای وظایف یا عملیات خاص ترکیب می شود.

امنیت

OpenShift از همان ابتدا سیاست های امنیتی سختگیرانه ای دارد. به عنوان مثال، به حداقل امتیازات کاربر حتی برای عملیات های اساسی نیاز دارد و همچنین کانتینرهای Docker را برای اجرا به عنوان تصاویر ساده محدود می کند.

ویژگی های امنیتی Kubernetes به تنظیمات پیچیده تری نیاز دارد، زیرا فاقد قابلیت های احراز هویت و مجوز بومی است و به یک API برای استفاده با ابزارهای شخص ثالث برای این منظور استفاده می شود. پروتکل امنیتی دقیقاً مانند OpenShift تعریف نشده است، زیرا هیچ رمزگذاری پیش فرضی در یک خوشه وجود ندارد و K8s را بیشتر مستعد حملات می کند.

داشبورد

OpenShift با یک کنسول وب ساده یک تجربه کاربری عالی را فراهم می کند. داشبورد ساده و مبتنی بر فرم به کاربران امکان می دهد تمام منابع را در یک محیط تمیز و ساده مدیریت کنند.

رابط کاربری Kubernetes سخت تر است. برای دسترسی به رابط کاربری گرافیکی (رابط کاربری گرافیکی)، توسعه دهندگان باید داشبورد اختصاصی Kubernetes را نصب کنند، سپس فرآیند احراز هویت و مجوز را برای دسترسی به آن راه اندازی کنند، زیرا این رابط حتی یک صفحه ورود نیز ندارد. توسعه دهندگان پیشرفته تر با این مشکلی نخواهند داشت، اما ممکن است از همان ابتدا مانعی برای مبتدیان شود.

به روز رسانی و پشتیبانی

OpenShift، به عنوان یک محصول تجاری، خدمات، پشتیبانی و راهنمایی اختصاصی به مشتریان ارائه می دهد. Kubernetes، به عنوان یک پروژه منبع باز، مبتنی بر جامعه و رایگان، این کار را نخواهد کرد. اگر توسعه دهندگان با مشکلی در Kubernetes مواجه شدند، باید به تجربه توسعه دهندگان دیگر در انجمن ها تکیه کنند و منتظر پاسخ سوالات خود باشند. OpenShift تیمی از مهندسان Red Hat دارد که 7/24 آماده کمک هستند.

Built-in vs. Third Party

مجموعه OpenShift به طور پیش فرض شامل ویژگی هایی مانند نظارت و شبکه است. Prometheus و Grafana دو ابزار نظارتی هستند که در مورد مسائل موجود در stack هشدار می دهند. با Open vSwitch، یک راه حل بومی OpenShift، شبکه خارج از جعبه فعال می شود

تفاوت های فنی بین OpenShift و Kubernetes

Integrated CI/CD

CI یا ادغام مداوم، بهترین روش DevOps است. CI به معنای اجرای تست های خودکار برای بررسی اینکه آیا ادغام تغییرات در کد اصلی برنامه را خراب نمی کند و اطمینان حاصل شود که هیچ چالش یکپارچه سازی با هر commit جدید وجود ندارد. CD یا تحویل مداوم، پس از یکپارچه سازی مداوم یا همراه با آن اتفاق می افتد. پس از مرحله ساخت، تمام تغییرات کد در محیط آزمایش و/یا تولید مستقر می شوند. OpenShift از Jenkins استفاده می کند، یک سرور اتوماسیون که پشتیبانی از منبع به image را فراهم می کند و می تواند به عنوان یک سرور CI استفاده شود. Kubernetes همچنین به یک ابزار شخص ثالث به نام CircleCI برای ایجاد یک جریان CI/CD متکی است.

Image Registry

توسعه دهندگان می توانند یک رجیستری Docker در Kubernetes راه اندازی کنند، اما یک رجیستری image یکپارچه ارائه نمی دهد. از سوی دیگر، OpenShift دارای یک رجیستری image یکپارچه برای استفاده با Red Hat یا Docker Hub از طریق کنسولی است که حاوی تمام اطلاعات مربوط به image موجود در پروژه است.

Deployment

OpenShift و Kubernetes رویکردهای متفاوتی در مورد استقرار دارند. OpenShift ممکن است پیچیده تر به نظر برسد، اما مزایای اضافه تری دارد، مانند راه اندازی برای استقرار خودکار. Kubernetes اجزا را با استفاده از کنترلرهای پیاده سازی می کند، در حالی که OpenShift از یک دستور استفاده می کند. دستور استقرار OpenShift از چندین به روز رسانی پشتیبانی نمی کند، اما اجزا استقرار Kubernetes می توانند به روز رسانی های همزمان را مدیریت کنند.

Kubernetes از Helm استفاده می کند، مجموعه ای از مانیفست های YAML که برای ساده سازی استقرار برنامه های کاربردی کانتینری ساخته شده است. این رویکرد ساده تر از قالب های OpenShift است که فاقد سادگی و پیچیدگی نمودارهای Helm هستند. استقرار تک پاد OpenShift ممکن است در سناریوهای پیچیده تر موثر نباشد.

