به نام خدا

Doc Docker

به زبان فارسی و خلاصه

برگرفته از کتاب :
Docker Up & Running

فصل ۱ : مقدمه

معرفی داکر: تحولی در دنیای نرم‌افزار

داکر در سال 2013 توسط سولومون هایکس، مؤسس شرکت dotCloud، معرفی شد. این ابزار در ابتدا در یک کنفرانس کوچک توسعه‌دهندگان پایتون رونمایی شد، اما به سرعت به ابزاری انقلابی تبدیل گردید که شیوه‌ی ساخت، تحویل و اجرای نرم‌افزار را دگرگون کرد.

داکر یک پلتفرم متن‌باز است که به توسعه‌دهندگان و تیم‌های DevOps اجازه می‌دهد تا برنامه‌ها را همراه با تمام وابستگی‌هایشان در یک بسته‌ی قابل حمل به نام کانتینر قرار دهند. این کانتینرها تضمین می‌کنند که نرم‌افزار بدون توجه به محیط میزبان، یکسان اجرا شود.

وعده‌های داکر: چرا داکر مهم است؟

حل مشکلات محیطی:

یکی از مشکلات رایج توسعه‌دهندگان، تفاوت در محیط‌های کاری مختلف است (معروف به مشکل "روی سیستم من کار می‌کند"). داکر با فراهم کردن یک محیط یکسان، این چالش را برطرف می‌کند.

تسریع فرآیند CI/CD:

داکر فرآیندهای توسعه‌ی مداوم (Continuous Development) و تحویل مداوم (Continuous Delivery) را بهبود می‌بخشد. هر مرحله از ایجاد تصویر (Image) تا آزمایش و استقرار کاملاً خودکار می‌شود.

کاهش مصرف منابع:

کانتینرها برخلاف ماشین‌های مجازی (VM)، بدون نیاز به اجرای یک سیستم‌عامل کامل، از هسته‌ی سیستم‌عامل میزبان استفاده می‌کنند. این موضوع باعث می‌شود کانتینرها سریع‌تر و سبک‌تر باشند.

مزایای داکر: چرا باید از آن استفاده کنیم؟

قابلیت حمل (Portability): یک کانتینر می‌تواند بدون تغییر در هر محیطی اجرا شود؛ از لپ‌تاپ توسعه‌دهنده گرفته تا سرورهای ابری.

مدیریت وابستگی‌ها: تمامی فایل‌ها و وابستگی‌های برنامه داخل کانتینر قرار می‌گیرند و از تداخل با سیستم میزبان جلوگیری می‌شود.
افزایش سرعت توسعه: فرآیندهای زمان‌بر مانند تنظیم محیط‌های توسعه و تست به طور چشم‌گیری ساده‌تر می‌شوند.
بهبود امنیت: کانتینرها به طور پیش‌فرض از محیط میزبان ایزوله هستند که این موضوع امنیت نرم‌افزار را افزایش می‌دهد.

تفاوت کانتینرها با ماشین‌های مجازی:

در حالی که ماشین‌های مجازی (VM) یک سیستم‌عامل کامل را اجرا می‌کنند، کانتینرها تنها شامل برنامه و وابستگی‌های آن هستند و از سیستم‌عامل میزبان استفاده می‌کنند.
مزایا:
راه‌اندازی سریع‌تر
مصرف کمتر منابع
امکان اجرای تعداد بیشتری کانتینر در مقایسه با VM روی یک سرور

داکر چه چیزی نیست؟

داکر، برخلاف تصور برخی، جایگزینی برای ابزارهای زیر نیست اما می‌تواند با آن‌ها ترکیب شود:
ماشین‌های مجازی: داکر به جای شبیه‌سازی سخت‌افزار، از قابلیت‌های کانتینرهای لینوکس بهره می‌برد.
ابزارهای مدیریت پیکربندی: مانند Puppet یا Chef. داکر می‌تواند با این ابزارها برای مدیریت بهتر ترکیب شود.
پلتفرم‌های ابری: داکر یک سیستم مدیریت کانتینر است و وظیفه‌ی ایجاد سرورهای جدید یا ذخیره‌سازی ابری را برعهده ندارد.

اصطلاحات کلیدی در داکر

تصویر (Image): قالبی که شامل برنامه و وابستگی‌های آن است. این قالب برای ساخت کانتینر استفاده می‌شود.
کانتینر: نمونه‌ای از تصویر که به عنوان یک فرآیند در حال اجرا در سیستم میزبان عمل می‌کند.

کلاینت و سرور داکر: کلاینت (docker) دستورات را اجرا می‌کند و سرور (dockerd) وظیفه‌ی مدیریت کانتینرها و تصاویر را بر عهده دارد.

