jupyterlab-discovery Documentation

Release 5.0.0

Vidar Tonaas Fauske

Apr 18, 2018

Installation and usage

Version 5.0.0

Source code Github page

jupyterlab-discovery is a JupyterLab extension for discovering and managing other JupyterLab extensions. The extension adds a side-bar component to JupyterLab, that allows the user to:

- View the currently installed extensions and their status.
- Search and discover other extensions published to the npm registry.
- Install, uninstall, enable, disable and update extensions.

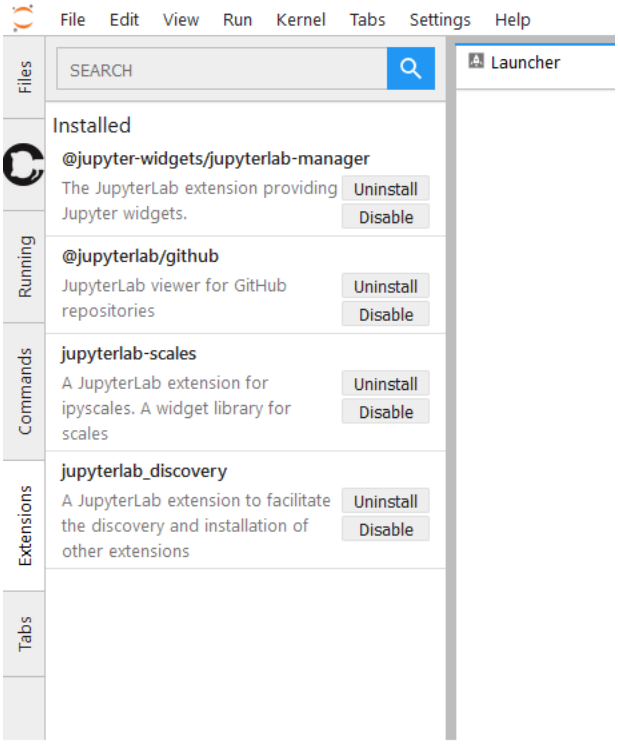

Fig. 1: Figure: Base view of the extension sidebar.

CHAPTER 1

Quickstart

To install the extension, run the following command on the JupyterLab server, e.g. from the JupyterLab terminal:

```
pip install jupyterlab-discovery
```

This will install both the front-end extension, as well as the required server extension. For torubleshooting and details, see the *Installation* section.

CHAPTER 2

Contents

2.1 Installation

There are two parts to the jupyterlab-discovery extension:

- A JupyterLab extension for the user interface.
- A jupyter notebook server extension for doing the management of the extension on the server.

Installing the Python package `jupyterlab-discovery` is the first step. It should be installed from the environment in which you normally run the `jupyter lab` command. From within JupyterLab itself, you can gain access to this environment by opening a terminal. The command to install the package is:

```
pip install jupyterlab-discovery
```

If you are on Jupyter Notebook version 5.3 or greater, that package and a restart of the notebook server should normally be sufficient to start using the extension. With older versions of the notebook package, you will also have to run the following commands:

```
jupyter serverextension enable [--sys-prefix | --user | --system] jupyterlab_discovery
jupyter labextension install [--sys-prefix | --user | --system] --py jupyterlab_
↪discovery
jupyter labextension enable [--sys-prefix | --user | --system] --py jupyterlab_
↪discovery
```

where the flags `[--sys-prefix | --user | --system]` are as specified here.