3. Hasicorp Nomad

Nomad یک پلت فرم ارکستراسیون از Hashicorp است که از کانتینرها پشتیبانی می کند. این فلسفه مشابهی از kubernetes در مدیریت برنامه های کاربردی در مقیاس دارد.

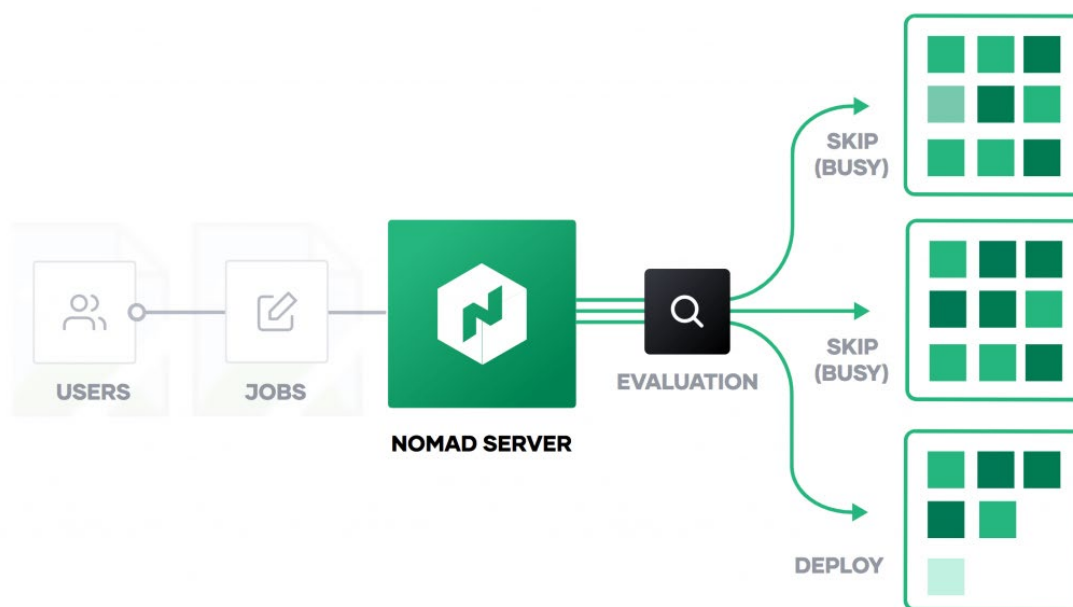
با این حال، Nomad از workload کانتینری و غیر کانتینری پشتیبانی می کند. همچنین Nomad با ادغام خوبی از سایر ابزارهای Hashicorp مانند Consul، Vault و terraform ارائه می شود.

موارد استفاده اولیه برای Nomad عبارتند از

1-ارکستراسیون کانتینری

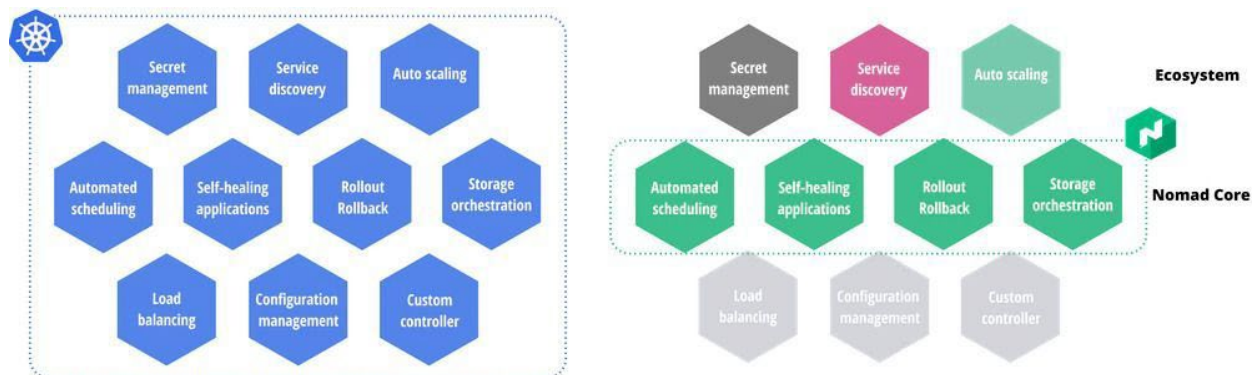
2-ارکستراسیون برنامه غیر کانتینری.

3-شبکه خدمات خودکار با کنسول.



تفاوت اصلی Nomad و Kubernetes این است که Nomad همه کاره تر و سبک وزن است. Nomad می تواند فقط به عنوان یک زمان بندی کار ساده کار کند یا با توجه به مشخصات پروژه نقش های ارکستراسیون سنگین تری را بر عهده بگیرد.

Kubernetes به عنوان یک پلتفرم با ویژگی‌های کامل همراه با تمام اجزای آن عرضه می‌شود. Nomad بسیاری از مؤلفه‌هایی را که در صورت لزوم می‌توانند بعداً اضافه شوند، کنار گذاشته و نیاز به وابستگی‌های خارجی را به حداقل می‌رساند.



