

Docker

Volumes and Persistent DB

CONTENTS

1. Docker Volume
2. Persist the To-Do App DB
3. Bind Mounts

Persist a DB

In case you didn't notice, our todo list is being wiped clean every single time we launch the container. Why is this? Let's dive into how the container is working.

The container's filesystem

When a container runs, it uses the various layers from an image for its filesystem. Each container also gets its own "scratch space" to create/update/remove files. Any changes won't be seen in another container, *even if* they are using the same image.

See this in practice

To see this in action, we're going to start two containers and create a file in each. What you'll see is that the files created in one container aren't available in another.

Files created in one container aren't available in another.

1. Start an ubuntu container that will create a file named `/data.txt` with a random number between 1 and 10000.

```
$ docker run -d ubuntu bash -c "shuf -i 1-10000 -n 1 -o /data.txt && tail -f /dev/null"
```

In case you're curious about the command, we're starting a bash shell and invoking two commands (why we have the `&&`). The first portion picks a single random number and writes it to `/data.txt`. The second command is simply watching a file to keep the container running.

2. Validate that you can see the output by accessing the terminal in the container. To do so, go to **Containers** in Docker Desktop, hover over the container running the **ubuntu** image, and select the **Show container actions** menu. From the dropdown menu, select **Open in terminal**.

You will see a terminal that is running a shell in the Ubuntu container. Run the following command to see the content of the `/data.txt` file. Close this terminal afterwards again.

```
$ cat /data.txt
```

Files created in one container aren't available in another.

1. Now, let's start another ubuntu container (the same image) and we'll see we don't have the same file.

```
$ docker run -it ubuntu ls /
```

And look! There's no data.txt file there! That's because it was written to the scratch space for only the first container.

Container Volumes

Volumes provide the ability to connect specific filesystem paths of the container back to the host machine. If a directory in the container is mounted, changes in that directory are also seen on the host machine. If we mount that same directory across container restarts, we'd see the same files.

Persist the todo app data

By default, the todo app stores its data in a SQLite database at `/etc/todos/todo.db` in the container's filesystem. If you're not familiar with SQLite, no worries! It's simply a relational database in which all of the data is stored in a single file. While this isn't the best for large-scale applications, it works for small demos. We'll talk about switching this to a different database engine later.

With the database being a single file, if we can persist that file on the host and make it available to the next container, it should be able to pick up where the last one left off. By creating a volume and attaching (often called "mounting") it to the directory the data is stored in, we can persist the data. As our container writes to the `todo.db` file, it will be persisted to the host in the volume.

As mentioned, we are going to use a volume mount. Think of a volume mount as an opaque bucket of data. Docker fully manages the volume, including where it is stored on disk. You only need to remember the name of the volume.

Persist the todo app data

1. Create a volume by using the `docker volume create` command.

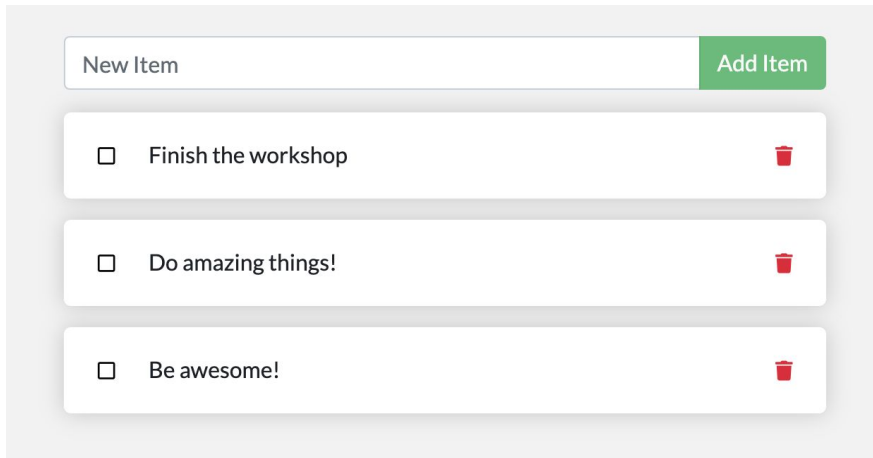
```
$ docker volume create todo-db
```

2. Stop and remove the todo app container once again in the Dashboard (or with `docker rm -f <id>`), as it is still running without using the persistent volume.
3. Start the todo app container, but add the `--mount` option to specify a volume mount. We will give the volume a name, and mount it to `/etc/todos` in the container, which will capture all files created at the path.




```
$ docker run -dp 3000:3000 --mount type=volume,src=todo-db,target=/etc/todos getting-started
```


Persist the todo app data

4. Once the container starts up, open the app and add a few items to your todo list.



The screenshot displays a user interface for a todo application. At the top, there is a text input field with the placeholder text "New Item" and a green button labeled "Add Item". Below this, there is a list of three todo items, each contained within a white rectangular card with rounded corners. Each card features a small square checkbox on the left, the text of the todo item in the center, and a red trash can icon on the right. The items are: "Finish the workshop", "Do amazing things!", and "Be awesome!".

Checkbox	Item	Action
<input type="checkbox"/>	Finish the workshop	
<input type="checkbox"/>	Do amazing things!	
<input type="checkbox"/>	Be awesome!	

Persist the todo app data

5. Stop and remove the container for the todo app. Use the Dashboard or `docker ps` to get the ID and then `docker rm -f <id>` to remove it.

6. Start a new container using the same command from above.

Open the app. You should see your items still in your list!

7. Go ahead and remove the container when you're done checking out your list.

Hooray! You've now learned how to persist data!

Dive into Docker Volumes

If you want to know, you can use the `docker volume inspect` command.

```
$ docker volume inspect todo-db
```

```
[
  {
    "CreatedAt": "2019-09-26T02:18:36Z",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/todo-db/_data",
    "Name": "todo-db",
    "Options": {},
    "Scope": "local"
  }
]
```

The Mountpoint is the actual location on the disk where the data is stored. Note that on most machines, you will need to have root access to access this directory from the host. But, that's where it is!

Bind Mounts

A bind mount is another type of mount, which lets you share a directory from the host's filesystem into the container. When working on an application, you can use a bind mount to mount source code into the container. The container sees the changes you make to the code immediately, as soon as you save a file. This means that you can run processes in the container that watch for filesystem changes and respond to them.

Things to do Next

1. Configuration Management using Ansible.
2. Cluster management or container management using Kebernetes
3. Code Checkout using git
4. Dont stop
 - a. Linux Scripting/Python scripting
 - b. AWS services and commands using free tier
 - c. Terraform module writing practice
 - d. Docker and Containerization
 - e. Debugging and Verbose Mode