The Journal of Open Source Software

basilisk: a Bioconductor package for managing Python environments

Aaron T. L. Lun [1]

1 Genentech Inc., South San Francisco, USA

DOI: 10.21105/joss.04742

Software

- Review
- Repository
- Archive

Editor: Nikoleta Glynatsi

Reviewers:

- @jsun
- @gtonkinhill

Submitted: 18 August 2022
Published: 17 October 2022

Summary

`basilisk` is an R/Bioconductor package for managing Python environments within the Bioconductor package ecosystem. Developers of other Bioconductor packages can use `basilisk` to automatically provision and load custom Python environments, providing a streamlined experience for their end-users by avoiding the need for any manual system configuration. `basilisk` also enables robust execution of Python code via `reticulate` in complex analysis workflows involving multiple Python environments. This package aims to provide a standardized mechanism for integration of Python functionality into the Bioconductor code base.

Statement of need

The Python package ecosystem provides a large number of algorithms and tools that are relevant to R/Bioconductor users. Interoperability between R and Python is facilitated by several popular tools - this includes the `reticulate` package to seamlessly call Python code from an R session (Ushey et al., 2022), and the `conda` package manager to provision environments with the appropriate Python packages (Anaconda Inc., 2022). However, the configuration and management of the Python instances is typically the responsibility of the end user. R/Bioconductor packages with Python functionality often rely on the user to manually ensure that the correct versions of all Python packages are installed. This is burdensome, error-prone and does not scale to widespread integration of Python code into the Bioconductor ecosystem. Moreover, `reticulate` only supports one Python environment for each R session. This compromises interoperability between multiple R/Bioconductor packages that have different (and possibly incompatible) Python dependencies. The `basilisk` package aims to automate the management of Python environments required by "client" R/Bioconductor packages, simplifying their installation and enabling their integration into complex analysis workflows.

Usage

A developer of a client package is expected to define one or more `BasiliskEnvironment` objects that describe the Python environments required by the package. I show an small example below from the `snifter` Bioconductor package (O'Callaghan & Lun, 2022):

```
snifter.env <- BasiliskEnvironment(
    ”fitsne”,
    pkgname = ”snifter”,
    packages = c(
      ”opentsne=0.4.3”,
      ”scikit-learn=0.23.1”,
      if (basilisk.utils::isWindows()) ”scipy=1.5.0” else ”scipy=1.5.1”,
      ”numpy=1.19.0”,
        "python=3.7"
    )
)
```

Once defined, any `BasiliskEnvironment` object can be used in the `basiliskRun()` function to execute arbitrary R code in the context of the associated Python environment. This is most typically combined with `reticulate` to provide an intuitive developer experience when calling Python from R. To demonstrate, I'll again use an example from `snifter`, paraphrased for brevity:

```
out <- basiliskRun(
    env = snifter.env,
    fun = function(x, ...) {
        openTSNE <- reticulate::import("openTSNE", convert = FALSE)
        obj <- openTSNE$TSNE(...)
        out <- obj$fit(x)
        list(
            x = reticulate::py_to_r(out),
            affinities = reticulate::py_to_r(out$affinities$P)
        )
    },
    x = input_matrix # for some observation x dimension matrix.
)
```

Technically, the above call to `basiliskRun()` consists of an internal `basiliskStart()` step to provision and load the appropriate environment, followed by the execution of the provided `fun=`. Advanced users can efficiently re-use the same environment across multiple Python steps by running `basiliskStart()` explicitly before any number of `basiliskRun()` calls:

```
proc <- basiliskStart(env = snifter.env)
# on.exit(basiliskStop(proc)) # for use inside functions

out <- basiliskRun(proc,
    fun = function(x, ...) {
        openTSNE <- reticulate::import("openTSNE", convert = FALSE)
        obj <- openTSNE$TSNE(...)
        out <- obj$fit(x)
        list(
            x = reticulate::py_to_r(out),
            affinities = reticulate::py_to_r(out$affinities$P)
        )
    },
    x = input_matrix
)

# Re-using the same basilisk process:
ver <- basiliskRun(proc, fun = function() {
    mod <- reticulate::import("openTSNE")
    mod$`__version__`
})
```

Managing Python environments

`basilisk` uses conda to automatically manage the creation of Python environments on the user's device. On the first use of `basiliskStart()` anywhere, a local copy of conda is installed using an appropriate Miniconda installer for the user's system. Each conda environment required by a client package is lazily instantiated on the first call to basiliskStart() that uses the corresponding BasiliskEnvironment object. Subsequent uses of that BasiliskEnvironment via basiliskStart() will then re-use the cached conda environment.

I used conda and lazy installation to reduce the burden on the user during installation of client packages. With conda, the user does not have to perform any system configuration such as installing Python or the relevant Python packages. Client packages can define any number of Python environments, but the use of lazy instantiation means that only the ones that are used will actually be created on the user's machine. Similarly, if a client package only uses Python for some optional functionality, the cost of installation is only paid when that functionality is requested.

