Docker Volumes

Container'ların neden ephemeral (geçici) olduğu, named volume'lar, docker run -v ile bir volume mount etmek, PostgreSQL'in verisini nerede sakladığı, named volume vs bind mount, ve kalıcılığın bilinçli bir testle doğrulanması.

Orta 30 dk

Son güncelleme: 31 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/docker-volumes

---

"Docker Networking", bu projenin kendi PostgreSQL ve uygulama container'larını, isimle bağlı, yan yana çalıştırdı. O kurulumun üstünkörü geçtiği rahatsız edici bir gerçek var: `docker rm learning-platform-db` -- tamamen sıradan bir komut, daha önce "Temizlemek: `docker rm`"de işlenmiş -- o veritabanının o güne kadar sakladığı her satırı, hiçbir uyarı olmadan, kalıcı olarak silerdi. Bu ders, bunun neden olduğu ve bunu önleyen mekanizma (bir **volume**) hakkında.

Container'lar Neden Ephemeral (Geçici)

Çalışan bir container'ın kendi dosya sistemine yazdığı her şey, başlatıldığı salt okunur image'ın üzerine oturan, o container'ın kendi **yazılabilir katmanında** yaşar. O katman container'ın kendisinin bir parçasıdır -- image'ın değil, ayrı bir şeyin de değil -- bu yüzden `docker rm` (daha önce "Temizlemek: `docker rm`"de işlenen), o belirli container örneği hakkındaki her şeyle birlikte onu da siler. Hiçbir volume yapılandırılmamış bir PostgreSQL container'ı, gerçek veritabanı dosyalarını tam olarak aynı yazılabilir katmanda saklar, ki bu, verinin tüm yaşam süresinin o tek container'ın yaşam süresine bağlı olduğu anlamına gelir -- *aynı* container'ı durdurup yeniden başlat, veri hâlâ orada olur (`docker stop` / `docker start` , yazılabilir katmana dokunmaz), ama onu kaldır, aynısıyla değiştirmek için bile olsa, veri kalıcı olarak gider. "Ephemeral" burada bunu ifade eder: bir container'ın verisinin hızlıca kaybolması değil, container'ın kendisi kaybolduğu anda tamamen kaybolması.

Named Volume'lar

Bir **named volume** (adlandırılmış volume), Docker'ın kendisinin yönettiği, tamamen herhangi bir tek container'ın yazılabilir katmanının dışında bir depolamadır -- birini oluşturmak, sonunda onu kullanacak herhangi bir container'dan bağımsız, tek bir komuttur.

```
docker volume create learning-platform-db-data
```

```
learning-platform-db-data
```

Bir volume bağımsız var olduğu için, bir container'ın kendi yazılabilir katmanını yok eden tam olarak o olayı atlatır: `docker rm`. Yeni bir container -- hatta farklı bir image'dan başlatılmış, tamamen farklı bir container bile -- aynı var olan volume'a yönlendirilebilir ve öncekinin verisinin bıraktığı yerden tam olarak devam edebilir.

Bir Volume'u Mount Etmek: `-v`

Bir volume, `docker run`'da `-v <volume-adı>:<container-içindeki-yol>` kullanılarak belirli bir yola **mount** edilene kadar kendi başına hiçbir şey yapmaz:

```
docker run --name learning-platform-db \
  -e POSTGRES_PASSWORD=secret \
  -v learning-platform-db-data:/var/lib/postgresql/data \
  -d postgres:16
```

O container'ın bakış açısından, `/var/lib/postgresql/data` sıradan bir dizin gibi görünür -- oraya yazılan her şey, gerçekte, container'ın kendi tek kullanımlık yazılabilir katmanına değil, Docker'ın yönettiği volume'a yazılıyor.

PostgreSQL Verisini Nerede Saklar

`/var/lib/postgresql/data`, bu ders için seçilmiş keyfi bir yol değil -- ("Docker CLI Temelleri" ve "Docker Networking" boyunca zaten kullanılan) resmi `postgres` image'ının gerçek veritabanı dosyalarını varsayılan olarak tam olarak sakladığı yer, image'ın kendisi tarafından bu şekilde belgelenmiş. *O belirli yola* bir volume mount etmek, "kaldırıldığı anda verisi kaybolan bir PostgreSQL container'ı" ile "verisi kendisinden daha uzun yaşayan bir PostgreSQL container'ı" arasındaki farkı yaratan şeydir -- PostgreSQL'in kendisinin nasıl çalıştığına dair hiçbir şey değişmez; yalnızca dosyalarının fiziksel olarak nerede bittiği değişir.

Kalıcılığı Doğrulamak

Bir volume'un gerçekten çalıştığının tek gerçek testi, bilinçli olarak yıkıcıdır: veri yaz, container'ı tamamen kaldır, aynı volume'a karşı yeni bir tane başlat, ve verinin hâlâ orada olduğunu doğrula.

```
docker exec -it learning-platform-db psql -U postgres -c "INSERT INTO proof (note) VALUES
('still here');"

docker stop learning-platform-db
docker rm learning-platform-db

docker run --name learning-platform-db -v learning-platform-db-data:/var/lib/postgresql
/data -d postgres:16

docker exec -it learning-platform-db psql -U postgres -c "SELECT * FROM proof;"
```

Kaldırmadan önceki satır hâlâ oradaysa, verinin gerçekten tüm süre boyunca yaşadığı yer -- container değil -- volume'du. Bu dersteki ilerideki "Hepsini Bir Araya Getirmek", bu tam sırayı baştan sona işliyor.