فصل ۲: چشم‌انداز داکر

ساده‌سازی فرآیندها

یکی از بزرگ‌ترین مزایای داکر، ساده‌سازی و تسریع فرآیندهای توسعه، تست و دیپلوی نرم‌افزار است. با استفاده از داکر، وابستگی‌ها و نیازمندی‌ها در همان ابتدای فرآیند شناسایی و حل می‌شوند، و فرآیند پیچیده ارتباط بین تیم‌های توسعه و عملیات ساده‌تر می‌شود.

فرآیند سنتی دیپلوی نرم‌افزار:

درخواست منابع: تیم توسعه برای اجرای برنامه، منابع مورد نیاز را از تیم عملیات درخواست می‌کند.

تخصیص منابع: تیم عملیات منابع لازم را تخصیص می‌دهد و ابزارهای مورد نیاز را نصب می‌کند.

آماده‌سازی اسکریپت‌ها: تیم توسعه اسکریپت‌های لازم برای نصب و پیکربندی را آماده می‌کند.

اصلاحات متوالی: توسعه‌دهندگان و تیم عملیات چندین مرتبه تنظیمات را برای هماهنگی بهینه‌سازی می‌کنند.

شناخت وابستگی‌های جدید: تیم توسعه وابستگی‌های بیشتری را شناسایی کرده و درخواست رفع می‌کند.

تکرار اصلاحات: تیم عملیات تغییرات جدید را اعمال می‌کند.

دیپلویمنت نهایی: نرم‌افزار پس از چندین مرحله بازنگری، به محیط عملیاتی منتقل می‌شود.

این فرآیند ممکن است هفته‌ها طول بکشد و علاوه بر زمان‌بر بودن، مستعد خطا نیز هست.

فرآیند دیپلویمنت با داکر:

تیم توسعه ایمیج داکر را ایجاد می‌کند که شامل تمام وابستگی‌ها و تنظیمات مورد نیاز است.

ایمیج در یک ریجستری (مانند Docker Hub) آپلود می‌شود.

تیم عملیات کانتینر را با استفاده از ایمیج اجرا می‌کند.

این روش:

زمان تعامل بین تیم‌ها را کاهش می‌دهد.

ریسک خطاهای محیطی را به حداقل می‌رساند.

فرآیند را ساده‌تر و خودکار می‌کند.

پشتیبانی و پذیرش گسترده

داکر از ابتدای عرضه تا امروز به سرعت تبدیل به استاندارد صنعتی در زمینه کانتینرها شده است. از این رو، اکثر پلتفرم‌ها و ابزارهای مدرن با آن سازگارند یا از آن پشتیبانی می‌کنند.

پشتیبانی در ابرهای عمومی:

AWS: شامل خدماتی مانند EKS، ECS و Fargate.

Google Cloud: با استفاده از Kubernetes Engine.

Azure: پشتیبانی از داکر در Azure Kubernetes Service.

پلتفرم‌های دیگر: IBM Cloud، Red Hat OpenShift و ...

ابزارهای متن‌باز مرتبط:

containerd: موتور اجرای کانتینر که به CNCF اهدا شد.

runc: ابزار استاندارد برای مدیریت کانتینرها.

gVisor و **Kata Containers**: برای افزایش امنیت و بهبود جداسازی.

تطبیق با استانداردها:

فرمت ایمیج داکر پایه استاندارد **OCI (Open Container Initiative)** است. این استاندارد امکان استفاده از ایمیج‌های داکر را در تمام ابزارهای سازگار با OCI فراهم می‌کند.

معماری

داکر بر اساس یک معماری ساده **کلاینت/سرور** کار می‌کند:

کلاینت داکر: دستورات را از کاربر دریافت کرده و به سرور ارسال می‌کند.

سرور داکر: مسئول اجرای دستورات، ایجاد ایمیج‌ها و مدیریت کانتینرها است.

ریجستری: مکانی برای ذخیره‌سازی و به اشتراک‌گذاری ایمیج‌های داکر.

نحوه ارتباط کلاینت و سرور:

سوکت یونیکس: /var/run/docker.sock (پیش‌فرض).

پورت‌های شبکه:

2375: برای ارتباط غیرامن.

2376: ارتباط امن با SSL.

2377: مختص Swarm Mode.

ابزارهای اصلی:

Docker CLI: ابزار خط فرمان داکر برای مدیریت کانتینرها.

Docker Engine API: واسط برنامه‌نویسی برای تعامل با سرور داکر.

ابزارهای اضافی: ابزارهای توسعه‌یافته مانند Docker Compose و Kubernetes.

فصل ۳: نصب داکر

پیش‌نیازها

سیستم‌عامل لینوکس: سیستم‌عاملی که از systemd پشتیبانی کند (مانند اوبونتو یا فدورا).

ویندوز و macOS: نیازمند Docker Desktop.

منابع سخت‌افزاری:

فضای ذخیره‌سازی کافی برای ایمیج‌ها.

دسترسی به اینترنت برای دانلود ایمیج‌ها و به‌روزرسانی‌ها.