نصب و راه اندازی

Nomad سبک وزن و نصب آسانی دارد. این به عنوان یک باینری ساده ارائه می‌شود که به سرعت در یک ماشین توسعه محلی یا محیط ابری با ثبات و عملکرد یکسان استقرار می‌یابد.

Kubernetes به زمان و منابع بیشتری برای استقرار نیاز دارد و فرآیند نصب پیچیده تر است. پیاده سازی‌های سبک تر Kubernetes دیگری نیز وجود دارند که تنها تعدادی از ویژگی‌های کامل را پوشش می‌دهند. این‌ها برای توسعه و آزمایش سریع استفاده می‌شوند، اما به خوبی در مرحله تولید ترجمه نمی‌شوند زیرا مستعد ناسازگاری‌های پیکربندی هستند.

مقیاس پذیری

Kubernetes (تا تاریخ انتشار) ادعا می‌کند که از خوشه‌هایی تا 5000 گره با 300000 کانتینر و حداکثر 150000 پاد پشتیبانی می‌کند.

Nomad نشان داد که می‌تواند به اندازه‌های خوشه‌ای بالاتر از 10000 گره مقیاس شود. و چالش 2 میلیون کانتینری ارجاع شده در سال 2020 ادعاهای آنها را در مورد عملکرد مقیاس پذیری برتر تأیید کرد.

شبکه سازی

در Kubernetes، پادها از طریق یک شبکه هم‌تا به هم‌تا ارتباط برقرار می‌کنند. این مدل شبکه به دو CIDR (روتر بین دامنه‌ای بدون کلاس) نیاز دارد: یکی برای آدرس‌دهی IP Node و دیگری برای خدمات.

در Nomad هر کار به صورت پیش فرض یک IP دریافت می‌کند. سپس می‌توان به پورت‌های مربوطه به‌طور مستقیم یا از طریق پراکسی‌های جانبی، با استفاده از شبکه میزبان، انتقال پویا یا استاتیک پورت با کمک مؤلفه Consul، دسترسی پیدا کرد.

مشخصات مورد نیاز

Kubernetes به سخت‌افزار بیشتری نیاز دارد که برای پروژه‌های بلندمدت و با سرمایه‌گذاری بیشتر در محیط‌های ابری عمومی مانند Google Cloud Platform، Azure یا AWS استفاده می‌شود.

Nomad برای تیم‌های کوچک‌تر، با ظرفیت محدود برای اهداف ارکستراسیون، با مهلت‌های توسعه کوتاه‌تر، کار بر روی محیط‌های ترکیبی یا درون محل مناسب است.

زبان ها

Kubernetes از YAML یا JSON برای تعریف و استقرار برنامه ها استفاده می کند، Nomad از زبان پیکربندی Hashicorp (HCL) استفاده می کند.

HCL هم یک syntax و هم یک API است که توسط Hashicorp برای ساخت فرمت های پیکربندی ساختاریافته طراحی شده است

Load Balancing

یک متعادل کننده بار ترافیک ورودی را از اینترنت به برنامه های کاربردی فرانت اند که مسئول رسیدگی به درخواست ها هستند توزیع می کند.

محبوب ترین راه حل در Kubernetes برای تعادل بار، Ingress است، یک کنترلر تخصصی (Kubernetes بیش از حد شبیه به یک pod)

Ingress شامل مجموعه ای از قوانین برای مدیریت ترافیک و دیمنی برای اعمال آنها می شود. این قوانین را می توان برای نیازهای پیشرفته تر تطبیق داد.

Nomad عملکردی مشابه کنترلر Ingress Kubernetes دارد که می تواند به راحتی با تغییرات پیکربندی و مقیاس سازگار شود.

Integration

Nomad با Docker درست مانند Kubernetes کار می کند و بارهای کاری غیر کانتینری (ویندوز، جاوا) را اجرا می کند.

رابط کاربری گرافیکی (GUI)

هر دو دارای داشبوردهای جذاب و کاربردی هستند که تجربه مدیریتی واضح و ساده ای را ارائه می دهند.

4. Docker Swarm

، درواقع مجموعه‌ای از ماشین‌هاست) سرورهای فیزیکی یا Virtual Machine ها (که همگی داکر را اصطلاحاً در حالت swarm mode بر روی خود نصب دارند و یک کلاستر واحد را تشکیل می‌دهند. این کلاستر از تعدادی manager (یا مسترهای کلاستر) و تعدادی worker (یا مینیون‌های کلاستر) تشکیل شده است که وظیفه manager ها مدیریت کلاستر و وظیفه worker ها اجرای سرویس‌های موردنظرما است. یک نود در کلاستر می‌تواند manger باشد، می‌تواند worker باشد یا اینکه هم manager باشد و هم worker باشد. هنگامی که یک سرویس را در کلاستر سوارم ایجاد می‌کنیم، وضعیتی که انتظار داریم آن سرویس به آن برسد و آن را حفظ کند را تعریف می‌کنیم. برای مثال ویژگی‌هایی مثل تعداد replica ها (مثلاً می‌خواهیم ۳ نسخه از سرویس روی ۳ نود مختلف اجرا شود)، مقدار منابع موردنیاز و در دسترس (مثل CPU و RAM و پورت‌هایی که می‌خواهیم expose بکنند را مشخص می‌کنیم. از اینجا به بعد، manager های داکر سوارم، سعی می‌کنند همواره سرویس ما در state و وضعیتی که تعریف کرده ایم باقی بماند؛ مثلاً اگر یکی از نودهای کلاستر از مدار خارج بشود، manager ها یک replica دیگر از سرویس بر روی یک نود دیگر بالا می‌آورند.

