Manage state and data in Docker applications

Article • 04/13/2022

> **Tip**
>
> This content is an excerpt from the eBook, .NET Microservices Architecture for Containerized .NET Applications, available on **.NET Docs** or as a free downloadable PDF that can be read offline.
>
> Download PDF
>
> 
>

In most cases, you can think of a container as an instance of a process. A process doesn't maintain persistent state. While a container can write to its local storage, assuming that an instance will be around indefinitely would be like assuming that a single location in memory will be durable. You should assume that container images, like processes, have multiple instances or will eventually be killed. If they're managed with a container orchestrator, you should assume that they might get moved from one node or VM to another.

The following solutions are used to manage data in Docker applications:

From the Docker host, as Docker Volumes :

- **Volumes** are stored in an area of the host filesystem that's managed by Docker.
- **Bind mounts** can map to any folder in the host filesystem, so access can't be controlled from Docker process and can pose a security risk as a container could access sensitive OS folders.
- **tmpfs mounts** are like virtual folders that only exist in the host's memory and are never written to the filesystem.

From remote storage:

- Azure Storage , which provides geo-distributable storage, providing a good long-term persistence solution for containers.
- Remote relational databases like Azure SQL Database or NoSQL databases like Azure Cosmos DB, or cache services like Redis .

From the Docker container:

- **Overlay File System**. This Docker feature implements a copy-on-write task that stores updated information to the root file system of the container. That information is "on top" of the original image on which the container is based. If the container is deleted from the system, those changes are lost. Therefore, while it's possible to save the state of a container within its local storage, designing a system around this would conflict with the premise of container design, which by default is stateless.

However, using Docker Volumes is now the preferred way to handle local data in Docker. If you need more information about storage in containers check on Docker storage drivers and About storage drivers .

The following provides more detail about these options:

Volumes are directories mapped from the host OS to directories in containers. When code in the container has access to the directory, that access is actually to a directory on the host OS. This directory is not tied to the lifetime of the container itself, and the directory is managed by Docker and isolated from the core functionality of the host machine. Thus, data volumes are designed to persist data independently of the life of the container. If you delete a container or an image from the Docker host, the data persisted in the data volume isn't deleted.

Volumes can be named or anonymous (the default). Named volumes are the evolution of **Data Volume Containers** and make it easy to share data between containers. Volumes also support volume drivers that allow you to store data on remote hosts, among other options.

Bind mounts are available since a long time ago and allow the mapping of any folder to a mount point in a container. Bind mounts have more limitations than volumes and some important security issues, so volumes are the recommended option.

tmpfs mounts are basically virtual folders that live only in the host's memory and are never written to the filesystem. They are fast and secure but use memory and are only meant for temporary, non-persistent data.

As shown in Figure 4-5, regular Docker volumes can be stored outside of the containers themselves but within the physical boundaries of the host server or VM. However, Docker containers can't access a volume from one host server or VM to another. In other words, with these volumes, it isn't possible to manage data shared between containers that run on different Docker hosts, although it could be achieved with a volume driver that supports remote hosts.

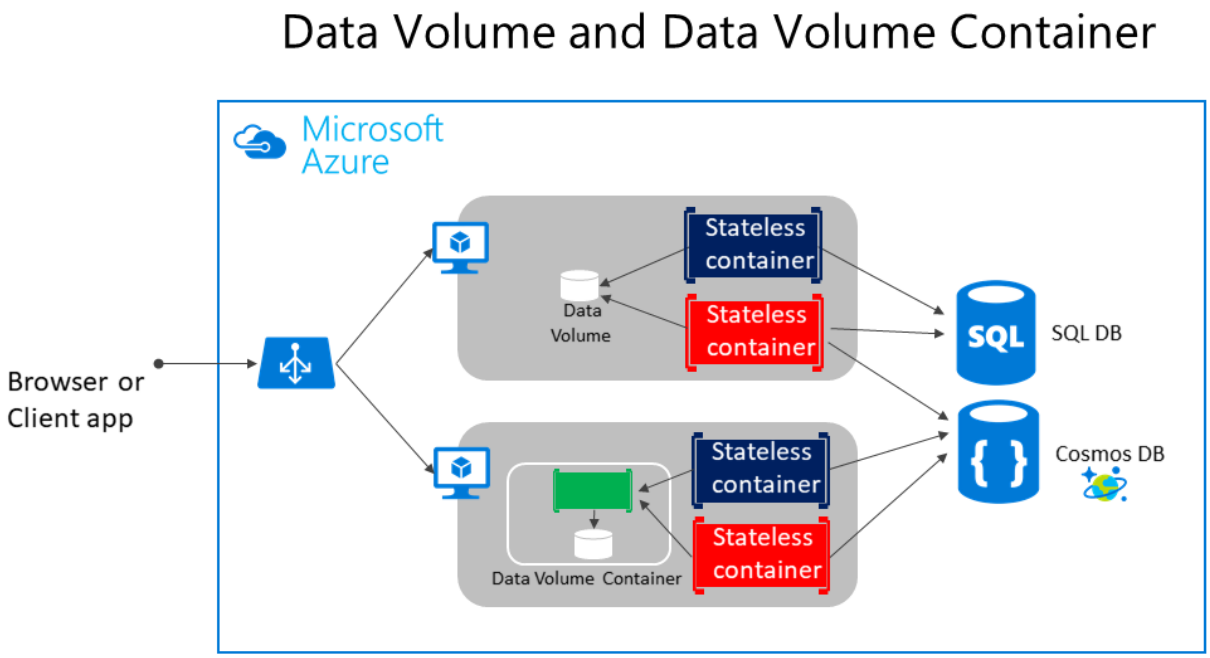

Figure 4-5. Volumes and external data sources for container-based applications

Volumes can be shared between containers, but only in the same host, unless you use a remote driver that supports remote hosts. In addition, when Docker containers are managed by an orchestrator, containers might "move" between hosts, depending on the optimizations performed by the cluster. Therefore, it isn't recommended that you use data volumes for business data. But they're a good mechanism to work with trace files, temporal files, or similar that will not impact business data consistency.

Remote data sources and cache tools like Azure SQL Database, Azure Cosmos DB, or a remote cache like Redis can be used in containerized applications the same way they are used when developing without containers. This is a proven way to store business application data.

Azure Storage. Business data usually will need to be placed in external resources or databases, like Azure Storage. Azure Storage, in concrete, provides the following services in the cloud:

- Blob storage stores unstructured object data. A blob can be any type of text or binary data, such as document or media files (images, audio, and video files). Blob storage is also referred to as Object storage.

- File storage offers shared storage for legacy applications using standard SMB protocol. Azure virtual machines and cloud services can share file data across application components via mounted shares. On-premises applications can access file data in a share via the File service REST API.
- Table storage stores structured datasets. Table storage is a NoSQL key-attribute data store, which allows rapid development and fast access to large quantities of data.

Relational databases and NoSQL databases. There are many choices for external databases, from relational databases like SQL Server, PostgreSQL, Oracle, or NoSQL databases like Azure Cosmos DB, MongoDB, etc. These databases are not going to be explained as part of this guide since they are in a completely different subject.

Previous Next