Projet TuxML :
Manuel d'utilisation

Corentin Chédotal Gwendal Didot Dorian Dumanget
Antonin Garret Erwan Le Flem Pierre Le Luron
Alexis Le Masle Mickaël Lebreton Fahim Merzouk

Encadrés par Mathieu Acher

4 Mai 2018

Remerciements

L'équipe du projet TuxML souhaiterait remercier monsieur Olivier Barais pour avoir pris le temps de présenter Docker et son utilisation ainsi que pour les pistes de travail concernant son emploi dans le cadre de notre projet.

Le logo du projet utilise des *emojis* modifiés du projet Noto mis à disposition gracieusement sous la licence Apache. Seul l'emoji `1f473 1f3fb` a été modifié avant son inclusion dans le logo afin d'en extraire et recolorer son turban. Puis l'ensemble du logo a été recoloré pour donner un aspect "plan de construction".
Le logo Linux Tux ainsi que le fond font partie du domaine public.

Table des matières

1 Fonctionnement général

1.1 Base de données

Pour fonctionner, TuxML nécessite trois tables :

— Compilations : Il s'agit de la table principale contenant, comme son nom l'indique, toutes les informations relatives à une compilation. Par exemple le temps de compilation (négatif si la compilation a échouée), la date, les fichiers de log, le .config etc.. Ainsi que toutes les information sur environnement : modèle du CPU, quantité de RAM, distribution Linux etc...

— Tests : Cette table va nous permettre de sauvegarder les résultats des tests. Pour l'instant elle n'est constituée que de la date du test, l'identifiant de la compilation et du temps de démarrage (négatif s'il y a eu un problème). Cette table est amenée à évoluer en fonction des tests que l'on effectua.

— Incremental_compilations : Lorsque le mode incrémental est activé on se base sur une première compilation, qui va effacer tous les fichiers issus des précédentes compilations, pour effectuer les compilations suivantes qui, elles, n'effaceront aucun fichier. Cette table nous permet de faire le lien entre la compilation de "base" et les compilations incrémentales. Elle contient donc deux clés étrangères faisant référence à deux entrées de la table "Compilations".

2 Programmes

2.1 Kanalyser

Kanalyser est un programme permettant d'analyser les options disponibles dans le Kconfig d'un kernel linux. Pour une version et une architecture donnée, Kanalyser va extraire toutes ses options ainsi que leurs types et les ajouter dans une base de données pour pouvoir générer les valeurs par défaut des options qui ne sont pas présentes dans les fichiers `.config`.

Le dossier de Kanalyser doit être placé à la racine du kernel cible. Le programme nécessite `python3` et la bibliothèque `mysqlclient`, installable via `pip3` pour pouvoir interagir avec la base de données.

Pour utiliser Kanalyser, il faut d'abord renseigner les informations de la ou les bases de données dans le fichier `DBCredentials.py` avec un utilisateur qui a les permissions pour `INSERT` et `DROP`. Le script `filltypes.sh` peut alors être lancé sans arguments pour remplir automatiquement la base.

2.2 TuxML Project Docker Image Manager : TPDIM

2.2.1 Principe :

TPDIM a pour vocation d'aider les développeurs du projet TuxML à pouvoir créer et utiliser des images Docker facilement, sans avoir à connaître les commandes Docker et sans connaissance de Docker autre que de savoir ce qu'est une image, et l'utilité d'une telle image dans le projet. TPDIM offre des commandes simplifiées par

rapport à celle par défaut de Docker, tout en essayant d'offrir une liberté d'utilisation au moyen d'option de customisation. Actuellement TPDIM ne supporte que les distributions Linux utilisant apt-get.

2.2.2 Commandes :

TPDIM possède 4 commandes, permettant de générer des DockerFile, de construire des images Docker, d'envoyer l'image sur un repository distant et une commande permettant de réaliser les 3 autres à la suite

- Générer : On utilise la commande `TPDIM -g [nom de l'image] -dep dependences.txt -t [tag de l'image]`
- Construire : on utilise la commande `TPDIM -b [nom de l'image] -t [tag de l'image]`
- Envoyer : on utilise la commande `TPDIM -p [nom de l'image] -t [tag de l'image]`
- All : on utilise la commande `TPDIM -a [nom de l'image] -t [tag de l'image]`

Pour les différentes commandes :
- Nom de l'image correspond au nom de la distribution qui doit être utilisée par l'image.
- Tag de l'image correspond au tag appliquée à l'image pour pouvoir l'identifier.

