Outil Makefile

Dans le but de simplifier le développement sur les projets Odoo installés localement sur un poste Mac OSX, la mise en place d'un **Makefile** a été réalisée, ce qui permet l'utilisation de commandes courtes réalisant une **suite d'opérations** précises.
D'une part, il permet ainsi de simplifier les commandes à utiliser, en réalisant le plus complexe par lui-même. D'autre part, il a permit de standardiser le Fonctionnement des dépôts, via l'utilisation de **variables** propres à chaque projet.

1. Fonctionnement général du Makefile

Le Makefile est structuré en *3 parties distinctes*, pour faciliter sa compréhension. En ce sens, il est nécessaire, en cas de modification, de suivre cette démarquation afin de laisser un fichier propre et maintenable par tous :

1.1 Variables

D'une part, une partie dédiée aux **variables**, modifiables en haut du fichier pour adapter le fonctionnement du Makefile à chaque projet client. Cette partie permet aussi de modifier les variables dans le but de modifier une commande pour réaliser une **opération spécifique**
Exemple :

```
# Fichier qui contient les processus

# Configuration pour Python
VENVS       = ~/.virtualenvs
CWD         = $(shell pwd)

PROJECT     = aln160402
CUSTOMER    = bobbies
VENV        = $(VENVS)/$(CUSTOMER)
MODULES     = $(CUSTOMER)_profile
CONFIGFILE  = odoo.conf
INTERFACE   = 127.16.4.2

# PostgreSQL
DBHOST      = localhost
DBPORT      = 5432
DBUSER      = $(CUSTOMER)
DBPASS      = $(DBUSER)
DBNAME      = $(DBUSER)
DBROLE      = $(DBUSER)
TMPL_DBNAME = template0
RESTOREFILE = database/restore.pgsql
```

1.2 Commandes seules

En second, viennent les commandes **standard** du Makefile. Ces commandes sont celles qui se suffisent à elles-mêmes, et réalisent une suite de commandes précise.
Exemple :

```
python:
    @test -d $(VENV) || python3 -m venv $(VENV)
    @. $(VENV)/bin/activate && pip install --upgrade pip
    @. $(VENV)/bin/activate && pip install -U coverage wheel flake8
    @. $(VENV)/bin/activate && pip install -r requirements.pip

postgresql:
    @echo "Create the PostgreSQL user $(DBUSER)"
    -@createuser -SdRE $(DBUSER)
    @echo "Change default password for this one $(DBPASS)"
    -@psql -c "ALTER USER \"$(DBUSER)\" ENCRYPTED PASSWORD '$(DBPASS)
    @grep "$(DBHOST):$(DBPORT):\*:$(DBUSER):$(DBPASS)" ~/.pgpass > /d
ifneq ($(DBUSER), $(DBROLE))
    @psql -c "GRANT $(DBROLE) TO $(DBUSER);" postgres
endif
```

1.3 Commandes combinées

Enfin, viennent les commandes qui représentent une suite de commandes **standardisées**. par exemple, la commande suivante : **`make init`** lance respectivement 3 commandes du Makefile : **`make python`** , **`make postgresql`** , **`make config`** .
Exemple :

```
init: python postgresql config
```

De nombreuses commandes combinées existent, utilisant plusieurs commandes seules afin d'effectuer une manipulation bien précise.

2. Variables

Au sein du Makefile, nous avons vu que les variables étaient initialement initialisées en haut du fichier Makefile, de sorte à accéder simplement à toutes les variables, si jamais il s'avère nécessaire de devoir en modifier une, telle que **l'interface** ou encore le **addons path**

