In this scenario, you'll learn how to use Docker Volumes to persistent data within Containers. Docker Volumes allow directories to be shared between containers and container versions.

Docker Volumes allows you to upgrade containers, restart machines and share data without data loss. This is essential when updating database or application versions.

Step 1 - Data Volumes
Docker Volumes are created and assigned when containers are started. Data Volumes allow you to map a host directory to a container for sharing data.

This mapping is bi-directional. It allows data stored on the host to be accessed from within the container. It also means data saved by the process inside the container is persisted on the host.

Task
This example will use Redis as a way to persist data. Start a Redis container below, and create a data volume using the -v parameter. This specifies that any data saved inside the container to the /data directory should be persisted on the host in the directory /docker/redis-data.

```
docker run  -v /docker/redis-data:/data \
  --name r1 -d redis \
  redis-server --appendonly yes
```

```
[root@host01 ~]# docker run  -v /docker/redis-data:/data \
>   --name r1 -d redis \
>   redis-server --appendonly yes
5dcd141594998b2eafee2e819da914f4dbf46d79b2d502be9a2356c9c644d2ce
```

The docker run command you provided starts a Redis container with the following configuration:

-v /docker/redis-data:/data: mounts the local directory /docker/redis-data as a volume in the container at the path /data. This allows data to persist between container restarts and ensures that the data is stored outside of the container, making it easier to manage and backup.

--name r1: sets the name of the container to r1.

-d redis: runs the Redis image in daemon mode, which means the container will run in the background.

redis-server --appendonly yes: runs the Redis server process with the --appendonly yes flag, which enables Redis to persist data to disk.

Overall, this command creates a Redis container with a mounted volume, sets the container name to r1, runs the container in the background, and enables Redis persistence to disk.

We can pipe data into the Redis instance using the following command.

```
cat data | docker exec -i r1 redis-cli --pipe
```

Redis will save this data to disk. On the host we can investigate the mapped direct which should contain the Redis data file.

```
ls /docker/redis-data
```

```
[root@host01 ~]# cat data | docker exec -i r1 redis-cli --pipe
All data transferred. Waiting for the last reply...
Last reply received from server.
errors: 0, replies: 1
[root@host01 ~]# ls /docker/redis-data
appendonly.aof
```

This same directory can be mounted to a second container. One usage is to have a Docker Container performing backup operations on your data.

```
docker run  -v /docker/redis-data:/backup ubuntu ls /backup
```

```
[root@host01 ~]# docker run  -v /docker/redis-data:/backup ubuntu ls /backup
appendonly.aof
[root@host01 ~]#
```

The command cat data | docker exec -i r1 redis-cli --pipe reads data from the data file and pipes it to the redis-cli command running inside the r1 container.

Here's how the command works:

cat data: Reads the contents of the data file and outputs it to stdout.

|: Pipes the output of the cat command to the next command in the pipeline.

docker exec -i r1 redis-cli --pipe: Runs the redis-cli command inside the r1 container and uses the --pipe option to read Redis commands from stdin. The -i option tells docker exec to keep stdin open, so that Redis can read the commands piped from cat data.

When you run this command, Redis reads the commands from stdin, which are the contents of the data file, and executes them. This allows you to populate the Redis instance with data from a file. The format of the data in the file should be Redis protocol-compliant, which means each command should be a single line in the format <command> <arguments>.

Step 2 - Shared Volumes
Data Volumes mapped to the host are great for persisting data. However, to gain access to them from another container you need to know the exact path which can make it error-prone.

An alternate approach is to use -volumes-from. The parameter maps the mapped volumes from the source container to the container being launched.

In this case, we're mapping our Redis container's volume to an Ubuntu container. The /data directory only exists within our Redis container, however, because of -volumes-from our Ubuntu container can access the data.

docker run --volumes-from r1 -it ubuntu ls /data

This allows us to access volumes from other containers without having to be concerned how they're persisted on the host.

```
-bash: vi: command not found
[root@host01 ~]# docker run --volumes-from r1 -it ubuntu ls /data
appendonly.aof
```

Step 3 - Read-only Volumes
Mounting Volumes gives the container full read and write access to the directory. You can specify read-only permissions on the directory by adding the permissions :ro to the mount.

If the container attempts to modify data within the directory it will error.

docker run -v /docker/redis-data:/data:ro -it ubuntu rm -rf /data

```
[root@host01 ~]# docker run -v /docker/redis-data:/data:ro -it ubuntu rm -rf /data
rm: cannot remove '/data/appendonly.aof': Read-only file system
[root@host01 ~]#
```