A
PROJECT REPORT ON

A CONTAINER CLONE TO REVERSE ENGINEER THE CONTAINERIZATION TECHNOLOGY

By

UTSAV PARMAR (CE-085) (17CEUSF124)

B.Tech CE Semester-VI
Subject: System Design Practice

Guided by:
Prof. Pandav K. Patel
Assistant Professor
Dept. of Comp. Engg.

Faculty of Technology
Department of Computer Engineering
Dharmsinh Desai University

Faculty of Technology
Department of Computer Engineering
Dharmsinh Desai University

CERTIFICATE

This is to certify that the practical / term work carried out in the subject of

System Design Practice and recorded in this journal is the

bonafide work of

UTSAV PARMAR (CE-085) (17CEUSF124)

of B.Tech semester **VI** in the branch of **Computer Engineering**

during the academic year **2019-2020**.

Prof. Pandav K. Patel
Assistant Professor,
Dept. of Computer Engg.,
Faculty of Technology
Dharmsinh Desai University, Nadiad

Dr. C. K. Bhensdadia,
Head,
Dept. of Computer Engg.,
Faculty of Technology
Dharmsinh Desai University, Nadiad

CONTENTS

Abstract

Container Clone refers to a set of processes isolated from each other in an Operating System allotted a well-defined specific set of resources to work in.

Introduction

Purpose

In cases when there are multiple instances running born from a single application, we need all the processes to access same part of memory and not indulge in into other programs' resources. We would also like it to abstract the processes' belonging to same set to stay away from the eccentricities of the host it is running on. Thus, to achieve

- Isolation
- Resource constraints
- Abstraction

we use 'Container Clone'.

The underlying purpose of this program is to reverse engineer Docker and learn the basic layers of its workings.

Technology Used:

- Python, Shell Script

Platform Used:

- Debian based Linux Distributions with kernel version 4.19 and above

Tools Used:

- Visual Studio Code

Software Requirement Specification

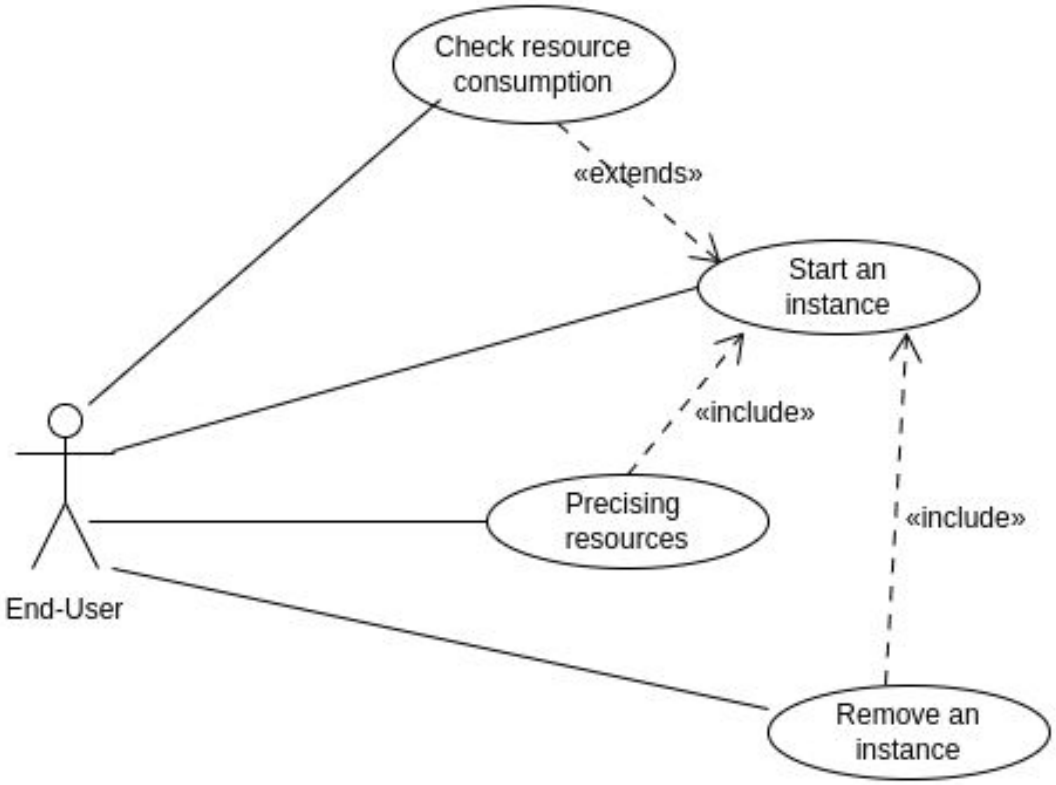
Purpose:

This document defines the requirements and functionalities that the system offers.