2.1 Initialisation

Afin de bien comprendre les variables utilisées, voici le code commenté d'un fichier Makefile récupéré au sein d'un dépôt de projet client :

```
# Fichier qui contient les processus

# Répertoire où se situe tous les environnements virtuels
VENVS       = ~/.virtualenvs
# Current Working Directory : Dossier dans lequel nous nous situons a
CWD         = $(shell pwd)

# Code projet : utilisé pour identifier le dépôt de Backup (Bup). Il
PROJECT     = aln160402
# Client : Utilisé pour définir le dossier accueillant le VENV, ou er
CUSTOMER    = bobbies
# Stocke le chemin vers le répertoire contenant l'environnement virtu
VENV        = $(VENVS)/$(CUSTOMER)
# Définit les modules concernés lors d'une commande : Par défaut, lor
MODULES     = $(CUSTOMER)_profile
# Le nom du fichier de configuration utilisé pour faire tourner le se
CONFIGFILE  = odoo.conf
# L'inteface locale à laquelle sera disponible le projet, une fois le
INTERFACE   = 127.16.4.2

# PostgreSQL : Contient les informations de connexion à la base de do
DBHOST      = localhost
DBPORT      = 5432
DBUSER      = $(CUSTOMER)
DBPASS      = $(DBUSER)
DBNAME      = $(DBUSER)
DBROLE      = $(DBUSER)
TMPL_DBNAME = template0
# Le restore file est utilisé lors de la restauration d'une base de d
# 1. Définir l'utilisateur d'ID "1" avec un username : admin / subter
# 2. Changer les mots de passe de chaque utilisateur : Password = log
# 3. Lancer le fichier 'restore-common.pgsql' qui effectue des opérat
RESTOREFILE = database/restore.pgsql

# Odoo
DBOPTIONS   = -r $(DBUSER) -w $(DBPASS) --db_host=$(DBHOST) --db_port
IFOPTIONS   = --xmlrpc-interface=$(INTERFACE)
# Les différents chemins absolus, séparés par une virgule, vers les m
ADDONS      = $(CWD)/modules/odoo/enterprise,$(CWD)/server/addons,$(C
# Le chemin vers le répertoire de filestore, contenant images, PDFs ,
FSTORE_PATH = ~/.local/share/Odoo/filestore
MODULE_CATEG = customer
# Modules testés lors de la commande 'make test'
TESTED_MODULE = modules/$(MODULE_CATEG)/addons/$(MODULES)

# Serveur distant de production : Informations utilisées notamment lo
REMOTE_DBHOST = localhost
REMOTE_DBPORT = 5432
REMOTE_DBUSER = odoomaster
```

```
REMOTE_DBNAME = odoo
REMOTE_DBROLE = $(REMOTE_DBUSER)
REMOTE_PATH   = /tmp
REMOTE_FSTORE_PATH = $(FSTORE_PATH)

# Branche Git : Utilisée notamment lors du 'make template'
BRANCH      = 12.0
CUR_BRANCH := $(shell git rev-parse --abbrev-ref HEAD)

# Translations configuration
POEDITOR    = $(shell which poedit || which /Applications/Poedit.app/
SED         = $(shell which gsed || which sed)
TAR         = $(shell which gtar || which tar)

# Bup : Informations de restauration de base de données de productior
BUP_RESTORE_DATE = $(shell date --iso-8601)
BUP_RESTORE_TIME = 23:59:59
BUP_PATH    = database/backups
BUP_REPO    = $(PROJECT)

# Borg (Nouveau système de backups)
BORG_SERVER = borg@51.68.95.82:22 # L'ip du serveur qui stocke les sa
BORG_MOUNTPATH = /tmp/backups # Le point de montage en local sur le m
BORG_ODOOPATH = opt/odoo-master
BORG_DBPATH = tmp/databases # L'endroit où se situe la base de donnée
BORG_FSTOREPATH = home/odoomaster/.local/share/Odoo/filestore # L'end
BORG_PATHFS = $(BUP_PATH)/$(DATABASE)
BORG_CUSTOMERMOUNT = $(BORG_MOUNTPATH)/$(PROJECT)
BORG_DB = $(BORG_CUSTOMERMOUNT)/$(BORG_DBPATH)/$(REMOTE_DBNAME).sql
BORG_FS = $(BORG_CUSTOMERMOUNT)/$(BORG_FSTOREPATH)/$(REMOTE_DBNAME)
BORG_DATE = $(shell borg list ssh://$(BORG_SERVER)/~/backups/$(PROJEC
PG_WORKERS = 8 # Le nombre de workers (coeur virtuels) qui seront uti
```

2.2 Modifications ponctuelles de variables

Lors de l'utilisation de commandes définies au sein du Makefile, il arrive que les variables soient modifiées, parce qu'une commande nécessite, pour son cas uniquement, des paramètres particuliers. Aussi, il est ainsi possible de voir ceci au sein du Makefile :

```
upgrade: DATABASE=$(DBNAME)$(INSTANCE)
upgrade: OPTIONS=-d $(DATABASE) --update $(MODULES)
upgrade: run
```

Ici, de nouvelles variables sont définies, par exemple, afin d'être utilisées pendant le traitement de la commande **`make upgrade`** .
C'est un point essentiel, qui peut expliquer que vous ne trouviez pas certaines variables utilisées dans les commandes du Makefile en haut de votre fichier, puisque celles-ci sont trop spécifiques.