2.3 CSVgen

CSVgen est un programme permettant d'extraire les informations de compilations générées par ProjetIrma et de les placer dans un fichier CSV, dans le but de l'exploiter dans le cadre du machine learning. Un fichier CSV formaté par CSVgen contient les noms des features dans la première ligne (nom des options du `.config`, et informations résultant de la compilation, comme la taille du kernel ou le temps de compilation), et pour chacune des lignes suivante, les valeurs respectives pour chaque sample (compilation).

CSVgen nécessite Python 3 et la bibliothèque `mysqlclient`, installable via `pip3` pour pouvoir interagir avec la base de données

Pour utiliser CSVgen, il faut d'abord renseigner les informations de la ou les bases de données dans le fichier `DBCredentials.py` avec un utilisateur qui a les permissions pour `SELECT`. Des bases de données d'exemple sont déjà données dans ce fichier car elles contiennent plusieurs milliers de compilations déjà effectuées à ce jour. Le script `genCSV.py` peut alors être lancé avec comme argument le nom du fichier CSV cible. Le processus peut prendre du temps en fonction du nombre de samples dans les bases de données.

2.4 `MLfood.py`, le script à lancer

2.4.1 Principe

`MLfood.py` est le script à la base de la compilation sur une machine. C'est ce programme qui se charge de lancer la compilation sur des conteneurs Docker. Dans

le but de remplir une base de données de taille importante, il était nécessaire d'avoir un fichier capable de lancer des compilations en série. `MLfood.py` est donc né de cette idée et mélange une boucle d'exécution de compilation à l'utilisation du logiciel Docker afin que chaque machine, quelque soit le système d'exploitation, puisse effectuer les commandes de compilation et en réduisant le nombre d'incompatibilités possible. Ce script se charge aussi de récupérer quatre fichiers depuis le conteneur Docker avant de le supprimer : *output.log, std.log, err.log* et le *.config* qui à été généré. Ils sont ainsi stockés dans le dossier Logs/ dans un dossier ayant pour nom la date et l'heure de l'exécution.

2.4.2 Utilisation

`MLfood.py` s'exécute dans sa forme la plus basique grâce à la commande :
`./MLfood.py [nombre de compilations]`

Cette commande effectuera le nombre de compilations demandées en série à la suite les unes des autres et indépendamment.
Si aucun paramètre supplémentaire n'est précisé, `-incremental` sera configuré à 0 par défaut. La commande exécuté sera en réalité :

```
./MLfood.py [nombre de compilation] 0
```

MLfood peut accepter plusieurs arguments :

```
MLfood.py [-h] [--no-clean] [--reset-logs] [--dev] nbcompil [incremental]
```

- `--no-clean` permet de ne pas supprimer les conteneurs Dockers à la fin de l'exécution de `MLfood.py`. Par défaut Docker garde en mémoire les dockers ayant terminé de s'exécuter mais ne sont pas supprimés, `MLfood.py` inverse ce comportement en les supprimant systématiquement, excepté si cette option est précisée.
- `--reset-logs`, `MLfood.py` créé des logs dans un dossier local appelé Logs/ et situé dans le dossier courant. Ces logs peuvent finir par s'accumuler et dans le cas où ils ne sont plus nécessaire, cette options permet de vider la totalité du contenu de ce dossier.
- `--dev`, ce paramètre optionnel permet de choisir quelle image Docker utiliser. Par défaut `MLfood.py` utilise une image Docker "stable" appelée *prod* et contenant une version fonctionnelle du projet. Avec l'option `--dev` utilisée, l'image Docker choisie sera l'image *dev* qui est en cours de développement, elle est instable et peut ne plus fonctionner. L'avantage d'activer cette option est que *dev* est par définition une image plus avancée en terme de développement que *prod* et contenant donc les dernières mises-à-jour.
- `nbcompil` C'est le paramètre principal et obligatoire à l'exécution de la commande, c'est le nombre de compilations que l'on veut exécuter indépendamment. Le nombre précisé doit être strictement supérieur à 0.

