UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Development of a virtualization framework with LXD

Bachelor's Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Òscar Pérez Castillo

In partial fulfillment
of the requirements for the degree in
Telecommunications Technologies and Services **ENGINEERING**

Advisor: Jose Luis Muñoz Tapia
Advisor: Rafael Genés Durán
Barcelona, Date 21 June 2021

Abstract

Containers are an operating system virtualization technology used to provide processes isolate environments. They provide a lightweight solution where a single Linux kernel is shared between the host and the containers.

Using this technology several project have emerged over the years which offer different use cases. One of these project is "LXD", which offers a way of running unmodified Linux distributions inside containers.

This thesis intends to provide a framework on top of "LXD" in order to improve some of it's functionalities and provide new user cases.

We have developed two command line tools and one minimal web application which integrate all the improvements on top of "LXD".

Resum

Els contenidors són una tecnologia utilitzada per el sistema operatiu que ens permet oferir alsprocessos un entorn aïllat. Ofereixen una tecnologia molt lleugera on només es comparteix un únix kernel entre el propi host i els contenidors.

A partir d'aquesta tecnologia diversos projectes han surgit al llarg dels anys. Un d'aquests projectes és "LXD", que permet utilizar distribucions Linux dintre de contenidors.

Aquesta tesis té l'objectiu d'oferir un sistema que utilitzant "LXD" permeti extendre i millorar les propies funcionalitats.

S'han desenvolupat dues aplicaciones de comandes i una aplicació web que utilitzant "LXD" han permès millorar les funcionalitats de "LXD".

Resumen

Los contenedores son una tecnología empleada por el sistema operativo que permite ofrecer a los procesos un entorno aislado. Ofrecen una solución muy ligera donde únicamente se comparte un kernel entre el propio host y los contenedores.

A partir de esta tecnología, diversos proyectos han surgido durante los últimos años que emplean dicha tecnología. Uno de estos proyecto es "LXD", que permite correr distribuciones de Linux no modificadas dentro de contenedores.

Esta tesis tiene el objectivo de ofrecer un sistema que usando "LXD" permite extender y mejoras sus funcionalidades.

Para eso, hemos desarollado dos comandos y una aplicación web que usando "LXD" nos han permitido mejorar dichas funcionalidades.

Acknowledgements

I would like to thank you first to Jose for giving me the opportunity of working in this thesis and next to Rafa for his continuos help and dedication during all the project. Their help has been essential for the development of the project.

Also to my family and friends for their continuos support during the project.

Revision history

Revision	Date	Purpose
0	01/06/2021	Document creation
1	20/06/2021	Document revision
2	21/06/2021	Document release

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Òscar Pérez Castillo	oscar.pz.castillo@gmail.com
Jose Luis Muñoz Tapia	jose.luis.munoz@upc.edu
Rafael Genés Duran	rafael.genes@upc.edu

Written by:		Reviewed and approved by:	
Date	20/06/2021	Date	20/06/2021
Name	Òscar Pérez Castillo	Name	Jose Luis Muñoz Tapia
Position	Project Author	Position	Project Supervisor

Contents

List of Figures

Listings

1 Introduction

Virtualization is a computer mechanism that allows a single computer to host multiple virtual machines, where each system has the ability of running a completely different operating system than the main machine.

One kind of virtualization it is the "OS-level virtualization" (or containerization), which is a paradigm in which the operating system, through different os level functionalities, can create user instances, where those instances are what we refer as "containers" as they have their own set of os-resources properties in their own environment.

On top of that technology, several systems and technologies have emerged over the years. In Linux, the "Linux Containers project [1]" has been working on containers for over ten years and has develop an open source containers platform that provides a set of utilities to provide a framework as close as what you get from a VM (virtual machine).

One of those utilities are "LXC/LXD [2][3]". These utilities are a set of tools that allow us to run unmodified Linux distributions inside containers without the overhead of creating a virtual machine. This is extremely helpful because we can create different linux distributions in one unique linux machine.

So, the objective of this thesis is to provide a framework on top of the "LXC/LXD" utilities to unify some of their commands and improve the management of the containers.

1.1 Requirements and specifications

The "LXC/LXD" set of tools are used for creating such "containers". Once created, we can start/stop them, add them shared folders, manage memory, manage cpu resources, set up linux distribution ... But a lot of commands for properly set up a container with different configurations (folders, proxies ...) were needed. Also, when the number of containers increase, we have no way or organize them of categorize them.

So the requirements, based on those problems, were:

- Be able to manage a common container configuration by a text file.
- Possibility to group containers by "domains".
- Tag containers by an alias name.
- Set up proxies based on a text file.

By developing the following base of tools:

- **lxce**: base command installed on top of "lxc" command line tool. Should be responsible for configuring all the containers based on configuration files and commands.
- **lxce-admin**: command for managing the different hosts with lxce installed in a centralized location.
- **web interface**: minimal web application for visualizing all the containers and manage them with a simple API.

1.2 Previous efforts

The thesis began with the two commands (lxce and lxce-admin) in an initial version:

- **lxce**: this command was in an initial version but it lacked a lot of different features along robustness.
- **lxce-admin**: this command was simple but should be extended for improve some features.

The two were written in Javascript.

For the web interface no versions were made, so it should be coded from the beginning.

1.3 Work plan

For the work plan we set the following goals, in order of preference:

- Develop a robust, well tested version for the "lxce" command.
- Integrate the "lxce-admin" improvements in the centralized "lxce-admin" command.
- Based on left time, develop the web interface application.

Where we can see summarize it in the following Gantt diagram:

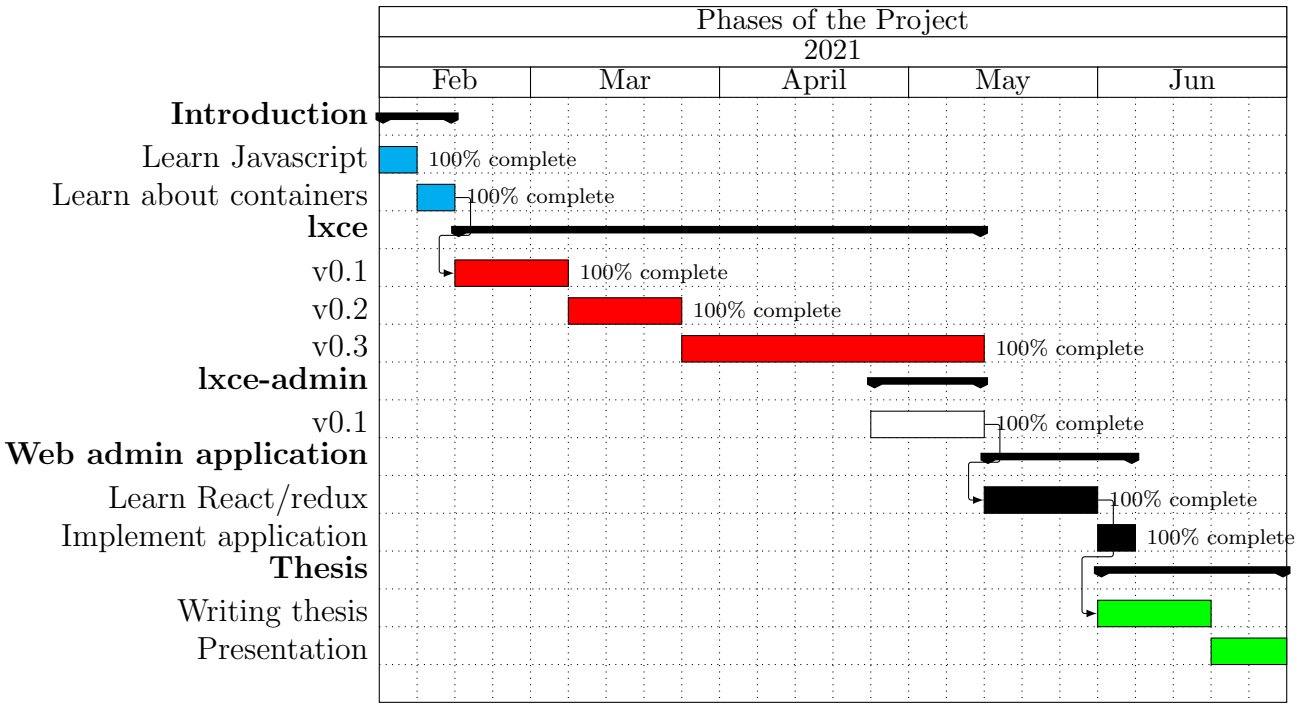

Figure 1: Gantt diagram of the project.

In term of deviations of the initial plan, we intended to provide a subcommand in the "lxce" command line tool which could generate automatic "nginx" configurations. It involved having to configure and work with web certificates but an external approval was necessary so at the end had to be canceled.

2 State of the art

This chapter will provide a general overview, in the context of Linux, of the different technologies used by the operating system to provide the foundation of "containers".

It will also expose a brief comparison between some systems which use containerization and explain which one suits our needs better.

And finally it will present the set of tools in which our framework resides on.

2.1 Container technology

Containers. Operating system main abstractions are processes. Processes act as instances of programs and are executed whenever the CPU schedules them. Depending on their properties they have the ability to execute different actions (read from file, send a packet, open a socket ...).

Containers are no different than this. They are mainly an abstraction for a process with a set properties provided the operating system by different technologies, and a supporting runtime. The main technologies are **namespaces** and **cgroups**.

And as the functionalities offered are implemented inside the kernel, they don't need to run any kind of hypervisor or virtualization. The following image illustrates this fact:

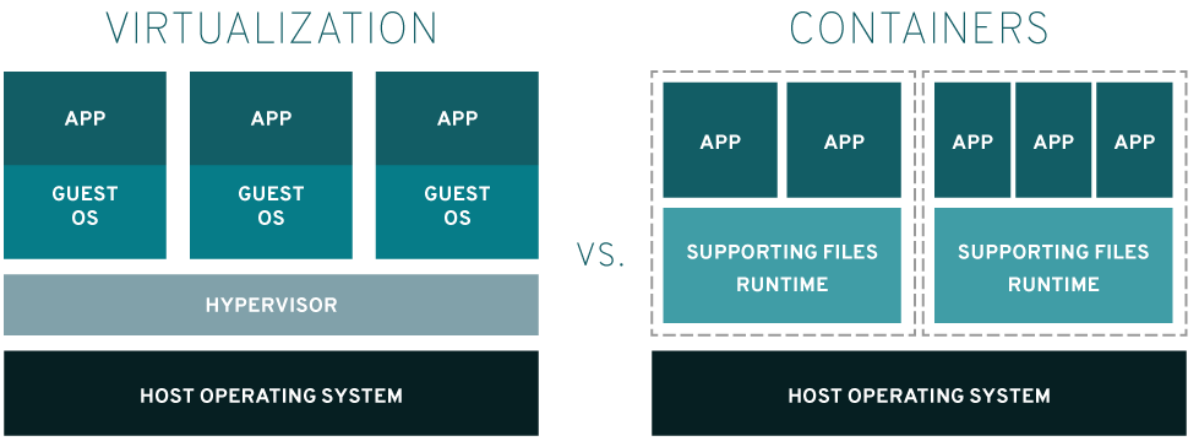

Figure 2: Virtualization vs Containers.

This creates a lightweight solution for applications where only one service will be running (such as a web server) without the need of setting up a whole VM with a separated kernel.

For enabling the existence of containers, the kernel offers some technologies for "isolate" containers and control their resources.

Namespaces. The first kernel feature provided by the kernel, which is the main foundation for the concept of containers, are the kernel namespaces. They are mainly and abstraction that enables the kernel to limit the context and visibility of the kernel objects. The kernel just label their resources and when it receives a request for viewing some of his objects, it only offers the ones according to the label.

In this way, different process with different labels can have separate views of the kernel objects and they are not able to access the objects different from their label.

The kernel provides 7 namespaces:

- Mount (mnt).
- Process ID (pid) (mnt).
- Network (net).
- Inter-process Communication (ipc).
- Control group (cgroup).
- UTS.
- User ID (user).

And they are manipulated using 3 syscalls:

- clone(): used with namespaces, creates a new process in the specified namespace.
- unshare(): modify the context of a process.
- setns(): allows attaching a process to an existing namespace.

Cgroups. Control groups ("cgroups") are a kernel feature that allows the kernel to allocate resources (CPU time, system memory) to a group of process. They are not dependant of namespaces, but they are used with namespaces to limit, control and isolate resource usage.

We won't go into details about the technologies mentioned before, but it is good to have to a general overview of the mechanisms used by the kernel.

2.2 Containerization systems

The concept of "container" is enabled by the different kernel technologies mentioned before, but there is another key element that takes part - the runtime.

The uses and systems in which containers are used nowadays vary a lot, but the key that they have in common is that they want to run some kind of application with all their dependencies in a confined environment (a.k.a the containers).

Different runtimes and systems have emerged over the recent years:

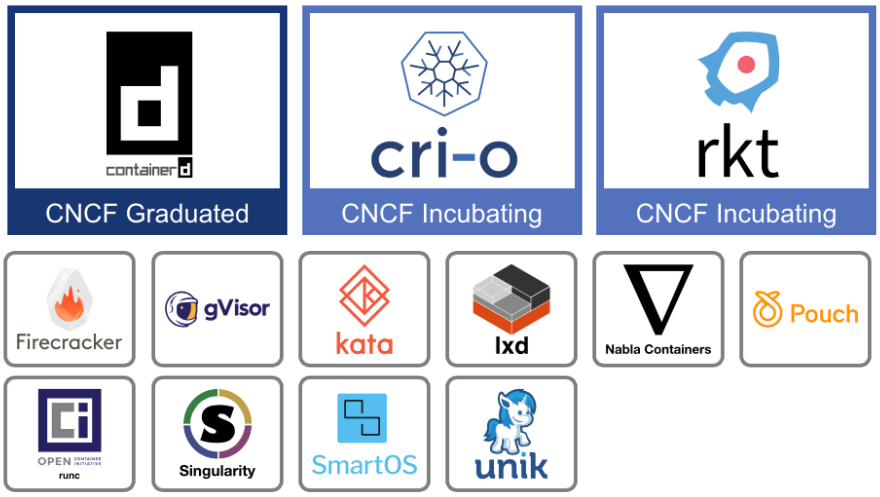

Figure 3: Runtimes landscape.

Where this thesis has been builded with the runtime of "LXD", as it is intended to provide a kind of full virtual machine "container" that behaves like a normal linux distribution, whereas other systems (such as Docker) are more focused in running applications (ex: running a database service).

2.3 LXC

As we have stated before, the framework developed in this thesis has been constructed in top of "LXC/LXD", which are both open source tools provided by the Linux Containers project.

