Arroved third party: https://cloud.google.com/appengine/docs/python/tools/built-in-libraries-27

Original Link: https://cloud.google.com/appengine/docs/python/tools/using-libraries-python-27

Using Built-in Libraries in Python 2.7

This page describes how to use supported libraries in the Google App Engine Python 2.7 runtime environment. By default, this runtime environment includes the Python standard library, the App Engine libraries, and a few bundled third- party packages. For a complete list of runtime-provided libraries, see the built-in third-party libraries reference.

Adding libraries

You can add a third-party library to your app in one of two ways: requesting the library or installing the library.

Requesting a library

You can request a library by using the `libraries:` directive in `app.yaml`.

```
libraries:
- name: PIL
  version: "1.1.7"
- name: webob
  version: "1.1.1"
```

Note that:

- The library must be one of the supported runtime-provided third-party libraries.
- When deployed, App Engine will provide the requested libraries to the runtime environment.
- Some libraries must be installed locally.

Installing a library

You can install the library into a folder in your project's source directory. The library must be implemented as pure Python code with no C extensions. The code is uploaded to App Engine with your application code, and counts towards file quotas.

The easiest way to manage this is with a `./lib` directory:

1. Use pip to install the library and the vendor module to enable importing packages from the third-party library directory.
2. Create a directory named `lib` in your application root directory:

   ```
   mkdir lib
   ```

3. To tell your app how to find libraries in this directory, create or modify a file named `appengine_config.py` in the root of your project, then add these lines:

   ```
   from google.appengine.ext import vendor

   # Add any libraries installed in the "lib" folder.
   vendor.add('lib')
   ```

4. Use `pip` with the `-t lib` flag to install libraries in this directory:

   ```
   pip install -t lib gcloud
   ```

Note: pip version 6.0.0 or higher is required for this method to work properly.**Warning:** If you are using Homebrew Python on OS X, you might encounter an exception when running **`pip install -t`**. This problem is related to a known issue with Homebrew's configuration of Python. The issue description has a workaround.

The `appengine_config.py` file above assumes that the current working directory is where the `lib` folder is located. In some cases, such as unit tests, the current working directory can be different. To avoid errors, you can explicity pass in the full path to the `lib` folder using:

```
vendor.add(os.path.join(os.path.dirname(os.path.realpath(__file__)),
'lib'))
```

Using `pip` requirements files

`pip` can read a list of libraries to install from a file, known as a *requirements file*. Requirements files make it easy to set up a new development environment for your app, and upgrade to new versions of libraries.

A requirements file is a text file with one line per library, listing the package name and version:

```
Flask==0.10
Markdown==2.5.2
google-api-python-client
```

To install the libraries from a requirements file, use the `-r` flag in addition to the `-t lib` flag:

```
pip install -t lib -r requirements.txt
```

Using libraries with the local development server

While several of the runtime-provided libraries are available to your local development environment through the App Engine SDK, the following libraries are platform-dependent and must be installed locally before you can use them with the development server:

- lxml
- matplotlib
- mysqldb
- numpy
- PIL
- crcmod
- pycrypto

You can use the pip command to install all of these packages from the Python package index (PyPI).

```
sudo pip install lxml==2.3.5
```

Depending on your platform, you might need to install build support tools and Python sources to install these libraries.

- On Linux, the package manager can provide these prerequisites and can often provide a pre-built version of the library.
- On Windows, installers for pre-built versions are usually available. *On OS X, the Xcode Command Line Tools are required to build some packages.

Note: The development server uses the package version you have installed locally regardless of the version specified in`app.yaml`. If desired, you can set up a virtualenv for your project to provide the exact package version. Note that the virtualenv is only used for these binary packages locally and will not be made available to your application once deployed. To add additional third-party libraries, use the vendoring method described in Installing a library.

Using Django or `matplotlib`

This section provides information you should know when using the Django or `matplotlib` libraries.

Using Django

Warning: Support for Django versions 1.2 and 1.3 is deprecated and will be removed. See the Django 1.2, 1.3 Turndowndocument for details and timetable.

Django is a full-featured web application framework for Python. It provides a full stack of interchangable components, including dispatch, views, middleware, and templating components, and many others.

The Django data modeling interface is not compatible with the App Engine datastore. You can use the App Engine data modeling libraries (db or ndb) in your Django applications. However, third-party Django applications that use the Django data modeling interface, most notably Django's Admin application, might not directly work with App Engine.