نصب داکر

روی لینوکس:
بروزرسانی مخازن:

```
sudo apt update
```

نصب داکر:

```
sudo apt install docker.io
```

فعال‌سازی سرویس داکر:

```
sudo systemctl enable --now docker
```

اضافه کردن کاربر به گروه داکر:

```
sudo usermod -aG docker $USER
```

تایید نصب:

```
docker --version
docker run hello-world
```

روی ویندوز و macOS:

دانلود Docker Desktop از سایت رسمی.

نصب و اجرا: با تنظیمات پیش‌فرض.

تست نصب:

docker run hello-world

فصل ۴ : کار کردن با داکر ایمیج ها

ساختار آناتومی داکر فایل ها

ساختار داکر فایل: برای ساخت یک داکر ایمیج با ابزار های پیش فرض نیاز هستش که با ساختار داکرفایل ها به خوبی آشنا باشید که در زیر نمونه ای آورده شده است:

دستورات مهم در Dockerfile شامل:

دستور FROM برای انتخاب بیس ایمیج، مانند:

FROM python:3.9-slim

دستور RUN برای اجرای دستورات نصب یا تنظیمات، مانند:

RUN apt-get update && apt-get install -y nginx

دستور CMD برای تعریف فرمان پیش‌فرض اجرا در کانتینر:

CMD ["nginx", "-g", "daemon off;"]

دستور COPY برای کپی فایل‌ها از میزبان به ایمیج:

COPY myapp/ /app/

دستور EXPOSE برای تعریف پورت‌های مورد استفاده کانتینر:

EXPOSE 8080

ساخت و اجرای ایمیج‌ها

برای ساخت ایمیج از دستور زیر استفاده کنید:

docker build -t myapp:1.0 .

برای اجرای یک ایمیج به عنوان کانتینر:

docker run -d -p 80:8080 myapp:1.0

مدیریت آرگومان‌ها و متغیرهای محیطی

تعریف آرگومان در Dockerfile:

ARG APP_VERSION=1.0

استفاده از آرگومان در زمان ساخت:

docker build --build-arg APP_VERSION=2.0 .

تعریف متغیر محیطی در Dockerfile:

ENV NODE_ENV=production

مدیریت ایمیج‌ها

مشاهده لیست ایمیج‌ها:

docker images

حذف ایمیج:

docker rmi myapp:1.0

راه‌اندازی رجیستری خصوصی:

```
docker run -d -p 5000:5000 --name registry registry:2
```

بهینه‌سازی ایمیج‌ها

استفاده از بیس ایمیج سبک:

```
FROM alpine:3.16
```

پاکسازی فایل‌های موقت:

```
*/RUN apt-get clean && rm -rf /var/lib/apt/lists
```

ادغام دستورات در یک RUN:

```
RUN apt-get update && apt-get install -y nginx && apt-get clean
```

ذخیره و انتشار ایمیج‌ها

ارسال ایمیج به Docker Hub:

```
docker push username/myapp:1.0
```

ارسال ایمیج به رجیستری خصوصی:

```
docker tag myapp:1.0 localhost:5000/myapp:1.0
docker push localhost:5000/myapp:1.0
```

عیب‌یابی ساخت ایمیج‌ها

فعال کردن حالت لاگ پیشرفته در زمان ساخت:

```
. docker build --progress=plain
```

فصل ۵: کار با کانتینرها

این فصل به نحوه ایجاد، مدیریت، و بهینه‌سازی کانتینرهای داکر می‌پردازد. کانتینرها واحدهای اجرایی اصلی در داکر هستند که از ایمیج‌ها ساخته می‌شوند.

مفهوم کانتینر

کانتینرها نمونه‌های قابل اجرا از ایمیج‌های داکر هستند. آن‌ها محیطی ایزوله ایجاد می‌کنند که تمام منابع مورد نیاز برنامه، شامل سیستم‌عامل و کتابخانه‌ها، در آن تعبیه شده است.

ایجاد و مدیریت کانتینرها

ایجاد یک کانتینر از ایمیج:

دستور docker run برای اجرای یک کانتینر از یک ایمیج استفاده می‌شود:

```
docker run -d -p 80:8080 nginx
```

این دستور:

-d: کانتینر را در پس‌زمینه اجرا می‌کند.

-p: پورت ۸۰ میزبان را به پورت ۸۰۸۰ کانتینر متصل می‌کند.

شروع، توقف، و حذف کانتینرها:

شروع کانتینر متوقف‌شده:

```
docker start container_id
```

توقف کانتینر:

```
docker stop container_id
```

حذف کانتینر متوقف‌شده:

```
docker rm container_id
```

مشاهده کانتینرهای در حال اجرا و متوقف:

برای مشاهده کانتینرهای در حال اجرا:

```
docker ps
```

برای مشاهده تمام کانتینرها (شامل متوقف‌شده‌ها):

```
docker ps -a
```

تنظیمات کانتینرها

ولوم‌ها (Volumes):

ولوم‌ها برای مدیریت داده‌های دائمی کانتینرها استفاده می‌شوند.

