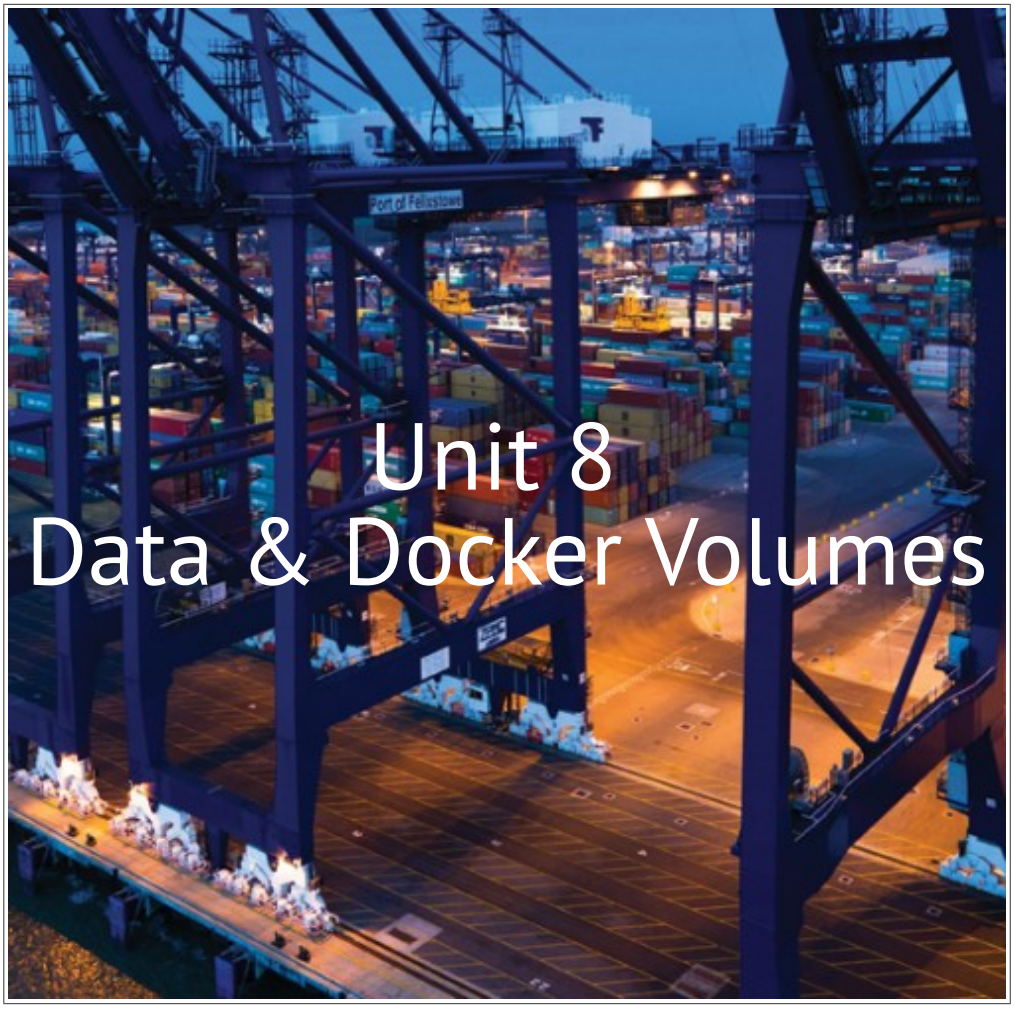

Unit 8
Data & Docker Volumes

Objectives

Upon completion of this unit, you will be able to:

- Explain how Docker enables data persistence
- Differentiate between different volume types
- Detail how Docker volumes are created and managed
- Describe how the host shares data with a container
- Take account of some practical considerations regarding volumes

Agenda

The following topics will be covered in this unit:

- Docker Volumes
- Different Volumes Types
- The docker volume Commands
- Legacy Data Volume Containers
- Practical Considerations
- Lab: Data Persistence With Docker Volumes

Docker Volumes

- Container filesystems are ephemeral
- Docker volumes allow data to persist beyond the life of a container
- Docker volumes bypass the container's copy-on-write filesystem
- Containers don't 'own' volumes, but can reference them for use

Volumes can be shared by all containers running on the Docker host, and may contain data from the host