2.2 Usage

When starting JupyterLab, the extension will query the server about which extensions are installed, and their status. Once this information has been obtained, the base view of the extension panel will look something like this:

In this view, you can see the installed extensions, and uninstall or disable extensions. The status of extensions can also be seen by colored borders on the left-hand side of each entry:

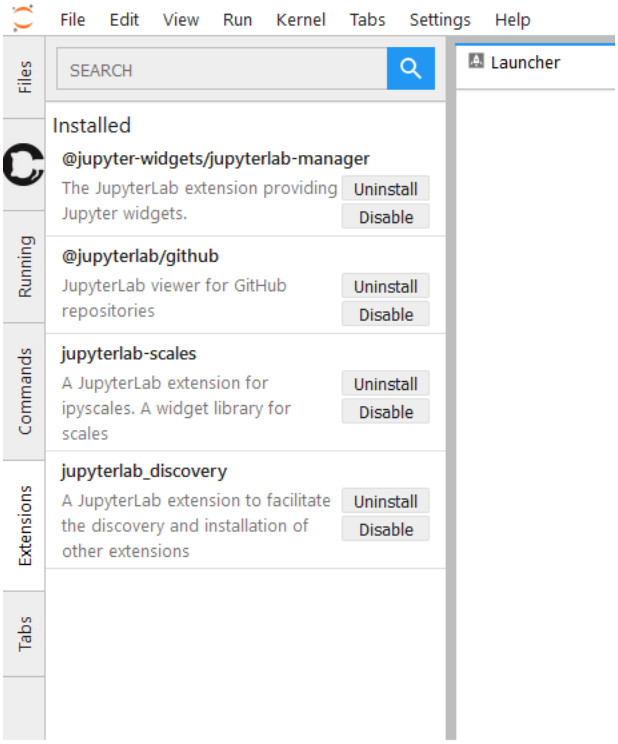

Fig. 2.1: Figure: Base view of the extension sidebar.

- a red border indicates an error with the extension, typically that the installed version is incompatible with the current version on JupyterLab.
- a yellow/orange border indicates that changes have been made to the extension, and that a *rebuild* of JupyterLab is needed.

2.2.1 Searching

You can search for extensions on the NPM registry by using the search bar on the top of the extension panel. Simply typing a space will allow you to see all available extensions.

Note: If you are an extension author, see the *section for extension authors* for instructions on how to make your extension discoverable.

2.2.2 Installing an extension

Once you have found an extension you want to install, simply click its 'Install' button.

Danger: Installing an extension allows it to execute arbitrary code on the server, kernel, and in the client's browser. You should therefore avoid installing extensions you do not trust, and watch out for any extensions trying to mascerade as a trusted extension.

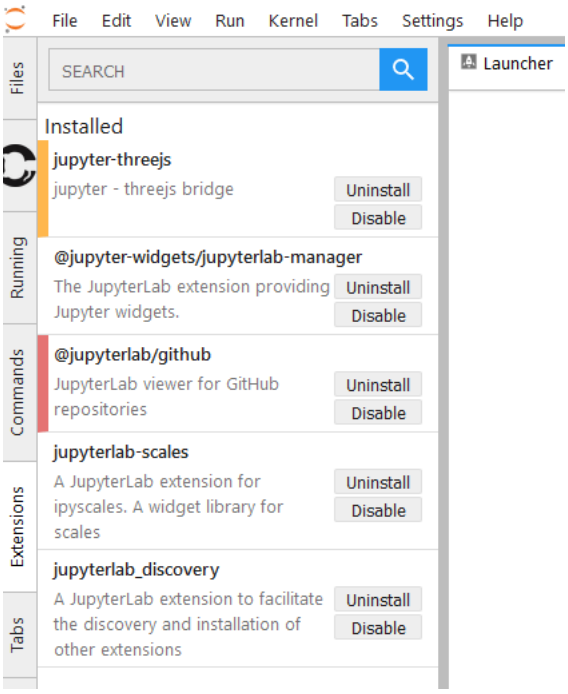

Fig. 2.2: Figure: The extension state indicators.