- Functional Requirements
  - The system must allow the client to create an instance
  - The system must allow the client to run the instance as a non-root user
  - The system must allow the client to use the required resources
  - The system must allow the client to see the resource consumption
  - The system must not allow the client's other process to interfere with the running instance
  - The system must allow the client to have full control over the instance, that is, to stop and/or remove the instance

- Non-Functional Requirements
  - The system must consume an efficient amount of memory
  - The system must be able to run on widely used Linux distributions
  - The security of the system must be optimal

Use Case Diagram



Building Blocks and Modules

➢ Namespaces

- Namespaces are a feature of the Linux kernel that partitions kernel resources such that one set of processes sees one set of resources while another set of processes sees a different set of resources.
- The feature works by having the same namespace for a set of resources and processes, but those namespaces refer to distinct resources. Resources may exist in multiple spaces.
- Examples of such resources are process IDs, hostnames, user IDs, file names, and some names associated with network access, and interprocess communication.
  (verbatim - link)

Finding pid of a process:

```
$ pidof zsh
$ sudo pidof zsh
```

Since, no pid is returned, it implies that the application is not running.

Let's create a namespace

This let us run a program in an unshared namespace than it's parent.

```
% pidof zsh
pid 1
```

Since, it is unshared from its parent, it has been assigned the pid 1

But, pid 1 is only reserved for boot processes such as systemd in Debian based distributions.

So, we open up a new terminal shell and look for the processed and pids.

```
$ ps 1
init
```

```
$ pidof zsh
7723
```

➢ Control Groups

- Control groups, usually referred to as cgroups, are a Linux kernel feature which allow processes to be organized into hierarchical groups whose usage of various types of resources can then be limited and monitored.
- Namespaces are not responsible for the restriction of physical resources but cgroups.
  ($man cgroups)

For example,

CPU : "Top cpuset"

/ \

CPUSet1 CPUSet2

| |

(Professors) (Students)

In addition (system tasks) are attached to topcpuset (so that they can run anywhere) with a limit of 20%

Memory : Professors (50%), Students (30%), system (20%)

Disk : Professors (50%), Students (30%), system (20%)

Network : WWW browsing (20%), Network File System (60%), others (20%)

/ \

Professors (15%) students (5%)

```
$ cat /proc/cgroups
#subsys_name    hierarchy       num_cgroups     enabled
cpuset  4       1       1
cpu     3       56      1
cpuacct 3       56      1
blkio   6       56      1
memory  7       91      1
devices 11      56      1
freezer 9       1       1
net_cls 8       1       1
perf_event      2       1       1
net_prio        8       1       1
hugetlb 10      1       1
pids    5       56      1
```

This command will create a sub-group “myapp” under the “blkio” system. Writing values to these files provides controlled access to various resources. You can check if your group is created, by running the command, lscgroup, which lists all the control groups.

```
$ cgcreate -g blkio:myapp
$ lscgroup | grep blkio:/myapp
blkio:/myapp
```

Thus, cgroups provide us with:

- Resource restrictions: Limit to a specific memory
- Prioritization: Some processes can be prioritized for more memory usage
- Control: All the power over processes - monitoring, freezing, stopping etc