The Datastore modeling library (DB) is the default. To use Django with the NDB storage API instead, add`'google.appengine.ext.ndb.django_middleware.NdbDjangoMiddleware',` to the `MIDDLEWARE_CLASSES` entry in your Django`settings.py` file. It's a good idea to insert it in front of any other middleware classes, since some other middleware might make datastore calls and those won't be handled properly if that middleware is invoked before this middleware. You can learn more about Django middleware in the project documentation.

To enable Django in your app, specify the WSGI application and Django library in `app.yaml`:

```
...
handlers:
- url: /.*
  script: main.app  # a WSGI application in the main module's global
scope

libraries:
- name: django
  version: "1.4"
```

The `DJANGO_SETTINGS_MODULE` environment variable must be set to the name of your Django settings module, typically `'settings'`, before packages are imported.

If your Django settings module is something other than `settings.py`, set the `DJANGO_SETTINGS_MODULE` environment variable accordingly either in your `app.yaml` file:

```
env_variables:
  DJANGO_SETTINGS_MODULE: 'myapp.settings'
```

Or in your Python code:

```
import os
# specify the name of your settings module
os.environ['DJANGO_SETTINGS_MODULE'] = 'myapp.settings'

import django.core.handlers.wsgi
app = django.core.handlers.wsgi.WSGIHandler()
```

Using `matplotlib`

Note: The experimental release of matplotlib is not supported on the development server. You can still add **`matplotlib`** to the **`libraries`** list, but it will raise an **`ImportError`** exception when imported.

Matplotlib is a plotting library that produces graphs and figures in a variety of image formats. On App Engine, the interactive modes of matplotlib are not supported, and a number of other features are also unavailable. This means you cannot use `pyplot.show()` as many matplotlib tutorials suggest. Instead, you should use `pyplot.savefig()` to write image data to theoutput stream, a`cStringIO.StringIO` instance, or the Google Cloud Storage using the Cloud Storage Client Library.

Matplotlib allows extensive customization through the use of the `matplotlibrc` configuration file, which should be placed in the application's top-level directory. Alternatively, you can set the `MATPLOTLIBRC` environment variable to a path relative to your application's directory.

The default backend is AGG, which allows writing files of all supported formats: PNG (the default format), RAW, PS, PDF, SVG and SVGZ. If you make the PIL library available by adding `PIL` to the `libraries` section of `app.yaml`, then the AGG backend will automatically support writing JPEG and TIFF image formats as well.

Matplotlib comes with a number of fonts which are automatically available. You can use custom fonts by uploading them in TTF format along with your application, and setting the `TTFPATH` environment variable to the path where they are located, relative to your application's directory. For more information, see the `app.yaml` reference.

A number of matplotlib features are not supported on App Engine. In particular:

- There is no `~/.matplotlib` directory. However, there are alternative locations to place the `matplotlibrc` configuration file, as described above.
- Interactive backends and GUI elements are not supported.
- The EMF, Cairo and GDK backends are not supported.
- There is no caching, and therefore a number of mechanisms will re-calculate or re-download data that would normally be cached. Specific caching mechanisms that have been disabled include font data calculated by`matplotlib.font_manager.FontManager.findfont`, sample data downloaded by `matplotlib.cbook.get_sample_data`and financial data downloaded by `matplotlib.finance.fetch_historical_yahoo`.
    - Because there is no caching, it is not possible to call `[matplotlib.cbook.get_sample_data](http://matplotlib.org/api/cbook_api.html#matplotlib.cbook.get_sample_data)` with `asfileobj=False`unless `examples.download` is set to `False`.
- All features that invoke external commands have been disabled.
    - Use of `fontconfig` has been disabled. Fonts are found through the mechanism described above.
    - Use of LaTeX for text rendering is not supported. Setting `text.usetex` to `True` will not work.
    - Use of an external PostScript distiller program is not supported. Setting `ps.usedistiller` to `ghostscript` or `xpdf` will not work.
    - Use of an external video encoding program is not supported. The `matplotlib.animation.Animation.save` method will not work, and therefore, the `matplotlib.animation` package is not useful.
    - The `matplotlib.cbook.report_memory` function and `matplotlib.cbook.MemoryMonitor` class are not supported.
- The `matplotlib.test` function has been disabled.

Note: The **`pylab`** and **`matplotlib.pyplot`** modules are stateful and not thread safe. If you use them on App Engine, you must set **`threadsafe: false`** in **`app.yaml`**, and be aware that the plotter state will be preserved between requests on the same instance. For example, you will need to call **`pyplot.clf()`** at the beginning of each request to ensure that previous plots are not visible. It is recommended that you use the thread-safe object-oriented API instead of the stateful pyplot API.