یکی از مزایای مهم داکر سوارم نسبت به اجرای کانتینرهای مستقل روی داکر در حالت معمولی این است که می‌توان سرویس را بدون down-time تغییر داد، یعنی بدون نیاز به restart کردن دستی سرویس می‌توان آن را update کرد. داکر تغییر جدید را روی replica های سرویس به ترتیب اعمال می‌کند، یعنی پس از update کردن یک replica، به سراغ بعدی می‌رود و در نتیجه کل عملیات بدون نیاز به restart کردن کل کلاستر انجام می‌شود.

زمانی که داکر در حالت swarm mode اجرا می‌شود، هم‌چنان می‌توان کانتینرهای مستقل را هم روی آن اجرا کرد؛ با این تفاوت که هر نودی می‌تواند کانتینرهای مستقل را اجرا کند اما فقط manager های کلاستر سوارم می‌توانند سرویس‌های سوارم را اجرا کنند.

۲. مزایا و ویژگی های Docker Swarm

یکپارچه شدن مدیریت کلاستر سوارم با : Docker Engine ایجاد و مدیریت کردن کلاستر سوارم و دیپلوی کردن سرویس ها روی آن با همان Docker CLI انجام می شود و هیچ ابزار orchestration اضافه ای برای کار با سوارم نیاز نیست.

طراحی غیرمتمرکز: تفاوت‌هایی که بین نودهای مختلف (manger) و (worker وجود دارد، در زمان اجرا (runtime) هندل می‌شود و این باعث می‌شود که تمام نودهای مختلف را از روی یک image واحد بتوان ساخت.

مدل Declarative برای تعریف سرویس‌ها و اپلیکیشن‌ها: داکر به ما این امکان را می‌دهد که با استفاده از یک روش declarative وضعیت و state مورد نظر سرویس خود را مشخص کنیم.

امکان افزایش یا کاهش تعداد replica ها : (scaling) برای هر سرویس می‌توان در هر زمان تعداد instance های در حال اجرا را افزایش یا کاهش داد؛ در نتیجه manager سوارم به‌صورت خودکار instance های جدیدی را اضافه یا برخی از instance های قبلی را حذف می‌کند.

نگه داشتن همیشگی سرویس‌ها در وضعیت تعریف شده: نود manager به طور پیوسته وضعیت کلاستر را بررسی می‌کند و اگر تفاوتی بین state تعریف شده و state حال حاضر یک سرویس مشاهده کند، تلاش می‌کند تا این تفاوت را از بین ببرد و همواره سرویس را در همان

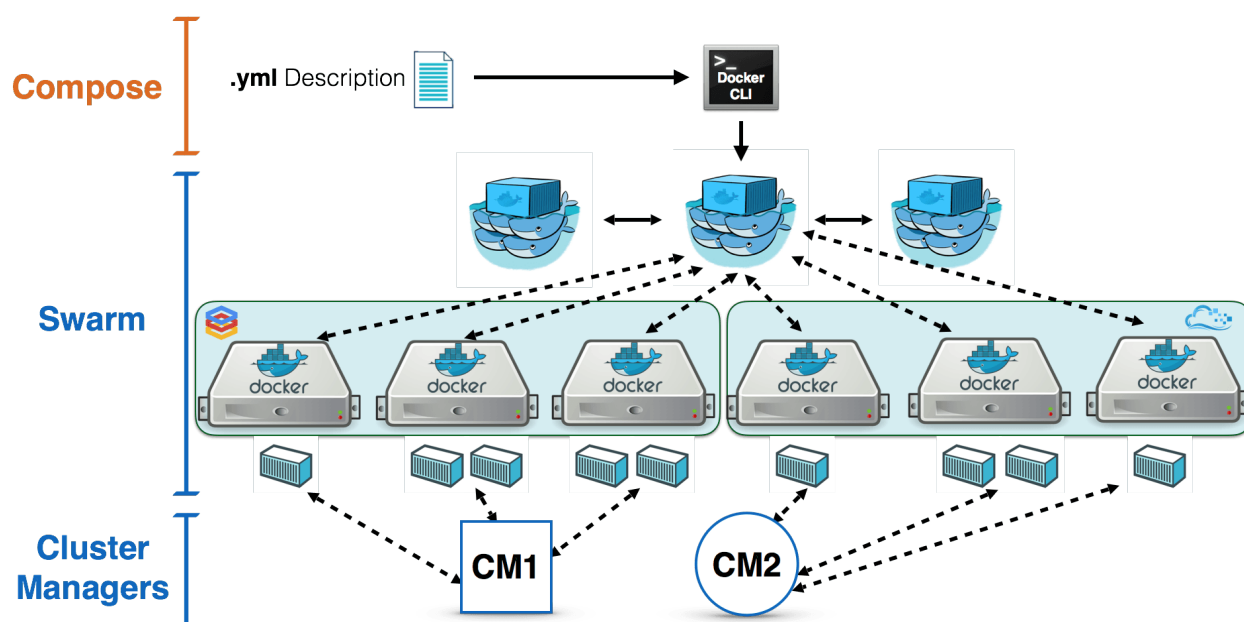
state تعریف شده نگه دارد. برای مثال، اگر تعریف کرده باشیم که از یک سرویس، ۱۰ تا replica موجود باشد اما به دلیل down شدن یکی از سرورها، ۲ تا از این replica ها از کار افتاده باشند، manager این ۲ تا instance را روی نود سالم دیگری بالا خواهد آورد تا state تعریف شده برای این سرویس حفظ شود.

