

# Notebook-scoped Python libraries

February 23, 2023

Notebook-scoped libraries let you create, modify, save, reuse, and share custom Python environments that are specific to a notebook. When you install a notebook-scoped library, only the current notebook and any jobs associated with that notebook have access to that library. Other notebooks attached to the same cluster are not affected.

Notebook-scoped libraries do not persist across sessions. You must reinstall notebook-scoped libraries at the beginning of each session, or whenever the notebook is detached from a cluster.

There are two methods for installing notebook-scoped libraries:

- Run the `%pip` magic command in a notebook. Databricks recommends using this approach for new workloads. This article describes how to use these magic commands.
- On Databricks Runtime 10.5 and below, you can use the Databricks library utility. The library utility is supported only on Databricks Runtime, not Databricks Runtime ML or Databricks Runtime for Genomics. See [Library utility \(dbutils.library\)](#).

To install libraries for all notebooks attached to a cluster, use [workspace](#) or [cluster-installed](#) libraries.

## Important!

``dbutils.library.install`` and ``dbutils.library.installPyPI`` APIs are removed in Databricks Runtime 11.0.

## In this article:

- [Requirements](#)
- [Install notebook-scoped libraries with %pip](#)

- [Manage libraries with %pip commands](#)
- [Manage libraries with %conda commands](#)
- [Frequently asked questions \(FAQ\)](#)
- [Known issues](#)

## Requirements

Notebook-scoped libraries using magic commands are enabled by default.

On a High Concurrency cluster running Databricks Runtime 7.4 ML or Databricks Runtime 7.4 for Genomics or below, notebook-scoped libraries are not compatible with [table access control](#) or [credential passthrough](#). An alternative is to use [Library utility \(dbutils.library\)](#) on a Databricks Runtime cluster, or to upgrade your cluster to Databricks Runtime 7.5 ML or Databricks Runtime 7.5 for Genomics or above.

To use notebook-scoped libraries with [Databricks Connect](#), you must use [Library utility \(dbutils.library\)](#).

## Driver node

Using notebook-scoped libraries might result in more traffic to the driver node as it works to keep the environment consistent across executor nodes.

When you use a cluster with 10 or more nodes, Databricks recommends these specs as a minimum requirement for the driver node:

- For a 100 node CPU cluster, use i3.8xlarge.
- For a 10 node GPU cluster, use p2.xlarge.

For larger clusters, use a larger driver node.

## Install notebook-scoped libraries with %pip

### Important!

- You should place all %pip commands at the beginning of the notebook. The notebook state is reset after any %pip command that modifies the environment. If you create Python methods or variables in a notebook, and then use %pip commands in a later cell, the methods or variables are lost.
- Upgrading, modifying, or uninstalling core Python packages (such as IPython) with %pip may cause some features to stop working as expected. For example, IPython 7.21 and above are incompatible with Databricks Runtime 8.1 and below. If

you experience such problems, reset the environment by detaching and re-attaching the notebook or by restarting the cluster.

# Manage libraries with %pip commands

The %pip command is equivalent to the [pip](#) command and supports the same API. The following sections show examples of how you can use %pip commands to manage your environment. For more information on installing Python packages with pip, see the [pip install documentation](#) and related pages.

## In this section:

- [Install a library with %pip](#)
- [Install a wheel package with %pip](#)
- [Uninstall a library with %pip](#)
- [Install a library from a version control system with %pip](#)
- [Install a private package with credentials managed by Databricks secrets with %pip](#)
- [Install a package from DBFS with %pip](#)
- [Save libraries in a requirements file](#)
- [Use a requirements file to install libraries](#)

## Install a library with %pip


Python

 Copy

```
%pip install matplotlib
```

## Install a wheel package with %pip

Python

 Copy


```
%pip install /path/to/my_package.whl
```

## Uninstall a library with %pip

**Note**

You cannot uninstall a library that is included in [Databricks Runtime](#) or a library that has been installed as a [cluster library](#). If you have installed a different library version than the one included in Databricks Runtime or the one installed on the cluster, you can use `%pip uninstall` to revert the library to the default version in Databricks Runtime or the version installed on the cluster, but you cannot use a `%pip` command to uninstall the version of a library included in Databricks Runtime or installed on the cluster.

Python

 Copy

```
%pip uninstall -y matplotlib
```

The `-y` option is required.

