# Les environnements virtuels Python avec Conda

G. COLLETAZ – Novembre 2025

## 1- Généralités

Ce sont des espaces isolés les uns des autres dans lesquels on va pouvoir installer des versions spécifiques des packages que l'on veut utiliser. On pourra par exemple avoir python 3.10 dans un environnement avec des packages qui requièrent cette version et toutes leurs dépendances, et un autre environnement avec python 3.13 et des packages compatibles avec leurs dépendances propres possiblement non compatibles avec celles avec l'environnement précédent.

L'exécutable `Conda` est un logiciel open source qui est à la fois un gestionnaire d'environnements, i.e. de ces espaces cloisonnés, et de packages, i.e. il va permettre de les installer, de les mettre à jour et de les supprimer. Pour en bénéficier vous pouvez télécharger et installer soit la distribution `Anaconda`, soit `Miniconda`. L'un et l'autre vont installer `Conda` et `Python` ainsi que d'autres packages utiles. Leurs commandes seront donc identiques. Les deux peuvent nécessiter l'acquisition de licences payantes pour accéder à certains dépôts de packages et nous reviendrons plus loin sur ce point.

La différence entre `Anaconda` et `Miniconda` tient notamment au nombre de packages qui seront installés :

- Anaconda : plus de 300 packages essentiellement scientifiques et plus de 300 GB d'espace disque, propose une interface graphique (GUI) via le navigateur Anaconda.
- Miniconda : installation des dépendances minimales de Conda d'environ 400MB, et une interface de ligne de commande (CLI).

Dans cette introduction aux environnements conda sous Window son vous conseille de choisir d'installer Miniconda sachant qu'il sera toujours temps d'installer aisément les packages dont vous auriez besoin pour tel ou tel projet tout en ayant une vision plus claire du contenu de vos environnements de travail. Par ailleurs la gestion des dépendances étant complexe (et lente), on ne doit pas sous-estimer l'arrivée de difficultés par exemple lors de la mise à jour des packages et dans ce cas, la parcimonie de Miniconda peut être un atout important pour leur correction.

Notez que pour les utilisateurs de Windows, Anaconda.org recommande d'utiliser l'interface CLI. Ce document ne s'intéresse qu'à cette dernière. Sacchez qu'il est toutefois possible de l'installer via la commande commande `conda install –name base anaconda-navigator`.

Des versions commerciales existent, mais une installation gratuite individuelle est toujours disponible. L'un ou l'autre peut être téléchargé cette adresse :

https://www.anaconda.com/docs/getting-started/getting-started

Pendant l'installation vous aurez plusieurs options possibles, le mieux est de valider les choix qui vous sont recommandés. En particulier l'option « Add Anaconda to my PATH environment variable » n'est pas recommandée car elle ajoute Anaconda dans les variables d'environnement système or sous Windows le system PATH est prioritaire sur le user PATH ce qui peut être source d'erreur pour des commandes entrées dans un environnement « utilisateur ». En revanche lorsque l'option choisie

est « just me », Anaconda est ajouté dans le user PATH ce qui élimine les sources d'erreurs précédentes.

Par défaut les environnements seront installés dans les directory suivantes :

- Anaconda Distribution - C:\Users\<USERNAME>\anaconda3\envs
- Miniconda - C:\Users\<USERNAME>\anaconda3\envs

`Conda` est réputé notamment pour sa gestion des dépendances : lors de l'installation d'un ou de plusieurs nouveaux packages dans un environnement existant ou de l'update d'un package, il va normalement s'assurer de la compatibilité des dépendances des nouvelles installations entre elles mais également avec celles des installations déjà présentes. Vous pourrez ainsi observer quelquefois qu'il n'installe pas obligatoirement la dernière version du ou des packages que l'on veut télécharger et qu'il peut même lui arriver de remplacer un package par une version plus ancienne afin d'assurer la compatibilité des dépendances propres à l'environnement dans lequel vous travaillez. En cela il est généralement reconnu que le risque de casser un environnement est plus faible avec `Conda` qu'avec PIP, un autre utilitaire très connu de gestion de packages sur lequel nous reviendrons.

