Lab STRG3 - Container Storage

Introduction and Prerequisites

The purpose of this lab session is to:

- Understand different options for data management in a docker context, including bind mounts, tmpfs, and volume mounts.

The following resources and tools are required for this laboratory session:

- Any modern web browser
- Modern ssh client
- Docker on a OSTK instance.

Time & Assessment

The entire session will take 90 minutes. A formal assessment (points relevant for the final mark) is not foreseen.

Task 0 – Set up and understand the working environment

In this task, you will set up the working environment you will use for the rest of the lab session.

Subtask 0.1 – Launch VMs and pull docker containers

Follow the steps below to set up the working environment for this lab:

- Log in to the Ned Openstack cluster using your provided credentials
- Launch 2 OSTK instances with the following characteristics:
    - Image: Latest available Ubuntu
    - Each VM requires a floating IP
    - Ensure that your keys are installed on the instances
- To install docker on your OSTK instance run the following commands:

```
sudo apt-get update
sudo apt install docker.io
```

- Hint: To run docker commands without need to always type sudo in front, add the user to the docker group with the following command in each instance: *sudo usermod -aG docker* `$USER`
- Log out and log in again in the instance for the last modification to take effect. Verify that docker is running correctly - you can do this by simply listing active containers

Subtask 0.2 – Perform basic queries to understand your environment

Before starting the lab properly, it is helpful to understand basic characteristics of the environment you have been provided with.

1. Check what file systems are mounted in your OSTK instances and what filesystems are used using the mount command
2. Check what storage driver is used by the docker engine you have been provided by getting **info** on docker

Task 1 - Understand the union file system AUFS

In this task, you will create an aufs union file system comprised of directories on the local filesystem.

Subtask 1.1 – Create layers and mount them via aufs

First, create two layers:

1. Create a directory in /home/ubuntu called myContainerLayers

2. Inside this directory create two more directories, base_layer and writable_layer

3. Inside each directory, create an empty file using an editor. These could be called base_layer-1-ro-file.txt and writable_layer-rw-file.txt

4. Create a new directory at root level in the file system using sudo mkdir /aufs

5. Mount the filesystem using the aufs driver to the /aufs mountpoint - check the lecture content for details on how to do this

6. Change directory to /aufs. What is in this directory? Is it what you expect?

7. Create a file in the /aufs directory with touch - call it union_file_system_file.txt. Do you see this file under /home/ubuntu?

8. Using an editor under /aufs, edit the file writable_layer-rw-file.txt - what happens in /home/ubuntu?

9. Create a file in /home/ubuntu/myContainerLayers/base_layer - call it base_layer-2-ro-file.txt - what happens inside /aufs?

10. Edit the file base_layer-1-ro-file.txt in the /aufs directory. What happens to the files in /home/ubuntu?

11. Remove the file base_layer-1-ro-file.txt in the /aufs directory - what happens inside /home/ubuntu? Note that you may need to look for hidden files to see what has happened. What happens if you remove the offending item from writeable layer?

Task 2 – Understanding bind and tmpfs mounts in docker

In this task you will experiment with different ways that file systems work in docker, considering three specific cases:

- Mount a single file from the host machine into a docker container using a bind mount;
- Mount a directory from the host machine into a docker container using a bind mount;
- Working with a tmpfs mounted filesystem in a docker container

Subtask 2.1 – Mounting a single file using a bind mount

Follow the steps below and answer the questions inline:

1. Create a text file containing a single sentence in the home directory of your VM - call it subtask21.txt

2. Launch a standard ubuntu container which binds the file on the host file system to a file with the same name inside the container (both the container and the host have the user ubuntu)/
    - Note that you will need to use the -v flag and the full absolute path of the file

3. Using vi, edit the file inside the container - add another sentence to the file from inside the container.
    - Note that the container does not have vi installed, so you will need to apt-get update and apt-get install vim

4. From outside the container, look at the file on the host OS - what has happened? Is this what you expect?

5. Using the chmod instruction, modify the permissions of the file from inside the container - what happens to the file from the perspective of the host filesystem?