In reality, LXD is built on top of LXC, so we will explain the two tools separated to have a general idea how their work.

LXC. LXC is a userspace interface for the kernel containment features, according to [2]. It provides a powerful API and simple tools to manage system or applications containers.

It combines namespaces and cgroups, along as other security mechanisms to provide isolated environments and contain processes.

It is formed basically by:

- C library (liblxc).
- Several languages bindings.
- Set of tools for controlling containers.
- Distribution templates.

It offers some programs for managing containers:

- Creating a container with an Ubuntu template:

```
[host] # lxc-create -n mycontainer -t ubuntu
```

- Run a command inside the container:

```
[host] # lxc-attach -n webserver -- ifconfig eth1 192.168.1.2/24
```

Where we can customize the containers in different ways such as:

- Attaching devices.
- Configure bridges, hardware addresses, network configurations ...
- Migrate containers from one host to other host.
- Set up unprivileged containers.

2.4 LXD

LXD. LXD is a tool written in Go, defined as a system container manager which offers a user experience similar to virtual machines but using Linux containers instead, according to [3].

Is composed basically by:

- A REST API over a local unix socket as well as over the network.

- A client, provided by a new command line tool "lxc", which talks with the REST API.

so we are able to manage the containers by a REST API in a flexible and composable way.

It has also different integrations with container services along other advanced features.

It is not a rewrite of the previous tool (LXC) but a tool builded on top of it through liblxc and the Go bindings.

Some examples for interacting with containers:

- Creating a container with an Ubuntu template:

```
[host] # lxc launch ubuntu:20.04 box
```

- Obtain a shell inside the container named box:

```
[host] # lxc exec box bash
```

- Create a proxy device connecting container port 80 with host port 80:

```
[host] # lxc config device add box testport80 listen=tcp:0.0.0.0:80
↪  connect=tcp:127.0.0.1:80
```

- Shared a host folder with the container test:

```
[host] # lxc config device add box devicewww disk source=/wwwdata
↪  path=/var/www/html
```

3 Methodology / project development

In order to construct our framework we had to develop a set of tools. Basically we developed two commands and a minimal web application:

- **lxce**.
- **lxce-admin**.
- **web-admin**.

This chapter will provide with the technical implementation of each tool and how they are constructed and organized.

3.1 lxce

The first tool developed in this thesis is what we have called "lxce".

It is basically a command line tool coded in Typescript built on top of the "lxc" command line tool with the idea of improving the management and set up of the containers.

We could already work only with the "lxc" tool but the problem is that in order to have a properly set up container we would have to do the following steps, for every container:

- Create the container with linux image specified:

```
[host]# lxc launch ubuntu:20.04 container
```

- Configure password inside container for user ubuntu:

```
[host]# lxc exec container -- bash -c "ubuntu:1234 | chpasswd"
```

- Set up shared folders between host and container:

```
[host]# lxc config device add containers myfolder disk
↪  source=/www/data path=/data
```

- Set up a proxy, connecting host port 4000 with container port 80:

```
[host]# lxc config device add myproxy proxy listen=tcp:0.0.0.0:4000
↪  connect=tcp:10.1.2.1:80
```

Then, if we would like to access the containers by ssh or vnc we would have to create also the corresponding configuration files.

Everything is managed individually, which is good for a basic set up, but for situations where we are working with +50 containers is unmanageable.

So the idea of this command is to resolve such limitations with a command which could:

- Manage containers by configuration files, with a default configuration file.
- Organize containers by "domains".
- Be able to reference containers by aliases.
- Configure proxies and shared locations with a configuration file.
- Generate SSH and VNC configuration files to be distributed.

The architecture of the command line tool is the following:

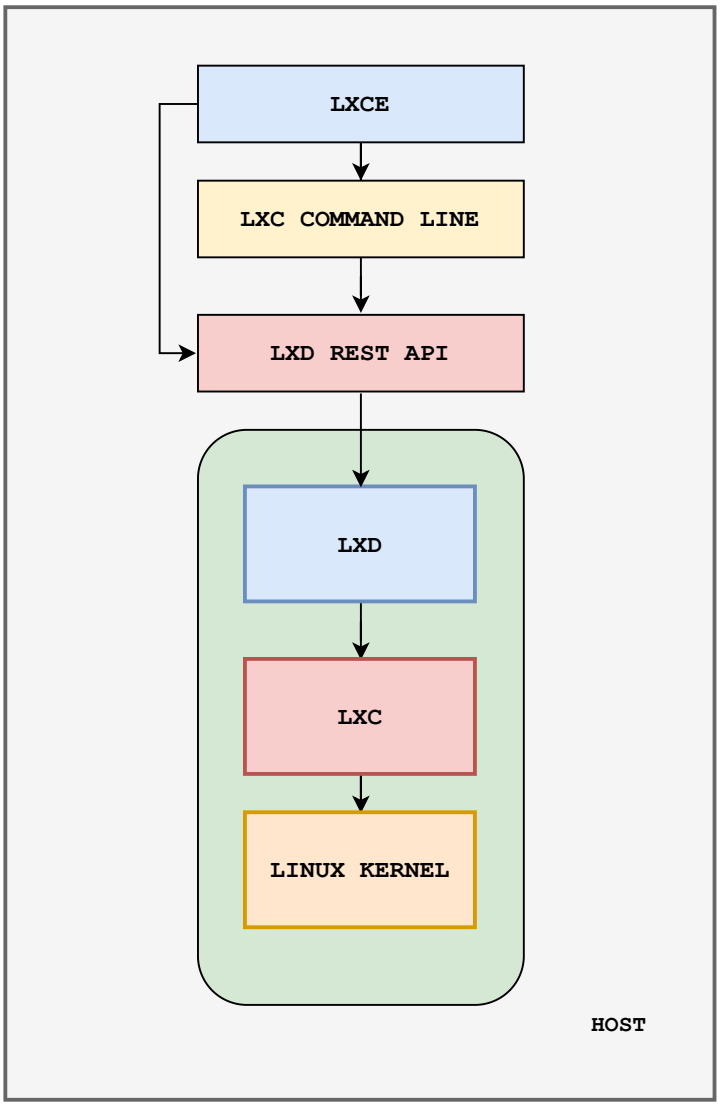

Figure 4: lxce architecture.

Once defined all the specifications for the command, we will explain how are the configuration files organized and the list of subcommands implemented.

3.1.1 Configuration files

Our command uses a series of configuration files for defining a list of properties (general and specific for each container). These configuration files are used by "lxce" and are updated acordingly.

The list of configuration files is the following:

- **container-default.conf**: default configuration file. Defines the default container

configuration template.

- **lxce.conf**: general command configuration. Defines properties such as the hostname of host.
- **individual container configuration files**: they follow the container based template and are updated based on their properties.
- **remmina**: defines a configuration file specific for a VNC client (Remmina [4]).
- **ssh**: ssh-config specific files for each container.

* see Appendix B for a further documentation of the configuration files

3.1.2 Commands

For the commands that are available for our command, we have develop the following commands:

- **lxce init**: initializes the command (configuration files and folder structure).
- **lxce alias**: allow us to define custom names for the containers, as the container names are random.
- **lxce delete**: deletes containers and configurations/folders related.
- **lxce launch**: launch containers with folders/proxies/permissions configured according to the configuration files.
- **lxce list**: output a table of the current containers and their properties.
- **lxce pass**: computes password of each container. They are all generated by a common seed that is stored in the main configuration file.
- **lxce proxy**: configures the proxies associated to the containers.
- **lxce rebase**: allow us to change a container base linux distribution without modifying the container properties.
- **lxce show**: outputs the container configuration files.
- **lxce start**: start containers in a group or individually.
- **lxce stop**: same as the start subcommand.
- **lxce uninstall**: removes all the configuration files and container running in the host.