Lors de l'installation d'`Anaconda` ou de `Miniconda`, un environnement particulier nommé `base` est créé, c'est dans celui-ci qu'est notamment installé le gestionnaire `conda` et Il est conseillé de ne rien y installer d'autre. Enfin vous y trouverez également un programme Anaconda Prompt qui ouvre une fenêtre dans laquelle seront entrées les commandes de l'interface CLI. Sachant que son utilisation est assez fréquente, il peut être utile de lui associer un bouton dans votre barre des tâches afin d'y accéder rapidement.

# 2- Avantages des environnements Conda

## 2.1- Isolation des dépendances :

- 2.1.1 : Chaque projet peut avoir ses propres versions de packages sans conflits
- 2.1.2 : Évite les problèmes de compatibilité entre différents projets
- 2.1.3 : Permet de travailler simultanément sur des projets nécessitant des versions différentes de Python

## 2.2- Reproductibilité :

- 2.2.1 : Facilite le partage d'environnements entre développeurs
- 2.2.2 : Garantit que le code s'exécute de la même manière sur différentes machines
- 2.2.3 : Simplifie le déploiement en production

## 2.3- Gestion simplifiée des packages :

- 2.3.1 : Installation et mise à jour simplifiée
- 2.3.2 : Résolution automatique des dépendances

## 2.4- Sécurité et stabilité :

- 2.4.1 : Tests de nouvelles versions sans risque pour l'environnement principal
- 2.4.2 : Possibilité de revenir aisément à une version stable
- 2.4.3 : Les environnements anaconda peuvent être créés dans l'espace utilisateur ce qui permet d'installer des packages sans disposer des droits administrateurs système

# 3- Les principales commandes

Nous présentons dans l'ordre les commandes relatives à `Conda` lui-même, puis à la gestion des environnements leur création, leur suppression, leur clonage,…) et enfin aux packages Python (installation, suppression, mise à jour).

Remarque : les invites de commandes font apparaître la directory dans laquelle se trouve les informations concernant l'utilisateur. Celles-ci se trouvent dans Windows dans c:\Users. Ainsi l'utilisateur de nom p447 verra la chaîne `C:\Users\p447>`

Le signe de fin `>` signifie qu'une commande est attendue à sa suite. Pour ne pas obscurcir la présentation, nous ne faisons pas apparaître ces chemins par le suite.

## 3.1- Commandes relatives à Conda

- Pour connaître la version de `Conda` qui est installée, vous pouvez utiliser au choix la commande

  `conda -V`, ou
  `conda –version`

Vous pouvez également utilser `conda info` qui vous donnera des informations sur votre installation de Conda et notamment sa version.

- Plus importante est la commande de mise à jour de `Conda` vers la dernière version : il est en effet conseillé d'utiliser la version la plus récente du gestionnaire préalablement à toute action sur les packages. Si ce n'est pas le cas vous recevrez d'ailleurs un message d'avertissement lors d'une demande d'installation ou de mise à jour de packages, message vous demandant d'installer sa version la plus récente, par exemple :

```
==> WARNING: A newer version of conda exists. <==
current version: 4.6.13
latest version: 4.8.0
```

Pour faire cette mise à jour la commande à exécuter est :

`conda update conda`

`Conda` va alors comparer la version installée et celle disponible en téléchargement. Si elles diffèrent, il posera la question

`Proceed ([y]/n)?`

Si vous tapez `y`, il affichera l'ensemble des packages qui seront également automatiquement mis à jour et réalisera l'update.

## 3.2- Commandes relatives aux environnements

### 3.2.1- La création d'un environnement

Pour créer un environnement, on dispose de la commande `create` dont la syntaxe est :

```
conda create --name <nom de l'environnement>
```

où <nom de l'environnement> est le nom que vous voulez utiliser. Par exemple :

```
conda create --name env1
```

va créer un environnement de nom `env1` qui, en l'état, sera vide.

Généralement un environnement est créé pour y mettre des packages et `create` va d'ailleurs autoriser simultanément la création et l'installation de programmes. Ainsi on pourra exécuter :

```
conda create --name env1 numpy
```

qui va créer `env1` et y installer Numpy. On peut également en profiter pour installer plusieurs packages et éventuellement des versions spécifiques en précisant le numéro de version désiré. Par exemple :

```
conda create --name env1 python=3.08 numpy pandas
```

créera `env1` et y installera la version 3.08 de Python ainsi que les packages Numpy et Pandas dans leurs versions les plus récentes possibles compatibles avec ce Python (et donc pas forcément les plus récentes). Notez déjà qu'Anaconda recommande d'installer plusieurs packages dans une seule commande plutôt que de le faire un par un afin de réduire les risques de conflits de dépendances.