— `incremental` Il s'agit d'un paramètre spécial concernant la compilation en mode incrémental de `tuxml.py`. En effet il est paramétré à 0 par défaut lorsqu'il n'est pas précisé. *incremental* représente le nombre de compilations en plus de la compilation basique d'un conteneur et qui sont dépendantes l'une par rapport à la précédente. Un paramètre incrémental positionné à 3 signifie que chaque conteneur fera 3 compilations supplémentaire par conteneur et ces compilations utiliseront les données des précédentes compilations de ce même conteneur. (Voir la section `tuxml.py` pour plus d'informations sur la compilation incrémentale)

Un exemple de commande pour `MLfood.py` :

```
./MLfood.py 10 4 --dev --no-clean
```

Dans cet exemple, `MLfood.py` va exécuter 10 conteneurs sur l'image Docker *dev*, sur chacun de ces conteneurs seront effectués 5 compilations, une basique suivie de quatre compilations incrémentale. Enfin les conteneurs seront sauvegardés par Docker en fin d'exécution et `MLfood.py` ne les supprimera pas grâce à "*--no-clean*".

Autre exemple : `./MLfood.py 1 --reset-logs`

Ici le nombre de compilations demandé sera ignoré car seul la demande de suppression des logs sera effectuée.

2.5 `runandlog.py`, premier script intermédiaire

2.5.1 Principe

Le script `runandlog.py` est le premier script intermédiaire exécuté entre `MLfood.py` et `tuxLogs.py`, il ne peut pas être exécuté en dehors du projet complet. Il est chargé de créer le fichier de log `output.log` qui est la sortie standard complète de l'exécution de `tuxLogs.py`. Ce programmes permet d'attraper les cas d'erreur fatale (crash) et donc d'écrire la *stacktrace* dans ce fichier. Lorsque un certain nombre de compilation est lancé, `MLfood.py` va appeler une fois par conteneur le programme `runandlog.py` qui se chargera de transmettre les paramètres d'appels tel que *incremental* à `tuxLogs.py`. `MLfood.py` lance l'exécution de `runandlog.py` pour pouvoir récupérer le fichier `output.log` à la terminaison.

Une fois le fichier `output.log` créé et la/les compilations du conteneur courant terminées, `runandlog.py` se charge de compresser et d'envoyer le fichier `output.log` à la base de données.

2.5.2 Utilisation

Ce script ne possède qu'un paramètre, le paramètre incrémental. Étant le premier script exécuté au démarrage du conteneur Docker dans lequel il se trouve, il n'a besoin d'appeler `tuxLogs.py` qu'en lui fournissant le paramètre incrémental de `tuxml.py`. Son utilisation se résume donc à :

```
./runandlog.py [incremental]
```

Si il n'est pas précisé, le paramètre incrémental est paramétré à 0 et le conteneur Docker se contentera de lancer une compilation basique.

`runandlog.py` est utilisé seulement dans le code de `MLfood.py` et ne peut pas fonctionner en dehors du contexte du projet TuxML.

2.6 tuxLogs.py, second script intermédiaire

2.6.1 Principe

Le script du fichier `tuxLogs.py` est un script intermédiaire nécessaire à l'exécution du projet TuxML, au même titre que `runandlog.py`. Il est situé entre ce dernier et `tuxml.py`. Son unique but est d'exécuter `tuxml.py` grâce aux arguments passés par les programmes précédents.

2.6.2 Utilisation

Son utilisation est la même que `runandlog.py`. Les deux scripts fonctionnent de la même manière car ils se contentent de transmettre les paramètres donnés à `MLfood.py`. On aura donc une commande de type :

```
./tuxLogs.py [incremental]
```

De même que `runandlog.py`, si aucune valeur incrémentale n'est précisée, elle sera paramétrée à 0. (Étant toujours appelée par `runandlog.py` qui donne une valeur par défaut à *incremental*, le paramètre *incremental* de `tuxLogs.py` aura toujours une valeur.)