شناسایی سرویس یا : Service discovery نودهای manger کلاستر سوارم، به هر سرویس یک نام DNS یکتا می‌دهند و بین containerهای در حال اجرای این سرویس در هنگام کوئری لودبالانس می‌کنند.

توزیع بار یا Load balancing: داکر سوارم به ما این امکان را می‌دهد که پورت‌های یک سرویس را expose کرد تا بتوان از یک load balancer برای توزیع بار بین container های مختلف استفاده کرد. هم‌چنین، سوارم به ما این قابلیت را می‌دهد که مشخص کنیم چگونه containerهای مختلف را بین نودهای مختلف توزیع کند.

امنیت بالا: به صورت پیش‌فرض، هر نود از TLS authentication و encryption استفاده می‌کند تا تمامی ارتباطات بین خودش و نودهای دیگر را با امنیت حداکثری فراهم کند.

قابلیت آپدیت به ترتیب containerها یا Rolling updates: داکر سوارم به ما این امکان را می‌دهد که بتوانیم containerهای یک سرویس را با فاصله از هم آپدیت کنیم و این دلیلی زمانی بین آپدیت یک container تا آپدیت container بعدی هم قابل تنظیم است. در نتیجه بدون نیاز به down-time می‌توانیم تغییرات مختلف را روی سرویس خود اعمال کنیم. هم‌چنین، در صورت بروز مشکل در اعمال کردن تغییرات جدید، امکان برگشتن به ورژن قبلی (roll back) وجود دارد.



مزایا و ویژگی های Docker Swarm

یکپارچه شدن مدیریت کلاستر سوآرم با Docker Engine: ایجاد و مدیریت کردن کلاستر سوآرم و دیپلوی کردن سرویس ها روی آن با همان Docker CLI انجام می شود و هیچ ابزار orchestration اضافه ای برای کار با سوآرم نیاز نیست.

طراحی غیرمتمرکز: تفاوت هایی که بین نودهای مختلف (manger و worker) وجود دارد، در زمان اجرا (runtime) هندل می شود و این باعث می شود که تمام نودهای مختلف را از روی یک image واحد بتوان ساخت.

مدل Declarative برای تعریف سرویس ها و اپلیکیشن ها: داکر به ما این امکان را می دهد که با استفاده از یک روش declarative وضعیت و state مورد نظر سرویس خود را مشخص کنیم.

امکان افزایش یا کاهش تعداد replica ها (scaling): برای هر سرویس می توان در هر زمان تعداد instance های در حال اجرا را افزایش یا کاهش داد؛ در نتیجه manager سوآرم به صورت خودکار instance های جدیدی را اضافه یا برخی از instance های قبلی را حذف می کند.

نگه داشتن همیشگی سرویس ها در وضعیت تعریف شده: نود manager به طور پیوسته وضعیت کلاستر را بررسی می کند و اگر تفاوتی بین state تعریف شده و state حال حاضر یک سرویس مشاهده کند، تلاش می کند تا این تفاوت را از بین ببرد و همواره سرویس را در همان state تعریف شده نگه دارد. برای مثال، اگر تعریف کرده باشیم که از یک سرویس، ۱۰ تا replica موجود باشد اما به دلیل down شدن یکی از سرورها، ۲ تا از این replica ها از کار افتاده باشند، manager این ۲ تا instance را روی نود سالم دیگری بالا خواهد آورد تا state تعریف شده برای این سرویس حفظ شود.

شناسایی سرویس یا Service discovery: نودهای manger کلاستر سوآرم، به هر سرویس یک نام DNS یکتا می دهند و بین container های در حال اجرای این سرویس در هنگام کوئری لودبالانس می کنند.