6. Create a new user inside the container called rick. When you list the contents of the directory, what do you see? Create another user inside the container: call it morty. Using the chown instruction, modify the ownership of the file to morty:morty from inside the container - what happens on the host?
    - (You can use the command adduser <username> to create a new user).

7. From inside the container, change the ownership back to rick:rick. From outside the container, edit the file using vi - add another sentence.
    - What happens now?
    - (Hint: you can use ls -li to get some understanding of what is happening)

8. From inside the container, change the permissions and ownership on the file
    - What happens on the host filesystem?

9. Terminate the container and remove it

Subtask 2.2 – Mounting a directory using a bind mount

1. Create a directory inside the ubuntu home directory on the VM - call it subtask22_dir

2. Launch a new ubuntu container mounting a directory inside the container - the directory can be mounted to /opt/subtask22

3. Using dd, create a 10GB file on the mounted directory inside the container - you can use a blocksize of 10240 and a count of 1000000
    a. What is the (quite obvious) issue with sharing resources between host OS and container using bind mounts? (Hint, you can use the df command for checking the disk usage)

4. Terminate the container and remove it

Subtask 2.3 – Understanding file systems in docker - working with tmpfs

1. Launch a new ubuntu container mounting a directory inside the container using tmpfs - the directory can be mounted to /opt/tmp. You can use the --mount flag for docker specifying the tmpfs filesystem type.

2. Inside the container - using dd - create an 800MB file on the mounted file system inside the container on the tmpfs filesystem - a blocksize of 1024 and a count of 800000 can be used
    - Do you see increased usage of the file system on the host?
    - Where does this data go? How can you see it? What are the implications for use of tmpfs filesystems in containers? (Hint, you may need to run top on the host filesystem and observe what happens when the file is created).

3. Terminate the container and remove it

Task 3 - Working with docker volumes

By now you should have an understanding of some of the issues associated with mounting files and directories directly to your container. Volumes were created to address some of these issues. Here, you will explore how they work.

Subtask 3.1 – Mount a volume to a container

1. Create a docker volume of maximum size 100MB using the tmpfs filesystem type. You will need to indicate the type of the filesystem and you should give it a name, eg task31_vol. Inspect the volume (docker volume inspect) to ensure it has the correct properties.

2. Create a container with the volume mounted on /opt/volume; use the mount command inside the container to ensure that the volume is available and mounted in the correct place.

3. Using dd create an 800MB file to fill up the volume as in subtask 2.3 above
    a. What happens?
    b. How does this work?
    c. Remove the file that was created

Subtask 3.2 – Mount the same volume to another container

1. Run another container and attach the volume in readonly mode. Confirm that the volume is visible in both containers and has read-write access in one container and read-only access in the other.

2. Exit and remove this container.

3. Run another container and attach the volume in read-write mode. Using mount, confirm that the volume has read-write access. Confirm that the volume is visible in both containers and has read-write access in both containers. You can use the echo command with output redirection to add content to a file, eg echo "file content" >> /opt/volume/file.txt

4. Exit both containers and remove them using docker rm. Remove all volumes using docker volume rm.

5. What have you learnt?

Task 4 - Create a complete Container Storage Environment with Linux Tools

Create a complete container storage environment like it was presented in the lecture. Use the Linux Overlay facility and make use of `pivot_root` and `unshare` to create a dedicated root and hierarchy with all the content required for a container.

Cleanup - Stop the Bills!

IMPORTANT: At the end of the lab session:

- **Delete** all - unneeded - OpenStack VMs, volumes, security group rules that were created by your team.
- **Release** all floating IPs back to the central pool for others to use.
    - Go to Network -> Floating IPs to release IPs back to the pool

Additional Documentation

The following documentation may be useful for this lab session:

- Docker command reference:
    - Docker run reference - https://docs.docker.com/engine/reference/run/
    - Docker CLI reference - https://docs.docker.com/engine/reference/commandline/cli/
    - Docker volume create reference - https://docs.docker.com/engine/reference/commandline/volume_create/
- Linux man pages
    - Man <command name> (e.g. man unshare)