Lab CMP II - Containers

Introduction

This laboratory is to:

- Experiment with cgroups for process management,
- Validate the application of cgroup rules applied to processes,
- Define a namespace and run a process in a Linux Container (cgroups plus namespaces)
- Experiment with basic docker management commands,
- Understand Docker architecture,
- Build your own docker image,
- Use that docker image in a composite application by linking it to an existing web-style application.

Prerequisites

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

- An account on docker hub (a Docker ID): https://docs.docker.com/docker-id/
- Any modern web browser,
- Any modern SSH client application
- OpenStack Horizon dashboard: https://ned.cloudlab.zhaw.ch

Time

The entire session will take 90 minutes.

Task 1 – Start Necessary VMs and Verify Environment

From previous labs you now know how to configure and launch a VM in OpenStack. Please perform the following steps:

1. Create 1 VM (use flavor: `m1.small`) and start an instance from a simple Ubuntu image
2. Enable SSH via security groups and assign this rule to the VM
3. Assign floating IPs to your VMs and SSH into it
4. Check if docker is installed properly by listing the contents of the local registry (`$docker images`) and if not, install it using:

   ```
   $ sudo apt update
   $ sudo apt install docker.io
   ```

   Add the current user (ubuntu) to the docker group to access without sudo:

   ```
   $ usermod -aG docker $USER
   ```

   And reboot the VM:

   ```
   $ sudo reboot now
   ```

Task 2 – Experimenting with Linux Containers

In this task you will manually define and evaluate a Linux Container. This requires a specific allocation of CPU resources using Linux cgroups, applying these rules to a process, which again will be run within its own namespaces. In other words, you will create a "**Box made of Rules and Run a Process therein**".

Subtask – Defining CGroups

In order to limit CPU allocation to processes use hierarchies and subsystems, groups, and define respective limits.

- Examine the default cgroups hierarchies and configuration of Ubuntu.
  Hints: check the /proc/cgroup file, the /sys/fs/cgroup filesystem, mounted controllers, systemd-cgls, etc (Hint: feel free to refer to your BSY materials)

- Create two new groups in a CPU hierarchy, one for high priority processes and one for low priority processes.
  Hints: cgrous are directories in the cgroups fs hierachy. Give access rights to user ubuntu.

- Define a 1:10 ratio, in terms of CPU time-share allocation, for the two groups. For more information about subsystem parameters see "Red Hat Enterprise Linux 6 Resource Management Guide" on Moodle (or the online version here)

Subtask – Evaluate the CGroups Configuration

With the two groups configured, create CPU-intensive processes, assign them to groups, and examine the effect.

- Create a number (minimum of 2) of CPU intensive processes[1] (aka workloads) for evaluation purposes. You may find the Linux command line application "stress" (man stress) useful.

- Assign the processes[2] to the groups
  Hint: See slides on attaching processes to cgroups. Or check the RedHat resource management Guide - check the alternative methods (without using the cgroup-tools)

- Demonstrate the result to the lab assistant and explain it. Display the hierarchies and the respective result.
  Hint: You may also take a screenshot of the resource distribution (top) and hierarchy (tree).

1 For e.g. you could use: `` `stress -c 1` ``, where 1 is the number of CPUs on the system **OR** `` `dd if=/dev/urandom | bzip2 -9 >> /dev/null` `` Runn the process in background adding '&' at the end.
2 `` `ps -aef --forest` `` can be of help to see child processes

Subtask – Defining Namespaces

Stop your workloads from the previous tasks and start over. This time, add at least one of your workloads into a dedicated namespace.

- Create your workload within a dedicated namespace
- Now validate that your workload is within its own namespace by displaying the process list from within this namespace.
- Compare and explain the PIDs within the container and on the host system.

Task 3 - Docker

In this task you will begin with the basic commands of docker, moving on to creating your own docker container image and finally using that docker container as part of a composite application (2 linked docker containers).