### 3.2.2- Activer et désactiver un environnement

- L'activation :

L'activation va consister à préciser l'environnement dans lequel vos commandes vont être exécutées. Ainsi, pour activer l'environnement env1, il suffira d'entrer :

```
conda activate env1
```

La commande `activate` va affecter des variables système et le PATH de sorte qu'alors `env1` est en fait un chemin qui mène à la directory dans laquelle sont stockés tous les packages que vous avez installés dans l'environnement en question. En particulier les commandes entrées ultérieurement seront en sous-main automatiquement précédées de env1 garantissant ainsi que ce sont bien les packages que vous désirez qui seront exécutés.

Par défaut l'environnement `base` est automatiquement activé.

Vous pourrez reconnaître visuellement votre environnement de travail par le fait qu'à l'affichage toutes vos invites de commandes seront précédées du nom de l'environnement actif mis entre

parenthèses. Par exemple, si vous exécutez la commande précédente après avoir ouvert la fenêtre Anaconda Prompt vous verrez à l'écran :

```
(base) conda activate env1
```

qui sera ensuite suivi par

```
(env1)  …
```

Ce qui montre que la première commande a été réalisée dans l'environnement `base` et que la prochaine le sera dans l'environnement `env1`.

Notez enfin qu'à tout moment que vous ne pouvez avoir un seul environnement actif.

- La désactivation :

Pour désactiver un environnement actif utilisez

```
conda deactivate
```

cela a pour effet d'effacer les variables système et le PATH associés à cet environnement actif et de et de retourner à celui qui était actif avant celui que l'on désactive.

Pour retourner directement à l'environnement `base` il suffit de faire :

```
conda activate
```

i.e. envoyer une commande d'activation sans préciser de nom d'environnement.

Un exemple d'enchainement de commandes facilite sans doute la compréhension de ces points notamment en repérant l'environnement actif qui suit l'exécution de chacune :

1. `(base) conda create –name env1`
2. `(base) conda create –name env2`
3. `(base) conda activate env1`
4. `(env1) conda activate env2`
5. `(env2) conda deactivate`
6. `(env1) conda activate env2`
7. `(env2) conda activate`
8. `(base)`

Remarque : évitez de faire

```
(base) conda deactivate
```

i.e. de faire un `deactivate` sur l'environnement racine. Cela peut entrainer un blocage sans gravité de Conda et vous obliger à démarrer une nouvelle session conda dans un nouveau terminal.

### 3.2.3- Afficher la liste de vos environnements

Pour connaître la liste des environnements que vous avez créé, vous pouvez exécuter au choix

```
conda env list   ou    conda info --envs
```

Les deux conduisent au même écran qui, outre leurs noms, va également indiquer leurs emplacements sur le disque dur. Par exemple :

```
(base) C:\Users\p447>conda env list
```

Pourra afficher :

```
# conda environments:
#
base                  *  C:\Users\p447\AppData\Local\miniconda3
ml                       C:\Users\p447\AppData\Local\miniconda3\envs\ml
darts                    C:\Users\p447\AppData\Local\miniconda3\envs\darts
dartsGPU                 C:\Users\p447\AppData\Local\miniconda3\envs\dartsGPU
finance                  C:\Users\p447\AppData\Local\miniconda3\envs\finance
pytorch                  C:\Users\p447\AppData\Local\miniconda3\envs\pytorch
sktime                   C:\Users\p447\AppData\Local\miniconda3\envs\sktime
stats                    C:\Users\p447\AppData\Local\miniconda3\envs\stats
```

Vous pouvez remarquer plusieurs choses :

- L'environnement actif est repéré par une astérisque,
- L'environnement racine  est dans la directory `miniconda3` et tous les autres dans une sous-directory de celle-ci nommée `envs`,
- Que plutôt de nommer de façon obscure les environnements, comme env1, env2, … il est préférable de leur donner un nom donnant une information se référant soit à leur contenu, soit au projet pour lequel ils ont été créés.