Docker Volumes

- Because of their ephemeral nature, care should be taken when using data in container workloads
- A container's filesystem is as ephemeral as the container itself; when a container is removed, it's filesystem is removed with it, along with any data created by the container's workload
- If data written to the read-write layer of the COW filesystem belonging to a container can be deemed ephemeral, then no extra consideration needs to be taken
- If, however, the container creates, amends or reads data that needs to be shared, or needs to persist, then a Docker volume is required as a repository for the data
- Docker volumes bypass the container's COW filesystem, are stored on the Docker host as regular files and directories, and are mounted into the container's filesystem for use by the process(es) running in the container
- Docker volumes can be shared between different containers or with the Docker host, and can persist beyond the life of the container
- When new Docker images are created using the `docker commit` command, any volumes used by the container being used as the basis of the commit, are NOT included in the resultant Docker image
- Docker volumes can be created in several ways; deterministically through a Docker image (covered in Unit 9: Building Images With Dockerfiles), or as a result of the use of Docker client commands
- Volumes can be of different types:

    1. **Named volume**: an independent volume entity, created and managed independently of containers
    2. **Container volume**: a volume created in conjunction with a specific container
    3. **Host directory/file bind mount**: not strictly a volume, but a means of sharing data with a container from the host

Notes

Container Volumes

- Docker volumes can be created dynamically at container runtime
- The volume is created on the host and mounted into the container at the specified mount point
- Control parameters can be applied to characterise the volume's use (e.g. ro|rw)

Docker volumes use pluggable drivers, and third-party drivers are available

Client Option	Description
`-v, --volume`	Specifies a volume to mount into container filesystem
`--volume-driver`	Driver to use when creating and operating the volume

Container Volumes

- Container volumes can be created dynamically using the Docker CLI configuration option `-v`, `--volume`, or if the image from which the container is derived, specifies the creation of a volume

`-v, --volume=[]`: specifies a volume that is to be created for use by the container
`--volume-driver=local`: specifies the driver to use when creating volumes

- In this form of use, a container directory accompanies the `-v`, `--volume` configuration option, which will be created if it doesn't already exist, e.g.

```
$ docker create -v /data/db mongo:latest
```

- The volume is created in the Docker host as a directory, and mounted into the container's filesystem at the location specified, which must be an absolute path
- If the container location already has content, it is copied to the volume on the host before the volume is mounted
- Multiple volumes can be created by using multiple declarations of the `-v`, `--volume` configuration option
- Controlling the characteristics of the volume is achieved by appending comma-seperated option control parameters to the `-v`, `--volume` configuration option, following a colon which must be placed after the container location
- One such control parameter, `ro|rw`, specifies whether the volume is to be seen as read-write or read-only by the container (`rw` by default)
- Another control parameter informs the Docker Engine how to treat SELinux labels (if appropriate); by default Docker inherits the OS level configuration, but if the control parameter `z` is specified, Docker re-labels to allow containers to share the volume, whereas `Z` results in the volume being private to the container
- To ensure volume sharing on a Docker host utilising SELinux:

```
$ docker create -v /data/db:z mongo:latest
```

Notes

Named Volumes

- Named volumes are created independently of containers with the docker volume create command
- Named volumes can be created with the default driver, or with a third-party plugin driver

Format of the docker volume create sub-command is:

```
docker volume create [options]
```

The config options available for docker volume create:

Client Options	Description
`-d, --driver=local`	Driver to use when creating the named volume
`--name=""`	Name to be applied to the volume
`-o, --opt=map[]`	Driver specific options to apply to the volume

Named Volumes

- As of Docker 1.9, a new `docker volume` command was introduced into the Docker CLI, with a `create` sub-command for explicitly creating named Docker volumes
- As with Docker's networking capability, volumes use a plugin architecture, thereby allowing volumes to be created based on a third-party driver
- An example of a third-party Docker data volume driver is ClusterHQ's Flocker
- The format of the docker volume create sub-command is:

```
docker volume create [options]
```

- The configuration options available for the `docker volume create` sub-command are:

`-d, --driver=local`: specifies the driver to use for creating the specified volume
`--name=""`: specifies a name to be used for the volume, which can subsequently be used with other Docker CLI commands
`-o, --opt=map[]`: specifies options to pass to the driver in relation to the volume (the default, built-in driver currently takes no options)

- If the `--name` configuration option is omitted, Docker uses a random 64-digit hex string for the volume name, otherwise the volume name must be unique, even when they are created with different volume drivers
- In order to create a volume with the default, built-in driver, called `archive`:

```
$ docker volume create --name archive
archive
```

- Volume names start with any of the characters `a-z0-9`, and can only contain the characters `a-z0-9`, `_`, `.` or `-`

Notes

Using Named Volumes

- Named volumes are used by containers when referenced as an argument to the -v, --volume config option of the Docker CLI:

```
$ docker run -v archive:/backup busybox tar cvf \
> /backup/archive-2015-03-11.tar /data
```

- The named volume must be specified **without** a preceding '/', which provides an altogether different meaning
- If the named volume referenced in a Docker CLI command doesn't exist, it will be created with the container