* see Appendix A for a complete description of each command

3.2 lxce-admin

The second command implemented is intended to be used as an administration tool for managing the hosts with "lxce" installed.

The idea is to have a central host with remote access to a list of hosts with the command line tool installed "lxce" in order to synchronize all configuration files from all the available hosts.

Because with all the configurations files in a centralized location we have:

- Complete view of all the containers across different hosts.
- Access to configuration files for SSH and VNC services.
- Ability to compute password for remote access to containers.

The synchronization is done using a sync tool (rsync[5]) that enable us to have synchronized folders between different hosts.

We can see how would look like in the following figure:

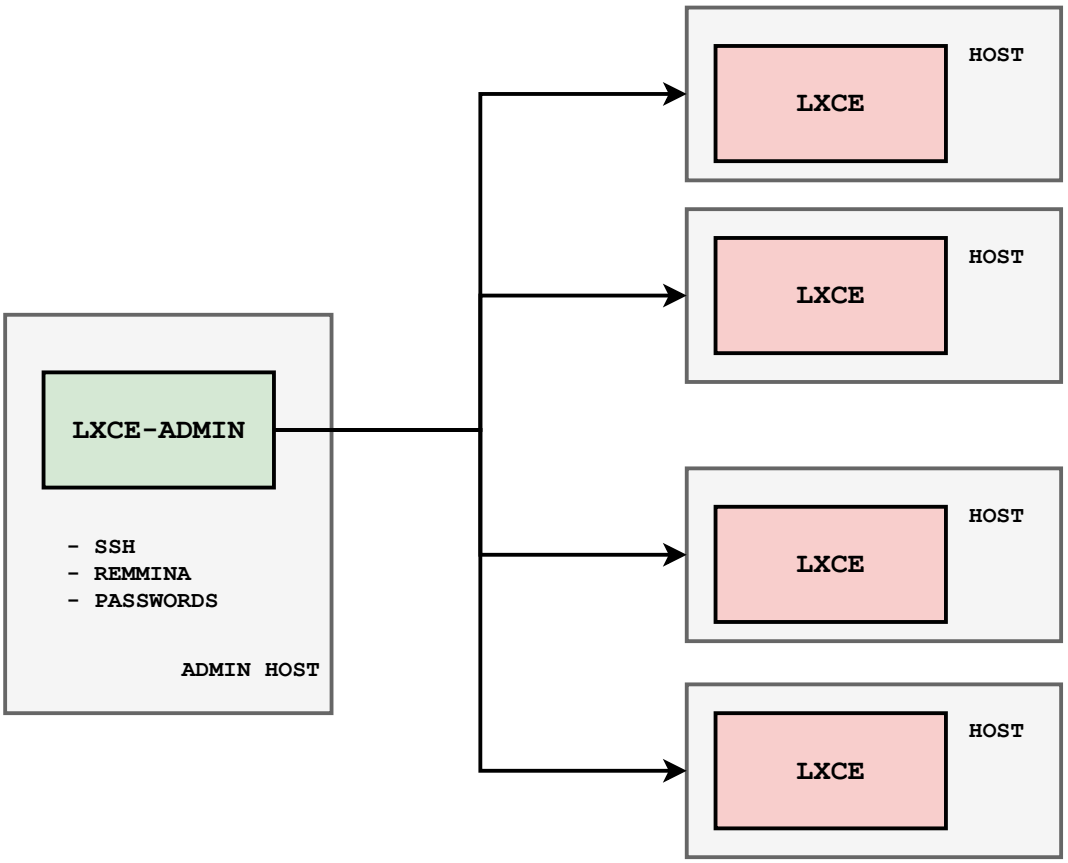

Figure 5: lxce-admin architecture.

3.2.1 Configuration files

The files that must be synchronized are mainly:

- SSH: ssh-config files to be distributed and used by the admin host.
- VNC: remmina configuration files to be used for the admin host and to be distributed.

in order to be able to use the command ssh correctly and have the automatic vnc configurations for the remmina VNC program (the command will also configure the passwords to be used along remmina).

3.2.2 Commands

The subcommands we have develop for the "lxce-admin" tool are:

- **lxce-admin config add**: configures a new host with lxce installed and syncronized all the configuration files for first time.
- **lxce-admin config list**: list a table with the hosts configured and the total number of domains and containers.
- **lxce-admin config remove**: removes a configured host and it's configuration files.
- **lxce-admin config update**: synchronized configuration files from specific host.
- **lxce-admin pass**: computes password for containers in specific host.
- **lxce-admin remmina**: init remmina client into container.
- **lxce-admin vnc**: starts a vnc connection with standard vnc client.

* see Appendix C for a complete description of each command

3.3 web-admin

The last tool implemented consist of a web application builded with React (framework of javascript [6]) along with a minimal server providing a REST API in each host with "lxce" installed.

It is basically a web front-end for our framework that enables to view all our hosts and containers in a detailed view in real time.

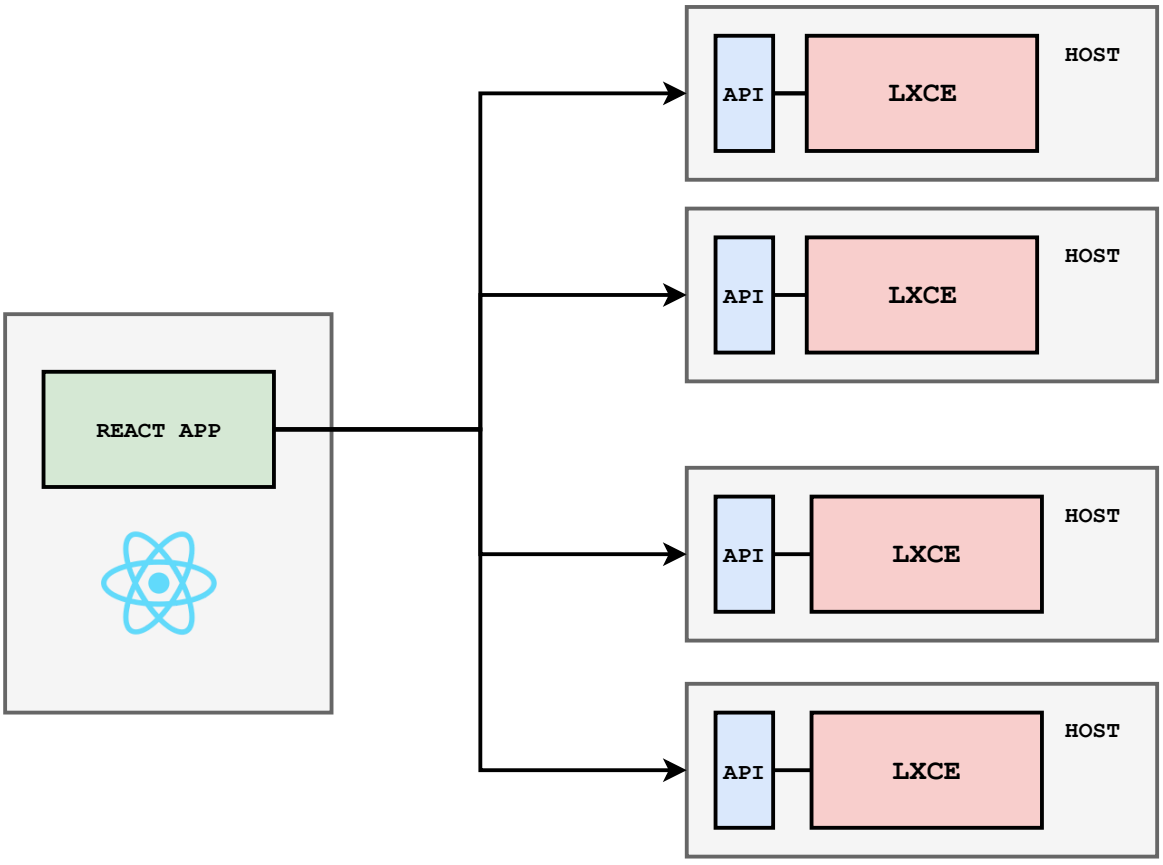

Figure 6: web-admin architecture.