### 3.2.4- Cloner un environnement

Cloner signifie que l'on veut reproduire un environnement identique à un environnement existant sur la même machine.  machine, soit sur une machine différente mais du même OS que la vôtre en utilisant un fichier de spécification

Pour cela on utilise

```
conda info --name env_clone --clone env_ini
```

Afin de vérifier que l'opération s'est bien déroulée, on peut exécuter

```
conda env list
```

La liste doit mentionner l'environnement de départ, ici env_ini, et son clone, env_clone.

### 3.2.5- Reconstruire un environnement dans un OS identique

On veut créer un environnement identique à un existant sur la même machine, ou sur une machine différente à condition qu'elle possède le même OS. Pour cela on passe par deux étapes et un fichier de spécifications.

La première étape consiste à créer ce fichier de spécification. On commence par activer l'environnement concerné, par exemple `env1` puis on exécute

```
(env1)      conda list --explicit > spec_file.txt
```

Le fichier de nom `spec_file.txt` est alors créé dans la directory courante. Naturellement vous pouvez lui donner le nom de votre choix. Si vous l'ouvrez, vous verrez la liste des packages présents dans `env1` avec leur numéro de version ainsi que l'indication de l'OS sous lequel tournait `env1`.

La deuxième étape utilise le fichier pour créer un nouvel environnement avec :

```
conda create --name new_env --file spec_file.txt
```

Remarque, on peut également installer les packages de l'environnement de départ dans un environnement déjà existant. Ainsi, si `env2` existe alors en exécutant :

```
conda install --name env2 --file spec_file.txt
```

L'environnement `env2` contiendra tous les packages de `env1` en plus de ses packages initiaux.

Attention, rappelez-vous que les OS de départ (ici celui de `env1`) et d'arrivée (ici celui de `new_env` ou de `env2)` doivent être identiques pour que tout se déroule sans erreur.

### 3.2.6- Exporter un environnement entre des OS différents

Voir : 4.3.1 dédié à la commande `export`.

### 3.2.7- Supprimer un environnement

Pour supprimer un environnement et tous les packages qu'il contient il faut le désactiver puis utiliser la commande `env remove` en spécifiant son nom. Ainsi, pour supprimer l'environnement `env2` et tous les packages qu'il contient on fera simplement :

(env2) `conda activate`

(base) `conda env remove --name env2`

### 3.2.8- Supprimer les packages inutilisés

```
conda clean --all --yes
```

Va supprimer tous les packages inutilisés **de tous les environnements, pas seulement de l'environnement actif**, et `--yes` évite la demande d'autorisation pour chacun des fichiers supprimés.

## 3.3- Commandes relatives aux packages

En présentant la commande create on a vu qu'il était possible de créer un environnement et d'y installer des packages avec l'exemple :

```
conda create --name env1 python=3.08 numpy pandas
```

Vous êtes alors en droit de vous demander depuis quels dépôts les packages en question vont être téléchargés. Dans la terminologie d'Anaconda ces dépôts sont appelés des channels. Certains vont être gérés directement par Anaconda.org, ce sont les channels par défaut d'Anaconda et d'autres par la communauté des utilisateurs, dans le cas de Conda il s'agit du channel conda-forge.

Avant d'étudier la gestion des packages, il importe surtout depuis 2004 de bien comprendre le statut de ces différents channels et des droits d'utilisation qu'ils imposent sur les packages dont ils sont issus.

### 3.3.1- Qui peut installer quoi avec Conda ?

Jusqu'en 2004, avant qu'Anaconda ne change les conditions d'utilisation de leurs dépôts, leur accès était gratuits. Depuis 2004, les termes des autorisations spécifient que toute organisation de plus de 200 salariés doivent acquérir une licence payante pour accéder à leurs channels, sachant que cette limite s'applique également aux organisations privées et à celles à but non lucratifs.

Alors que le secteur de l'enseignement était encore exempté de cette obligation Anaconda a fait savoir que leur usage non payant était toutefois limité aux cours affichés dans les maquettes des formations, Anaconda se réservant le droit de s'assurer que les softs étaient utilisés strictement dans des objectifs éducatifs.