## Install a library from a version control system with `%pip`

Python

 Copy

```
%pip install git+https://github.com/databricks/databricks-cli
```

You can add parameters to the URL to specify things like the version or git subdirectory. See the [VCS support](#) for more information and for examples using other version control systems.

## Install a private package with credentials managed by Databricks secrets with `%pip`

Pip supports installing packages from private sources with [basic authentication](#), including private version control systems and private package repositories, such as [Nexus](#) and [Artifactory](#). Secret management is available via the Databricks Secrets API, which allows you to store authentication tokens and passwords. Use the [DBUtils API](#) to access secrets from your notebook. Note that you can use `$variables` in magic commands.


To install a package from a private repository, specify the repository URL with the `--index-url` option to `%pip install` or add it to the pip config file at `~/.pip/pip.conf`.

Python

 Copy

```
token = dbutils.secrets.get(scope="scope", key="key")
```


Bash

 Copy

```
%pip install --index-url https://<user>:$token@<your-package-repository>.com/<path/to/repo> <package>==<version> --extra-index-url https://pypi.org/simple/
```


Similarly, you can use secret management with magic commands to install private packages from version control systems.

Python

 Copy

```
token = dbutils.secrets.get(scope="scope", key="key")
```

Bash

 Copy


```
%pip install git+https://<user>:$token@<gitprovider>.com/<path/to/repo>
```

## Install a package from DBFS with %pip

You can use %pip to install a private package that has been saved on DBFS.

When you upload a file to DBFS, it automatically renames the file, replacing spaces, periods, and hyphens with underscores. For wheel files, pip requires that the name of the file use periods in the version (for example, 0.1.0) and hyphens instead of spaces or underscores, so these filenames are not changed.


Python

 Copy

```
%pip install /dbfs/mypackage-0.0.1-py3-none-any.whl
```

## Save libraries in a requirements file

Python

 Copy

```
%pip freeze > /dbfs/requirements.txt
```

Any subdirectories in the file path must already exist. If you run

%pip freeze > /dbfs/<new-directory>/requirements.txt, the command fails if the directory /dbfs/<new-directory> does not already exist.

## Use a requirements file to install libraries

A [requirements file](#) contains a list of packages to be installed using pip. An example of using a requirements file is:

```
%pip install -r /dbfs/requirements.txt
```

See [Requirements File Format](#) for more information on `requirements.txt` files.

## Manage libraries with %conda commands

### Important!

%conda commands have been deprecated, and will no longer be supported after Databricks Runtime ML 8.4. Databricks recommends using %pip for managing notebook-scoped libraries. If you require Python libraries that can only be installed using conda, you can use [conda-based](#) docker containers to pre-install the libraries you need.

Anaconda Inc. updated their [terms of service](#) for anaconda.org channels in September 2020. Based on the new terms of service you may require a commercial license if you rely on Anaconda's packaging and distribution. See [Anaconda Commercial Edition FAQ](#) for more information. Your use of any Anaconda channels is governed by their [terms of service](#).

As a result of this change, Databricks has removed the default channel configuration for the Conda package manager. This is a breaking change.

To install or update packages using the %conda command, you must specify a channel using `-c`. You must also update all usage of %conda `install` and %sh `conda install` to specify a channel using `-c`. If you do not specify a channel, conda commands will fail with `PackagesNotFoundError`.

The %conda command is equivalent to the [conda](#) command and supports the same API with some restrictions noted below. The following sections contain examples of how to use %conda commands to manage your environment. For more information on installing Python packages with conda, see the [conda install documentation](#).

Note that %conda magic commands are not available on Databricks Runtime. They are only available on Databricks Runtime ML up to Databricks Runtime ML 8.4, and on Databricks Runtime for Genomics. Databricks recommends using `pip` to install libraries. For more information, see [Understanding conda and pip](#).

If you must use both %pip and %conda commands in a notebook, see [Interactions between pip and conda commands](#).

### Note

The following conda commands are not supported when used with %conda:


- activate
- create
- init
- run
- env create
- env remove

### In this section:

- [Install a library with %conda](#)
- [Uninstall a library with %conda](#)
- [Save and reuse or share an environment](#)
- [List the Python environment of a notebook](#)
- [Interactions between pip and conda commands](#)

## Install a library with %conda


Python

 Copy

```
%conda install matplotlib -c conda-forge
```

## Uninstall a library with %conda

Python

 Copy

```
%conda uninstall matplotlib
```

## Save and reuse or share an environment

When you detach a notebook from a cluster, the environment is not saved. To save an environment so you can reuse it later or share it with someone else, follow these steps.