The web application will mainly consult each host with HTTP requests to the API, and based on the responses will construct the view of the application.

It has only been implemented the view of the containers, but the idea of the web application is to be able to manage of all the "lxce" commands through the API provided and offer a web alternative for the "lxce-admin" command line tool.

The API is provided by this simple express server:

```
const express = require("express")
const child = require("child_process")
const cors = require("cors")

const app = express()
const PORT = process.argv[2]

app.use(cors())

app.get("/containers", (req, res) => {
    const response = child.execSync("lxce list -f json").toString()

    res.setHeader('Content-Type', 'application/json');
    res.send(response)
})

app.listen(PORT, () => {
    console.log(`[*] Server listening on port ${PORT}`)
})
```

Listing 1: Express server.

which mainly listens in a specific port and exposes a single API endpoint.

4 Implementation and results

In this chapter we will explain the different use cases that our framework provides along with the programs captures of the tools explained in chapter 3.

We will explain one workflow for each tool.

4.1 lxce

For the first workflow, we will explain how to initialize the command and manage some containers configurations.

In specific we will:

- Initialize the command.
- Create some containers.
- Change linux distributions for containers.
- Delete some containers.
- Add and delete proxies on containers.
- Delete the command and configurations.

Initialize the command The first thing we have to do is initialize the command in order to generate the default configurations files and select different parameters.

```
root@oscar-vm: # lxce init
? lxce.conf: Select hypervisor hostname: localhost
? lxce.conf: Select ssh suffix: oscar-vm
? lxce.conf: Select vnc server: localhost
? lxce.conf: Select vnc port: 5901
? lxce.conf: Select data location [full path]: /datasdd
? Want to add another data location (just hit enter for YES)? No
? container.default: Select containers base: ubuntu:20.04
? container.default: Select default container location: /datasdd
[] Good!
root@oscar-vm: #
```

Listing 2: Init lxce command.

Create containers Then we can create 3 containers with different alias in the domain test with:

```
root@oscar-vm: # lxce launch -r 3 -d test -a alice bob peter
[*] ------------------------------------------------------------
[*] Checking ...
[*] Initialized
[*] Initialized: ok!
[*] Access
[*] Access: ok!
[*] Checks: ok!
[*] ------------------------------------------------------------
[*] Launching container with managing-harlequin
[**] launching ...
[**] waiting for container...
[**] Getting user
[**] Getting user: ubuntu !!
[**] Password created: fa89a2eaca
[**] launching: ok!
[**] creating configurations
[**] creating configurations: ok!
[**] read only directories
[**] added data-test shared folder
[**] added data-managing-harlequin shared folder
[**] read only directories: ok!
[**] adding proxies
[**] added proxy-ssh
[**] added proxy-test
[**] adding proxies: ok!
[**] dns resolution: managing-harlequin.lxd -> 10.10.1.171
[] Launching container with managing-harlequin
[*] Launching container with excited-amethyst
[] Launching container with excited-amethyst
[*] Launching container with coloured-purple
[] Launching container with coloured-purple
[*] ------------------------------------------------------------
[*] Success!!
```

Listing 3: Launch 3 containers.

Where we can see the containers created, along with their properties, with:

```
root@oscar-vm:~# lxce list
+--------------------+-------+--------+---------+-------------+--------------------------+
|        NAME        | ALIAS | DOMAIN | STATUS  |    IPV4     |          PORTS           |
+--------------------+-------+--------+---------+-------------+--------------------------+
|  coloured-purple   | peter |  test  | Running | 10.10.0.241 |  22/tcp -> 0.0.0.0:11020 |
|                    |       |        |         |             | 3000/tcp -> 0.0.0.0:11021 |
|                    |       |        |         |             |                          |
+--------------------+-------+--------+---------+-------------+--------------------------+
|  excited-amethyst  |  bob  |  test  | Running | 10.10.1.41  |  22/tcp -> 0.0.0.0:11010 |
|                    |       |        |         |             | 3000/tcp -> 0.0.0.0:11011 |
|                    |       |        |         |             |                          |
+--------------------+-------+--------+---------+-------------+--------------------------+
| managing-harlequin | alice |  test  | Running | 10.10.1.171 |  22/tcp -> 0.0.0.0:11000 |
|                    |       |        |         |             | 3000/tcp -> 0.0.0.0:11001 |
|                    |       |        |         |             |                          |
+--------------------+-------+--------+---------+-------------+--------------------------+
```

Figure 7: lxce list.

Change container base We have set up all the containers to be run with an ubuntu:20.04 base, but if we we would like to change one container (peter for example) to use ubuntu:18.04 instead we could do it by:

```
root@oscar-vm: # lxce rebase -d test -a peter -b ubuntu:18.04
? Do you want to rebase coloured-purple container within test with
 ↪  ubuntu:18.04? Yes
[*] Rebasing coloured-purple
[**] Removing coloured-purple
[**] launching container with base: ubuntu:18.04 ...
[**] waiting for container
[**] Getting user
[**] Getting user: ubuntu !!
[**] added proxy-ssh
[**] added proxy-test
[**] added data-ubuntu
[**] added data-test
[**] dns resolution: coloured-purple.lxd -> 10.10.0.168
[] Rebasing coloured-purple
```

Listing 4: lxce rebase.

where all the properties of the container will remain the same.

So then we would have the following:

```
root@oscar-vm: # lxce list -c nadb
+--------------------+-------+--------+--------------+
|        NAME        | ALIAS | DOMAIN |     BASE     |
+--------------------+-------+--------+--------------+
|  coloured-purple   | peter |  test  | ubuntu:18.04 |
+--------------------+-------+--------+--------------+
|  excited-amethyst  |  bob  |  test  | ubuntu:20.04 |
+--------------------+-------+--------+--------------+
| managing-harlequin | alice |  test  | ubuntu:20.04 |
+--------------------+-------+--------+--------------+
```

Listing 5: lxce list custom.

Delete containers Now if we want to delete a specific container, it's configuration and shared folder:

```
root@oscar-vm: # lxce delete -d test -a alice
[*] Init: ok!
[*] Permission checked
? Do you want to delete managing-harlequin? Yes
[**] Removing managing-harlequin
```

Listing 6: lxce delete.

Uninstall command Finally, if we want to uninstall the command (i.e: remove all containers, configurations files and shared locations folders) we simply:

```
root@oscar-vm: # lxce uninstall
[*] Init: ok!
[*] Permission checked
? Do you want to uninstall the lxce command and all it's configurations?
 ↪  Yes
[*] Deleting and stopping current containers
[**] Removing coloured-purple
[**] Removing excited-amethyst
[*] Delete /etc/lxce/
```

Listing 7: Uninstall lxce command.