Named Volume'lar vs. Bind Mount'lar

Bir **bind mount**, host depolamayı bir container'a bağlamanın diğer yoludur -- Docker'ın yönettiği bir volume yerine, host makinedeki belirli, gerçek bir yolu doğrudan container'a eşler (`-v <volume-adı>:/app` yerine `-v /home/dev/my-project:/app`). İkisi de aynı `-v` flag'ini kullanır, ama farklı problemleri çözerler: bir bind mount, belirli bir host yolunun önemli olduğu durumlar içindir -- örneğin, yerel geliştirme sırasında bir projenin kendi kaynak kodunu bir container'a mount etmek -- named volume ise bir container'ın yönettiği ve Docker'ın yaşam döngüsünü sahiplenmesi gereken veri içindir, tam olarak PostgreSQL'in kendi veri dosyaları gibi. Bu kurs tam olarak bu nedenle named volume'lara bağlı kalıyor; bir veritabanının veri dizini, host makinenin kendi dosya sisteminden doğrudan düzenlenmesi amaçlanan bir şey değildir.

Volume'ları Yönetmek: `docker volume ls` / `inspect` / `rm`

```
docker volume ls
```

```
DRIVER    VOLUME NAME
local     learning-platform-db-data
```

`docker volume inspect learning-platform-db-data`, Docker'ın o volume'un verisini host makinenin kendi diskinde gerçekte nerede sakladığını tam olarak gösterir -- var olduğunu bilmeye değer, doğrudan dokunmaya nadiren değer. `docker volume rm learning-platform-db-data`, bir volume'u kalıcı olarak siler, ve yalnızca hiçbir container onu şu anda mount etmemişse başarılı olur.

> `docker volume rm`, bu dersteki `docker rm`'in yapabileceği gibi GÖRÜNDÜĞÜ şeyi gerçekten yapan tek komuttur -- veriyi kalıcı olarak siler. `docker rm` kadar sık kazara çalıştırılmaz, tam olarak volume'ların sıradan container temizliğinden sağ çıkan şey olmak için var olması yüzünden -- ama bunun, gerçekten, bilinçli olarak kalıcı veriyi yok eden tek Docker komutu olduğunu bilmeye değer.

Hepsini Bir Araya Getirmek

Tam sıra: bir volume oluştur, PostgreSQL'i ona karşı çalıştır, gerçek veri yaz, container'ı tamamen kaldır, ve yepyeni bir container'ın hâlâ o veriyi gördüğünü doğrula:

```
#!/bin/sh
# Proving that data written to a named volume survives even after the
# container that wrote it is completely removed -- not just stopped.

docker volume create learning-platform-db-data
# Output:
# learning-platform-db-data

docker run --name learning-platform-db \
  -e POSTGRES_PASSWORD=secret \
  -v learning-platform-db-data:/var/lib/postgresql/data \
  -d postgres:16
# Output:
# 7f8e9a0b1c2d3e4f5a6b7c8d9e0f1a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e8f

# Write some real data.
docker exec -it learning-platform-db psql -U postgres -c \
  "CREATE TABLE proof (id SERIAL PRIMARY KEY, note TEXT); INSERT INTO proof (note) VALUES
('still here after removal');"
# Output:
# CREATE TABLE
# INSERT 0 1

# Destroy the container entirely -- not just stop it.
docker stop learning-platform-db
docker rm learning-platform-db
# Output:
# learning-platform-db
# learning-platform-db

# The container is gone. The volume is not -- confirm it's still listed.
docker volume ls
# Output:
# DRIVER    VOLUME NAME
# local     learning-platform-db-data

# Start a brand-new container, pointed at the SAME volume.
docker run --name learning-platform-db \
  -e POSTGRES_PASSWORD=secret \
  -v learning-platform-db-data:/var/lib/postgresql/data \
  -d postgres:16
# Output:
# 3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e8f9a0b1c2d3e4f5a6b7c8d9e0f1a2b3c4

docker exec -it learning-platform-db psql -U postgres -c "SELECT * FROM proof;"
# Output:
#  id |            note
# ----+----------------------------
#   1 | still here after removal
# (1 row)

# The row survived a full container removal -- because it was never stored
# inside the container's own filesystem to begin with.
```

Bu, "Docker Networking"deki eksik parça -- o dersteki aynı `learning-platform-db` container'ı, şimdi bağlı bir named volume'la, bu projenin gerçek verisini kaybetmeden (kazara ya da

bilinçli) bir `docker rm` çalıştırmayı güvenli kılan şeydir. Bu kategorideki bir sonraki ders "Docker Compose", böyle bir volume'un elle yazılmaktan çıkıp bildirilmiş, tekrarlanabilir bir kurulumun parçası hâline geldiği yer.