ایجاد ولوم:

```
docker volume create my_volume
```

اتصال ولوم به کانتینر:

```
docker run -v my_volume:/data nginx
```

محدودیت منابع:

برای کنترل مصرف منابع سخت‌افزاری کانتینرها:

محدودیت CPU:

```
docker run --cpus="1.5" nginx
```

محدودیت حافظه:

```
docker run --memory="512m" nginx
```

اتوماسیون راه‌اندازی مجدد کانتینرها:

تنظیم راه‌اندازی مجدد خودکار با گزینه --restart:

```
docker run --restart always nginx
```

پاکسازی و مدیریت سیستم

پاکسازی کانتینرهای متوقف‌شده:

برای حذف همه کانتینرهای متوقف‌شده:

```
docker container prune
```

پاکسازی ایمیج‌ها و ولوم‌ها:

حذف ایمیج‌های استفاده‌نشده:

```
docker image prune
```

حذف ولوم‌های استفاده‌نشده:

```
docker volume prune
```

مدیریت پیشرفته کانتینرها

مشاهده لاگ‌های کانتینر:

با دستور docker logs می‌توانید لاگ‌های یک کانتینر را مشاهده کنید:

```
docker logs container_id
```

وارد شدن به کانتینر در حال اجرا:

با دستور docker exec وارد کانتینر شوید:

```
docker exec -it container_id bash
```

توقف سریع یک کانتینر:

دستور docker kill بلافاصله کانتینر را متوقف می‌کند:

```
docker kill container_id
```

کانتینرهای ویندوز

کانتینرها می‌توانند روی سرورهای ویندوز نیز اجرا شوند، اما به کرنل ویندوز نیاز دارند. اجرای کانتینرهای لینوکس روی ویندوز نیز با استفاده از WSL2 امکان‌پذیر است.

فصل ۶: اکتشاف داکر

این فصل به بررسی ابزارها و دستورات داکر می‌پردازد که برای نظارت، عیب‌یابی، و جمع‌آوری اطلاعات از کانتینرها و ایمیج‌ها طراحی شده‌اند. این قابلیت‌ها برای توسعه‌دهندگان و مدیران سیستم بسیار ارزشمند هستند.

بررسی نسخه و اطلاعات سرور

برای اطلاع از نسخه کلاینت و سرور داکر:

```
docker version
```

این دستور اطلاعات مربوط به نسخه کلاینت، سرور و پروتکل API را نشان می‌دهد.

دریافت اطلاعات عمومی سیستم:

```
docker info
```

این دستور اطلاعاتی مانند تعداد کانتینرها، ایمیج‌ها، ولوم‌ها و جزئیات بیشتری درباره تنظیمات سرور داکر فراهم می‌کند.

مدیریت ایمیج‌ها و کانتینرها

به‌روزرسانی ایمیج‌ها:

با کشیدن نسخه جدید یک ایمیج از رجیستری:

```
docker pull nginx:latest
```

بازبینی اطلاعات کانتینر:

با دستور docker inspect می‌توانید اطلاعات دقیق در مورد کانتینرها یا ایمیج‌ها دریافت کنید:

```
docker inspect container_id
```

این دستور اطلاعاتی درباره تنظیمات شبکه، ولوم‌ها، و وضعیت کانتینر ارائه می‌دهد.

ورود به محیط کانتینرها

اجرا کردن دستورات در کانتینر:

با استفاده از docker exec می‌توانید دستورات خاصی را در کانتینر اجرا کنید:

```
docker exec -it container_id bash
```

این دستور پوسته Bash را در کانتینر باز می‌کند.

بررسی فایل‌های کانتینر:

با استفاده از docker container cp می‌توانید فایل‌ها را از کانتینر به میزبان کپی کنید یا برعکس:

کپی فایل از کانتینر به میزبان:

```
docker container cp container_id:/path/to/file ./local/path
```

مشاهده لاگ‌ها و مانیتورینگ کانتینرها

مشاهده لاگ‌های کانتینر:

```
docker logs container_id
```

برای مشاهده لحظه‌ای لاگ‌ها:

```
docker logs -f container_id
```

مانیتورینگ مصرف منابع:

```
docker stats
```

این دستور اطلاعاتی مانند مصرف CPU، حافظه، و I/O شبکه را به صورت زنده نشان می‌دهد.