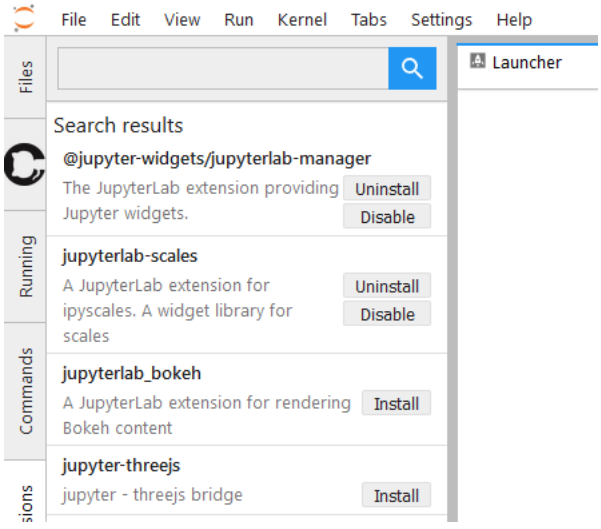

Fig. 2.3: Figure: An empty search (single space) will list all available extensions.

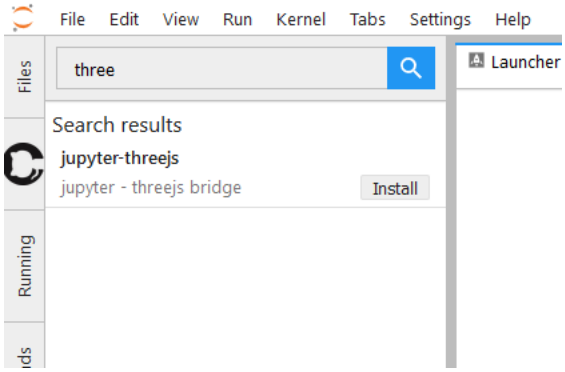

Fig. 2.4: Figure: By entering text in the search bar, the search is limited.

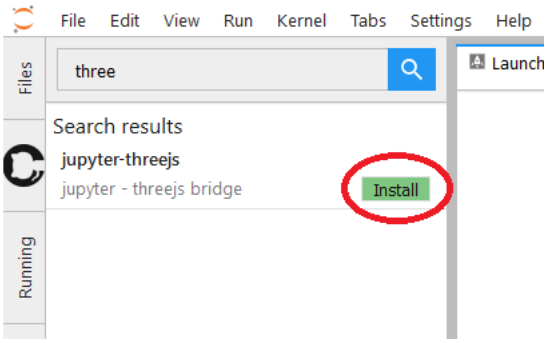

Fig. 2.5: Figure: The install button.

2.2.3 Rebuilding

A while after starting the install of an extension, a drop-down should appear under the search bar indicating that the extension has been downloaded, but that a rebuild is needed to complete the installation.

To trigger a rebuild, click the 'Rebuild' button. This will start the rebuild in the background. Once the rebuild completes, a dialog will pop up, indicating that a reload of the page is needed in order to load the latest build into the browser.

Instead of rebuilding immediately, you can choose to postpone the rebuild to a more appropriate time by clicking the 'Ignore' button on the drop-down. When you are ready, reload the page (or open a new tab to the same server) to trigger a new build check.

2.2.4 Companion packages

During installation of an extension, Discovery will inspect the package metadata for any *instructions on companion packages*. Companion packages can be:

- Notebook server extensions (or any other packages that need to be installed on the Notebook server).
- Kernel packages. An example of companion packages for the kernel are Jupyter Widget packages, like the ipywidgets Python package for the @jupyter-widgets/jupyterlab-manager package.

If Discovery finds instructions for companion packages, it will prompt you about what to do.

The available actions are:

Install Extension Only: Only install the JupyterLab extension, ignoring any companion packages.

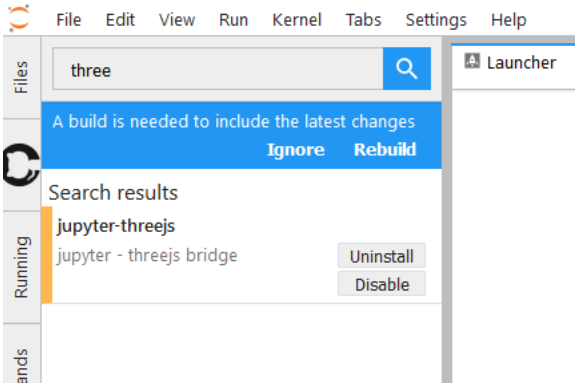

Fig. 2.6: Figure: The rebuild indicator.

