Skip to content

[![logo](../../../assets/logo-letter.svg)](../../.. "uv")

uv

Using uv with Jupyter

Initializing search

[uv](https://github.com/astral-sh/uv "Go to repository")

[![logo](../../../assets/logo-letter.svg)](../../.. "uv") uv

[uv](https://github.com/astral-sh/uv "Go to repository")

  * [Introduction](../../..)
  * [Getting started](../../../getting-started/)

Getting started

    * [Installation](../../../getting-started/installation/)
    * [First steps](../../../getting-started/first-steps/)
    * [Features](../../../getting-started/features/)
    * [Getting help](../../../getting-started/help/)
  * [Guides](../../)

Guides

* [Installing Python](../../install-python/)
* [Running scripts](../../scripts/)
* [Using tools](../../tools/)
* [Working on projects](../../projects/)
* [Publishing packages](../../package/)
* [Integrations](../)

Integrations

* [Docker](../docker/)
* Jupyter [Jupyter](./) Table of contents
  * Using Jupyter within a project
    * Creating a kernel
    * Installing packages without a kernel
  * Using Jupyter as a standalone tool
  * Using Jupyter with a non-project environment
  * Using Jupyter from VS Code
* [GitHub Actions](../github/)
* [GitLab CI/CD](../gitlab/)
* [Pre-commit](../pre-commit/)
* [PyTorch](../pytorch/)
* [FastAPI](../fastapi/)
* [Alternative indexes](../alternative-indexes/)
* [Dependency bots](../dependency-bots/)
* [AWS Lambda](../aws-lambda/)

* [Concepts](../../../concepts/)

Concepts

* [Projects](../../../concepts/projects/)

Projects

* [Structure and files](../../../concepts/projects/layout/)
* [Creating projects](../../../concepts/projects/init/)
* [Managing dependencies](../../../concepts/projects/dependencies/)
* [Running commands](../../../concepts/projects/run/)
* [Locking and syncing](../../../concepts/projects/sync/)
* [Configuring projects](../../../concepts/projects/config/)
* [Building distributions](../../../concepts/projects/build/)
* [Using workspaces](../../../concepts/projects/workspaces/)

* [Tools](../../../concepts/tools/)
* [Python versions](../../../concepts/python-versions/)
* [Resolution](../../../concepts/resolution/)
* [Caching](../../../concepts/cache/)

* [Configuration](../../../configuration/)

Configuration

* [Configuration files](../../../configuration/files/)
* [Environment variables](../../../configuration/environment/)
* [Authentication](../../../configuration/authentication/)

    * [Package indexes](../../../configuration/indexes/)
    * [Installer](../../../configuration/installer/)
  * [The pip interface](../../../pip/)

The pip interface

    * [Using environments](../../../pip/environments/)
    * [Managing packages](../../../pip/packages/)
    * [Inspecting packages](../../../pip/inspection/)
    * [Declaring dependencies](../../../pip/dependencies/)
    * [Locking environments](../../../pip/compile/)
    * [Compatibility with pip](../../../pip/compatibility/)
  * [Reference](../../../reference/)

Reference

    * [Commands](../../../reference/cli/)
    * [Settings](../../../reference/settings/)
    * [Troubleshooting](../../../reference/troubleshooting/)

Troubleshooting

      * [Build failures](../../../reference/troubleshooting/build-failures/)
      * [Reproducible examples](../../../reference/troubleshooting/reproducible-examples/)
    * [Resolver](../../../reference/resolver-internals/)
    * [Benchmarks](../../../reference/benchmarks/)
    * [Policies](../../../reference/policies/)

Policies

* [Versioning](../../../reference/policies/versioning/)
* [Platform support](../../../reference/policies/platforms/)
* [License](../../../reference/policies/license/)

Table of contents

Using uv with Jupyter

The [Jupyter](https://jupyter.org/) notebook is a popular tool for interactive computing, data analysis, and visualization. You can use Jupyter with uv in a few different ways, either to interact with a project, or as a standalone tool.

Using Jupyter within a project

If you're working within a [project](../../../concepts/projects/), you can
start a Jupyter server with access to the project's virtual environment via
the following:

```
$ uv run --with jupyter jupyter lab
```

By default, `jupyter lab` will start the server at
<http://localhost:8888/lab>.

Within a notebook, you can import your project's modules as you would in any
other file in the project. For example, if your project depends on `requests`,
`import requests` will import `requests` from the project's virtual
environment.

If you're looking for read-only access to the project's virtual environment,
then there's nothing more to it. However, if you need to install additional
packages from within the notebook, there are a few extra details to consider.

