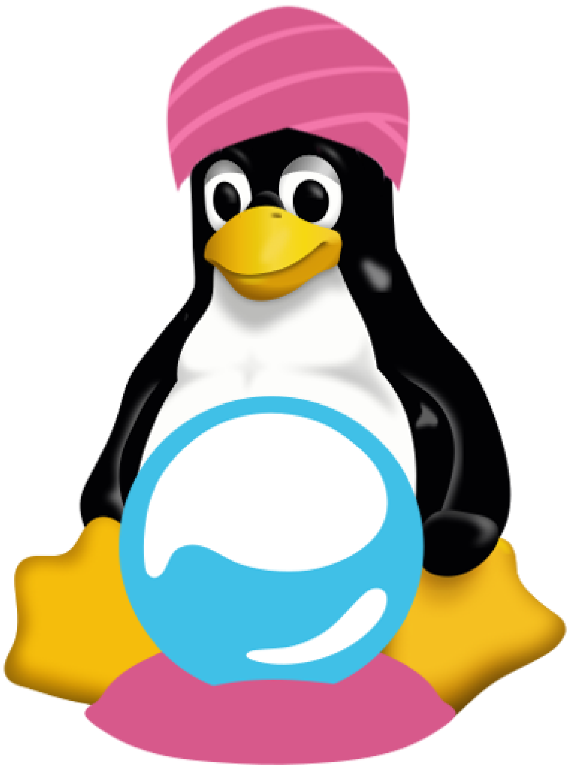
Projet TuxML
User Manual
ISTIC - Université de Rennes 1

Valentin PETIT Julien ROYON CHALENDARD
Cyril HAMON Paul SAFFRAY Michaël PICARD
Malo POLES Luis THOMAS Alexis BONNET

Encadrés par Mathieu ACHER



Lundi 22 Avril 2019

Ce rapport sera en Anglais, étant donné que son contenu sera aussi disponible sur le dépôt git du projet.

This report will be in English, since its content will also be available on the git project repository.

Table of Contents

1 Introduction

This report aims to provide help to the users of the TuxML scripts and programs, we'll cover which options to use for which use case. While last year's report is still relevant for some utilities (those sections have been copied), others have greatly changed. We also provide tutorials for advanced use cases.

1.1 Requirement

But first let's go over what's necessary on your machine to get you started. Obviously an OS, Windows, MacOS or any flavour of GNU/Linux should do it. Most of the development team was working on GNU/Linux but the others are just fine. If you have any problems don't hesitate to open an issue on our GitHub repository.

1.1.1 Python

The project is almost entirely written in Python so you'll need that. But not just any version, Python 3 and at least version 3.5. In order to check whether or not python is installed: open a command prompt or terminal (depending on your OS) and type "python –version" if it returns an error or a version number lower than 3.5 you'll need to update your Python install or even install it.

On Windows and MacOS go here: https://www.python.org/downloads/ on GNU/Linux it will depend on your distribution, if you're on a Debian-like system just type as root (or sudo) "apt install python3".

1.1.2 Docker

The other absolutely needed component is Docker. Docker is a tool that allows us to create a stable environment in order to compile the Linux Kernel. That way even if we're not all on the same machine we have the same versions of the tools, what's especially important is that we have the same compiler versions, otherwise we won't get the same results (We tested different versions of GCC).

Now if you don't have docker on your machine head over to:
https://www.docker.com/products/docker-desktop if you're on Windows or MacOS. For Linux users either your distribution supports a version in its repositories, or you'll have to install it manually, here is the procedure for Debian users: https://docs.docker.com/install/linux/docker-ce/debian/, follow that tutorial and you should be good to go.

2 User entry point : kernel_generator.py

2.1 Goals and functionalities

This standalone script is a way for the user to use this project with ease, without any need to go through all the project and understand everything.

This script provides to the user a simpler usage of our docker image, by managing all by himself the fetch and building, providing also a way to use different versions of the Linux kernel (this part is still in development and we can't assure that every single Linux kernel version 4.x.x will work at the moment), while also providing a way to use the stable (prod) or the latest (dev) version of the image.

This script also gives an easy way to the user to test a specific Linux config file and fetch its compilation logs, to compile a whole bunch of random config or to simply check if the project should not crash on your system.

Let's go over on how to do it.