Pour résumer :

- Les entreprises de moins de 200 salariés peuvent aujourd'hui utiliser la distribution Anaconda et les channels par défaut gratuitement,
- Les entreprises de plus de 200 salariés doivent acquérir une licence payante pour utilser la distribution et les channels par défaut,
- Les étudiants et les organismes d'enseignement sont dispensés de payer sous réserve que les accès soient justifiés par des cours affichés dans les maquettes de leur formation.

Miniconda reste gratuit pour tous aussi longtemps qu'on n'utilise pas les channels par défaut, sinon les exigences relatives à la distribution Anaconda s'appliquent.

Anaconda justifie cette politique par le fait que ses équipes d'ingénieurs révisent, améliorent la sécurité et optimisent les packages que l'on y trouve leur donnant ainsi une plus-value importante relativement à ceux présents dans les dépôts communautaires.

Compte-tenu de ces éléments, il devient important, sauf pour vous et pour encore quelques mois, de gérer le channel de téléchargement lorsqu'on décide d'installer un package via Conda, ce qui se fait soit au niveau des commandes en spécifiant le channel, soit par le biais d'un fichier .condarc sur lequel nous reviendrons.

Pour finir sachez que lorsque vous installez la distribution Anaconda ou Minicona, les deux produits sont configurés pour accéder les dépôts par défaut gérés par Anaconda.or, dépôts qui sont au nombre de 3 :

- main - https://repo.anaconda.com/pkgs/main
- r - https://repo.anaconda.com/pkgs/r
- msys2 - https://repo.anaconda.com/pkgs/msys2 (Windows only)

Sauf indication contraire c'est donc dans ces dépôts que seront cherchés les packages que vous installerez. Vous pouvez remarquerez que le second signale que les environnements Conda sont aussi utilisables pour ceux travaillant avec le langage R.

### 3.3.2- L'installation de packages

On utilise la commande `install`. Dans sa forme la plus simple, elle installe le package désiré dans l'environnement actif. Par exemple :

```
(env1)      conda install numpy
```

Demande d'installer Numpy dans l'environnement `env1`, le package étant descendu depuis l'un des trois dépôts définis par défaut.

Important :

- pour réaliser une installation, Conda va prendre dans les dépôts les versions les plus récentes possible du package et de ses dépendances compatibles avec l'environnement cible, ici l'environnement actif.
- On sait que des packages ou certaines de leurs versions ne sont compatibles qu'avec certaines versions de Python : si vous demandez d'installer un package dans un certain environnement qui n'est pas compatible avec la version de Python présente dans cet environnement, Conda ne l'installera pas. La solution est de créer un nouvel environnement et d'y installer une version de Python compatible avec le produit que vous voulez utiliser que vous installerez évidemment également dans ce nouvel environnement.

Il est possible de spécifier le channel et l'environnement cible dans la commande install comme suit :

```
(env1)conda  install -n sklearn-env -c conda-forge scikit-learn
```

Cette commande envoyée dans l'environnement env1 réclame l'installation de `scikit-learn` dans l'environnement `sklearn-env` en utilisant le dépôt `conda-forge` (`-n` est équivalent à `--name` utilisé jusqu'ici, et `-c` est équivalent à `--channel`).
Si l'environnement cible n'existe pas, vous devez maintenant comprendre qu'il est également possible avec une seule commande de le créer et de réaliser une installation en utilisant non pas `install` mais `create` . Ainsi, si `sklearn-env` n'existe pas, on peut faire :

```
(env1)conda  create -n sklearn-env -c conda-forge scikit-learn
```

Un point mérite d'être abordé : que fait Conda dans les deux commandes précédentes si le package demandé où certaines de ses dépendances ne se trouve pas dans le channel spécifié ? La réponse est qu'il va alors rechercher dans les autres channels qu'il connaît, ici dans les channels par défaut d'Anaconda. Si on veut éviter ce comportement on dispose de l'argument –override-channel qui a pour effet de limiter la recherche au channel spécifié et de forcer Conda à ignorer les autres channels comme par exemple dans :

```
conda install -c conda-forge --override-channels scikit-learn-intelex
```

Rappelez-vous enfin que `create` comme `install` admettent une liste de packages comme argument avec éventuellement la précision de la version souhaitée pour chacun d'eux. Il est d'ailleurs préférable d'installer tous les packages dans une même commande afin que toutes les dépendances soient elles-mêmes installées en même temps.