بررسی وضعیت سلامت کانتینرها:

می‌توانید تست‌های سلامت (Health Checks) را در Dockerfile تعریف کنید:

```
HEALTHCHECK --interval=30s CMD curl -f http://localhost/ || exit 1
```

با دستور زیر می‌توانید نتیجه این تست‌ها را بررسی کنید:

```
docker inspect --format='{{json .State.Health}}' container_id
```

مدیریت شبکه کانتینرها

مشاهده شبکه‌های موجود:

```
docker network ls
```

ایجاد یک شبکه سفارشی:

```
docker network create my_network
```

اتصال کانتینر به شبکه:

```
docker network connect my_network container_id
```

سیستم رخدادها (Events)

مشاهده رخدادهای سیستم داکر:

```
docker events
```

این دستور تمام رخدادهای مربوط به کانتینرها، ایمیج‌ها و شبکه‌ها را به صورت زنده نمایش می‌دهد.

ابزارهای پیشرفته برای مانیتورینگ

cAdvisor:

یک ابزار قدرتمند برای مانیتورینگ کانتینرها است که اطلاعات جامعی درباره مصرف منابع ارائه می‌دهد.

نصب و اجرا:

```
docker run -d --name=cadvisor -p 8080:8080 google/cadvisor
```

Prometheus Monitoring:

داکر قابلیت یکپارچه‌سازی با Prometheus را دارد. با استفاده از این ابزار می‌توانید داده‌های نظارتی را جمع‌آوری و تحلیل کنید.

فصل ۷: اشکال‌زدایی کانتینرها

این فصل به بررسی مشکلات رایج کانتینرها و نحوه عیب‌یابی آن‌ها می‌پردازد. با استفاده از ابزارها و دستورات مناسب، می‌توانید مشکلات مربوط به کانتینرها، شبکه، فایل‌سیستم و فرآیندهای داخل آن‌ها را شناسایی و رفع کنید.

مشاهده خروجی فرآیندها

مشاهده لاگ‌های کانتینر:

لاگ‌ها یکی از اولین مکان‌هایی هستند که برای اشکال‌زدایی باید بررسی شوند.

برای مشاهده لاگ‌ها:

```
docker logs container_id
```

برای مشاهده زنده لاگ‌ها (به‌روزرسانی لحظه‌ای):

```
docker logs -f container_id
```

برای نمایش تعداد خاصی از خطوط اخیر:

```
docker logs --tail 50 container_id
```

اجرای دستورات در کانتینر:

برای مشاهده خروجی دستورات یا اجرای تست‌های خاص:

```
docker exec -it container_id bash
```

اجرای مستقیم دستور:

```
docker exec container_id ls /app
```

بررسی فرآیندها

مشاهده فرآیندهای فعال در کانتینر:

با دستور زیر می‌توانید تمام فرآیندهای فعال داخل کانتینر را مشاهده کنید:

```
docker top container_id
```

استفاده از پوسته برای بررسی وضعیت:

وارد کانتینر شوید و ابزارهای استاندارد لینوکس مانند ps, htop یا top را اجرا کنید:

```
docker exec -it container_id bash
```

کنترل فرآیندها

متوقف کردن فرآیندها:

اگر فرآیندی در کانتینر دچار مشکل شده، می‌توانید با ابزارهایی مانند kill آن را متوقف کنید:

```
docker exec container_id kill -9 process_id
```

توقف یا از سرگیری کانتینر:

توقف کانتینر:

```
docker pause container_id
```

از سرگیری کانتینر:

```
docker unpause container_id
```

بررسی شبکه کانتینر

بررسی شبکه‌های متصل به کانتینر:

برای مشاهده شبکه‌هایی که کانتینر به آن‌ها متصل است:

```
docker network inspect network_name
```

تست ارتباط داخل کانتینر:

وارد کانتینر شوید و از ابزارهایی مانند curl، ping یا telnet استفاده کنید:

```
docker exec -it container_id ping 8.8.8.8
```

بررسی پورت‌های باز کانتینر:

با دستور docker port پورت‌های باز کانتینر و ارتباط آن‌ها با میزبان را مشاهده کنید:

```
docker port container_id
```

بررسی تاریخچه ایمیج‌ها

مشاهده لایه‌های ایمیج:

با دستور زیر می‌توانید تاریخچه ایمیج‌ها و تغییرات هر لایه را ببینید:

```
docker history image_name
```

تجزیه و تحلیل لایه‌ها:

این اطلاعات برای بررسی مشکلاتی که در فرآیند ساخت ایمیج ایجاد می‌شوند، مفید است.

بررسی فایل‌سیستم کانتینر

مشاهده فایل‌سیستم:

برای دسترسی به فایل‌سیستم کانتینر، از دستور زیر استفاده کنید:

```
docker exec -it container_id ls /path/to/directory
```

کپی فایل‌ها از کانتینر:

اگر نیاز به کپی فایل‌های خاصی دارید، از این دستور استفاده کنید:

```
docker cp container_id:/path/to/file ./local_path
```

بررسی فضای ذخیره‌سازی:

برای مشاهده جزئیات فضای ذخیره‌سازی کانتینرها:

```
docker system df
```

استفاده از ابزارهای پیشرفته برای عیب‌یابی

استفاده از nsenter:

ابزار nsenter به شما امکان می‌دهد وارد فضای نام (namespace) کانتینر شوید و به طور مستقیم مشکلات را بررسی کنید.