Databricks recommends that environments be shared only between clusters running the same version of Databricks Runtime ML or the same version of Databricks Runtime for Genomics.

1. Save the environment as a conda YAML specification.

```
%conda env export -f /dbfs/myenv.yml
```

2. Import the file to another notebook using `conda env update`.

```
%conda env update -f /dbfs/myenv.yml
```

## List the Python environment of a notebook

To show the Python environment associated with a notebook, use `%conda list`:

```
%conda list
```

## Interactions between `pip` and `conda` commands

To avoid conflicts, follow these guidelines when using `pip` or `conda` to install Python packages and libraries.

- Libraries installed using the [API](#) or using the [cluster UI](#) are installed using `pip`. If any libraries have been installed from the API or the cluster UI, you should use only `%pip` commands when installing notebook-scoped libraries.
- If you use notebook-scoped libraries on a cluster, init scripts run on that cluster can use either `conda` or `pip` commands to install libraries. However, if the init script includes `pip` commands, use only `%pip` commands in notebooks (not `%conda`).
- It's best to use either `pip` commands exclusively or `conda` commands exclusively. If you must install some packages using `conda` and some using `pip`, run the `conda` commands first, and then run the `pip` commands. For more information, see [Using Pip in a Conda Environment](#).

## Frequently asked questions (FAQ)

- [How do libraries installed from the cluster UI/API interact with notebook-scoped libraries?](#)
- [How do libraries installed using an init script interact with notebook-scoped libraries?](#)



- [Can I use %pip and %conda commands in job notebooks?](#)
- [Can I use %pip and %conda commands in R or Scala notebooks?](#)
- [Can I use %sh pip, !pip, or pip? What is the difference?](#)
- [Can I update R packages using %conda commands?](#)

## How do libraries installed from the cluster UI/API interact with notebook-scoped libraries?

Libraries installed from the cluster UI or API are available to all notebooks on the cluster. These libraries are installed using `pip`; therefore, if libraries are installed using the cluster UI, use only `%pip` commands in notebooks.

## How do libraries installed using an init script interact with notebook-scoped libraries?

Libraries installed using an init script are available to all notebooks on the cluster.

If you use notebook-scoped libraries on a cluster running Databricks Runtime ML or Databricks Runtime for Genomics, init scripts run on the cluster can use either `conda` or `pip` commands to install libraries. However, if the init script includes `pip` commands, then use only `%pip` commands in notebooks.

For example, this notebook code snippet generates a script that installs fast.ai packages on all the cluster nodes.

Python

 Copy

```
dbutils.fs.put("dbfs:/home/myScripts/fast.ai", "conda install -c pytorch -c fastai fastai  
-y", True)
```

## Can I use %pip and %conda commands in job notebooks?

Yes.

## Can I use %pip and %conda commands in R or Scala notebooks?

Yes, in a [Python magic cell](#).

# Can I use `%sh pip`, `!pip`, or `pip`? What is the difference?

`%sh` and `!` execute a shell command in a notebook; the former is a Databricks [auxiliary magic command](#) while the latter is a feature of IPython. `pip` is a shorthand for `%pip` when [automagic](#) is enabled, which is the default in Databricks Python notebooks.

On Databricks Runtime 11.0 and above, `%pip`, `%sh pip`, and `!pip` all install a library as a notebook-scoped Python library. On Databricks Runtime 10.4 LTS and below, Databricks recommends using only `%pip` or `pip` to install notebook-scoped libraries. The behavior of `%sh pip` and `!pip` is not consistent in Databricks Runtime 10.4 LTS and below.

## Can I update R packages using `%conda` commands?

No.

## Known issues

- When you use `%conda env update` to update a notebook environment, the installation order of packages is not guaranteed. This can cause problems for the `horovod` package, which requires that `tensorflow` and `torch` be installed before `horovod` in order to use `horovod.tensorflow` or `horovod.torch` respectively. If this happens, uninstall the `horovod` package and reinstall it after ensuring that the dependencies are installed.
- On Databricks Runtime 10.3 and below, notebook-scoped libraries are incompatible with batch streaming jobs. Databricks recommends using [cluster libraries](#) or the [IPython kernel](#) instead.