Python and venv

Python is supposed to be simple. but if you want your scripts to not break at arbitrary times, you'll need to deal with some complexity. this page explains it mostly specific to red hat enterprise linux, but it should be similar on other distros.

problem

First step when using python is imply using the system provided python. Linux distributions come with packaged python - red hat specifcally with python3 (the system default - for RHEL 8 it's python 3.6, for RHEL 9 it's python 3.9) and python36 (3.6) , 38, 39, and python3.11 specific packages.

This works fine for simple scripts without dependencies - likelyhood of something breaking when updating is extremely low while the major version stays static. When updating from RHEL 8 to 9, there can be some breakage - but a few small simple scripts will be manageable.

However, if your scripts get bigger and start including some dependencies, this stops being a good option. On a RHEL version upgrade, there's a good chance your script breaks, and dependencies might change or become unavailable - leading to a lot of effort, while you're probably already very busy with the server upgrade itself! you might get away with running your scripts with the previous standard version for a bit if available, but most likely any dependencies you had aren't available in the repos anymore, and installing system wide through pip is very much discouraged, and might install different versions of those packages too, breaking your scripts.

So we've established to not use the system default for anything bigger than the simplest of scripts. what if we just use a specific python of the available ones, python3.9 for example?

The biggest issue with that is that while most dependencies might be available as packages on the default, the 3.9 specific versions of the packages are not. only a small subset of packages are available in 3.9 on a RHEL 8 system. this brings us to the issue of system wide installation being discouraged again. so if you need any dependencies, do not use any system provided python.

With this, i think it's reasonable to say we cannot use any system provided python, and that likely means we need a venv.

venv

sidenote: a venv is a virtual environment - a place you can install your python dependencies with pip, without affecting any of the system wide packages. this venv can be updated and modified separately from the system python environment, so updates of our server do not break scripts.

The thing with venv is - you need to activate a venv before running a script. there are many ways to activate a venv:

- interactively: in bash, you can run something like "source ./venv/bin/activate" and you're in the venv.
- wrapper script: a small bash script that does "source ./venv/bin/activate" and then runs your python script
- shebang line: the first line of your python script can be #!/path/to/your/venv/python3.9 to run it with the venv
- python sys.path: you can tell python where to look for dependencies

We'll look at the problems each solution has and pick one.

interactively: i want my team to be able to run my scripts without having to enable a venv first - it has to be automatic.

Wrapper script is a common option - a small bash script that does the "source /venv/activate" bit and then runs your script will make it use the venv. but it feels stupid. for every script i want to run, i need a different script that calls it? i feel like there has to be a better way.

shebang line: this would work well if your venv is always available in the same path. however, i want to be able to clone the git repo and run it in my user directory, using a different venv while developing or updating things - so we cannot hardcode an absolute path to our venv. and according to chatgpt, the shebang line does NOT support relative paths. so we cannot use this option.

python sys.path: this is set from within the python script. so when running the script initially, it uses the system provided python, only later changing the path to see dependencies in your venv. Running system python but grabbing dependencies from non-system venv is just asking for trouble and can introduce all kinds of compatibility issues if you're not careful. this is a non-option too.

conclusion

And would you look at that - all the options are bad. I cannot believe it's this hard to run python scripts that don't arbitrarily break. But despite that, i prefer python to bash or perl, so i'll pick the least bad one - and to me, that appears to be wrapper scripts. yes having two files sucks, but it's simple enough and it'll work reliably.

additional info for why we don't install system wide:

many packages in distributions rely on the system python packages and do not bring their own venv. If we mess with system python, we mess with system packages. usually system packages

are cool and stable and work well, so if we do non-package stuff we do it separately.

Fun facts:

relative path shebang question on stackoverflow, with people coming up with all kinds of weird hacks: https://stackoverflow.com/questions/20095351/shebang-use-interpreter-relative-to-the-script-path

---

Revision #5
Created 2024-02-01 08:08:43 UTC by Admin
Updated 2024-02-01 10:39:23 UTC by Admin