نصب:

```
apt-get install util-linux
```

اجرا:

```
nsenter --target container_pid --mount --uts --ipc --net --pid
```

استفاده از tcpdump:
ابزار tcpdump برای بررسی ترافیک شبکه مفید است.
نصب:

apt-get install tcpdump

اجرا در کانتینر:

docker exec container_id tcpdump -i eth0

عیب‌یابی مشکلات در زمان ساخت ایمیج‌ها

مشاهده خطاها:
هنگام ساخت ایمیج، اگر خطایی رخ دهد، پیام آن در خروجی نمایش داده می‌شود. با استفاده از گزینه زیر می‌توانید جزئیات بیشتری ببینید:

. docker build --progress=plain

تست دستورات مشکل‌دار:
دستورات Dockerfile را به صورت مستقل اجرا کنید و از صحت آن‌ها مطمئن شوید.

عیب‌یابی با BuildKit:
فعال کردن BuildKit برای بهبود اشکال‌زدایی:

. DOCKER_BUILDKIT=1 docker build

فصل ۸: آشنایی با Docker Compose

این فصل به شما می‌آموزد که چگونه با استفاده از Docker Compose، کانتینرهای چندگانه را مدیریت کرده و برنامه‌های پیچیده‌تر را به صورت کارآمد اجرا کنید. Docker Compose ابزاری است که برای تعریف و اجرای چندین کانتینر به‌صورت همزمان طراحی شده است و برای توسعه و تست سرویس‌های توزیع‌شده کاربرد دارد.

مفهوم Docker Compose

Docker Compose با استفاده از یک فایل YAML، تنظیمات لازم برای اجرای چندین کانتینر را به‌صورت هماهنگ مدیریت می‌کند. این ابزار مناسب برای برنامه‌هایی است که شامل چندین سرویس مرتبط هستند، مانند یک وب‌سرور، پایگاه داده، و کش.

ساختار فایل docker-compose.yml

فایل docker-compose.yml تعریف کاملی از سرویس‌های کانتینرها، شبکه‌ها، و ولوم‌های مورد نیاز فراهم می‌کند.

ساختار اصلی:

```
version: "3.8"
services:
  web:
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - ./html:/usr/share/nginx/html
    networks:
      - app_network
  database:
    image: postgres:13
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
    volumes:
      - db_data:/var/lib/postgresql/data
    networks:
      - app_network
volumes:
  db_data:
```

```
networks:
  app_network:
```

اجزای اصلی:

version: نسخه فایل Compose (در اینجا 3.8).

services: تعریف سرویس‌های کانتینر. هر سرویس شامل:

image: ایمیج مورد استفاده.

ports: اتصال پورت‌های میزبان و کانتینر.

volumes: مدیریت داده‌های پایدار.

networks: اتصال کانتینرها به شبکه‌های مشخص.

volumes: تعریف ولوم‌های مشترک بین کانتینرها.

networks: تعریف شبکه‌های اختصاصی.

اجرای سرویس‌ها با Docker Compose

شروع سرویس‌ها:

برای اجرای سرویس‌های تعریف‌شده در فایل docker-compose.yml:

```
docker-compose up
```

با افزودن گزینه -d سرویس‌ها در پس‌زمینه اجرا می‌شوند:

```
docker-compose up -d
```

متوقف کردن سرویس‌ها:

برای توقف تمام سرویس‌ها:

```
docker-compose down
```

مشاهده وضعیت سرویس‌ها:

با دستور زیر می‌توانید وضعیت اجرای سرویس‌ها را مشاهده کنید:

```
docker-compose ps
```

بازسازی سرویس‌ها:

برای بازسازی سرویس‌ها پس از تغییر در فایل docker-compose.yml:

```
docker-compose up --build
```

مدیریت محیط با فایل‌های متغیر (dotenv)

برای مدیریت تنظیمات و متغیرهای محیطی:

فایل .env ایجاد کنید:

```
POSTGRES_USER=myuser
POSTGRES_PASSWORD=mypassword
```

در فایل docker-compose.yml از متغیرهای تعریف‌شده استفاده کنید:

```
environment:
  POSTGRES_USER: ${POSTGRES_USER}
  POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
```

راه‌اندازی پروژه‌های پیچیده‌تر

اتصال چند فایل Compose:

می‌توانید از چندین فایل Compose برای مدیریت تنظیمات مختلف استفاده کنید:

```
docker-compose -f docker-compose.yml -f docker-compose.override.yml up
```

مدیریت چند شبکه:

شبکه‌های جداگانه برای سرویس‌های مختلف تعریف کنید تا از تداخل جلوگیری شود.

```
networks:
  frontend:
  backend:
```

سرویس‌های وابسته:

در صورتی که سرویسی نیاز به اجرای یک سرویس دیگر داشته باشد، از depends_on استفاده کنید:

```
services:
  web:
    depends_on:
      - database
```

بررسی مثال عملی

مثال Rocket.Chat:

این برنامه شامل چندین سرویس است:

سرور Rocket.Chat

پایگاه داده MongoDB

فایل نمونه Compose:

```
version: "3.8"
services:
  rocketchat:
    image: rocketchat/rocket.chat
    environment:
      MONGO_URL: mongodb://mongo:27017/rocketchat
    ports:
      - "3000:3000"
  mongo:
    image: mongo:4.4
    volumes:
      - mongo_data:/data/db
volumes:
  mongo_data:
```

با اجرای docker-compose up، تمام سرویس‌ها به صورت هماهنگ اجرا می‌شوند.