2.2 How to use it

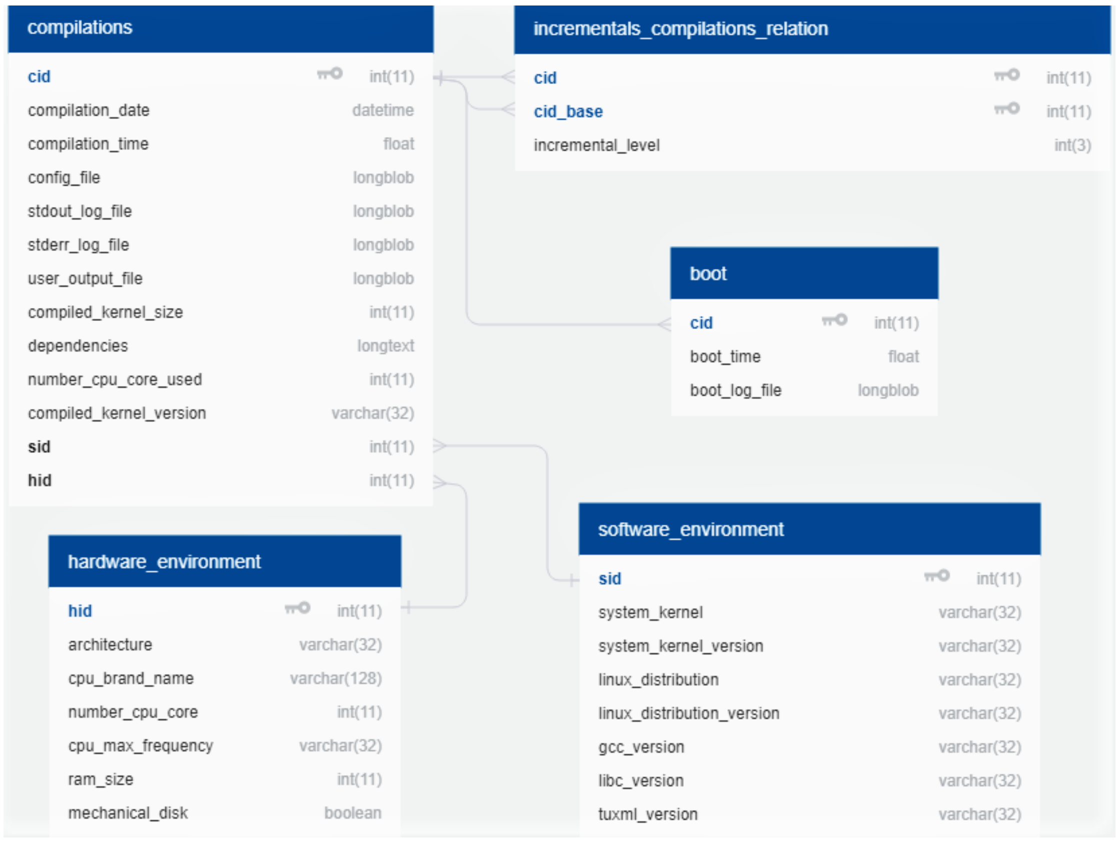
```
$ ./kernel_generator.py --help
usage: kernel_generator.py [-h] [--dev] [--local] [--tiny] [--config CONFIG]
                           [--linux4_version LINUX4_VERSION] [--logs LOGS]
                           [-s] [--unit_testing] [-n NUMBER_CPU]
                           [nbcontainer] [incremental]

positional arguments:
  nbcontainer           Provide the number of container to run. Have to be
                        over 0.
  incremental           Optional. Provide the number of additional incremental
                        compilation. Have to be 0 or over.

optional arguments:
  -h, --help            Show this help message and exit
  --dev                 Use the image with dev tag instead of prod's one.
  --local               Don't try to update the image to run, i.e. use the local
                        version.
  --tiny                Use Linux tiny configuration. Incompatible with
                        --config argument.
  --config CONFIG       Give a path to specific configuration file.
                        Incompatible with --tiny argument.
  --linux4_version LINUX4_VERSION
                        Optional. Give a specific linux4 version to compile.
                        Note that it's local, will take some time to download
                        the kernel after compiling, and that the image used to
                        compile it will be deleted afterward.
  --logs LOGS           Optional. Save the logs to the specified path.
  -s, --silent          Prevent printing on standard output when compiling.
                        Will still display the feature warnings.
  --unit_testing        Optional. Run the unit testing of the compilation
                        script. Prevent any compilation to happen. Will
                        disable --tiny, --config, --linux4_version, --silent,
                        --fetch_kernel and incremental feature during runtime.
  -n NUMBER_CPU, --number_cpu NUMBER_CPU
                        Optional. Specify the number of cpu cores to use while
                        compiling. Useful if your computer can't handle the
                        process at full power.
```