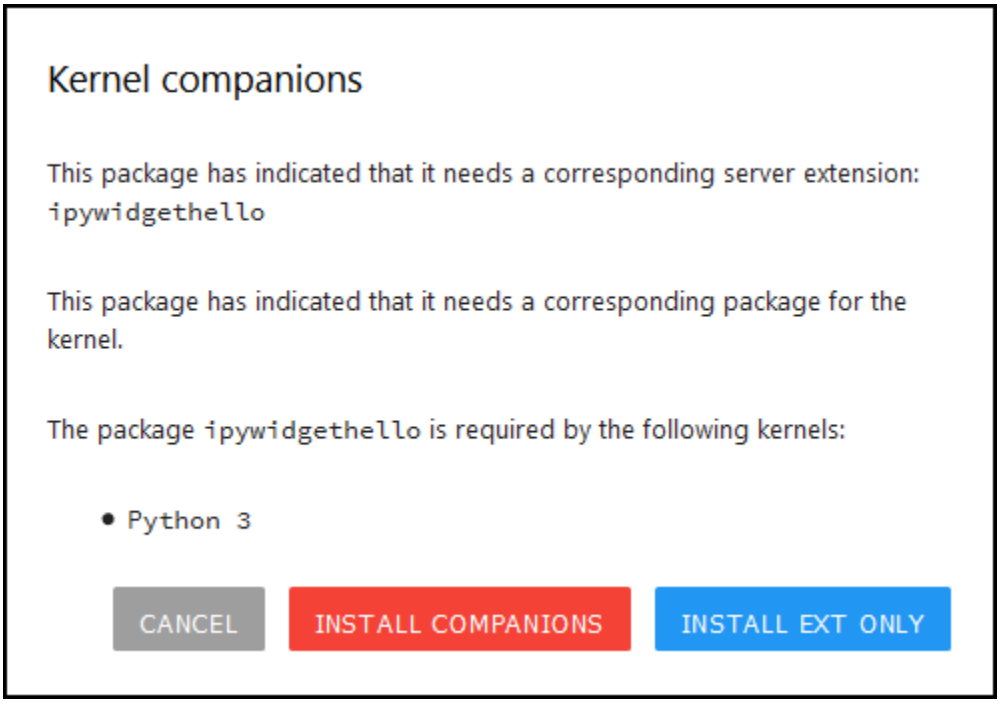

Fig. 2.7: Figure: The companion package information dialog.

Install Companions / Install in Kernel / Install Server Extension: The text of this button depends on which companion package types the metadata indicates are available. In all cases, it will show another dialog asking for more input on what to install, and how. After that, it will try to install the packages into the kernel and/or Notebook server.

Cancel: Do nothing.