4.2 lxce-admin

The second workflow will consist in how to use the "lxce-admin" tool to manage and existing host with the lxce command installed.

For this example we will do it everything in local but the same applies for external machines with remote access.

But before starting typing commands in the admin host, we must set up the following in each of the hosts with lxce installed:

- Install lxce.
- Init lxce and configure container bases with graphical support for enabling VNC access.
- Configure public key access to host.
- Set up a localhost VNC server listening according to the lxce configuration file.

Add host The first thing that we must do is to add a remote host:

```
root@oscar-vm: # lxce-admin config add
? Select host (ssh config): oscar-vm
? Select hostname: localhost
? Select ssh port: 22
? Select private key location [full path]:
 ↪  /home/oscar/.ssh/localhost_oscar
[*] Updating files
[**] Updating passwords
[*] Updating files: ok
```

Listing 8: Add host lxce-admin.

```
root@oscar-vm: # lxce-admin config list
+----------+---------+------------+
|   HOST   | DOMAINS | CONTAINERS |
+----------+---------+------------+
| oscar-vm |    1    |     3      |
+----------+---------+------------+
```

Listing 9: lxce-admin config list.

Test SSH Once set up the host, we have already access to the ssh configuration file of each container.

We can test it by ssh [host.domain.containerName/containerAlias]:

```
root@oscar-vm: # ssh oscar-vm.google.itchy-bronze
ubuntu@192.168.122.118's password:

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@itchy-bronze:~$
```

Listing 10: ssh container.

VNC Another service that is available is VNC access to every container.

For connecting to the container through VNC we can use two methods:

- **lxce-admin vnc**: will open a normal VNC client:

```
root@oscar-vm: # lxce-admin vnc --host oscar-vm -d google -n
↪  real-black --scale 1
```

- **lxce-admin remmina**: will open remmina (VNC client). The advantage is that remmina is able to use system passwords saved in the computer chain generated by the command, so we won't have to type the ssh password not the vnc password:

```
root@oscar-vm: # lxce-admin remmina --host oscar-vm -d google -n
↪  real-black
```

4.3 web-admin

For configuring the web-admin web app we should:

- Start the web application in an admin host.
- Start the express server in each of the hosts with lxce installed in order to serve the corresponding API.

Once everything is set up, the main page of the web application is the following:

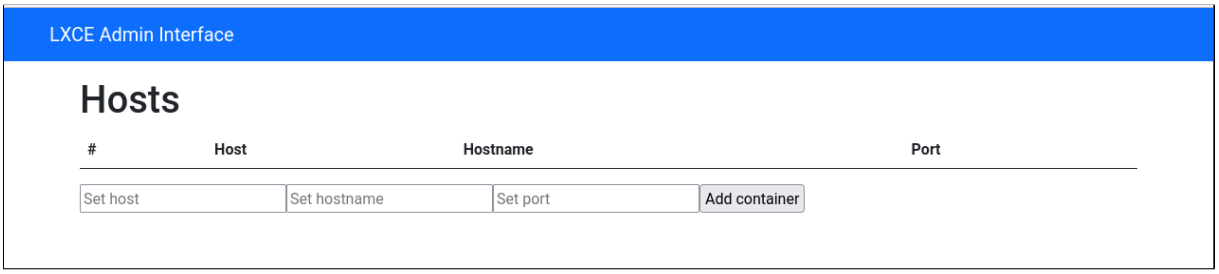

Figure 8: Web-admin principal view.

Where we can add a host:

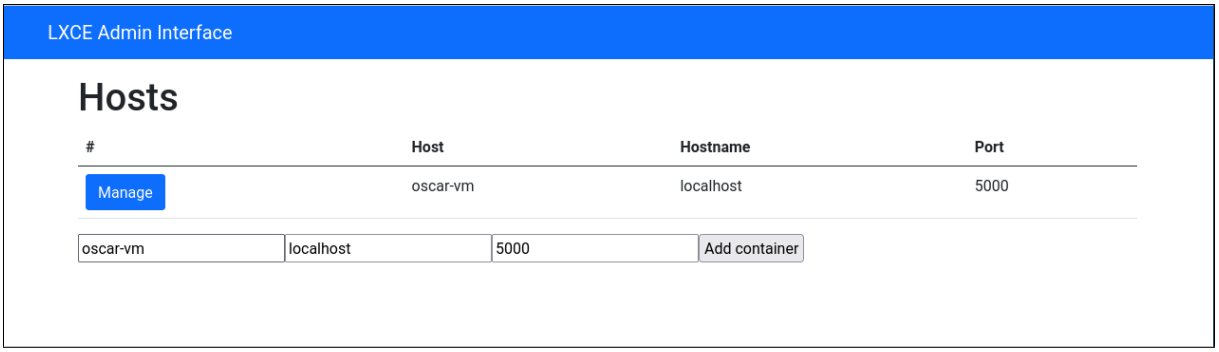

Figure 9: Adding host in the web application.

The action of adding a host in the web application will cause a request to the corresponding host the list of current containers running in the system. This is done by a simply GET request to an API which has the following structure:

```
http --json GET localhost:5000/containers
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: keep-alive
Content-Length: 824
Content-Type: application/json; charset=utf-8
Date: Mon, 14 Jun 2021 18:12:46 GMT
ETag: W/"338-Zebo3shS16lkGZNcHRE+1oMR2Oc"
Keep-Alive: timeout=5
X-Powered-By: Express
```

```
{
    "hostname": "oscar-vm",
    "ip": "localhost",
    "status": "running",
    "containers": [
        {
            "alias": "",
            "base": "ubuntu:20.04",
            "cpu": "23.54 (s)",
            "domain": "default",
            "ipv4": "10.10.1.201",
            "ipv6": "fd42:7c8c:7fab:4125:216:3eff:fe34:89d4",
            "name": "voiceless-blue",
            "ports": "22:10000-3000:10001-",
            "ram": "110.44 MB",
            "status": "Running",
            "user": "ubuntu"
        },
    ],
}
```

Listing 11: GET /containers from host.

Where it is basically a json will all the properties of the current containers of a host. This json response is then saved in the internal state of the application and is continuously updated with the last information of each host.