Using Named Volumes

- To make use of the volume, the volume must be referenced when using a Docker CLI command, whence it will be mounted into the container at the location specified e.g.:

```
$ docker run -v archive:/backup busybox tar cvf \
> /backup/archive-2015-03-11.tar /data
```

- In this example, if the `archive` named volume doesn't already exist, Docker will create it using the name specified
- Its important to note that the name of the volume must not contain a leading '`/`', as this has an entirely different meaning
- As with container volumes, it's also possible to optimise the behaviour of named volumes using the optional, comma-seperated control parameters `rw|ro` and `z|Z`

Notes

Host Files and Directories

- In addition to volumes, files & directories from the host can be mounted into containers
- The -v, --volume config option is used for this purpose:

```
$ docker run -d -v /data:/var/lib/mysql mariadb
```

- If container location exists, the host file or directory is mounted over the top of the container location

The following control parameters can be applied as part of the -v, --volume config option:

Control Parameters	Description
`rw|ro`	Specifies whether mount is read-write or read-only
`z|Z`	For SELinux labels, specifies if mount is private or not
`[r]shared|[r]slave|[r]private`	Sets propagation properties of bind mount

Host Files and Directories

- As well as allowing you to create Docker volumes, Docker also allows you to bind mount a file or directory from the host system into a container:

```
$ docker run -d -v /data:/var/lib/mysql mariadb
```

- In this circumstance, Docker doesn't create a volume, but simply bind mounts the host file or directory into the container at the specified location
- If the location already exists in the container, the host's file or directory is mounted over the top of the location inside the container
- If the specified host file or directory doesn't exist, Docker will create it before bind mounting it into the container*
- To bind mount a host file or directory into a container, the host pathname must be an absolute pathname rather than a relative pathname (i.e. it must start with a '`/`')
- When mounting a file, be aware that some editing tools (e.g. `sed`) will produce unexpected results, because they rely on creating a new file (inode) as part of their editing process
- As with volumes in general, it's also possible to optimise the behaviour of host bind mounts using the optional, comma-seperated control parameters `rw|ro` and `z|Z`
- In addition, for bind mounted host files and directories, Docker also provides the means for controlling mount propagation
- Mount propagation properties for bind mounts can be controlled with the parameter `[r]shared`, `[r]slave` or `[r]private`, where `r` denotes recursive propagation
- By default, any new mounts under the bind mount inside the container are `private`, but can be set `shared` (new mounts in the host or container are reflected in the other), or `slave` (new mounts in the host are reflected in the container)
- The host mount has to be a `shared` mount in order for the bind mount to be `shared`, and either `shared` or `slave` for the bind mount to be `slave`
- Be aware that the `systemd` unit file for the Docker daemon sets the mount propagation property to `slave` by default, thereby prohibiting `shared` propagation

* This behaviour is deprecated, and will change in a future release; an error will be issued if the file or directory doesn't exist

Notes

docker volume rm & ls

- The docker volume rm command removes volumes
- Volumes cannot be removed if they are in use
- The format of the docker volume rm sub-command is:

```
docker volume rm volume [volume ...]
```

- The docker volume ls command is for listing volumes
- The format of the docker volume ls sub-command is:

```
docker volume ls [options]
```

Config options available for docker volume ls:

Client Options	Description
`-f, --filter=[]`	Filter output based on set criteria (dangling=true)
`-q, --quiet=false`	Only display volume names in the output

docker volume rm & ls

- Docker volumes can be removed using the `docker volume rm` sub-command
- The format of the `docker volume rm` sub-command is:

```
docker volume rm volume [volume ...]
```

- There are currently no configuration options for the `docker volume rm` sub-command
- Docker volumes cannot be removed when they are in use, and calls to `docker volume rm` will error in such cases
- Unnamed volumes can also be removed, along with their associated containers, when the `-v` configuration option for the `docker rm` command is used
- If a container is removed that is the sole remaining container referencing a volume, and it is not removed with the `-v` configuration option, the data volume becomes orphaned and is said to be 'dangling'
- If a container is started with the `--rm` CLI configuration option (e.g. `docker run --rm`), and references a container volume, if it exits with no other container referencing the volume, the volume will also be removed
- Named data volumes cannot be removed with the `docker rm -v` command and option
- Listing the volumes registered with the Docker daemon can be achieved with the `docker volume ls` sub-command
- The format of the `docker volume ls` sub-command is:

```
docker volume ls [options]
```

- The configuration options available for the `docker volume ls` sub-command are:

`-f, --filter=[]`: filters the output based on the criteria provided in the form of a key/value pair; the only available filter at present is `dangling=true` (volumes not referenced by any existing containers)
`-q, --quiet=false`: specifies that the output is limited to the volume's name only