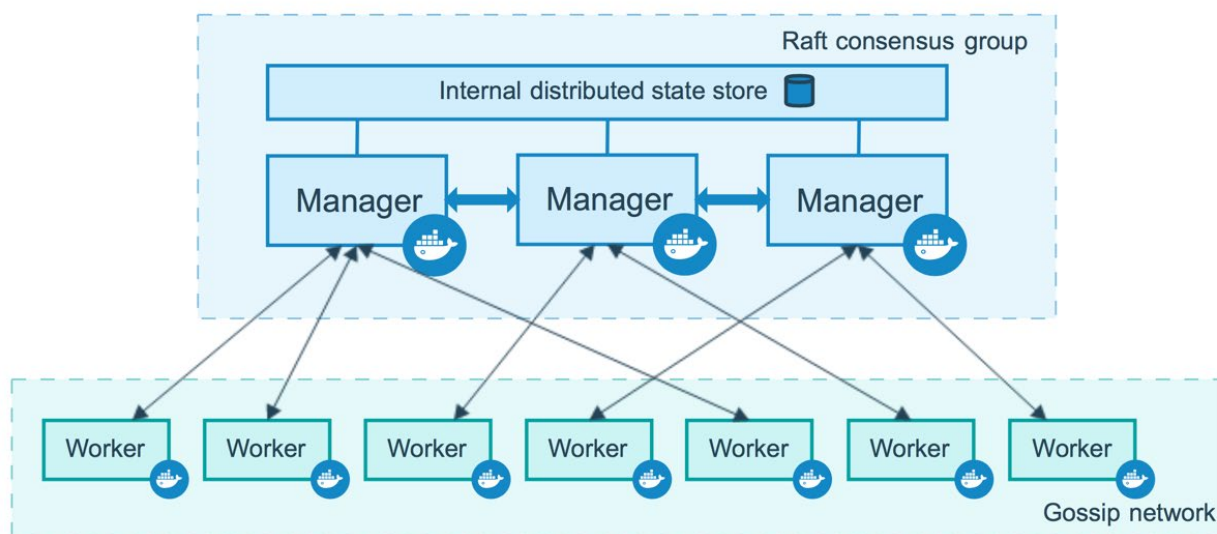
توزیع بار یا Load balancing: داکر سوآرم به ما این امکان را می دهد که پورت های یک سرویس را expose کرد تا بتوان از یک load balancer برای توزیع بار بین container های مختلف استفاده کرد. همچنین، سوآرم به ما این قابلیت را می دهد که مشخص کنیم چگونه container های مختلف را بین نودهای مختلف توزیع کند.

امنیت بالا: به صورت پیش فرض، هر نود از TLS authentication و encryption استفاده می کند تا تمامی ارتباطات بین خودش و نودهای دیگر را با امنیت حداکثری فراهم کند.

قابلیت آپدیت به ترتیب container ها یا Rolling updates: داکر سوآرم به ما این امکان را می دهد که بتوانیم container های یک سرویس را با فاصله از هم آپدیت کنیم و این دلیلی زمانی بین آپدیت یک container تا آپدیت container بعدی هم قابل تنظیم است. در نتیجه بدون نیاز به down-time می توانیم تغییرات مختلف را روی سرویس خود اعمال کنیم. همچنین، در صورت بروز مشکل در اعمال کردن تغییرات جدید، امکان برگشتن به ورژن قبلی (roll back) وجود دارد

مفاهیم کلیدی

نودها (Nodes)



یک نود به معنای یک instance یا نمونه از Docker engine است که در کلاستر swarm حضور دارد. مثلاً زمانی که روی یک کامپیوتر Docker نصب می‌کنیم و آن را در حالت swarm mode قرار می‌دهیم، این کامپیوتر یک نود از کلاستر سوارم ما خواهد بود. بر روی یک سرور فیزیکی می‌توان یک یا بیشتر از یک نود داشت اما در محیط production معمولاً نودهای کلاستر سوارم بر روی سرورهای فیزیکی متفاوتی قرار دارند.

برای اینکه یک اپلیکیشن را روی کلاستر سوارم دیپلوی کنیم، تعریف سرویس خود را به یک نود manager می‌دهیم. این سرویس به یک سری task شکسته می‌شود و این نود این task ها را به نودهای worker برای اجرا می‌فرستد.

نودهای manager کلاستر هم‌چنین وظیفه orchestration و مدیریت کردن کلاستر را برعهده دارند تا سرویس‌های تعریف شده روی کلاستر سوارم همواره در state خواسته شده قرار داشته باشند. نودهای manager بین خود، یک لیدر انتخاب می‌کنند که وظیفه orchestration برعهده نود لیدر است.

نودهای worker هم در طرف مقابل task هایی که نودهای manager به آن‌ها می‌فرستند را دریافت و اجرا می‌کنند. به صورت پیش‌فرض، نودهای manager به عنوان worker هم عمل می‌کنند؛ یعنی سرویس‌ها روی آن‌ها هم اجرا می‌شوند. اما می‌توان کلاستر را به گونه‌ای کانفیگ کرد که نودهای manager فقط وظیفه مدیریت کردن کلاستر را برعهده داشته باشند و به اصطلاح manager-only باشند. یک agent روی هر نود worker قرار دارد که پیوسته وضعیت task هایی که به آن نود assign شده است را به اطلاع نود manager می‌رساند و بدین ترتیب manager می‌تواند state هر worker را مدیریت و نگهداری کند.

سرویس‌ها و task ها

سرویس (Service)، تعریف کارها و task هایی است که باید روی نودهای manager یا worker اجرا شود و درواقع محل اصلی تعامل کاربر با کلاستر سوارم است. در زمان ایجاد کردن یک سرویس جدید، کاربر مشخص می‌کند که از چه image استفاده شود و چه دستورات و command هایی داخل container های ساخته شده اجرا شوند.