Here are some examples to get you started:

- A simple compilation: kernel_generator.py (nothing else !)
- With unit testing : kernel_generator.py –unit-testing
- With a custom .config file: kernel_generator.py –config /.config
- Running 100 compilations on 2 cpu cores: kernel_generator.py 100 -n 2 or kernel_generator.py 100 –number_cpu 2
- Running 4 compilations on 16 cpu cores with kernel version 4.15.1: kernel_generator.py 4 -n 16 –linux4_version 15.1

3 Database : what do we retrieve?

In order to work, the TuxML project needs an access to the internet, and more specifically to the database server holding all the data.



compilations	
cid	int(11)
compilation_date	datetime
compilation_time	float
config_file	longblob
stdout_log_file	longblob
stderr_log_file	longblob
user_output_file	longblob
compiled_kernel_size	int(11)
dependencies	longtext
number_cpu_core_used	int(11)
compiled_kernel_version	varchar(32)
sid	int(11)
hid	int(11)

incrementals_compilations_relation	
cid	int(11)
cid_base	int(11)
incremental_level	int(3)

boot	
cid	int(11)
boot_time	float
boot_log_file	longblob

hardware_environment	
hid	int(11)
architecture	varchar(32)
cpu_brand_name	varchar(128)
number_cpu_core	int(11)
cpu_max_frequency	varchar(32)
ram_size	int(11)
mechanical_disk	boolean

software_environment	
sid	int(11)
system_kernel	varchar(32)
system_kernel_version	varchar(32)
linux_distribution	varchar(32)
linux_distribution_version	varchar(32)
gcc_version	varchar(32)
libc_version	varchar(32)
tuxml_version	varchar(32)

Above is a graph showing those data. As you can see the only things concerning your hardware is your machine's CPU, RAM and Storage type (HDD, SDD ...) in the hardware_environment table. As for the software environment we save on which Os and Kernel version it was done (in the software_environment). The rest of the fields should stay consistent across all rows because those data are gathered inside the container.

Now about the data needed for the machine learning process, all of them are contained in the 3 remaining tables (compilations, boot and incrementals compilations relations), here we save everything related to the time taken to compile the configuration, the config file and even the compressed kernel sizes. All of this will help to predict some aspects of the kernel depending on the configuration.

4 Tutorials

4.1 Compilation inside the docker image tuxml

The goal of this tutorial is to use the docker container in order to build a specific kernel configuration, to do that we need to manipulate the container.

First of all we need to install the docker image locally if it isn't done yet, using this command you can download the image:

```
$ python3 kernel_generator.py --unit_test
```

First we need to run the container based on the tuxml/tuxml:prod, then copy the file, you can use this command to run a container:

```
$ docker run -it tuxml/tuxml:prod
```

(Hint: to list all images use:

```
$ docker images
```

or

```
$ docker image ls
```

Both commands have the same output.)

This will open you a shell inside the container, from that point you could exit the container to start it again then copy then attach. or you could simply open another terminal, list the running containers, and then copy your file in the right one.

- open another terminal
- use $ docker container ls to find your container's id or alias (funny two words names).
- then: $ docker cp <file> <container-id>:/<rest of the path>

So as an example:

```
$ docker cp .config objective_montalcini:/TuxML/linux-4.13.3/.config
```

The other method (one terminal):

- exit the shell with the "exit" command.
- restart the container, to know which one it is use $ docker container ls -a, once you find it use: $ docker restart <container-id>
- then: $ docker cp <file> <container-id>:/<rest of the path>

So as an example:

```
$ docker cp .config objective_montalcini:/TuxML/linux-4.13.3/.config
```

And then do what you have to do, that is installing dependencies through apt or using make on the kernel to see what happens (as an example if you need git: "apt install git").

4.2 Compilation and boot testing with specific options

You may want to test a specific configuration, e.g to study the impact of some options. In order to do this, you can use kernel_generator.py and give it the configuration file that you want to test (use –config followed by the path to the .config file).

Starting from a basic configuration, here is a command to switch an option with bash. To disable an option :

```
sed 's/CONFIG_<option>=y/# CONFIG_<option> is not set/' -i .config
```

To activate a disabled option :

```
sed 's/# CONFIG_<option> is not set/CONFIG_<option>=y/' -i .config
```

Note : this method is only for the study of options and their impact, and you should be careful about the dependancies before any changings. You can find information about an option and its dependencies here : `https://cateee.net/lkddb/web-lkddb/`