Notes

docker volume inspect

- The docker volume inspect sub-command provides detailed information for the specified volume(s)
- The format of the docker volume inspect sub-command is:

```
docker volume inspect [option] volume [volume ...]
```

The single config option available for docker volume inspect is:

Client Options	Description
`-f, --format=""`	Golang text/template to apply to format the output

docker volume inspect

- The `docker volume inspect` sub-command provides detailed information regarding volumes, in much the same way as `docker network inspect` and `docker inspect` do for networks and containers, respectively
- The format of the docker volume inspect sub-command is:

```
docker volume inspect [option] volume [volume ...]
```

- The single available configuration option for the docker volume inspect sub-command is:

`-f, --format=""`: specifies a Go text/template to apply to the data objects in order to filter the results

- If no format is specified, `docker volume inspect` returns the complete configuration of the specified volume(s) as a JSON object
- In order to inspect the volume named `archive`:

```
$ docker volume inspect archive
[
    {
        "Name": "archive",
        "Driver": "local",
        "Mountpoint": "/var/lib/docker/volumes/archive/_data"
    }
]
```

Notes

Data Volume Containers

- Docker's legacy approach to data sharing involved the use of data volume containers
- Data volume containers referenced a volume, but didn't even need to be running
- Containers requiring access to the data volume used the --volumes-from config option to gain access
- It is recommended to use named volumes instead of data volume containers

The config option used with docker create or docker run:

Client Options	Description
`--volumes-from=[]`	Use all the volumes referenced by the specified container

Data Volume Containers

- Instead of a named volume, Docker's legacy approach to data sharing between containers was to use a 'Data Volume Container' (or a 'Data Only Container')
- A data volume container is a container, usually with a name, that has one or more volumes that it references, and which often performs no other task (it doesn't even need to be running)
- Volumes can be shared by containers with the use of the `--volumes-from` Docker CLI configuration option

`--volumes-from=[]`: specifies a container(s), whose volumes will be mounted into the container in question

- In the following example, a container volume is created (and mounted on to `/data/maps`) at the same time as the container named `mapdata`:

```
$ docker create -v /data/maps --name=mapdata busybox false
```

- Subsequently, a container can be invoked referencing the volumes used by the container `mapdata` using the `--volumes-from` configuration option:

```
$ docker run -d --volumes-from=mapdata --name map_srv \
> jbloggs/map_app
```

- The volume physically exists on the Docker host, and is mounted into the container's filesystem, and will persist until explicitly removed
- Other containers created using the configuration option `--volumes-from` can also share the volume, and can use the same container (`mapdata`) to reference the volume, or use container `map_srv` to achieve this, and so on
- As with the `docker create` and `docker run` command line option `-v`, `--volume`, multiple `--volumes-from` options can be used to accumulate a myriad of data sources
- The `--volumes-from` configuration option can be appended with `rw|ro` (e.g. `:ro`) and `z|Z`, which have the same effect as those parameters previously discussed

Notes

Practical Considerations

- When the Docker host is remote from the Docker client, use a third-party plugin (or other data sharing capabilities) for sharing remote data
- Docker Toolbox shares /Users for Mac OS X, and C:\Users for Windows, with the Docker host, mounted at /Users
- Named volumes and volumes created with Docker CLI are owned by UID 0
- To avoid permission problems when container's process is not run as UID 0, create a mount point owned by a defined user in the Docker image

Practical Considerations

Mounting external data sources

- For making data sources external to the Docker host available as Docker volumes, either use a third-party volume plugin (e.g. Flocker or Convoy), or make use of the data sharing capabilities of the external data source (e.g. Virtualbox)
- If you're using Docker Toolbox on Mac OS X or Microsoft Windows, then `/Users` and `C:\Users`, are shared from the local machine via Virtualbox's `vboxsf`, respectively, and mounted on the Docker host at `/Users`

Volume ownership and permissions

- Named volumes, and volumes created dynamically via the Docker CLI, are owned by UID 0
- Accordingly, a container's process' ability to read, create, amend or remove data in such volumes is entirely dependent on standard Linux file and directory ownership and permissions
- However, it is recommended that a process(es) running in a container, does so as an alternative user, with a UID other than 0
- As a consequence, a process with a UID other than 0 running in a container, connected to a named or dynamically created volume, will be unable to write to the volume due to insufficient permissions
- A technique for circumventing this problem is to create a user in the Docker image, along with a mount point owned by the user
- When a container is created form the image, and a named volume or dynamically created volume is mounted onto a mount point owned by a specified user, the directory is owned by that user, and the process is able to operate within the directory with full privileges

Notes