provadis
Hochschule

Exposé

Comparing the Performance and Package Management Capabilities of Python Virtual Environments: An evaluation on venv, conda/mini-conda, Pipenv, and Virtualenv for efficient Pytest pipelines

Zur Veranstaltung Programmieren

im Studiengang Informatik

vorgelegt dem Fachbereich Informatik und Wirtschaftsinformatik der
Provadis School of International Management and Technology
von

Jean-Luc Makiola
D990
jean-luc.makiola@stud-provadis-hochschule.de

Erstgutachter: Prof. Dr. Olaf Grebner

Ende der Bearbeitungsfrist: 31.01.2025

Inhalt

Python environments are essential for creating isolated environments that bundle an independently managed Python instance with its own dependencies and versions, independent of the systems Python installation. In modern software development requires constant testing and continuous integration (CI) workflows. The right tool for the creation and management of a virtual environment is crucial due to its direct impact on a developer’s efficiency and the applications security. Later is in the form of version control, especially of dependencies, making sure to minimize the chance of possible security loopholes and following breaches. The study aims to compare these tools on the following criteria:

Test execution speed: How efficiently can each tools created environment run a fixed suite of tests?

Package management: How well does the tool manage dependencies, by keeping the up to date and resolving conflicting dependencies, with minimal human workload?

1 Relevance for the business

The business has a variety of different python projects in use and development. The use of Python virtual environments can often be in CI Pipelines and in the developer's personal testing suite. For both time saving and efficiency are key in assuring waiting and downtimes for developers and services are kept at a minimum, maintaining a fast feedback loop for developers. Slow setup or excessive test execution time can lead to bottlenecks, delaying the release cycle and negatively affecting developer productivity. Thus, evaluating the performance of each virtual environment tool in these contexts is of significant practical relevance.

Furthermore, maintaining project dependencies is essential for secure, reproducible and consistent software. It must be evaluated weather the overhead introduced by more robust and feature rich solutions like Pipenv and conda outweighs the resulting time saver due to improved and automated dependency management.

From a theoretical perspective, this research contributes to the understanding of best practices in virtual environment management and its relationship with software development performance. The comparison of venv, conda, Pipenv, and virtualenv also adds to the body of knowledge on dependency management in Python programming, particularly in the context of testing and CI/CD pipelines.

2 Hypothesis

The use of venv or virtualenv will result in the fastest environment creation and pytest execution times, as these tools focus solely on managing Python dependencies. This simplicity leads to quicker setup and execution compared to Pipenv and conda/miniconda, which introduce additional overhead. Conda/miniconda adds complexity by managing both Python and system-level dependencies, leading to longer setup and runtime. Similarly, Pipenv introduces overhead due to its dependency resolution features and the management of a Pipfile.lock, further increasing environment creation and execution times.

In terms of package management and security, conda/miniconda and Pipenv provide better solutions for managing up-to-date dependencies, which helps reduce security vulnerabilities. Conda/miniconda not only handles Python packages but also system libraries, offering more comprehensive security and package resolution. Pipenv, with its automatic locking of dependencies via a Pipfile.lock, makes it easier to track, update, and ensure a more secure, reproducible environment. In contrast, venv and virtualenv require more manual management of dependencies, making it harder to maintain security and keep packages up to date without the automation features found in Pipenv and conda.

We use LeanIX to manage software artifacts (microservices (ms)) and the LeanIX Connector that checks for different and configurations in the ms is written in Python and testes using Pytest running in venv, which can take quite some time, just to see whether a small change one implemented work as it is meant to.

3 Methodology

To compare the performance and package management capabilities of the four virtual environment tools (venv, conda/miniconda, pipenv, and virtualenv), a series of tests will be conducted. Test execution performance will be measured by running a consistent set of unit and integration tests using pytest within each environment. The time taken to execute these tests will be tracked, and resource utilization, such as CPU and memory, will be monitored with profiling tools. In terms of package management and dependency handling, each tool will be evaluated based on its ability to manage and resolve package dependencies. This will include testing how each tool handles dependency conflicts, such as differing versions of the same package, and how it updates dependencies to secure, stable versions.

For data collection, standard Python libraries like time and timeit will be used to measure pytest execution time, while profiling tools such as cProfile and psutil will monitor resource utilization during test execution. To assess security and package updates, tools like safety will be employed to evaluate each tool's ability to maintain up-to-date and secure dependencies. The collected data will be analyzed through statistical methods for tests that yield hard data, such as runtime and resource consumption, where metrics like mean, median, and standard deviation will be calculated. For tests involving dependency resolution and security, the tools will be evaluated on their ability to handle version mismatches and update packages. Security will also be assessed by using safety to check for outdated or vulnerable packages.

4 Related works

This topic does not have any papers on the difference of between any tools rather just on that it is a good practice to use them. Still the documentations of the different tools as well as their wikis provide a starting point to evaluate their capabilities.

Bhanot, Karan. 2019. “Comparing Python Virtual Environment Tools.” Medium. February 1, 2019. https://towardsdatascience.com/comparing-python-virtual-environment-tools-9a6543643a44.

“Conda Documentation — Conda 24.11.1 Documentation.” n.d. Accessed December 11, 2024. [https://docs.conda.io/projects/conda/en/stable/](https://docs.conda.io/projects/conda/en/sta-ble/).

David, Henrique Miguel Castilho. 2020. “Development of a Machine Learning Platform.” Universidade Nova de Lisboa. https://run.unl.pt/bitstream/10362/160388/1/David_2021.pdf.

Gonzalez-Rivas, Garrett, Zhihui Du, and David A Bader. n.d. “A Deployment Tool for Large Scale Graph Analytics Framework Arachne,” 7.

“Managing Environments — Conda 24.11.1 Documentation.” n.d. Accessed December 11, 2024. https://docs.conda.io/projects/conda/en/stable/user-guide/tasks/manage-environments.html.

“Miniconda — Anaconda Documentation.” n.d. Accessed December 12, 2024. https://docs.anaconda.com/miniconda/.

“Pipenv: Python Dev Workflow for Humans — Pipenv 2024.4.0 Documentation.” n.d. Accessed December 11, 2024. https://pipenv.pypa.io/en/latest/.

“Terms of Services Info.” n.d. Anaconda. Accessed December 12, 2024. https://www.anaconda.com/pricing/terms-of-service-faqs.

“Venv — Creation of Virtual Environments.” n.d. Python Documentation. Accessed December 11, 2024. https://docs.python.org/3/library/venv.html.

Wilkes, Matthew. 2020. “Prototyping and Environments.” In _Advanced Python Development: Using Powerful Language Features in Real-World Applications_, edited by Matthew Wilkes, 1–49. Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4842-5793-7_1.

5 Outline draft

- Introduction
    - Introduction to Python virtual environments and the tools (venv, conda/mini-conda, pipenv, virtualenv).
    - Research aim: Compare the performance and package management in automated testing with pytest.
- Literature Review
    - Overview of virtual environments, their role in dependency management, and testing efficiency.
    - Review of each tool's features and previous research on performance and package management.
- Methodology
    - Data Collection: Test execution with pytest, resource consumption, and security scans using tools like time, psutil, and safety.
    - Evaluation: Statistical analysis of performance, dependency management, and security.
- Results and Analysis
    - Comparison of test execution times, resource usage, dependency management, and security.
- Discussion
    - Interpretation of results and recommendations for the best tool based on performance and package management needs in CI/CD pipelines.
- Conclusion
    - Summary of findings and final recommendations for choosing the right tool.
- References
    - List of all references used.