### 3.3.3- La mise à jour des packages

Avec la commande update admet un nom ou une liste de nom de packages présents dans un environnement et va tenter de remplacer leurs versions actuelles par les versions les plus récentes compatibles entre elles et avec les autres programmes de l'environnement.

Notez que la demande d'update d'un package peut entraîner l'update d'autres packages dont dépend le premier ou qui dépendent de celui-ci ainsi que l'installation de nouveaux programmes dont dépendrait la nouvelle version.

Ainsi :

```
(env1)conda update pandas numpy
```

Cherche à mettre à jour pandas et numpy présents dans l'environnement actif env1

Comme toujours les arguments -n, -c, --override-channels. On vous laisse comprendre la commande suivante :

```
(env1)conda update -n env2 -c conda_forge matplotlib
```

Enfin, il est possible demander l'update de tous les packages contenus dans un environnement avec l'argument –all comme dans :

```
(env1)conda update --all
```

Important : il est conseillé d'utiliser la dernière version de Conda avant de faire des mises à jour de packages, et donc d'exécuter

```
conda update conda
```

avant de faire les

`conda update` sur les packages

3.3.4- La suppression d'un package

Pour supprimer un ou une liste de packages on utilisera la commande remove :

```
(env1) conda remove polars scipy
```

Va supprimer les packages polars et scipy de l'environnement `env1`.

On peut supprimer tous les packages et donc l'environnement lui-même avec l'argument `--all`. Naturellement l'environnement concerné doit être désactivé, par exemple :

```
(env2) conda activate
(base) conda remove --name env2 --all
```

Cette dernière commande est donc équivalente à :

```
(base) conda env remove --name env2
```

# 4- Autres éléments d'information

## 4.1- Le fichier .condarc

C'est un fichier de configuration pour Conda dont l'emploi est optionnel. Il n'est pas créé lors de l'installation de Conda mais l'est automatiquement lorsque vous utilisez pour la première fois la commande :

```
conda config
```

et est normalement situé dans la directory c:\Users\<UserName>. Vous pouvez le vérifier en entrant

```
conda info
```

Il s'agit d'un fichier au format YAML possédant l'extension yml qui peut être ouvert avec votre éditeur de texte usuel. Si vous le faites vous lirez quelque chose ressemblant à ceci :

```
channels:
  - conda-forge
  - defaults
channel_priority: strict
```

Ces lignes configurent Conda de sorte que les seuls channels autorisés sont conda-forge et les 3 channels par défauts d'Anaconda. Retenez que l'ordre de la liste est important : Conda doit d'abord chercher dans conda-forge et s'il ne le trouve pas aller chercher dans les defaults.

La ligne channel_priority: strict assure que toutes les dépendances viendront également de conda-forge, c'est la valeur recommandée pour cette option.

Plutôt que d'utiliser un éditeur pour modifier à votre convenance le fichier .condarc et sauvegarder la nouvelle version, vous pouvez utliser l'interface CLI et exécuter une commande telle que :

```
conda config --add channels conda-forge
```

dont l'effet sera de placer conda-forge en première place de la liste des channels dans .condarc, suivie par :

```
conda config --set channel_priority strict
```

Si on ne veut pas donner à conda-forge la priorité la plus haute mais au contraire la plus basse, on peut utiliser l'argument --append comme dans :

```
conda config --add channels --append conda-forge
```

qui placera conda-forge en dernière position dans la liste des channels.

Sachez enfin que le fichier .condarc placé dans le répertoire racine va s'imposer à tous les environnements. On peut toutefois affecter une configuration particulière à un environnement en sauvegardant un fichier .condarc avec les options que l'on désire dans la directory qui le contient. Il n'existe pas de commande Conda permettant de réaliser cette sauvegarde qu'il faut donc réaliser manuellement.

Par exemple lorsque nous avons affiché la liste des environnements nous avions la ligne :

```
stats                    C:\Users\p447\AppData\Local\miniconda3\envs\stats
```

Si dans la directory C:\Users\p447\AppData\Local\miniconda3\envs\stats nous enregistrons un .condarc contenant les lignes suivantes :

```
allowlist_channels:
  - conda-forge
denylist_channels:
  - defaults
```

alors nous interdisons à cet environnement d'accéder aux default channels et ne lui autorisons que conda-forge.