Yaygın Hatalar

- PostgreSQL'i (ya da herhangi bir stateful servisi) hiçbir volume mount edilmemiş bir container'da çalıştırmak, ve sıradan bir `docker rm`'den sonra verinin gittiğine şaşırmak -- bkz. "Container'lar Neden Ephemeral (Geçici)".
- Bir volume'u yanlış yola mount etmek -- bir volume yalnızca mount edildiği tam yola yazılanı yakalar; resmi `postgres` image'ı için özellikle `/var/lib/postgresql/data` (bkz. "PostgreSQL Verisini Nerede Saklar").
- Named volume doğru araç olduğunda bir bind mount'a başvurmak, ya da tam tersi -- bir bind mount, bir container'ı belirli bir host yoluna bağlar; named volume, tam olarak bu tür veri için tasarlanmış, Docker'ın yönettiği depolamadır.
- Hâlâ kullanımda olan verinin gerçekten o volume'da tutulup tutulmadığını kontrol etmeden `docker volume rm` çalıştırmak -- bir container'daki `docker rm`'in aksine, bunun "yalnızca çalışan örnek" güvenlik ağı eşdeğeri yoktur.

Best Practices

- Gerçek bir veritabanı çalıştıran her container için -- PostgreSQL dahil -- her zaman bir named volume mount et -- kalıcı olması amaçlanan veri için asla bir container'ın kendi yazılabilir katmanına güvenme.
- "Kalıcılığı Doğrulamak" ve "Hepsini Bir Araya Getirmek"in yaptığı gibi kalıcılığı bilinçli olarak doğrula -- veri yaz, container'ı kaldır, aynı volume'a karşı yeniden oluştur, verinin hayatta kaldığını doğrula, bir volume'un doğru yapılandırıldığını varsaymak yerine.
- Yerel geliştirme sırasında host kaynak kodu için özellikle bir bind mount'a, ve bir container'ın kendisinin yönettiği veri için özellikle bir named volume'a başvur -- ikisinden birine de ayrımı aklında tutmadan alışkanlıkla başvurma.
- `docker volume rm`'i, gerçekten yıkıcı herhangi bir komutla aynı özenle ele al -- onu çalıştırmadan önce hiçbir şeyin hâlâ o volume'un verisine ihtiyaç duymadığını doğrula.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- Bir container'ın kendi dosya sistemi yazmaları, `docker rm` çalıştığı anda geri kalan her şeyle birlikte silinen, tek kullanımlık yazılabilir katmanında yaşar -- bir container'ı "ephemeral" (geçici) yapan şey budur.

- Bir named volume (`docker volume create`), herhangi bir tek container'dan bağımsız var olan, ve bir container'ın yazılabilir katmanını yok eden tam olarak o olaydan sağ çıkan, Docker'ın yönettiği depolamadır.
- `docker run`'da `-v <volume-adı>:<yol>`, bir container içindeki belirli bir yola bir volume mount eder -- resmi `postgres` image'ı için o yol `/var/lib/postgresql/data`'dır.
- Bir bind mount, belirli bir host yolunu doğrudan bir container'a eşler (geliştirme sırasında kaynak kod için kullanışlı); named volume, bir container'ın yaşam döngüsünün belirlememesi gereken veri için tasarlanmış, Docker'ın yönettiği depolamadır.
- `docker volume rm`, kalıcı veriyi gerçekten, kalıcı olarak silen tek komuttur -- doğru yapılandırılmış bir volume'un özellikle sağ çıkması amaçlanan, container'ın kendisini kaldırmaktan farklı olarak.

Cheat Sheet

```
docker volume create <isim>                     # named bir volume oluştur
docker run -v <isim>:<container-içindeki-yol> ...  # onu bir container'a mount et
docker volume ls                                 # tüm volume'ları listele
docker volume inspect <isim>                     # diskte gerçekte nerede yaşadığını göster
docker volume rm <isim>                          # bir volume'un verisini kalıcı olarak sil
```

```
postgres veri yolu: /var/lib/postgresql/data
```

Terimler Sözlüğü

- **Yazılabilir katman (writable layer)**: salt okunur bir image'ın üzerine oturan, container'a özel, tek kullanımlık dosya sistemi katmanı; `docker rm` ile container'la birlikte silinir.
- **Named volume**: herhangi bir container'dan bağımsız oluşturulan, `docker rm`'den sağ çıkan, ve öncekinin bıraktığı yerden devam etmek için yeni bir container'a mount edilebilen, Docker'ın yönettiği depolama.
- **Bind mount**: belirli, gerçek bir host dosya sistemi yolunu doğrudan bir container'a eşlemek -- named volume'un aksine, host yolunun kendisinin önemli olduğu durumlar için.
- **Mount**: bir volume'u (ya da bind mount'u) `-v` üzerinden bir container içindeki belirli bir yola bağlamak, o yoldaki okuma ve yazmaların container'ın kendi yazılabilir katmanı yerine volume'a gitmesini sağlamak.