2.3 Variables en argument

Pour conclure la partie **VARIABLES**, il est possible de **forcer** ou **définir** une variable. Par exemple, je souhaite lancer la commande **`make upgrade`** , pour une base de données nommée **<client>_upgraded**, il m'est possible de lancer ceci :

`make upgrade DATABASE=nom_base_de_donnees`

Voici la commande dans le Makefile qui est alors lancée :

```
upgrade: DATABASE=$(DBNAME)$(INSTANCE) # Ici, DATABASE prends à la pl
upgrade: OPTIONS=-d $(DATABASE) --update $(MODULES)
upgrade: run
```

Cela clôt la documentation sur les variables du Makefile

3. Customisation du Makefile

Dans certains cas, il est possible que vous vous retrouviez face à un besoin qui n'est pas encore forcément traité par le **Makefile** aujourd'hui. Dans ce cas, n'hésitez pas à mettre en place vos propres commandes !
Dans la majorité des cas, la commande que vous souhaitez pouvoir utiliser n'est qu'une combinaison de commandes existantes, où même une très légère customisation du Makefile pour votre besoin.

4. Utilisation

Lorsque vous récupérer un dépôt de projet client, que vous avez forké :
`make init` : Cette commande initialise le projet afin qu'il soit utilisable. Il crée ainsi l'environnement virtuel, l'utilisateur postgresql, et accède à cet environnement virtuel. Il met aussi en place la configuration du projet.
A la fin de cette commande, si celle-ci s'est passé sans accroc, le reste des commandes du Makefile sont prêtes à être utilisées et fonctionnelles.

4.1 Utilisation d'une base de données de développement

Pour éviter de devoir créer une base de données complète à chaque fois qu'une base vierge est nécessaire, il est possible de créer une base template. C'est une base de données dans laquelle tous les modules nécessaires au projet sont installés, mais qui ne contient aucune donnée.
La création d'une nouvelle base vierge se fait ensuite par une simple duplication de la base template, puis mise à jour des modules. La première fois, il est donc nécessaire de créer la base template, avec la commande suivante :

`make template`

Par défaut, cette commande bascule vers la branch master, il est possible de spécifier une branche à utiliser en faisant :

`make template BRANCHE=<nom_branche>`

Ensuite, pour créer une base de développement vierge, il suffit de lancer la commande :

`make dev_db`

La base de données alors créée ne contient aucune donnée.

4.2 Lancer le service Odoo

Sur Mac OS, avant de lancer le service Odoo pour la première fois, il faut utiliser la commande :

`make loopback_alias`

ATTENTION: Tout redémarrage du MAC entraîne une perte de l'adresse loopback, il sera alors nécessaire de refaire tourner la commande.

Ensuite, il est nécessaire de lancer le service Odoo afin de pouvoir l'utiliser dans le navigateur web. Cela se fait avec la commande :

`make run_<instance>` , où <instance> correspond à la base de données à charger, qui est soit **dev**, soit **master**. Dans le cas où vous ayez suivi la partie **4.1**, la commande sera donc :

`make run_dev`

Après cette commande, le serveur odoo sera lancé. Pour savoir sur quelle URL, vous pouvez chercher une ligne de log "INFO" ressemblant à celle-ci, dans les 20 premières lignes données par Odoo :

`2019-08-09 15:21:34,319 4216 INFO ? odoo.service.server: HTTP service (werkzeug) running on 127.19.8.3:8069`

Vous aurez ensuite accès à une instance, en vous connectant via l'IP et le port spécifié, vid"e de toute données vous permettant d'effectuer les tests désirés. Désormais, le nom d'utilisateur par défaut est **subteno**, avec le **même mot de passe**.

Si ce n'est pas celui-ci, alors le login & mot de passe est : **admin**.

4.3 Utiliser une copie de production (à J-1)

Dans le cas où vous ayez besoin d'une base sauvegardée de la production, il est possible de la récupérer via la commande suivante :

`make bup_master` (Nouveau système –> Borg)

Cette commande ne nécessite pas de construire une base de données template.