Then whenever we open the view of a specific host we get a table page of the current containers. For the moment only the view is implemented, but we could add the management of the containers in the current table:

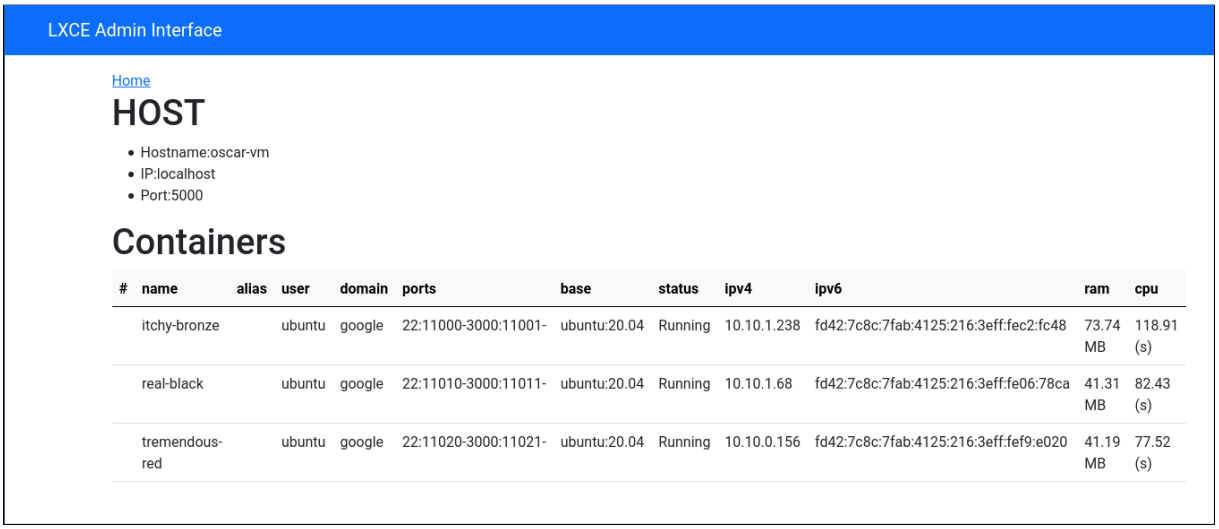

Figure 10: View of the current containers in host.

where the different tables with the containers will be updated each time we open the corresponding page.

5 Budget

For the budget of this project, we should include:

- Single developer role. Total time of 600h at 12€/h.
- Double role for supervisor. Total time of 150h at 15€/h.
- Single laptop for development. Cost at 1000€.
- Server for development. Cost at 5000€.

All sum ups a total of **17.700€** for the project.

6 Conclusions

In this thesis we have achieved the following accomplishments:

- Improvement of the "lxce" command line tool in a more improved, tested version.
- Added functionalities to the "lxce-admin" tool improving the management of the containers.
- Complete build of an initial web application with a view for improvements and extensibility.

The main limitations were working and learning a new technology ("containerization") not known at the beginning of the project, plus developing the set of tools in a programming language (javascript/typescript) without any previous experience. Also for development of the web application learning about React was needed.

Also, some systems administration research was needed for setting up the development environment.

7 Future work

The next steps for this project would involve mainly:

- Integrate into the "lxce" command a new functionality involving web services in a way in we could define a web proxy and expose a service inside the container by a configuration file.
- Improve the web application integrating all the "lxce" commands along with extending the current API.

References

[1] Linux containers project. https://linuxcontainers.org/.

[2] lxc: container tool utility. https://linuxcontainers.org/lxc/introduction/.

[3] lxd: container tool utility. https://linuxcontainers.org/lxd/introduction/.

[4] Remmina access screeen and file sharing. https://remmina.org/.

[5] Rsync tool. https://rsync.samba.org/.

[6] React: javascript framework. https://reactjs.org/.

[7] Linux containers project. https://linuxcontainers.org/.

[8] Virtualization. https://en.wikipedia.org/wiki/Virtualization.

[9] Redux: State container. https://redux.js.org/.

[10] Information and history of containers. https://github.com/saschagrunert/demystifying-containers.

[11] Red hat containers. https://www.redhat.com/en/topics/containers/whats-a-linux-container.

Appendices

A lxce

For the commands that are available for our command, we have the following structure:

```
Usage: lxce [command] <options> <flags>

Commands:
  lxce alias       Manage containers aliases
  lxce completion  Output completions scripts
  lxce delete      Delete containers and configurations/folders related
  lxce init        Initialize lxce command
  lxce launch      Launch containers
  lxce list        List containers properties
  lxce pass        Compute password from containers
  lxce proxy       Delete and restart proxies
  lxce rebase      Relaunch container with new base specified
  lxce show        Show containers configurations files
  lxce start       Start containers
  lxce stop        Stop containers
  lxce uninstall   Remove all configurations from the lxce command

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose
```

lxce alias

```
Usage: lxce alias [command] <options> <flags>

Commands:
  lxce alias set    set container alias
  lxce alias unset  unset container alias
  lxce alias check  check container alias

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose
```

lxce alias set

```
Usage: lxce alias set [options] <flags>

Options
  -d, --domain  container domain
  -n, --name    container name
  -a, --alias   new container alias

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose

Examples:
  lxce alias set -d google                Set alias alice to container
  -n front -a alice                       front within google domain
```

lxce alias unset

```
Usage: lxce alias unset [options] <flags>

Options
  -d, --domain  container domain
  -n, --name    container name
  -a, --alias   new container alias

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose

Examples:
  lxce alias unset -d google -n front  Unset alias to container front
                                       within google domain
  lxce alias unset -d google -a alice  Unset alias to container with
                                       alice alias within google
                                       domain
```

lxce alias check

```
Usage: lxce alias check [options] <flags>

Options
  -d, --domain  container domain
  -a, --alias   new container alias
  -f, --format  output format            ["plain", "json", "csv"]

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose

Examples:
  lxce alias check -d google -a alice  check alice alias existence
```

```
                                        within google domain
```

lxce delete

```
Usage: lxce delete <options> <flags>

Options
  -g, --global  apply to all containers
  -d, --domain  domain name for a group of containers
  -n, --name    container name
  -a, --alias   container alias
  -y, --yes     yes to questions

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose

Examples:
  lxce delete --global                  Deletes all containers and
                                        configurations related
  lxce delete -d google                 Deletes all containers within
                                        google domain
  lxce delete -d google -n still-yellow  Deletes container referenced
                                        by name
  lxce delete -d google -a alice        Deletes container referenced
                                        by alias
```

lxce init

```
Usage: lxce init <flags>

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose
```

lxce launch

```
Usage: lxce launch <options> <flags>

Options
  -d, --domain   domain for the container/containers
  -r, --range    range of container (ex: -r 5)
  -n, --names    names/name of the containers/container
  -a, --aliases  aliases/alias of the containers/container

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose

Examples:
  lxce launch -d google                     Launch one container within
                                            google with a random name
  lxce launch -d google -r 3                Launch three containers
                                            within google with
                                            random names
  lxce launch -d google -r 3 -n back front  Launch three containers
  base                                      within google with
                                            specified names
  lxce launch -d google -r 3 -n back front  Launch three containers with
  base -a alice bob eve                     name and alias
                                            specified
  lxce launch -d google -r 3 -a alice bob   Launch three containers
  eve                                       with random names and
                                            alias specified
```

lxce list

```
Usage: lxce <options> <flags>

Format options
==============
-n: "name"
-a: "alias"
-u: "user"
-b: "base"
-r: "ram (MB)"
-p: "ports"
-4: "ipv4"
-6: "ipv6"
-s: "status"
-d: "domain"
-c: "cpu usage (s)"

Options
  -c, --columns  Values to show
  -f, --format   Output format

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose

Examples:
  lxce list -c naubr
  lxce list -f json
```

lxce pass

```
Usage: lxce pass <options> <flags>

Options
  -g, --global  Apply to all containers
  -d, --domain  Domain name for a group of containers
  -n, --name    Container name
  -a, --alias   Container alias
  -p, --plain   plain output

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose

Examples:
  lxce pass --global            Compute all container passwords
  lxce pass --domain google     Compute all domain passwords
  lxce pass -d google -n front  Compute container name password
  lxce pass -d google -a alice  Compute container alias password
```

