ActiveState®

Managing Dependencies

ActiveState Deminar

About ActiveState

- **Track-record:** 97% of Fortune 1000, 20+ years open source
- **Polyglot:** 5 languages - Python, Perl, Tcl, Go, Ruby
- **Runtime Focus:** concept to development to production

vmware

NORTHROP GRUMMAN

Rockwell Collins

Alcatel·Lucent

SIEMENS

ActiveState

Managing Dependencies and Runtime Security

Welcome

Pete Garcin, Developer Advocate, ActiveState (@rawktron)

ActiveState®

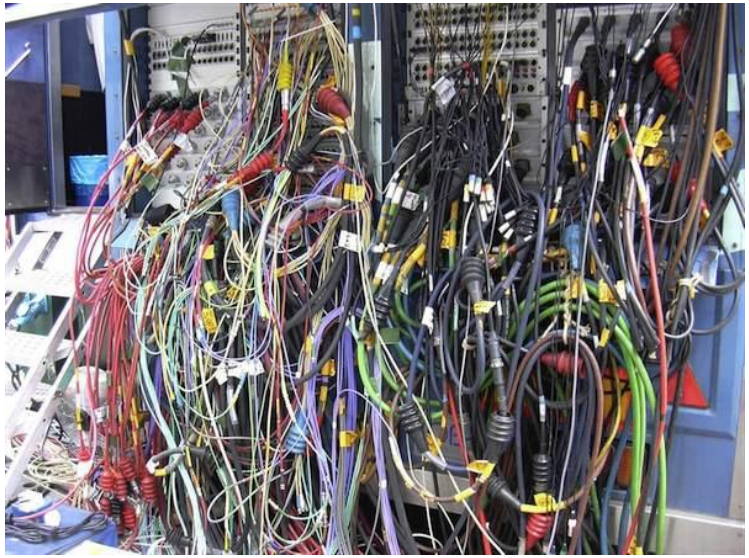
Overview

- Managing Project Dependencies
  - Pip/requirements
  - ActivePython
- Virtual Environments
  - PipEnv
- Q&A

Configuring Dev Environment

```
git clone https://github.com/ActiveState/activedeminar
```

Your dependency tree:



ActiveState®

Managing Deps

- Vendored Deps
  - Advantages: guaranteed security, compatibility, stability, availability
  - Disadvantages: larger repo, you have to manually maintain - could be out of date, conflicts with system installs

Managing Deps

- Requirements.txt/Pipfile
  - Have to ‘install’ and build from a repo BUT you don’t have to maintain the code and ship it yourself
  - You need to pin versions to prevent bleeding edge
  - Use a virtualenv for isolation

Managing Deps

- Pre-built distributions
  - No discipline approach
  - Most popular packages already pre-built, tested, and included in your distro, quarterly updates
  - As the standard install across a large org or team can work well
  - Not updated frequently
  - Not customized to your needs
  - Overall may not fit your use case

Vendoring Deps in Python

- Requires a virtualenv to prevent conflicts
- May involve generating your own wheels for local pip servers
- Not widely used
- Higher maintenance overhead
- Can be good/necessary if you have custom patches

Creating requirements.txt

- Can use “pip freeze” but this gives us everything in our system environment.
- Let’s use pipreqs:
  - https://github.com/bndr/pipreqs
  - `pip3 install pipreqs`
  - `pipreqs .`

Pinning Versions

- Pinning means forcing a specific version to be installed
- Why? Reproducible builds.
- Syntax:
  - Framework==0.9.4
  - Library>=0.2

Reproducible Builds

- Guarantee the exact same build in two locations
- Ensure you have the same versions of every package
- Requires a lockfile, or a “pip freeze”

Virtual Environments

- A Virtual Environment is a self-contained, sandboxed environment -- just for your app.
- It only has the packages you specify and they are totally distinct from the system installed ones.

Virtual Environments

- Complex but critical for app deployment, development.
- Can use ‘virtualenv’ to create and manage them but there is a new tool combining pip and virtualenv.

PipEnv

- Enter PipEnv: New “Community Standard” application combines Pip/virtualenv and extends their functionality in a single app.
- Let’s install it here:
  - https://github.com/pypa/pipenv

    ```
    pip3 install pipenv
    ```
- You can initialize a clean environment, Python 3:

  ```
  pipenv -three
  ```

Generating Pipfile

- We can generate a pipfile from our requirements.txt using the following command:

```
pipenv install
```

HANDY TIP

We can generate a virtualenv of ActivePython using:

```
pipenv
--python="/home/para
llels/AP36/bin/pytho
n3" --site-packages
install
```

Generating Pipfile

```
[[source]]
url = "https://pypi.python.org/simple"
verify_ssl = true
name = "pypi"

[packages]
numpy = "==1.14.3"
tensorflow = "==1.8.0"
Flask = "==1.0.1"

[dev-packages]

[requires]
python_version = "3.6"
```

Generating Pipfile.lock

- Generate a lockfile that contains the fully resolved dep tree for our project:

  ```
  pipenv lock
  ```

- Required for a deterministic build.
- **Warning**: could fail to resolve a dependency conflict!

Install all Dependencies

- Let's spawn a shell inside our virtualenv:

  ```
  pipenv shell
  ```

- The "sync" command will install everything in the .lock file:

  ```
  pipenv sync
  ```

Project Complete

- We now have a project that has:
  - A virtualenv created for it distinct from our system install
  - A pipfile that defines all the deps for our project generated from our requirements.txt
  - A lockfile that is a fully resolved version of all deps for this project.
  - All deps installed for our project in that virtualenv
  - Our project ready to go!

Running Project

- Remember to spawn a shell inside our virtualenv:

  ```
  pipenv shell
  ```

- We can deploy our flask server using this command:

  ```
  python3 app.py
  ```

Verify it works

- Let's check that our service is running:

```
curl http://localhost:8000?file=./mypoodle.jpg
```

Success!

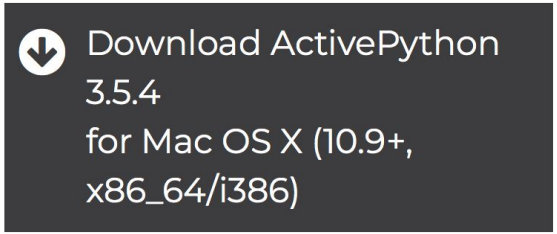
ActiveState®

Packaging and Distribution

- Further topics:
  - Generating a setup.py
  - Generating a docker image

Installing ActivePython

- Easy option: Install ActivePython (includes everything we need)
- https://www.activestate.com/activepython/downloads



ActiveState®

Future Platform Support

What if we could reduce ALL of what we just did to a single command?

Future Platform Support

- Working to streamline and simplify this process.
- Tight integration and compatibility with community tools is key.
- Share your pain points working with dependency management and environment configuration:
  - peteg@activestate.com

Future Platform Support

- Dependency Resolution.
- Reproducible Builds.
- Customized Builds/Environments.
- “One click” Environment Configuration.

- https://start.activestate.com/platform-home/

Q & A

ActiveState®

Thank you!

- Learn more about our Platform: https://www.activestate.com/platform
- Download & try our ActivePython: https://www.activestate.com/activepython
- Contact platform@activestate.com for more information.

ActiveState®