Lorsque vous utilisez cette commande, vous récupérez la base de données de production de la nuit passée. En effet, la base de production est sauvegardée toutes les nuits, et envoyée sur un serveur **BORG** qui sert de dépôt de backups.

Cette commande se charge de télécharger les sauvegardes, restaurer la base de données, la mettre à jour avec le code présent sur la machine de développement, et remplace automatiquement certaines valeurs, comme les mots de passe des utilisateurs (égaux au login pour chaque utilisateur) ou les adresses email des partenaires (pour éviter d'envoyer des emails aux vrais clients lors des tests).

Lorsque la commande est utilisée, vous allez ainsi récupérer une sauvegarde sur le dépôt de backup BORG, qui gère toutes les sauvegardes clientes.

> ***Lorsque vous lancez cette commande pour la première fois, pour certains clients, il est possible qu'elle prenne un certain temps. En effet, le filestore doit être importé une première fois en entier. Par la suite, seul le différentiel sera récupéré.***

Il est aussi possible de restaurer une sauvegarde faite à une date définie, en précisant le paramètre **`BORG_DATE`** de cette manière : **`make bup_master BORG_DATE='2020-09-07T03:46:59'`** . Il est possible de récupérer la liste de tous les backups disponibles en allant sur le serveur de pré-production du client, via la commande : **odoo-borg-backup-list**

Le réalisation d'un **`make bup_master`** met par la suite à jour **tous les modules odoo.**

CAS PARTICULIERS: (Ancient système –> Bup)

Lorsqu'un projet est utilisé pour plusieurs clients (Dans le cas de bup), il est nécessaire de préciser le nom du client, l'emplacement où placer le dépôt de sauvegarde sur le disque ainsi que le nom du dépôt de sauvegarde des autres clients du projet. Cela peut être fait en utilisant les paramètres **`CUSTOMER`** (nom du client), **`BUP_PATH`** (chemin local) et **`BUP_REPO`** (nom du dépôt sur le serveur). Pour restaurer une base de production du client Acta depuis le dépôt de GIS, on utilisera donc cette commande : **`make bup_master CUSTOMER=acta BUP_PATH=database/backups-acta BUP_REPO=aln150607-acta`**

4.4 Sauvegarder et restaurer une base de données locale

Il est possible d'effectuer une sauvegarde d'une base de données locale, par exemple après avoir effectué un paramétrage complexe, mais avant d'effectuer des tests, pour pouvoir retrouver simplement une base propre contenant le paramétrage effectué. La sauvegarde se fait avec la commande make backup_instance, et la restauration se fait avec la commande make restore_instance, où instance correspond à la base de données à restaurer (souvent dev ou master). Il est aussi possible de nommer la sauvegarde, afin d'en conserver plusieurs, par exemple : **`make backup_master NAME=base_parametrage puis make restore_master NAME=base_parametrage`** .

4.5 Mise à jour de module(s)

Lorsque vous réalisez certaines modifications au sein d'un projet client, notamment lors de la mise à jour de vues XML, il est nécessaire de faire un upgrade des modules concernés afin d'intégrer les modifications dans la base de données Odoo.
Pour cela, la commande suivante est à votre disposition :
`make upgrade_instance` : Où _**instance** est soit **`dev`** , soit **`master`** , de sorte à mettre à jour la base de développement, où bien la base de production précédemment récupérée.
Dans le cas où votre base n'est ni la base de production, ou encore la base de développement, vous avez la possibilité d'exécuter la commande **`make upgrade DATABASE=<nom_bdd>`** seule. De cette façon, vous précisez la base que vous voulez mettre à jour.
IMPORTANT: Par défaut, seul le module profile est mis à jour lorsque vous lancez un **`make upgrade`** (dev, master, etc …). Si vous souhaitez mettre à jour tous les modules Odoo, il est possible d'ajouter un argument à la commande :
`make upgrade_master MODULES=all` : Met à jour tous les modules sur la base copie de production.
Il est aussi possible de ne mettre à jour qu'une liste précise de modules :
`make upgrade_dev MODULES=module_1,module_2` : Met à jour **module_1** et **module_2** sur la base de développement.
À la fin de la mise à jour, le service Odoo n'est pas stoppé, il est donc possible de se connecter à l'instance sans commande supplémentaire.