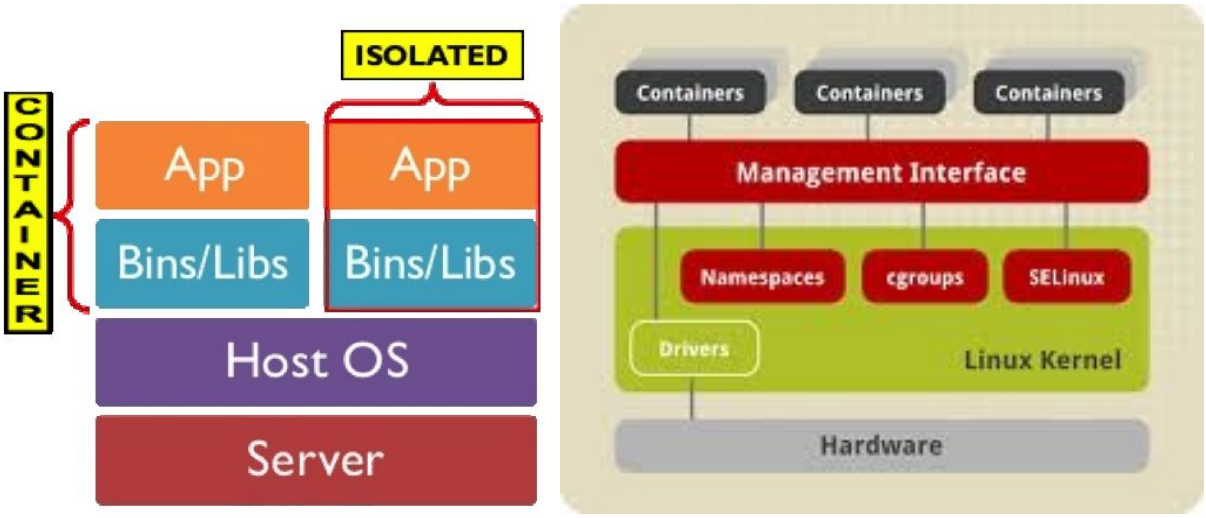
➢ Chroot

- Chroot is used to change a specific directory to the calling process’s root directory. This directory will be further referred to by ‘/’ by all the child processes.
- This is neither to be used for sandboxing a process not to restrict filesystem system calls.
- Chroot thus can be said to be a restricted filesystem view, so it is also referred to as “chroot jail” too.

  ($man chroot)

Architecture

Let's take a closer look at the containers:



As we can observe from the above images, the fundamental blocks of a container-namespaces, chroot and cgroups are what constitute a container. A container is but a standard set of software that packs up the whole application and it's binaries and libraries, that is to say, its dependencies together to facilitate efficient and same working behaviour on different machines.

Containers are also referred to as 'lightweight', that is to say, that they work using the operating system's kernel and do not require the overhead of a whole operating system. Thus, they inherently become smaller in capacity and faster in performance than VM. The idea behind a container is "**write once and run everywhere**".

Testing

- Testing Scope
  - In-Scope: Chroot functionality, resource limiting
  - Out-of-Scope: Performance Testing

- Testing Method:
  - Regression Testing
    - Each time a new build was deployed, testing was performed to detect defects.
    - Test cases for the new functionality kept on adding on top of the old ones.
  - Mocks and Monkeypatching
    - Monkeypatching is dynamically changing a piece of software at runtime such as a module, object, method, or function.
    - The monkeypatch fixture helps you to safely set/delete an attribute, dictionary item or environment variable, or to modify sys.path for importing.
    - Mocks:
      - To isolate certain external dependencies, we mock, usually external APIs to check for behaviours.

It is easy to patch modules functions and/or variables due to getting easy reference to most of them in Python.

```python
# contents of test_module.py with source code and the test
from pathlib import Path


def getssh():
    """Simple function to return expanded homedir ssh path."""
    return Path.home() / ".ssh"


def test_getssh(monkeypatch):
    # mocked return function to replace Path.home
    # always return '/abc'
    def mockreturn():
        return Path("/abc")

    # Application of the monkeypatch to replace Path.home
    # with the behavior of mockreturn defined above.
    monkeypatch.setattr(Path, "home", mockreturn)

    # Calling getssh() will use mockreturn in place of Path.home
    # for this test with the monkeypatch.
    x = getssh()
    assert x == Path("/abc/.ssh")
```

In this example, **monkeypatch.setattr()** is used to patch Path.home so that the known testing path Path("/abc") is always used when the test is run. This removes any dependency on the running user for testing purposes. After the test function finishes the Path.home modification will be undone.
(verbatim: link)

Practially performed tests:

Test Cases		
Planned	Passed	Failed
30	27	3

Conclusion

- Chroot jail has been implemented successfully with an image of ubuntu 18.04 with kernel version 5.0
- Namespaces can be assigned easily to the process/es
- Resources such as memory, CPU usage can be allocated in limited view to process/es
- A tar-ball or a binary file is containerized successfully

Limitations

- The system does not have a graphical interface
- The system is still subjected to performance overhead
- The system is not yet suitable for graphical applications
- The system's optimal usage is gained from microservies at best and not other kinds of services

Further Extensions

- Embed the idea of container clusters and it's usage over distributed computing
- Implement other high level docker commands

Bibliography

- For the creation of diagrams, Umlet tool was used
- To understand the concept behind containers and docker, the following references were used:
    - https://www.docker.com/
    - https://opensource.com/article/19/10/namespaces-and-containers-linux
    - https://medium.com/@teddyking/linux-namespaces-850489d3ccf
    - https://www.kernel.org/doc/Documentation/
    - Linux Man Pages
    - https://www.lizrice.com/
    - https://www.ibm.com/cloud/learn/containerization
- To understand the working of system calls used in linux and to incorporate in the project linuxjourney.com was used
- For testing purposes:
    - http://softwaretestingfundamentals.com/black-box-testing/
    - https://docs.pytest.org/en/latest/monkeypatch.html
- Other references:
    - https://stackoverflow.com
    - https://en.wikipedia.org/wiki/
    - https://github.com
    - https://docs.python.org/3.6/
    - https://air.imag.fr/index.php/Main_Page
    - https://www.geeksforgeeks.org/