در سرویس‌هایی که replication دارند، manager بسته به تعداد replica های خواسته شده توسط کاربر در تعریف سرویس، تسک‌هایی را به نودهای worker کلاستر assign می‌کند.

برای سرویس‌های global، سوارم بر روی تمام نودهای کلاستر یک task از سرویس را اجرا می‌کند.

یک task، از یک container داکر و دستورات و command هایی که باید داخل آن container اجرا شوند تشکیل می‌شود task.، کوچکترین واحد اتمی scheduling در سوارم محسوب می‌شود. نودهای manager بسته به اینکه تعداد replica ها چقدر است، تسک‌هایی را به نودهای worker تخصیص می‌دهند. لحظه ای که یک task به یک نود اختصاص داده شد، دیگر نمی‌تواند به نود دیگری انتقال یابد؛ یا روی نود اجرا می‌شود یا fail می‌شود.

تبادل بار (Load balancing)

سوارم از ingress load balancing برای در دسترس قراردادن سرویس‌هایی که می‌خواهیم از خارج از کلاستر سوارم دیده شوند استفاده می‌کند. بدین منظور، هم امکان این وجود دارد که manager سوارم به صورت اتوماتیک یک PublishedPort در رنج 30000-32767 به سرویس موردنظر ما assign کند و هم امکان این وجود دارد که یک پورت مشخص به آن تخصیص دهیم.

با چنین مکانیزمی، سرویس ما به راحتی از طریق پورت مشخص شده (PublishedPort) بر روی هر یک از نودهای کلاستر سوارم در دسترس است؛ حتی اگر روی آن نود هیچ container از آن سرویس وجود نداشته باشد. درواقع تمام نودهای کلاستر توانایی route کردن ترافیک ورودی به یک instance از سرویس ما را دارند.

در حالت swarm به صورت خودکار به هر سرویس یک DNS اختصاص داده می‌شود و manager سوارم براساس نام DNS هر سرویس، ریکوئست‌های درون کلاستر را بین سرویس‌های مختلف پخش می‌کند که در واقع یک load balancing داخلی برای کلاستر محسوب می‌شود.

Features	Kubernetes	Docker Swarm
Installation	Complex Installation but a strong resultant cluster once set up	Simple installation but the resultant cluster is not comparatively strong
GUI	Comes with an inbuilt Dashboard	There is no Dashboard which makes management complex
Scalability	Highly scalable service that can scale with the requirements. 5000 node clusters with 150,000 pods	Very high scalability. Up to 5 times more scalable than Kubernetes. 1000 node clusters with 30,000 containers
Load Balancing	Manual load balancing is often needed to balance traffic between different containers in different pods	Capability to execute auto load balancing of traffic between containers in the same cluster
Rollbacks	Automatic rollbacks with the ability to deploy rolling updates	Automatic rollback facility available only in Docker 17.04 and higher if a service update fails to deploy
Logging and Monitoring	Inbuilt tools available for logging and monitoring	Lack of inbuilt tools. Needs 3rd party tools for the purpose
Node Support	Supports up to 5000 nodes	Supports 2000+ nodes
Optimization Target	Optimized for one single large cluster	Optimized for multiple smaller clusters
Updates	The in-place cluster updates have been constantly maturing	Cluster can be upgraded in place
Networking	An overlay network is used which lets pods communicate across multiple nodes	The Docker Daemons is connected by overlay networks and the overlay network driver is used
Availability	High availability. Health checks are performed directly on the pods	High availability. Containers are restarted on a new host if a host failure is encountered

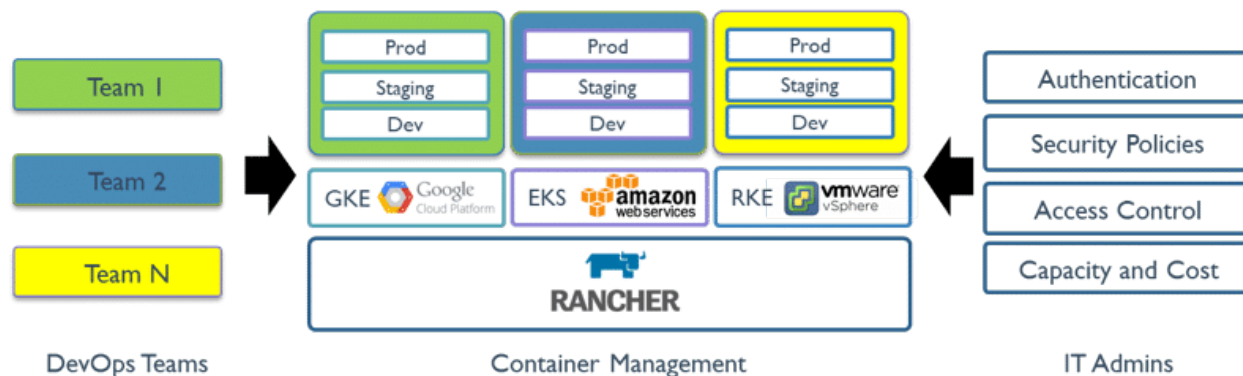
Kubernetes vs Docker Swarm 1 versus

5.Rancher

Rancher یک ابزار ارکستراسیون کانتینر منبع باز است. در هسته، از kubernetes به عنوان ارکستراتور کانتینر استفاده می کند.

