Introduction to containers - Conda

John A. Juma

Animal and Human Health

International Livestock Research Institute (ILRI)

March, 2023

j.juma@cgiar.org

Objectives

- Understand why you should use a package and environment management system and benefits of Conda
- Working with environments
- Using packages and channels
- Sharing environments
- Setup a conda environment

Packages and Environments

When you begin working with a programming language such as Python, you may begin to wonder why such a language can do almost everything. The fact is that Python does not perform everything on its own. It depends on external libraries or packages to enable it perform the functions you desire it to do.

1. A **module** is a collection of functions and variables, e.g., in a script
2. A **package** is a collection of modules with an initialization function (e.g., init.py) e.g., a directory with scripts
3. A **library** is a collection of packages with related functionality

In **R – packages** while in **Python – libraries**. Can be used interchangeably to refer to the same thing.

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

Dependencies

Things get complicated when you realize that many packages do not just do everything on their own.

- **Packages depend on other packages for their functionality.**

- For example, the **Scipy** package is used for numerical routines in Python. The package does not reinvent the wheel in its functionality since it uses other packages such as **numpy** (numerical python) and **matplotlib** (plotting) among others. Here **numpy** and **matplotlib** are dependencies of scipy.

- Further developments in packages and these developments may **add** or **delete** functionalities leading to different **versions**. If one package can depend on the other, this may create issues.

- Its important to know that for the example above, scipy depends on **numpy version >= 1.6** and **matplotlib version >= 1.1** (If the numpy version is 1.5, this would be insufficient for scipy).

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

Environments

When beginning your programming journey, you will realize that you don’t need many packages. However, as you progress, at some point a package or a programming language is phased out.

A workaround would be to have another computer or virtual machine to run older versions of the packages.

A practical solution is to use environments which act as isolated computers with project-specific packages.

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

Environment management

An environment management system solves several problems including:

1. An application you need for a research project requires different versions of packages or base programming language (e.g., python2.7 versus python3.6)
2. An application you developed as part of previous research project that worked fine on your system six months ago now no longer works.
3. Code that was written for a joint research project works on your machine but not on your collaborators’ machines
4. An application that you are developing on your local machine does not provide similar results when run on a remote cluster.

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

Environment management

An **environment management system** enables you to set up a **new, project specific software environment** containing **specific Python versions** as well as the **versions of additional packages and required dependencies** that are all **mutually compatible**.

- Environment management systems help **resolve dependency issues** by allowing you to use different versions of a package for different projects.
- Make your projects **self-contained and reproducible** by capturing all package dependencies in a single requirements file.
- Allow you to **install packages on a host** on which you do not have admin privileges.

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

Environment management

An **environment management system** enables you to set up a **new, project specific software environment** containing **specific Python versions** as well as the **versions of additional packages and required dependencies** that are all **mutually compatible**.

- Environment management systems help **resolve dependency issues** by allowing you to use different versions of a package for different projects.
- Make your projects **self-contained and reproducible** by capturing all package dependencies in a single requirements file.
- Allow you to **install packages on a host** on which you do not have admin privileges.
- You can manage python packages using **virtualenv, pipenv, venv, pyenv**

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

Package management

A good package management system greatly simplifies the process of installing software by:

1. identifying and **installing compatible versions** of software and all required **dependencies**.
2. handling the process of **updating software** as more recent versions become available.

Why should I use a package and environment management system

Installing software system-wide has several drawbacks:

- It can be **difficult to figure out** what software is required for any research project.
- It is often **impossible to install different versions** of the same software package at the same time.
- Updating software required for one project can often "**break**" the software installed for another project.