فصل ۹: مسیر به سوی کانتینرهای تولیدی (Production Containers)

در این فصل به مفاهیم و چالش‌های مرتبط با استفاده از کانتینرهای داکر در محیط‌های تولیدی پرداخته می‌شود. اجرای موفقیت‌آمیز کانتینرها در مقیاس تولیدی مستلزم توجه به جزئیات مربوط به پیکربندی، نظارت، امنیت، و بهینه‌سازی است.

انتقال به محیط تولید

برای آماده‌سازی کانتینرها جهت استفاده در محیط تولید، لازم است:

حداقل‌سازی وابستگی‌ها:

ایمیج‌ها باید کوچک و بهینه باشند. از نسخه‌های Slim یا Alpine بیس ایمیج‌ها استفاده کنید.

پایداری و قابلیت اطمینان:

کانتینرها باید به گونه‌ای طراحی شوند که در صورت بروز خطا به راحتی راه‌اندازی مجدد شوند.

مستقل بودن:

تمام وابستگی‌های نرم‌افزار باید در کانتینر موجود باشند یا از سرویس‌های خارجی (مانند پایگاه داده‌ها) استفاده کنند.

نقش داکر در محیط‌های تولیدی

داکر به تیم‌های توسعه و عملیات کمک می‌کند تا فرآیندهای پیچیده را ساده‌تر کنند:

کنترل شغل‌ها (Job Control):

مدیریت فرآیندهای داخلی کانتینرها با استفاده از ابزارهای موجود در سیستم‌عامل لینوکس یا تنظیمات Restart Policy.

محدودیت منابع:
اعمال محدودیت بر مصرف منابع کانتینرها (CPU و حافظه) برای جلوگیری از تداخل کانتینرها.
مثال:

```
docker run --cpus="1" --memory="512m" myapp
```

پیکربندی شبکه:
استفاده از شبکه‌های سفارشی برای جداسازی و کنترل ارتباطات کانتینرها.

تنظیمات:
مدیریت تنظیمات حساس و پیکربندی‌ها با استفاده از متغیرهای محیطی یا ابزارهایی مانند Docker Secrets.

مدیریت لاگ‌ها
ذخیره‌سازی لاگ‌ها:
داکر به طور پیش‌فرض لاگ‌ها را در فایل‌های محلی ذخیره می‌کند.

ارسال لاگ‌ها به سیستم‌های خارجی:
با استفاده از درایورهای لاگ مانند fluentd ،syslog یا json-file، لاگ‌ها را به سیستم‌های مانیتورینگ ارسال کنید.
مثال:

```
docker run --log-driver=syslog myapp
```

نظارت و مانیتورینگ کانتینرها
آمار کانتینرها:
استفاده از دستور docker stats برای مشاهده مصرف منابع.

استفاده از ابزارهای مانیتورینگ:
Prometheus: جمع‌آوری داده‌های مربوط به عملکرد.
Grafana: نمایش بصری داده‌های مانیتورینگ.
cAdvisor: نظارت بر عملکرد کانتینرها.

امنیت کانتینرها

اجتناب از دسترسی روت:

کانتینرها را بدون دسترسی روت اجرا کنید تا از حملات امنیتی جلوگیری شود:

```
docker run --user 1001 myapp
```

استفاده از Rootless Mode:

داکر از نسخه‌هایی با قابلیت Rootless پشتیبانی می‌کند که امکان اجرای کانتینرها بدون دسترسی به روت سیستم میزبان را فراهم می‌کند.

بروزرسانی منظم:

اطمینان حاصل کنید که ایمیج‌ها و نرم‌افزارهای کانتینر همیشه به‌روز باشند.

اکوسیستم DevOps و داکر

کانتینرها بخش مهمی از فرآیند DevOps را تشکیل می‌دهند و به ادغام توسعه، تست، و تولید کمک می‌کنند:

یکپارچگی با CI/CD:

استفاده از کانتینرها برای خودکارسازی فرآیندهای Test ،Build، و Deploy.

ابزارهایی مانند GitLab CI/CD یا Jenkins به‌خوبی با داکر سازگار هستند.