Rancher عملکردهای زیر را ارائه می دهد.

1. Centralized Cluster Provisioning that supports on-prem, cloud, and edge.
2. Streamlined Kubernetes Operations by controlling cluster operations from a single console.
3. Centralized Kubernetes Security through centralized user policies.
4. Intuitive Workload Management using native kubernetes API or kubectl utility.
5. Integrated Monitoring and Logging using Prometheus, Fluentd, and Grafana.
6. Supports management for Amazon EKS clusters & Google Kubernetes Engine (GKE)
7. Global Application Catalog to make application installation and upgrade easier.

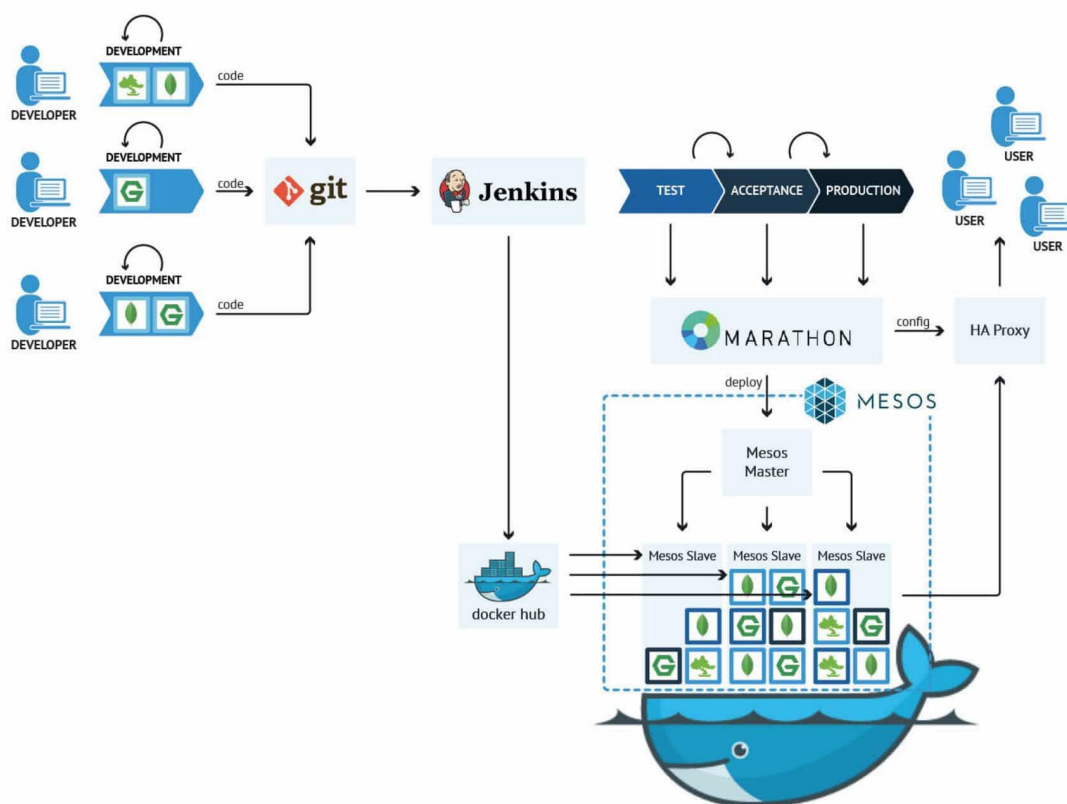


6.Mesos

Mesos یکی دیگر از ابزارهای مدیریت کلاستر است که می تواند ارکستراسیون کانتینر را بسیار کارآمد مدیریت کند. توسط توییتر برای زیرساخت خود ایجاد شد و سپس منبع باز شد. این توسط شرکت هایی مانند eBay، Airbnb و غیره استفاده می شود. Mesos یک ابزار اختصاصی برای کانتینرها نیست.

Mesos یک ابزار اختصاصی برای کانتینرها نیست. در عوض، می توانید از آن برای کلاستر بندی VM یا ماشین فیزیکی برای اجرای بارهای کاری (داده های بزرگ و غیره) به غیر از کانتینرها استفاده کنید.

شما همچنین می توانید یک خوشه Kubernetes را بر روی یک خوشه Mesos اجرا کنید.



منابع

[/https://devopscube.com/docker-container-clustering-tools](https://devopscube.com/docker-container-clustering-tools)

<https://www.imaginarycloud.com/blog/openshift-vs-kubernetes-differences/#:~:text=The%20biggest%20difference%20between%20them,of%20tools%20and%20dedicated%20support>

[/https://www.imaginarycloud.com/blog/nomad-vs-kubernetes](https://www.imaginarycloud.com/blog/nomad-vs-kubernetes)