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

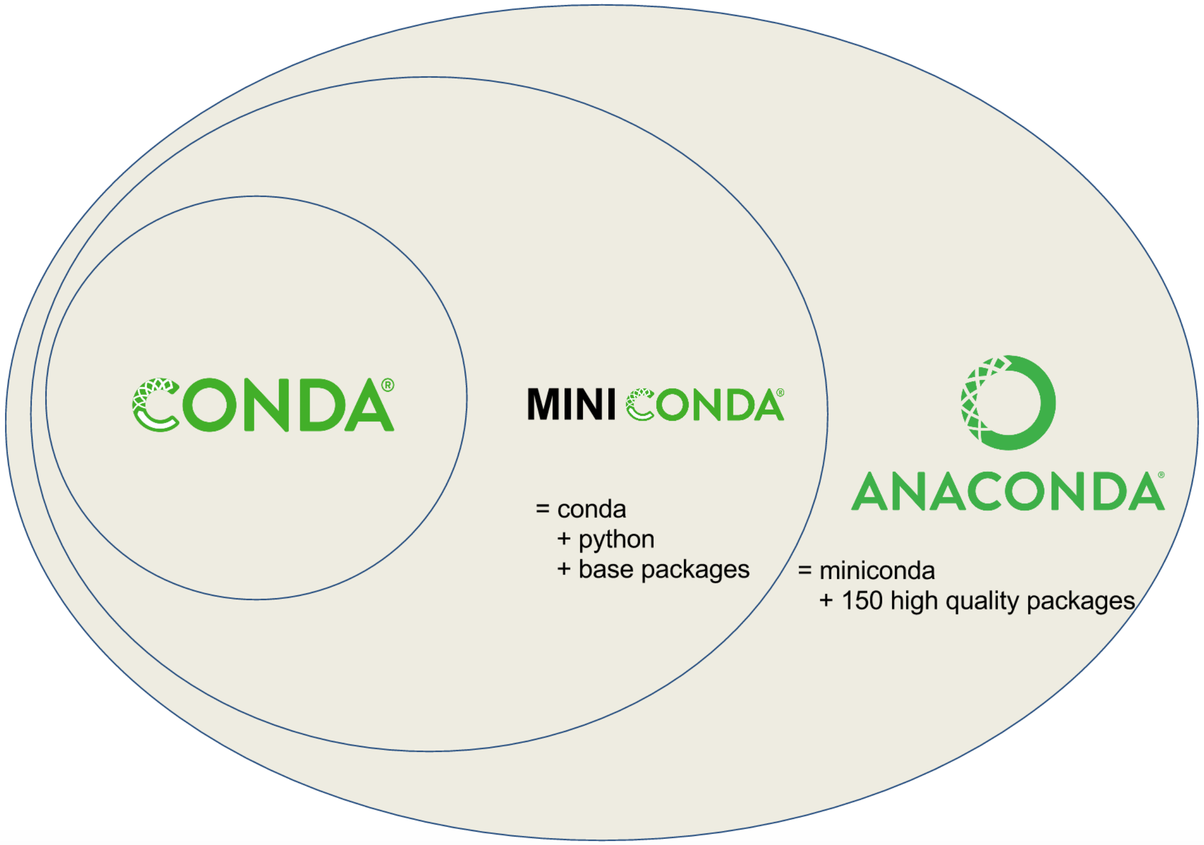
Conda

Conda is an open-source package and environment management system that runs on Windows, Mac OS and Linux.

- Conda can quickly install, run, and update packages and their dependencies.
- Conda can create, save, load, and switch between project specific software environments on your local computer.
- Although Conda was created for Python programs, Conda can package and distribute software for any language such as R, Ruby, Lua, Scala, Java, JavaScript, C, C++, FORTRAN.

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

Conda vs miniconda vs anaconda



Conda is a tool for managing environments and installing packages.

Miniconda combines Conda with Python and a small number of core packages.

Anaconda includes Miniconda as well as a large number of the most widely used Python packages.

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

Why use conda

- Conda provides prebuilt packages, avoiding the need to deal with compilers, or trying to work out how exactly to set up a specific tool.
- Conda is cross platform, with support for Windows, MacOS, GNU/Linux, and support for multiple hardware platforms, such as x86 and Power 8 and 9.
- Conda allows for using other package management tools (such as pip) inside Conda environments, where a library or tools is not already packaged for Conda.

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

What is a conda environment

A Conda environment is a directory that contains a specific collection of Conda packages that you have installed.

Conda has a default environment called **base** that include a Python installation and some core system libraries and dependencies of Conda. It is a “best practice” to avoid installing additional packages into your base software environment.

Additional packages needed for a new project should always be installed into a newly created Conda environment.

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

Creating conda environments

To create a new environment for Python development using conda you can use the `conda create` command.

```
$ conda create --name python3-env python
$ conda create --name python36-env python=3.6
```

Explicitly specify the version number for each package that you install into an environment

Search packages

```
$ conda search $PACKAGE_NAME
$ conda search scikit-learn
```

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

Creating conda environments

Installing multiple packages

```
$ conda create --name basic-scipy-env ipython=7.13 matplotlib=3.1 numpy=1.18 scipy=1.4
```

Creating an environment

Activating an environment

1. Add entries to the PATH for the environment
2. Runs activation scripts that the environment may contain.

```
$ conda activate basic-scipy-env
```

Deactivating an environment

```
$ conda deactivate
```

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

Installing package(s) into existing environment

You can install a package into an existing environment using the `conda install` command.

This command accepts a list of package specifications (i.e., **numpy=1.18**) and installs a set of packages consistent with those specifications *and* compatible with the underlying environment. If full compatibility cannot be assured, an **error** is reported, and the environment is ***not*** changed.

```
$ conda activate basic-scipy-env
$ conda install numba
```

To prevent existing packages from being updating when using the `conda install` command, you can use the `--freeze-installed` option.

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

Installing package(s) into existing environment

You can install a package into an existing environment using the `conda install` command.

This command accepts a list of package specifications (i.e., **numpy=1.18**) and installs a set of packages consistent with those specifications *and* compatible with the underlying environment. If full compatibility cannot be assured, an **error** is reported, and the environment is ***not*** changed.

```
$ conda activate basic-scipy-env
$ conda install numba
```

To prevent existing packages from being updating when using the `conda install` command, you can use the `--freeze-installed` option.

https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/

Exercises