4.6 Traduction des développements réalisés

La traduction nécessite d'avoir au préalable généré une base template vierge.
La traduction peut ensuite être faite, en lançant la commande **`make translate`**`
Cette commande extrait les termes à traduire depuis Odoo, met à jour le fichier de traduction depuis la liste des termes à traduire (ajoute les nouveaux et retire les termes disparus), puis ouvre le fichier de traduction dans poedit, s'il est installé sur la machine, ou vi si vous préférez

(re)vivre dans les années 80.
Dès la fermeture du programme éditant le fichier de traduction, les traductions sont mises à jour dans la base de données, puis ré-extraites, pour permettre de commiter un fichier toujours tout juste extrait d'Odoo.
Cela permet de réduire les diffs, en étant assurés que les termes sont toujours placés dans le même ordre. Dans le cas où il y a plusieurs modules (par exemple, le dépôt GIS/ACTA), pour faire la traduction d'un module spécifique, il faut s'assurer que le module est bien présent dans le template - par exemple :

```
make template MODULES=acta_profile
```

Avant de faire le :

```
make translate MODULES=acta_profile
```

4.7 Exécution des tests unitaires

L'exécution des tests unitaires nécessite aussi d'avoir au préalable généré la base template.
Pour l'exécution des tests unitaires : **`make test`**
Après l'exécution des tests unitaires, une rapport des fichiers non entièrement testés est affiché dans la console. Ce rapport indique, pour chaque fichier, le nombre de lignes non testées, ou testées partiellement, et les numéros de lignes non ou pas totalement testées.
Un rapport en HTML est aussi généré, disponible dans le répertoire **htmlcov**, placé à la racine du dépôt.

5. Commandes utiles

Voici une liste de commandes pouvant être ajoutées au Makefile, qui peuvent s'avérer utile :

5.1 Mise à jour du standard Odoo

> Warning
>
> La section 5.1 n'est plus à jour, et sera corrigée sous peu. Nous utilisons aujourd'hui les subrepos, à la place des subtrees !

Ajouter la ligne suivante dans la section des variables :

```
ODOO_VERSION = 12.0 # Numéro de version odoo
```

Puis créer cette commande dans la liste des commandes du makefile :

```
update_odoo:
git subtree pull --squash --prefix=server git@github.com:odoo/odoo $(
git subtree pull --squash --prefix=modules/odoo/enterprise git@github
```

Vous mettrez ainsi le standard Odoo à jour.

5.2 Changement de version majeure du standard Odoo

> Warning
>
> La section 5.2 n'est plus à jour, et sera corrigée sous peu. Nous utilisons aujourd'hui les subrepos, à la place des subtrees !

Ajouter la ligne suivante dans la section des variables :

```
ODOO_VERSION = 13.0 # Numéro de la nouvelle version Odoo
```

Puis créer cette commande dans la liste des commandes du makefile :

```
upgrade_odoo:
git rm -rf server
git ci -m "[REM] Odoo CE"
git rm -rf modules/odoo/enterprise
git ci -m "[REM] Odoo Enterprise"
git clean -fdx server/ modules/
git subtree add --squash --prefix=server git@github.com:odoo/odoo $((
git subtree add --squash --prefix=modules/odoo/enterprise git@github.
```

5.3 Récupérer une base de pré-production ou de testing

Parfois, il peut être utile de récupérer en local une base de données de pré-production ou d'instance de test. Cette récupération ne fait pas partie du process "standard" Subteno IT, mais il est possible d'ajuster les commandes `make` pour réussir à parvenir à ses fins.
Pour récupérer la base **archive** d'un serveur de pré-production, on peut lancer la commande suivante :

```
$ make download_instance INSTANCE=_preprod REMOTE_DBNAME=archive REM(
```

La syntaxe s'explique de cette façon :

- INSTANCE = l'instance à laquelle on va se connecter, précédée d'un **underscore**
- REMOTE_DBNAME = La base distante que l'on veut restaurer localement
- REMOTE_DBROLE et REMOTE_DBUSER = Rôle et utilisateur de la base de données à utiliser. Seul l'utilisateur odoopreprod a accès aux bases de données de preprod, comme archive !

> Tip
>
> Vous l'aurez compris, si je souhaite restaurer une base sur un serveur testing, je dois alors modifier l'instance à _**testing**, choisir la bonne base à restaurer, et utiliser l'utilisateur et le rôle lié aux instances testing : **odootesting**