2.7 TuxML dans sa version standalone

2.7.1 Principe

TuxML peut être lancé indépendamment des dockers sur une seule machine, très utile dans la phase de développement pour déboguer et avoir un choix de paramètres plus fin qu'en utilisant `MLFood.py`.

Pour ce faire il faut télécharger le contenu du dossier core/ sur GitHub (https://github.com/TuxML/ProjetIrma/tree/master/core) et exécuter le fichier `tuxml.py`.

TuxML gardera le même comportement qu'à l'accoutumée. Il compilera les sources Linux passées en paramètres, analysera les fichiers de log en cas d'erreur, effectuera un test sur le noyau et enverra les résultats à la base de données.

TuxML dispose d'un large panel de paramètres :

```
tuxml.py [-h] [-v {1,2,3,4}] [-V] [-c NB_CORES] [-d KCONFIG] [--incremental
NINC] [--database {prod,dev}] source_path
```

Parmi eux seul le paramètre `source_path` est obligatoire, il s'agit du chemin vers les sources Linux. Les paramètres optionnels sont constitués de :

- `-h, --help` : Permet d'afficher l'aide
- `-v {1,2,3,4}, --verbose {1,2,3,4}` : Niveau de verbosité, par défaut à 3.
- `-V, --version` : Permet d'afficher la version de TuxML
- `-c N, --cores N` : Spécifie à TuxML le nombre de cœurs processeurs à utiliser, par défaut TuxML utilise tous les coeurs disponibles.
- `-d [KCONFIG], --debug [KCONFIG]` : permet de compiler/déboguer un `KCONFIG` spécifique. Le `KCONFIG` peut être donné sous la forme d'un seed (e.g. `0xdeadbeef`), et dans ce cas TuxML générera le .config associé, ou directement sous la forme d'un chemin vers un .config. Si aucun `KCONFIG` n'est spécifié, TuxML utilisera le .config déjà existant dans le dossier des sources Linux. S'il n'existe pas cela provoquera une erreur.
- `--incremental N` : Le mode incrémental, actuellement dans une phase expérimentale, permet de compiler plusieurs .config sans effacer les fichiers issus de la compilation précédente. D'abord on lance une première compilation appelée `BASE_CONFIG` (générée aléatoirement). Lors de cette compilation on efface tous les fichiers issus des compilations précédentes. Ensuite on effectue N compilations sans effacer les fichiers compilés précédemment. Il est possible de spécifier un .config spécifique pour la BASE_CONFIG en combinant le paramètre `-debug` au mode incrémental.
- `--incrementalVS KCONFIG1 KCONFIG2` : permet d'utiliser le mode incrémental avec deux .config spécifiques. KCONFIG1 jouera le rôle de la BASE_CONFIG tandis que KCONFIG2 utilisera le mode incrémental. (incompatible avec `--incremental`)
- `--database {prod,dev}` : permet d'envoyer les résultats sur l'une des deux bases de données, par défaut `prod`.

2.7.2 Utilisation

La commande suivante permet de lancer TuxML avec les paramètres par défaut :

```
./tuxml.py ./sources-linux-4.13.3
```

Quelques exemples d'utilisations pour le paramètre `-debug` :

```
./tuxml.py ./sources-linux-4.13.3 -d

./tuxml.py ./sources-linux-4.13.3 -d 0xdeadbeef

./tuxml.py ./sources-linux-4.13.3 -d ./config_backup/my.config
```

Pour utiliser le mode incrémental :

```
./tuxml.py ./sources-linux-4.13.3 --incremental 10

./tuxml.py ./sources-linux-4.13.3 --incremental 10 -d 0xdeadbeef
```

La commande pour utiliser le mode incremental versus :

```
./tuxml.py ./sources-linux-4.13.3 --incrementalVS ./config_backup/1.config
./config_backup/2.config
```

2.8 Variante TuxML : tuxml_depML.py

2.8.1 Principe

tuxml_depML.py se lance sur un docker Debian très minimaliste dans le sens où il contient que des paquets indispensables à toutes les compilations *(python3 apt-file apt-utils gcc make binutils util-linux kmod e2fsprogs python3-dev default-libmysqlclient-dev git wget)* et avant chaque compilation *cid*, il faut supprimer les paquets qui sont installés à la compilation *cid - 1*.