Creating a kernel

If you need to install packages from within the notebook, we recommend
creating a dedicated kernel for your project. Kernels enable the Jupyter

server to run in one environment, with individual notebooks running in their own, separate environments.

In the context of uv, we can create a kernel for a project while installing Jupyter itself in an isolated environment, as in `uv run --with jupyter jupyter lab`. Creating a kernel for the project ensures that the notebook is hooked up to the correct environment, and that any packages installed from within the notebook are installed into the project's virtual environment.

To create a kernel, you'll need to install `ipykernel` as a development dependency:

```
$ uv add --dev ipykernel
```

Then, you can create the kernel for `project` with:

```
$ uv run ipython kernel install --user --env VIRTUAL_ENV $(pwd)/.venv --name=project
```

From there, start the server with:

```
$ uv run --with jupyter jupyter lab
```

When creating a notebook, select the `project` kernel from the dropdown. Then use `!uv add pydantic` to add `pydantic` to the project's dependencies, or `!uv pip install pydantic` to install `pydantic` into the project's virtual environment without persisting the change to the project `pyproject.toml` or `uv.lock` files. Either command will make `import pydantic` work within the notebook.

Installing packages without a kernel

If you don't want to create a kernel, you can still install packages from within the notebook. However, there are a few caveats to consider.

Though `uv run --with jupyter` runs in an isolated environment, within the notebook itself, `!uv add` and related commands will modify the _project's_ environment, even without a kernel.

For example, running `!uv add pydantic` from within a notebook will add `pydantic` to the project's dependencies and virtual environment, such that `import pydantic` will work immediately, without further configuration or a server restart.

However, since the Jupyter server is the "active" environment, `!uv pip install` will install package's into _Jupyter's_ environment, not the project

environment. Such dependencies will persist for the lifetime of the Jupyter server, but may disappear on subsequent `jupyter` invocations.

If you're working with a notebook that relies on pip (e.g., via the `%pip` magic), you can include pip in your project's virtual environment by running `uv venv --seed` prior to starting the Jupyter server. For example, given:

```
$ uv venv --seed
$ uv run --with jupyter jupyter lab
```

Subsequent `%pip install` invocations within the notebook will install packages into the project's virtual environment. However, such modifications will _not_ be reflected in the project's `pyproject.toml` or `uv.lock` files.

Using Jupyter as a standalone tool

If you ever need ad hoc access to a notebook (i.e., to run a Python snippet interactively), you can start a Jupyter server at any time with `uv tool run jupyter lab`. This will run a Jupyter server in an isolated environment.

Using Jupyter with a non-project environment

If you need to run Jupyter in a virtual environment that isn't associated with a [project](../../../concepts/projects/) (e.g., has no `pyproject.toml` or

`uv.lock`), you can do so by adding Jupyter to the environment directly. For example:

macOS and LinuxWindows

```
$ uv venv --seed
$ uv pip install pydantic
$ uv pip install jupyterlab
$ .venv/bin/jupyter lab
```

```
uv venv --seed
uv pip install pydantic
uv pip install jupyterlab
.venv\Scripts\jupyter lab
```

From here, `import pydantic` will work within the notebook, and you can install additional packages via `!uv pip install`, or even `!pip install`.

Using Jupyter from VS Code

You can also engage with Jupyter notebooks from within an editor like VS Code. To connect a uv-managed project to a Jupyter notebook within VS Code, we

recommend creating a kernel for the project, as in the following:

```
# Create a project.
$ uv init project

# Move into the project directory.
$ cd project

# Add ipykernel as a dev dependency.
$ uv add --dev ipykernel

# Open the project in VS Code.
$ code .
```

Once the project directory is open in VS Code, you can create a new Jupyter notebook by selecting "Create: New Jupyter Notebook" from the command palette. When prompted to select a kernel, choose "Python Environments" and select the virtual environment you created earlier (e.g., `.venv/bin/python` on macOS and Linux, or `.venv\Scripts\python` on Windows).

Note

VS Code requires `ipykernel` to be present in the project environment. If you'd prefer to avoid adding `ipykernel` as a dev dependency, you can install

it directly into the project environment with `uv pip install ipykernel`.

If you need to manipulate the project's environment from within the notebook, you may need to add `uv` as an explicit development dependency:

```
$ uv add --dev uv
```

From there, you can use `!uv add pydantic` to add `pydantic` to the project's dependencies, or `!uv pip install pydantic` to install `pydantic` into the project's virtual environment without updating the project's `pyproject.toml` or `uv.lock` files.

February 1, 2025

Back to top [Previous Docker](../docker/) [Next GitHub Actions](../github/)