Lazy instantiation involves the construction of a user-owned cache of conda environments. These environments can consume a large amount of disk space, so basilisk will automatically remove environments that have not been recently used. Some care is also taken to ensure that cache management is thread-safe - if multiple processes attempt to create or delete a particular environment, only one will proceed while the others will wait for its completion. This ensures that multiple basilisk-dependent tasks can be run concurrently without corrupting the cache.

In some scenarios, it is preferable to pay the environment instantiation cost during client package installation. This avoids any delay on first use of basiliskStart() within the client package, which provides more predictable end-user experiences for R-based applications like Shiny. To do this, administrators of an R installation can set the BASILISK_SYSTEM_DIR environment variable, which will cause the conda environments to be created in the client package's installation directory. This "system-wide" installation is also useful on shared systems where a single environment is provisioned for any number of users, rather than requiring each user to create and cache their own.

For developers, the use of conda provides a consistent cross-platform experience for easier maintenance and debugging. It also allows client packages to easily switch between Python versions in different environments, e.g., to run legacy code that is only compatible with older Python versions. However, some Python packages may not be available from conda's repositories, so we provide the pip= argument in the BasiliskEnvironment constructor to pull those packages from PyPI instead.

Integrating with reticulate

basilisk naturally integrates with reticulate to seamlessly call Python code from R. basiliskStart() will automatically load the appropriate Python instance before basiliskRun() evaluates fun=, ensuring that the correct packages are available. If a different Python instance is already loaded into the current R session, basiliskStart() will automatically spin up a new R process to run fun= before transferring the results back to the current session. In this manner, basilisk supports the use of reticulate with multiple Python environments in a single analysis, despite the fact that reticulate is limited to only one Python instance for the lifetime of any given R session (Muenchow et al., 2019).

The use of new R processes ensures that a basilisk client package will always be able to successfully execute its Python-related code via reticulate. The client package remains functional even if other packages - or indeed, the user themselves - load a different Python instance into the current session. In fact, client packages can be forced to always start a new process in basiliskStart() by turning off the getBasiliskShared() option, which avoids interfering with non-basilisk usage of other Python instances via reticulate in the current session. However, this robustness comes at the cost of performance due to the need to spin up a new R process (with the associated delay from package loading) as well as the overhead of communication between different R processes. As such, loading of Python into the current session is preferred by default.

It is also possible to obtain the path to the environment's directory for execution of Python code outside of `reticulate`. This is more onerous but allows clients to directly call executables that are provided in the environment. For example, the `crisprScore` package (Hoberecht et al., 2022) relies on Python 2 environments that will no longer be supported by `reticulate` (Kalinowski, 2022). By directly acquiring the path to the provisioned environment, `crisprScore` can locate the Python 2 executable for execution of its legacy code.

Further comments

The current set of `basilisk` clients can be found on its Bioconductor landing page, including `snifter`, `crisprScore`, `zellkonverter`, `velociraptor` and `BiocSklearn`, to name a few.

The name "basilisk" is based on the mythological snake monster (Rowling, 1998). The original purpose of the `basilisk` package was to freeze Python package versions, much like how the monster was able to Petrify its victims.

Acknowledgements

Thanks to Vince Carey, one of the first developers using `basilisk` in his `BiocSklearn` package; Hervé Pagès, for helping me to get `basilisk` through the Bioconductor build system; Jean-Philippe Fortin, a `basilisk` power user with his `crisprScore` package; and Luke Zappia, Alan O'Callaghan and Kevin Rue-Albrecht, for their feedback as client package developers.

References

Anaconda Inc. (2022). *Conda*. https://conda.io/projects/conda/en/latest

Hoberecht, L., Perampalam, P., Lun, A., & Fortin, J.-P. (2022). A comprehensive Bioconductor ecosystem for the design of CRISPR guide RNAs across nucleases and technologies. *bioRxiv*. https://doi.org/10.1101/2022.04.21.488824

Kalinowski, T. (2022). *Restore Python 2 compatibility; deprecate Python 2 compatability*. https://github.com/rstudio/reticulate/pull/1242

Muenchow, J., Allaire, J., & Markov, N. (2019). *Change path to Python binary*. https://github.com/rstudio/reticulate/issues/27

O'Callaghan, A., & Lun, A. (2022). *Snifter: R wrapper for the python openTSNE library*. https://doi.org/10.18129/B9.bioc.snifter

Rowling, J. K. (1998). *Harry Potter and the Chamber of Secrets*. Bloomsbury.

Ushey, K., Allaire, J., & Tang, Y. (2022). *Reticulate: Interface to 'Python'*. https://rstudio.github.io/reticulate/