2.8.2 Utilisation

La commande se lance de la facon suivante :

```
./tuxml_depML.py ./sources-linux-4.13.3
```

2.9 Le Machine Learning : stats.ipynb

2.9.1 Principe

Une fois l'ensemble des informations récupérées grâce au autres fonctions il est important de pouvoir en obtenir quelque chose. La partie `Machine Learning` sert justement à analyser l'ensemble des données que l'on récupère afin de pouvoir en déduire des informations sur les compilations à venir.

Ce que l'on souhaite obtenir est, dans un premier temps, la taille du noyau que l'on va obtenir en fonction des options que l'on a choisi lors de la configuration. Pour cela, après une série de compilation on récupère pour chaque kernel l'ensemble des options choisis ainsi que la taille du noyau qui en résulte. On applique ensuite un algorithme de `Machine Learning`, ici soit un arbre de décision soit une forêt de décision aléatoire, afin de leur faire "apprendre" les données d'entrées. Une fois le model créé (grâce a l'apprentissage des données), on peut alors utiliser ce model afin d'en déduire les propriétés d'un nouvel exemple.

2.9.2 Utilisation

Pour ce projet, nous avons utiliser `Jupyter` permettant d'obtenir un interpréteur accessible depuis un navigateur, les programmes créés sont alors des notebook. Il est alors très simple en utilisant `Jupyter` de modifier, enregistrer ou exécuter le notebook fournis.

A Annexes

A.1 Figures

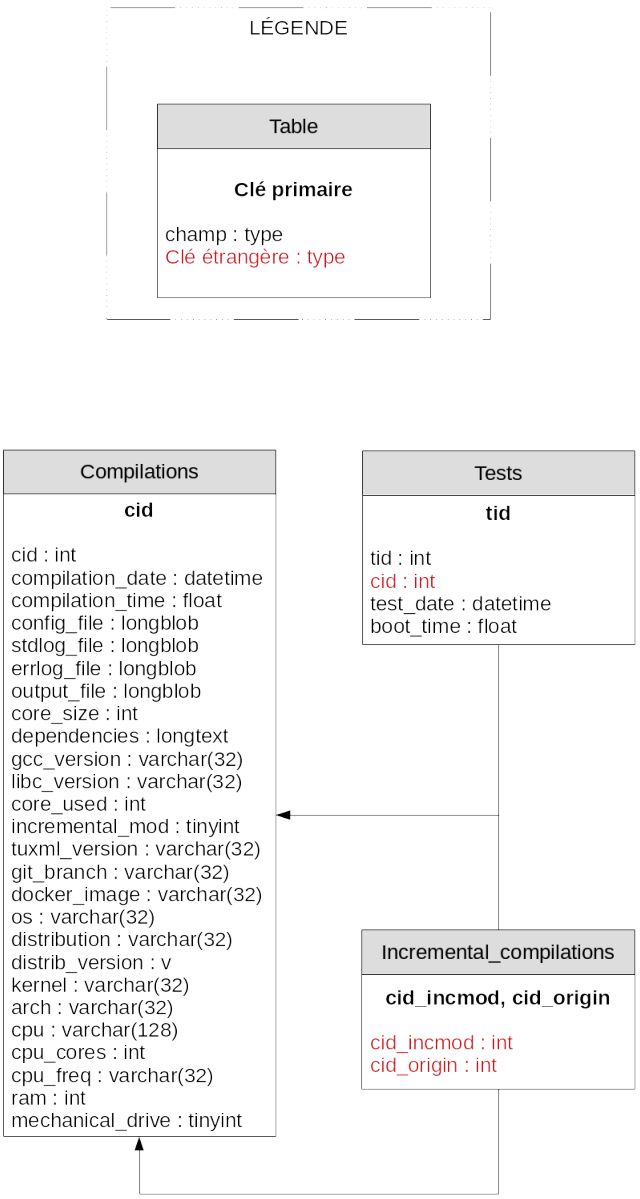

FIGURE 1 – Schéma représentant l'architecture de la base de données