Python Packages in OpenWhisk

Apr 27, 2017
openwhisk **python** **serverless**
6 min read

OpenWhisk's Python runtime includes popular third-party libraries like requests, scrapy and simplejson. Developers don't have to manually install packages to use those libraries.

Great, but what about using other libraries that aren't pre-installed?

In a previous blog post, we showed how to deploy Node.js actions from zip files containing third-party modules. These modules are then made available in the Node.js runtime.

Recent updates to OpenWhisk allow us to use the same approach with the Python runtime!

Python Packages

Python packages can be installed using the pip tool. This can be used to install individual packages or a series of dependencies from an external file.

```
$ pip install blah
$ pip install -r requirements.txt
```

pip defaults to installing packages in a global location (site-packages) which is shared between all users. This can cause issues when different projects require different versions of the same package.

virtualenv

virtualenv is a tool that solves this issue by creating virtual python environments for projects. The virtual environment includes a custom `site-packages` folder to install packages into.

```
$ virtualenv env
Using base prefix '/Library/Frameworks/Python.framework/Versions/3.6'
New python executable in /private/tmp/env/bin/python3.6
Also creating executable in /private/tmp/env/bin/python
Installing setuptools, pip, wheel...done.
```

OpenWhisk recently added support for using virtualenv in the Python runtime.

custom packages on openwhisk

OpenWhisk actions can be created from a zip file containing source files and other resources.

If the archive includes a virtual Python environment folder, the platform runs the `./bin/activate_this.py` script before executing Python actions. This script modifies the module search path to include the local `site-packages` folder.

This will only happen during "cold" activations.

This feature comes with the following restrictions.

- Virtual Python environment must be in a folder called `virtualenv` under the top-level directory.
- Packages must be available for the Python runtime being used in OpenWhisk (2.7 or 3.6).

Let's look at an example of building an OpenWhisk Python action which uses an external Python package.

Python Package Example

The pyjokes package provides a library for generating (terrible) jokes for programmers. Let's turn this package into an API (Jokes-as-a-Service!) using the Python runtime on OpenWhisk.

Start by creating a new directory for your project and set up the virtual Python environment.

```
$ mkdir jokes; cd jokes
$ virtualenv virtualenv
Using base prefix '/Library/Frameworks/Python.framework/Versions/3.6'
New python executable in /tmp/jokes/virtualenv/bin/python3.6
Also creating executable in /tmp/jokes/virtualenv/bin/python
Installing setuptools, pip, wheel...done.
$ source virtualenv/bin/activate
(virtualenv) $ pip install pyjokes
Collecting pyjokes
  Using cached pyjokes-0.5.0-py2.py3-none-any.whl
Installing collected packages: pyjokes
Successfully installed pyjokes-0.5.0
(virtualenv) $
```

In the project directory, create a new file (`__main__.py`) and paste the following code.

```
import pyjokes

def joke(params):
    return {"joke": pyjokes.get_joke()}
```

Check the script works with the Python intepreter.

```
(virtualenv) $ python -i .
>>> joke({})
```

```
{'joke': 'What do you call a programmer from Finland? Nerdic.'}
>>>
```

Add the `virtualenv` folder and Python script to a new zip file.

```
$ zip -r jokes.zip virtualenv __main__.py
  adding: virtualenv/ (stored 0%)
  adding: virtualenv/.Python (deflated 65%)
  adding: virtualenv/bin/ (stored 0%)
  adding: virtualenv/bin/activate (deflated 63%)
  ...
$ ls
__main__.py  jokes.zip   virtualenv
```

Create a new OpenWhisk action for the Python runtime using the `wsk` cli.

```
$ wsk action create jokes --kind python:3 --main joke jokes.zip
ok: created action jokes
```

Invoking our new action will return (bad) jokes on-demand using the third-party Python package.

```
$ wsk action invoke jokes --blocking --result
{
    "joke": "Software salesmen and used-car salesmen differ in that the
latter know when they are lying."
}
```

Installing Packages With Docker

In the example above, the Python runtime used in development (v3.6) matched the OpenWhisk runtime environment. Packages installed using `virtualenv` must be for the same major and minor versions of the Python runtime used by OpenWhisk.