Subtask - Basic Docker

In this section we look into the basic docker commands. For the purpose of consistency, we will name the container we create as **`my_c`**. If you want quick help on a particular command (`docker help` provides a list of commands) or parameter simply execute: `docker $COMMAND --help`

Container Execution

- Run your first container using the docker image `ubuntu:16.04` and running the command `/bin/echo 'Hello Lab!'`

Docker Management

- Once done, exit your container, verify that no docker process is running on the VM beside the docker daemon. Hint: show running docker container processes.
- What does the output tell you? What other detail can you get using the command? Is the process completely cleaned up? If it's not how do you clean it up?
  Hint: show stopped container processes and stop container processes
- You have executed a container however, you might need to have shell access with a container. Do this with using the command to execute a shell (requires a terminal and be interactive) from the Ubuntu 16.04 LTS container image.
- Once the terminal session (shell) is attached, check the PIDs from within the container. What has actually happened?
- Why might you need such access? How can you make this container run in the background and then get access to an existing running container?

Container Logs

- You might also need the output of a docker container, especially where a container runs as a background process (daemon). Display the logs.

- If you wanted to stream/follow the docker log outputs of a container how would you do this?

Container Runtime Information

- Finally, while a docker container is running you might need to extract runtime information. How to get and what can you tell from this output?
  Hint: which command gives more information on a specific container than docker ps?

Container Removal

Once you are finished with your container, as with VMs, you should destroy them and release the resources associated with them with the appropriate docker command.

- What should you do in order to make it work with the appropriate destroy command when the container is running in the background (`-d`)?

Subtask - Docker and Linux Container Architecture

In the lecture you were told that Docker (or any container tooling) is using the Linux container infrastructure, that is cgroups, namespaces, etc. Start a container and analyse the present (running) container technology architecture (containerd/runc) together with cgroups and namespaces configuration. Hint: see processes and process hierarchy, systemd-cgls

Can you show the relation between Docker and containerd on the running system? Hint, check the following command: $ systemctl cat help

Check the configuration options of the $ docker command and compare it with the options of $ runc. Discuss what you notice.

Run a container and analyze the cgroups configuration. Hint: $ man systemd-cgtop

Subtask - Building Docker Container Images

The power of docker is realised through its container creation system. Here you will create your own image that allows you run MongoDB[3]. However we do not want to do things manually. We want to be able to create automated and repeatable builds. `Dockerfiles` are used to accomplish this in docker. A `Dockerfile` is simply a set of sequential commands run to create and run a process within a docker container. To install MongoDB the following guide shows how to do it manually [MONGO_UBU]. For completeness the steps are:

```
1. apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
   0C49F3730359A14518585931BC711F9BA15703C6
2. echo "deb [ arch=amd64,arm64 ]
   http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4
   multiverse" | tee /etc/apt/sources.list.d/mongodb-org-3.4.list
3. apt-get update
4. apt-get install -y mongodb-org
5. mkdir -p /data/db
```

3 http://mongodb.org/ a document-oriented DB allowing for easy persistence of JSON data

Run the DB process "`/usr/bin/mongod`". The DB is then accessible on its default port `27017`. Take these steps and create a `Dockerfile` to build the container based on Ubuntu 16.04 LTS. You will need to use the following docker commands: `FROM`, `RUN`, `EXPOSE` and `ENTRYPOINT`. A `Dockerfile` is just a text file with docker build commands within [BUILDER].

Once you have created your `Dockerfile`, you can build a docker container image. *Note*: you should name your container in the following way:

```
<MY_CONTAINER_IMAGE_NAME> = <DOCKER_ID>/my_mongo
```

- Can you explain what the `-t` parameter does and why it's important?
- Once built, run the DB container with an external port mapping using the default MongoDB port 27017.
- Now that you've built your docker container image you can share it within a docker registry. By default docker push places the docker into the public docker hub registry. You need to login[4] into docker hub to allow pushing. Once logged in, push your newly built docker image.
- How can you add version tags to your published image using docker push?

Stop the Bills!

IMPORTANT: At the end of the lab session:

- **Delete** all - not needed - OpenStack VMs, volumes, security group rules that were created by your team. This will also destroy any containers you have running within the VM.
- **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

OpenStack Horizon documentation can be found on the following pages:

- User Guide: https://docs.openstack.org/horizon/latest/
- Linux manpages
- [BUILDER] Docker build (Dockerfile) reference https://docs.docker.com/engine/reference/builder/

4 https://docs.docker.com/docker-id/