مدیریت وابستگی‌ها:

از ایمیج‌های استاندارد برای کاهش پیچیدگی و تکرارپذیری استفاده کنید.

فصل ۱۰: داکر در مقیاس بزرگ (Scaling Docker)

این فصل به موضوع مقیاس‌بندی کانتینرها می‌پردازد و نشان می‌دهد چگونه می‌توان از داکر برای مدیریت سیستم‌های پیچیده و توزیع‌شده در مقیاس بزرگ استفاده کرد. با بهره‌گیری از ابزارهای مدیریت و هماهنگ‌سازی، مانند Docker Swarm و Kubernetes، می‌توان مقیاس‌پذیری، انعطاف‌پذیری، و اطمینان در اجرای سرویس‌ها را تضمین کرد.

چرا مقیاس‌بندی مهم است؟

افزایش تقاضا:

با رشد کاربران یا حجم داده‌ها، برنامه‌ها باید به‌صورت دینامیک پاسخگوی تقاضا باشند.

بهبود دسترس‌پذیری:

توزیع سرویس‌ها در چندین نود از وقوع خرابی‌های کلی جلوگیری می‌کند.

انعطاف‌پذیری:

برنامه‌ها باید بتوانند بدون تغییرات اساسی با محیط‌های مختلف سازگار شوند.

روش‌های مقیاس‌بندی در داکر

مقیاس‌بندی افقی (Horizontal Scaling):

افزودن تعداد کانتینرهای مشابه برای توزیع بار کاری.

مثال: اجرای چندین کانتینر از یک سرویس با Load Balancing.

مقیاس‌بندی عمودی (Vertical Scaling):

افزایش منابع سخت‌افزاری یک کانتینر مانند CPU و حافظه.

مثال:

```
docker run --cpus=4 --memory=4g myapp
```

استفاده از Docker Compose برای مقیاس‌بندی

Docker Compose به شما امکان می‌دهد با کمترین تلاش تعداد کانتینرهای یک سرویس را افزایش دهید:

تعریف سرویس‌ها:
در فایل docker-compose.yml، سرویس‌ها را تعریف کنید.

مثال:

```
version: "3.8"
services:
  web:
    image: nginx
    ports:
      - "8080:80"
```

افزایش مقیاس:
افزایش مقیاس:
از دستور زیر برای افزایش تعداد کانتینرها استفاده کنید:

```
docker-compose up --scale web=5
```

استفاده از Docker Swarm

Docker Swarm ابزار داخلی داکر برای مدیریت خوشه‌ای (Cluster Management) است. این ابزار امکان مقیاس‌بندی سرویس‌ها و توزیع آن‌ها بین چندین نود را فراهم می‌کند.

مراحل راه‌اندازی Docker Swarm:
فعال‌سازی Swarm:
روی نود اصلی (Manager):

```
docker swarm init
```

افزودن نودها به Swarm:
دریافت Token با دستور:

```
docker swarm join-token worker
```

افزودن نود Worker:
اجرای دستور ارائه‌شده در خروجی روی نود Worker.

تعریف و اجرای سرویس‌ها در Swarm:

تعریف سرویس:

```
docker service create --name web --replicas=3 -p 8080:80 nginx
```

مشاهده وضعیت سرویس‌ها:

```
docker service ls
```

افزایش مقیاس:

```
docker service scale web=10
```

استفاده از Kubernetes برای مقیاس‌بندی پیشرفته

Kubernetes (K8s) یکی از قدرتمندترین ابزارهای مدیریت و هماهنگ‌سازی کانتینرها است.

ویژگی‌های اصلی Kubernetes:

Auto-Scaling:

مقیاس‌بندی خودکار سرویس‌ها بر اساس میزان مصرف منابع.

Load Balancing:

توزیع بار کاری بین پادها (Pods).

Fault Tolerance:

راه‌اندازی مجدد خودکار کانتینرهای خراب‌شده.

مراحل اساسی کار با Kubernetes:

تعریف یک Deployment:

یک فایل YAML ایجاد کنید:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

ذخیره فایل به نام deployment.yml.

ایجاد Deployment:

kubectl apply -f deployment.yml

افزایش مقیاس:

kubectl scale deployment web --replicas=10

مشاهده وضعیت:
kubectl get pods

نکات مهم در مقیاس‌بندی
نظارت بر مصرف منابع:
استفاده از ابزارهایی مانند Prometheus و Grafana برای نظارت بر عملکرد.
تنظیمات Load Balancer:
اطمینان حاصل کنید که بار کاری به‌طور مساوی بین نودها و کانتینرها توزیع می‌شود.
مدیریت Data Persistence:
برای جلوگیری از از دست رفتن داده‌ها، از ولوم‌های پایدار یا سیستم‌های ذخیره‌سازی توزیع‌شده استفاده کنید.
شبکه‌های قابل اعتماد:
شبکه‌های سفارشی ایجاد کنید تا ارتباط بین کانتینرها سریع و ایمن باشد.