OpenWhisk publishes the runtime environments as Docker images on Docker Hub.

Running containers from those runtime images provides a way to download packages for the correct environment.

```
$ docker run --rm -v "$PWD:/tmp" openwhisk/python3action sh \
  -c "cd tmp; virtualenv virtualenv; source virtualenv/bin/activate; pip
install pyjokes;"
Using base prefix '/usr/local'
New python executable in /tmp/virtualenv/bin/python3.6
Also creating executable in /tmp/virtualenv/bin/python
Installing setuptools, pip, wheel...done.
Collecting pyjokes
  Downloading pyjokes-0.5.0-py2.py3-none-any.whl
```

```
Installing collected packages: pyjokes
Successfully installed pyjokes-0.5.0
$
```

This will leave you a `virtualenv` folder in the current directory with packages for the correct Python runtime.

Speeding Up Deployments

Peeking inside the `virtualenv` folder reveals a huge number of files to set up the virtual Python environment. If we just want to use a third-party package from the local `site-packages` folder, most of those files are unnecessary.

Adding this entire folder to the zip archive will unnecessarily inflate the file size. This will slow down deployments and increase execution time for cold activations. OpenWhisk also has a maximum size for action source code of 48MB.

Manually including individual `site-packages` folders, rather than the entire `virtualenv` directory, will ensure the archive file only contains packages being used. We must also add the Python script (`virtualenv/bin/activate_this.py`) executed by OpenWhisk to modify the module search path.

```
$ zip -r jokes_small.zip virtualenv/bin/activate_this.py virtualenv/lib
/python3.6/site-packages/pyjokes __main__.py
updating: virtualenv/bin/activate_this.py (deflated 54%)
updating: virtualenv/lib/python3.6/site-packages/pyjokes/ (stored 0%)
updating: virtualenv/lib/python3.6/site-packages/pyjokes/__init__.py
(deflated 20%)
updating: virtualenv/lib/python3.6/site-packages/pyjokes/jokes_de.py
(deflated 29%)
updating: virtualenv/lib/python3.6/site-packages/pyjokes/jokes_en.py
(deflated 61%)
updating: virtualenv/lib/python3.6/site-packages/pyjokes/jokes_es.py
(deflated 40%)
updating: virtualenv/lib/python3.6/site-packages/pyjokes/pyjokes.py (deflated
68%)
updating: __main__.py (deflated 18%)
$ ls -lh
total 40984
-rw-r--r--  1 james  wheel    74B 21 Apr 11:01 __main__.py
-rw-r--r--  1 james  wheel    20M 21 Apr 11:07 jokes.zip
-rw-r--r--  1 james  wheel   9.3K 21 Apr 13:36 jokes_small.zip
drwxr-xr-x  6 james  wheel   204B 21 Apr 11:25 virtualenv
```

The archive file is now less than ten kilobytes! 🏃

With The Serverless Framework

The Serverless Framework is a popular open-source framework for building serverless applications. This framework handles the configuration, packaging and deployment of your serverless application.

OpenWhisk is supported through a provider plugin. Recent versions of the plugin added support for the Python runtime environment.

Using the application configuration file for the framework, users can add `include` and `exclude` parameters to control the contents of the archive file before deployment.

Here's an example of the configuration needed to only include the necessary files for the application above.

```
service: pyjokes

provider:
  name: openwhisk
  runtime: python:3

functions:
  jokes:
    handler: handler.joke

plugins:
  - serverless-openwhisk

package:
  exclude:
    - virtualenv/**
    - '!virtualenv/bin/activate_this.py'
    - '!virtualenv/lib/python3.6/site-packages/pyjokes/**'
```

conclusion

Python has a huge community of third-party packages for everything from parsing JSON, making HTTP requests and even generating jokes. OpenWhisk already provided a number of the most popular packages within the Python runtime.

Users can install additional packages locally using the `pip` and `virtualenv` tools. Bundling those files within the deployment archive means they are extracted into the OpenWhisk Python runtime environment.

Recent changes to the Python runtime allows the platform to automatically add local package folders to the module search path.

This means Python functions running on OpenWhisk can now use any third-party library as if it was installed globally.

Hurrah 👌!