Installing OCW using the conda package manager

Introduction

conda is an easy to use and maintain software package management system. It provides a simple and painless method for installing OCW and its dependencies across a variety of platforms.

OCW Installation Instructions

Requirements

- The latest version of conda. The recommended way to obtain this is to install either the Anaconda or Miniconda scientific python distributions. Be sure to allow the installation to update your PATH for you.

Installation Information

In addition to the standard scientific python packages bundled with most anaconda installations, the following packages will be installed:

- **requests:** Used to make HTTP requests. The OCW UI and Toolkit use this.
- **bottle:** Simple Python backend webserver used by the OCW UI.
- **pydap:** A Python library for connecting to OpenDAP servers. The Toolkit uses this to handle OpenDAP connections.
- **pyesgf:** A Python library for downloading files from the Earth System Grid Federation database.

How To

Before invoking conda to install the dependencies, it's always a good idea to ensure that your version of conda is up to date. You can do this with:

```
conda update conda
```

Then to install OCW and its dependencies into your conda environment (by default in **~/anaconda** or **~/miniconda**), you may use:

```
conda config --add channels conda-forge
conda install ocw
```

Testing

To give yourself some peace of mind that such a simple installation method actually works, it is recommended that you test you installation. The main OCW codebase has some simple examples which may be obtained using:

```
cd
git clone https://github.com/apache/climate.git
```

If you do not wish to directly use git, you may download and extract a copy of the code directly from our GitHub page. Once you've obtained the code, head on over to the **examples** and run one of them.

```
cd climate/examples
python simple_model_to_model_bias.py
```

After the evaluation runs you should find a **.png** file in the examples directory. Congratulations, your install was successful!

Fetching Updates and changing versions

Keeping OCW up to date is even more simple than the installation itself. Use:

```
conda update ocw
```

What if you wish to use a particular release of OCW, say version 1.1.0? Just specify the version in the install command as follows:

```
conda update ocw=1.1.0
```

Creating a clean environment

It should be noted that the above instructions will install OCW and its dependencies into your default conda environment. For those users who wish to perform development and testing in a clean and isolated environment, they should instead perform their installations using:

```
conda create -n ocw ocw
```

This will create a new virtual environment named "ocw" with only OCW and its dependencies installed. To activate this environment, use:

Linux/OSX

```
source activate ocw
```

Windows

```
activate ocw
```

To deactivate the environment, use:

Linux/OSX

```
source deactivate
```

Windows

```
deactivate
```

See this page for further documentation on virtual conda environments.

Maintaining the conda packages (for advanced users)

The following section contains some useful information for those in the OCW developer community who are interested in maintaining the conda packages.

Although not required, we recommend reading the conda build documentation first.

Getting Started

We use conda-forge to host the ocw package. To get started, you should fork our feedstock repo. The only file you will need to concern yourself with is **reci pe/meta.yaml**.

The meta.yaml recipe file

There are only a few entries in this file that should be updated. These are as follows:

Package Version

This should be in the form X.Y.Z which corresponds to the latest release. Note that the ocw source distribution for the given version must first be uploaded to PyPI before making this change, or else conda will fail to build the package. This is main field that you need to change when creating a new release.

Ex:

```
{% set version = "X.Y.Z" %}
```

Where X.Y.Z should be replaced by the version number of the release.

Hash

We use sha256 to populate the hash field. This too must be changed with every new release. To do this, you'll need to make sure OpenSSL is installed on your machine:

```
conda install openssl
```

Then to generate the checksum, you may use:

```
pip download ocw
openssl sha256 ocw-X.Y.Z.tar.gz
```

Where X.Y.Z should be replaced by the version number of the release. Then you may use the resulting checksum to set the hash in the recipe file. For example:

```
{% set sha256 = "8f12405ec02ea327b1e0cf65547e409232f67577be6eb348d2ea9011827e2c4a" %}
```

Build Section

A few notes:

- **number** should be set to 0 for all new releases. For updating an existing release, increment this by 1 for each update.
- **skip** Should only be set to True for unsupported platforms. This can be achieved using preprocessing selectors. For example, to skip building for Python 3, you would use:

  ```
  skip: True  # [py3k]
  ```

  Changes in compatibility between platforms or python versions may require you to update the CI build configuration files as well. See re-rendering the feedstock for more information.
- **Do not change the script field!**

Requirements Section

Every listed dependency *must* be available on some anaconda channel, preferably either the default anaconda channel or the conda-forge channel. If not, consider adding a recipe by submitting a pull request to the conda-forge staged-recipes repo! See the README in the repo for more detailed guidelines.

This section lists all of the dependencies needed for ocw at build time (not really important since we don't use any extensions) and runtime (much more important). As with the **skip** field in the build section, certain dependencies can and should be excluded if they are not supported on certain platforms. For example, if a package named "foo" is not compatible with Windows, it may be excluded using:

```
    - foo # [not win]
```

Maintainers Section

This section contains a list of GitHub usernames which have commit rights to the ocw-feedstock repo. If you are a regular contributor to the project and want to help maintain the recipe file, you should add your own GitHub username to this list.

Guidelines for updating the recipe

- Any updates to the recipe should be done through a pull request to the feedstock repo.
- All changes should be done from your own fork (not on an upstream branch).
- Finally, make sure that all of the CI builds (Circle, AppVeyor, and Travis) are passing before merging the PR. While this might seem inconvenient for committing very trivial changes to the recipe, it is necessary because it is the very same CI services which automate the process of building and uploading the binary package files to the conda-forge channel.

Re-rendering the Feedstock

Introduction

So far in our discussion of packaging we have only needed to be concerned with the meta.yaml recipe file. However it is also important to understand that conda-forge, in its efforts to support as many platforms as possible, uses CI services (remote virtual machines) to actually build the packages from the recipe. These are Circle (GNU/Linux), AppVeyor (Windows) and Travis (OSX), which in the feedstock repo are configured respectively in **circle.yml**, **appveyor.yml**, and **.travis.yml**.

Generally speaking, the conda-forge webservices should automatically update these files as needed. This is typically done with periodic "maintenance" pull requests in the feedstock repo made by the **conda-forge-admin** account (which maintainers of this project are then responsible for merging). We refer to this process as **re-rendering** the feedstock, as the actual CI configuration files are generated from the jinja2 templates found here.

Manually re-rendering the feedstock

While the process of re-rendering should happen automatically most of the time (as described above), there are a few situations where it will be necessary for you to do it manually. These can include but are not limited to the following:

- Changing supported platforms (eg, GNU/Linux, OSX, and Windows)
- Changing supported python versions (2.7, 3.5, and/or 3.6)
- Your CI builds have errors which are not related to the recipe file itself (very rare, here is an example)

In order to rerender the feedstock, you will need to use conda-smithy. Please follow the instructions in the link and make sure your version of conda-build is up to date as well. We also recommend using the --commit option with conda-smithy as this will automatically commit the re-rendered files to the feedstock repo with the conda-smithy version in the commit message. Afterwards you may push the changes to your fork and make a pull request (or to your release branch if you are doing the re-rendering on top of a new release).