lxce proxy

```
Usage: lxce proxy <options> <flags>

Options
  -g, --global  Apply to all containers
  -d, --domain  Domain name for a group of containers
  -n, --name    Container name
  -a, --alias   Container alias

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose

Examples:
  lxce proxy --global            Restart all containers proxies based
                                 on their configuration files
  lxce proxy -d google           Restart all domain containers proxies
                                 based on their configuration files
  lxce proxy -d google -n front  Restart container proxies
  lxce proxy -d google -a alice  Restart container proxies
```

lxce rebase

```
Usage: lxce rebase <options> <flags>

Options
  -g, --global  Applied to all containers
  -d, --domain  Domain name for a group of containers
  -n, --name    Container name
  -a, --alias   Container alias
  -b, --base    Container base

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose

Examples:
  lxce rebase --global                  Applies new base to existing
                                        containers and future ones
  lxce rebase -d google                 Applies new base to all
                                        containers withing
                                        google domain
  lxce rebase -d google -n still-yellow  Applies new base to container
  lxce rebase -d google -a alice         Applies new base to container
```

lxce show

```
Usage: lxce show <options> <flags>

Options
  -g, --global  Apply to all containers
  -d, --domain  Domain name for a group of containers
  -n, --name    Container name
  -a, --alias   Container alias
  -e, --extra   Show extra information

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose

Examples:
  lxce show --global                   Show all containers configurations
  lxce show -d google                  Show all containers configurations
                                       within domain
  lxce show -d google -n still-yellow  Show container configurations
                                       defined by name
  lxce show -d google -a alice         Stop container configuration
                                       defined by alias
```

lxce start

```
Usage: lxce start <options> <flags>

Options
  -g, --global  Apply to all containers
  -d, --domain  Domain name for a group of containers
  -n, --name    Container name
  -a, --alias   Container alias

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose

Examples:
  lxce start --global                   Start all containers
  lxce start -d google                  Start all container within domain
  lxce start -d google -n still-yellow  Start container defined by name
  lxce start -d google -a alice         Start container defined by alias
```

lxce stop

```
Usage: lxce stop <options> <flags>

Options
  -g, --global  Apply to all containers
  -d, --domain  Domain name for a group of containers
  -n, --name    Container name
  -a, --alias   Container alias

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose

Examples:
  lxce stop --global                   Stop all containers
  lxce stop -d google                  Stop all container within domain
  lxce stop -d google -n still-yellow  Stop container defined by name
  lxce stop -d google -a alice         Stop container defined by alias
```

lxce uninstall

```
lxce uninstall <options> <flags>

Options
  -y, --yes

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose
```

B lxce configuration files

The "lxce" configuration files structure is the following one:

```
/etc/lxce
|--- container.conf.d                    # Container configurations files
|   |--- default                         # Default domain configurations
|   |   '--- voiceless-blue
|   '--- derecho
|       '--- relieved-beige
|--- container_default.conf              # Default container template
|--- lxce.conf                           # lxce command configuration
|--- remmina                             # remmina configurations files
|   |--- default
|   |   '--- oscar-vm.default.voiceless-blue.remmina
|   '--- derecho
|       '--- oscar-vm.derecho.relieved-beige.remmina
'--- ssh                                 # ssh-config files
    |--- default
    |   '--- voiceless-blue.conf
    '--- derecho
        '--- relieved-beige.conf
```

Where the configurations files content is the following:

- **container-default.conf**

  This file acts as a template for every container to be created.

```
{
  "name": "",
  "alias": "",
  "user": "",
  "id_domain": 0,
  "id_container": 0,

  "domain": "default",
  "base": "ubuntu:20.04",
  "userData": "/datasdd",

  "proxies": [
    {
      "name": "ssh",
      "type": "tcp",
      "listen": "0.0.0.0",
      "port": 22
    },
    {
      "name": "test",
      "type": "tcp",
      "listen": "0.0.0.0",
      "port": 3000
    }
  ],
}
```

- **lxce.conf**

  This file specifies different parameters of the host where the command is installed, such as:

  - SSH IP
  - Hostname
  - Local VNC server configuration
  - Seed used for generating passwords
  - List of container domains currently in the host
  - List of locations available for the shared containers folders location

```
{
  "hypervisor": {
    "SSH_hostname": "localhost",
    "SSH_suffix": "oscar-vm",
    "VNC_server": "localhost",
    "VNC_port": 5901
  },
  "seed": "4b5a003f0e1715df",
  "domains": [
    {
      "id": 0,
      "name": "default"
    },
    {
      "id": 1,
      "name": "derecho"
    }
  ],
  "locations": [
    "/datasdd"
  ]
}
```

- **container configuration file**

  This files list the configured parameters for each container and the ids that uniquelly identifies it

```
{
  "name": "voiceless-blue",
  "alias": "",
  "user": "ubuntu",
  "id_domain": 0,
  "id_container": 0,
  "domain": "default",
  "base": "ubuntu:20.04",
  "userData": "/datasdd",
  "proxies": [
    {
      "name": "ssh",
      "type": "tcp",
      "listen": "0.0.0.0",
      "port": 22
    },
    {
      "name": "test",
      "type": "tcp",
      "listen": "0.0.0.0",
      "port": 3000
    }
  ],
}
```

- **VNC configuration**

  Remmina configuration file used internally by the tool:

```
[remmina]
ssh_tunnel_privatekey=
name=oscar-vm.default.voiceless-blue          # Container name
ssh_tunnel_passphrase=
password=.                                    # Indicate saved
...                                             password
server=localhost:5901                         # VNC server
disablepasswordstoring=0
ssh_tunnel_username=ubuntu
disableclipboard=0
window_maximize=1
ssh_tunnel_password=.                         # Indicate saved
                                                password
enable-autostart=0
proxy=
ssh_tunnel_server=localhost:10000             # Container SSH Port
ssh_tunnel_auth=0
group=oscar-vm.upc.edu
...
protocol=VNC
username=ubuntu                               # VNC username
showcursor=0
colordepth=32
```

- **SSH configuration**

  Ssh-config files for each container:

```
Host oscar-vm.default.voiceless-blue
   Hostname localhost
   User ubuntu
   Port 10000
   TCPKeepAlive yes
   ServerAliveInterval 300
```

C lxce-admin

lxce-admin config

```
Usage: lxce-admin [command] <flags>

Commands:
  lxce-admin config add     Add host and sync files
  lxce-admin config list    List configured hosts
  lxce-admin config remove  Remove host and associated files
  lxce-admin config update  Update host associated files

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose
```

lxce-admin config add

```
Usage: lxce-admin config add <options> <flags>

Options
      --dry-run

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose
```

lxce-admin config list

```
Usage: lxce-admin config list

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose
```

lxce-admin config remove

```
Usage: lxce-admin config remove <options> <flags>

Options
      --host     configured host
      --dry-run

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose
```

lxce-admin config update

```
Usage: lxce-admin config update <options> <flags>

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose
```

lxce-admin pass

```
Usage: lxce-admin pass <options> <flags>

Options
      --host    configured host
  -d, --domain  container domain
  -n, --name    container name
  -a, --alias   container alias

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose
```

lxce-admin remmina

```
Usage: lxce-admin remmina <options> <flags>

Options
      --host    configured host
  -d, --domain  container domain
  -n, --name    container name
  -a, --alias   container alias

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose
```

lxce-admin vnc

```
Usage: lxce-admin vnc <options> <flags>

Options
      --host     configured host
  -d, --domain   container domain
  -n, --name     container name
  -a, --alias    container alias
      --scale    scale vnc viewer
      --dry-run

Flags
      --version  Show version number
  -h, --help     Show help
  -v, --verbose
```