When working with Python applications, there are two important tasks to consider:

1. How to manage Python packages.
2. How to configure environments for Python applications.

As everyone knows, owning abundant Python application packages is the key to success for the Python community. To make the most of various Python application packages, you need a handy package manager, such as pip or easy_install. For now, the most popular Python package manager is pip.

Additionally, due to the incompatibility among different major versions of Python -- 3.x, 2.7, and 2.6, as well as the resulting incompatibility among various dependencies, you should always prepare a proper environment for each of your Python applications. Virtualenv provides a feasible solution to this issue: constructing a dedicated and isolated Python environment for each of your Python applications. Every application can enjoy the most suitable Python environment without messing up other applications' environments.

In this article, we will introduce to you how to use pip and virtualenv to manage Python packages and environments on a CentOS 6 server instance.

Before moving on, you need to:

1. Setup a Vultr CentOS 6 x64 server instance from the ground up, and
2. Create a non-root user who has sudo privileges and log in with it.

Install and use pip

First, let's have a look at pip. Install the latest pip with the following commands:

sudo yum update sudo yum install -y python-devel python-setuptools python-pip sudo pip install --upgrade pip

Once pip has been installed, you will be able to use it to manage Python packages, including but not limited to searching for, installing, upgrading, and uninstalling Python packages. In order to give you some hands-on instructions, I will list some common pip commands below:

Search for a Python package using pip
pip search [package name] **Install a Python package using pip**

a) Install a package by the package name:

sudo pip install [package name]

b) Install a specific version of a Python package:

sudo pip install [package name]==[version]

c) Install a Python package from a URL:

sudo pip install [URL] **List Python packages installed with pip**
pip list **Show the details of a Python package installed with pip**
pip show [package name] **Upgrade a Python package using pip**
sudo pip install --upgrade [package name] **Uninstall a Python package using pip**
sudo pip uninstall [package name] **Display pip help**
pip help **Install and use virtualenv**

As previously mentioned, the incompatibility among different dependencies is an issue worthy of your concern.

In order to avoid issues that occur due to incompatibilities, you can use virtualenv to prepare a virtual environment to contain the suitable dependencies for each of your Python applications. In this fashion, incompatible dependencies can coexist without conflict, and Python applications depending on them can coexist without conflict as well.

An additional benefit of using virtualenv is that you don't need root/sudo privileges to modify dependencies in the virtual environment, because every operation is performed in the current user's own directory.

Now, let's explore the virtual environment created by virtualenv.

1. Install virtualenv using pip
sudo pip install virtualenv **2. Create a dedicated virtual environment**

Before you deal with a new Python application, you can use virtualenv to create a dedicated directory—a virtualenv environment—to store your following modifications to the system dependencies.

Say that you want to use a directory "env1" under your home directory to contain the virtual environment:

```
cd ~ virtualenv env1
```

The two commands above will create the environment directory "env1" in your home directory and initiate the virtual environment in it, namely copy the global/system Python environment you are using into your virtual environment directory and adjust related configurations, making it an isolated Python environment.

Now, you need to activate the virtual environment:

```
source ~/env1/bin/activate
```

As you see, a string env1 will be inserted in front of your shell prompt, notifying you that you have entered the isolated virtual environment.

You can use the command which python to confirm your entrance. The system will tell you that you are using ~/env1/bin/python rather than the original /usr/bin/python.

From now on, you can deal with your Python application as you wish, all of your modifications to system dependencies will be recorded in this directory, avoiding the potential tampering to other Python applications.

3. Exit the virtual environment

After finishing your tasks, use the following command to exit the virtual environment:

```
deactivate
```

The string (env1) will disappear accordingly.

If you want to know more about virtualenv, use the following command:

```
virtualenv --help
```