## 4.2- La résolution des dépendances – recours au solveur Libmamba

Si après l'installation d'un package vous faites un conda list, vous pouriez être surpris du nombre de programmes qui ont été installés aux côtés de celui initialement demandé. Ceci vient du fait que votre package possède des dépendances qui doivent également être installées pour son bon fonctionnement. Dans un environnement de travail le nombre de dépendances peut très vite devenir élevé et le problème de résolution des compatibilités des dépendances croisées devient particulièrement complexe. Dans ce cas le solveur de Conda, reconnu pour sa fiabilité, a l'inconvénient d'être lent. Aussi, si vous passez plusieurs minutes ou dizaine de minutes à attendre que les actions d'installation ou de mises à jour s'achèvent, il peut être temps de vous tourner vers une autre solution entièrement compatible avec Conda mais beaucoup plus rapide : le solveur de Libmamba.

Sa mise en œuvre est particulièrement simple :

1. Comme toujours avant une installation on installe si nécessaire la dernière version de Conda :

   ```
   conda update -n base conda
   ```

2. On installe le solveur Libmamba compatible avec Conda :

   ```
   conda install -n base conda-libmamba-solver
   ```

3. On le définit comme solveur par défaut dans .condarc :

   ```
   conda config --set solver libmamba
   ```

On peut retourner si on le désire au solveur de Conda avec :

```
conda config --set solver classic
```

## 4.3- Travailler avec Conda et PIP

PIP est un gestionnaires de paquets Python qui est natif : il est installé en même temps que Python et accède au dépôt PYPI qui propose plus de choix et des versions de packages souvent plus récentes que conda-forge et les installe généralement plus rapidement. A l'origine, il ne gérait que des packages Python alors que Conda pouvait installer des programmes binaires précompilés mais aujourd'hui, grâce aux wheels, il peut installer des programmes ayant des extensions écrites par exemple en C ou C++ compilées et exécutables avec Python. Même si chaque wheel exige une version de Python, une compatibilité avec l'API CPython, une plateforme spécifique (Windows, Macos, Linux), on peut maintenant les trouver pour la plupart des packages, et de manière transparente PIP descend s'il le trouve le wheel correspondant aux contraintes de votre machine. De ce point de vue, l'avantage relatif de Conda a donc diminué.

PIP n'est pas un gestionnaire d'environnement, la gestion de ceux-ci est déléguée à un autre programme (venv, virtualenv,…), en revanche Conda prend en compte l'ensemble de l'environnement (Python, packages, librairies système) et étudie toutes les contraintes (version et compatibilité) pour chaque demande d'installation. PIP par exemple ne vérifie pas les dépendances système et installe les paquets dans l'ordre demandé ce qui peut être source de conflits ou d'erreurs

d'environnement. En conséquence, on admet généralement que le solveur de Conda est plus robuste que celui de PIP.

Pour résumer, les projets qui bénéficient le plus de Conda sont ceux qui ont des dépendances complexes tout particulièrement sur des bibliothèques non-Python, multi-langages, surtout en data science, ou nécessitant un environnement GPU (CUDA par exemple).

Il peut arriver que l'on ait besoin de recourir à PIP lorsque l'on travaille avec Conda ; par exemple on veut utiliser un package présent dans PYPI mais absent de conda-forge ou des default channels. La bonne nouvelle est que cela est possible simplement mais que des précautions doivent être prises, et être conscient du fait que le risque de casser l'environnement de travail augmente de toute façon. Les étapes conseillées sont les suivantes :

1. Créer un environnement conda et l'activer
2. Y installer tous les packages dont on a besoin avec `conda install` avant d'utiliser PIP et notamment faire un `conda install PIP`. Le PIP alors installé est configuré pour n'installer ses paquets que dans l'environnement actif, en revanche le PIP standard installé en même temps que Python utilisé en dehors d'un environnement actif peut modifier les packages partout dans le système
3. Faire du `PIP install` sur les packages non disponibles avec Conda
4. Vérifier que tout s'est bien passé avec un `conda list` : les packages installés avec PIP doivent apparaître dans la liste.

Normalement les packages présents dans cet environnement sont alors utilisables qu'ils proviennent de `conda install` ou de `PIP install`.