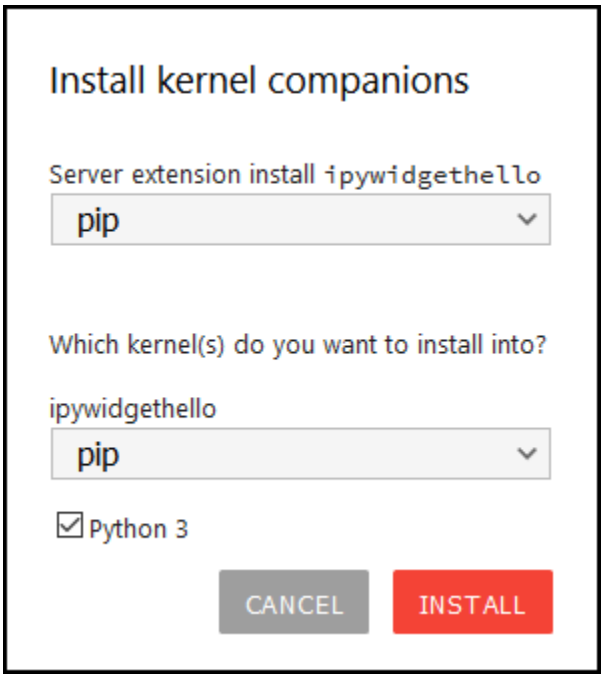

Fig. 2.8: Figure: The companion package install dialog. This example package includes both an server extension and a kernel package for Python. The drop-downs select how to install the packages. The available options are an intersection between what Discovery supports and what the package metadata indicates as valid options.

> **Warning:** The option to install companion packages is still experimental, and while it works for the most common setups, it makes no guarantees. Use at your own risk.

2.3 For Extension Authors

If you have developed an extension for JupyterLab, please ensure that your extension is discoverable by jupyterlab-discovery by adding the following keyword to your package.json:

```
"keywords": [
    "jupyterlab-extension",
    ... any other keywords you have
]
```

that is, 'jupyterlab-extension' as *one* keyword. This allows jupyterlab-discovery to make a clear distinction of actual extensions for jupyterlab.

Danger: Installing an extension allows for arbitrary code execution on the server, kernel, and in the client's browser. You should therefore take steps to protect against malicious changes to your extension's code. This includes ensuring strong authentication for your npm account.

2.3.1 Companion Packages

If your package depends on the presence of one or more packages in the kernel, or a notebook server extension, you can indicate this to jupyterlab-discovery by adding metadata to your package.json file. The full options available are:

```
"jupyterlab": {
  "discovery": {
    "kernel": [
      {
        "kernel_spec": {
          "language": "<regexp for matching kernel language>",
          "display_name": "<regexp for matching kernel display name>"    // optional
        },
        "base": {
          "name": "<the name of the kernel package>"
        },
        "overrides": {    // optional
          "<manager name, e.g. 'pip'>": {
            "name": "<name of kernel package on pip, if it differs from base name>"
          }
        },
        "managers": [    // list of package managers that have your kernel package
            "pip",
            "conda"
        ]
      }
    ],
    "server": {
      "base": {
        "name": "<the name of the server extension package>"
      },
      "overrides": {    // optional
        "<manager name, e.g. 'pip'>": {
          "name": "<name of server extension package on pip, if it differs from base␣
↪name>"
        }
      },
      "managers": [    // list of package managers that have your server extension␣
↪package
          "pip",
          "conda"
      ]
    }
  }
}
```

A typical setup for e.g. a jupyter-widget based package will then be:

```
"keywords": [
    "jupyterlab-extension",
    "jupyter",
```

```
    "widgets",
    "jupyterlab"
],
"jupyterlab": {
  "extension": true,
  "discovery": {
    "kernel": [
      {
        "kernel_spec": {
          "language": "^python",
        },
        "base": {
          "name": "myipywidgetspackage"
        },
        "managers": [
            "pip",
            "conda"
        ]
      }
    ]
  }
}
```

Currently supported package managers are:

- pip
- conda

2.4 Developer install

To install a developer version of jupyterlab-discovery, you will first need to clone the repository:

```
git clone https://github.com/vidartf/jupyterlab_discovery.git
cd jupyterlab_discovery
```

Next, install it with a develop install using pip:

```
pip install -e .
```

Enable the server extension with the appropriate flag:

```
jupyter serverextension enable [--sys-prefix | --user | --system] jupyterlab_discovery
```

Finally, install the labextension locally:

```
jupyter labextension install .
```

This will cause lab to check for changes to jupyterlab-discovery on reload, and will rebuild the extension on lab builds. It will also watch the extension build output if you run the server with the `--watch` flag (picks up the output from `npm run build` in the extension directory). However, running the server in watch mode is not generally conductive to testing the operation of jupyterlab-discovery, as it prevents lab from checking for added extensions (at least at the time of writing).