Attention : S'ils sont utilisables, les paquets installés par PIP n'en sont pas moins ignorés par Conda et par exemple, la commande

```
conda update --all
```

ne mettra à jour que ceux installés par Conda pouvant entraîner des conflits de dépendances. Pour la même raison, sur ceux installés par PIP il faut éviter de faire un

```
PIP install –upgrade <nom du package>
```

Pour garder la mémoire de cet environnement et/ ou le dupliquer, il peut être utile de générer un fichier yaml qui va le décrire plus ou moins précisément. On va présenter 3 méthodes qui peuvent être utilisées en donnant les caractéristiques du fichier yaml que chacune génère :

#### 4.3.1- command export avec description détaillée et non portabilité entre OS différents

```
conda env export > environment.yml
```

**Caractéristiques** : crée un fichier `environment.yml` très détaillé qui inclut toutes les dépendances avec builds spécifiques :

- Tous les packages installés via Conda

- Tous les packages installés via PIP (dans une section séparée)
- Les numéros de version exacts
- Toutes les dépendances avec leurs builds, un build étant un identifiant unique (exemple ; `h06a4308_0)` qui correspond à une compilation spécifique d'un package pour une plateforme donnée, ce qui explique la non portabilité entre OS différents
- Le nom de l'environnement

Le fichier `environment.yml` ressemblera à ceci :

```
name: myproject
channels:
  - conda-forge
  - defaults
dependencies:
  - _libgcc_mutex=0.1=main
  - blas=1.0=mkl
  - ca-certificates=2023.08.22=h06a4308_0
  - certifi=2023.7.22=py39h06a4308_0
  - intel-openmp=2023.1.0=hdb19cb5_46305
  - ld_impl_linux-x64=2.38=h1181459_1
  - mkl=2023.1.0=h213fc3f_46343
  - numpy=1.24.3=py39hf6e8229_1
  - pandas=2.0.3=py39h1128e8f_0
  - pip=23.2.1=py39h06a4308_0
  - python=3.9.18=h955ad1f_0
  - zlib=1.2.13=h5eee18b_0
  - pip:
    - charset-normalizer==3.2.0
    - idna==3.4
    - requests==2.31.0
    - urllib3==2.0.4
prefix: /home/user/anaconda3/envs/myproject
```

4.3.2- command export avec bonne portabilité entre plateformes mais perte de précision

```
conda env export --from-history > environment.yml
```

**Caractéristiques** : Minimaliste, très portable ne garde que les packages explicitement installés par l'utilisateur avec des numéros de version parfois absents ou simplifiés.

L'argument `--from-history` impose que seuls les packages listés dans une commande `install` associée à Conda et/ou PIP sont renseignés, leur dépendances ne le sont pas ce qui impose de les résoudre lors de la recréation de l'environnement initial.

Le fichier `environment.yml` résultant ressemblera à :

```
name: myproject
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3.9
  - numpy
  - pandas
  - pip
  - pip:
    - requests
prefix: /home/user/anaconda3/envs/myproject
```

#### 4.3.3- command export avec moindre portabilité entre plateformes mais gain en précision

```
conda env export --no-builds > environment.yml
```

**Caractéristiques** : Compromis entre les deux commandes précédentes - garde toutes les dépendances avec leurs versions, mais retire les builds spécifiques à la plateforme.

Elle génère `environment.yml` un de la forme :

```
name: myproject
channels:
  - conda-forge
  - defaults
dependencies:
  - _libgcc_mutex=0.1
  - blas=1.0
  - ca-certificates=2023.08.22
```

```
  - certifi=2023.7.22
  - intel-openmp=2023.1.0
  - ld_impl_linux-x64=2.38
  - mkl=2023.1.0
  - numpy=1.24.3
  - pandas=2.0.3
  - pip=23.2.1
  - python=3.9.18
  - zlib=1.2.13
  - pip:
    - charset-normalizer==3.2.0
    - idna==3.4
    - requests==2.31.0
    - urllib3==2.0.4
prefix: /home/user/anaconda3/envs/myproject
```

#### 4.3.3- Création d'un environnement à partir du fichier `environment.yml`

- Nouvel environnement sous le nom de celui de départ :
  `conda env create -f environment.yml`
- Nouvel environnement avec un nouveau nom :
  `conda env create -f environment.yml -n nouveau_nom`