aiida-testing

Release 0.1.0.dev1

unknown

Nov 23, 2022

CONTENTS

A pytest plugin to simplify testing of AiiDA plugins. It implements fixtures to cache the execution of codes:

- `mock_code`: Caches at the level of the code executable. Use this for testing calculation and parser plugins, because input file generation and output parsing are also being tested.
- `export_cache`: Uses the AiiDA caching feature, in combination with an automatic database export / import. Use this to test high-level workflows.

`aiida-testing` is available at http://github.com/aiidateam/aiida-testing

CHAPTER

ONE

USER GUIDE

1.1 Getting started

This page should contain a short guide on what the plugin does and a short example on how to use the plugin.

1.1.1 Installation

Use the following commands to install `aiida-testing`:

```
pip install aiida-testing
```

1.1.2 Usage

Once installed the pytest fixtures should show up in:

```
pytest --fixtures
```

1.2 Using `mock_code`

mock_code provides two components:

1. A command-line script `aiida-mock-code` (the *mock executable*) that is executed instead of the *actual* executable and acts as a *cache* for the outputs of the actual executable
2. A pytest fixture *mock_code_factory()* that sets up an AiiDA Code pointing to the mock executable

In the following, we will set up a mock code for the `diff` executable in three simple steps.

First, we want to define a fixture for our mocked code in the `conftest.py`:

```
import os
import pytest

# Directory where to store outputs for known inputs (usually tests/data)
DATA_DIR = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'data')

@pytest.fixture(scope='function')
def mocked_diff(mock_code_factory):
```

(continues on next page)

(continued from previous page)

```
    """
    Create mocked "diff" code
    """
    return mock_code_factory(
        label='diff',
        data_dir_abspath=DATA_DIR,
        entry_point='diff',
        # files *not* to copy into the data directory
        ignore_files=('_aiidasubmit.sh', 'file*')
    )
```

Second, we need to tell the mock executable where to find the *actual* `diff` executable by creating a `.aiida-testing-config.yml` file in the top level of our plugin.

Note: This step is needed **only** when we want to use the actual executable to (re)generate test data. As long as the mock code receives data inputs whose corresponding outputs have already been stored in the data directory, the actual executable is not used.

```
mock_code:
  # code-label: absolute path
  diff: /usr/bin/diff
```

Note: Why yet another configuration file?

The location of the actual executables will differ from one computer to the next, so hardcoding their location is not an option. Even the names of the executables may differ, making searching for executables in the PATH fragile. Finally, one could use dedicated environment variables to specify the locations of the executables, but there may be many of them, making this approach cumbersome. Ergo, a configuration file.

Finally, we can use our fixture in our tests as if it would provide a normal `Code`:

```
def test_diff(mocked_diff):
    # ... set up test inputs

    inputs = {
        'code': mocked_diff,
        'parameters': parameters,
        'file1': file1,
        'file2': file2,
    }
    results, node = run_get_node(CalculationFactory('diff'), **inputs)
    assert node.is_finished_ok
```

When running the test via `pytest` for the first time, `aiida-mock-code` will pipe through to the actual `diff` executable. The next time, it will recognise the inputs and directly use the outputs cached in the data directory.

Note: `aiida-mock-code` "recognizes" calculations by computing a hash of the working directory of the calculation (as prepared by the calculation input plugin). It does *not* rely on the hashing mechanism of AiiDA.

Don't forget to add your data directory to your test data in order to make them available in CI and to other users of your plugin!

Since the `.aiida-testing-config.yml` is usually specific to your machine, it usually better not to commit it. Tests will run fine without it, and if other developers need to change test inputs, they can easily regenerate a template for it using `pytest --testing-config-action=generate`.

For further documentation on the pytest commandline options added by mock code, see:

```
$ pytest -h
...
custom options:
  --testing-config-action=TESTING_CONFIG_ACTION
                        Read .aiida-testing-config.yml config file if present
                        ('read'), require config file ('require') or generate
                        new config file ('generate').
  --mock-regenerate-test-data
                        Regenerate test data.
```

1.2.1 Limitations

- No support for remote codes yet
- Not tested with MPI

1.3 Using `export_cache`

TODO: an introduction to export_cache

CHAPTER

TWO

DEVELOPER GUIDE

2.1 Full setup

The following commands give you a complete development setup for `aiida-testing`. Make sure to run this in the appropriate virtual environment:

```
git clone https://github.com/aiidateam/aiida-testing.git
cd aiida-testing
pip install -e .[dev]
pre-commit install
```

Commands to install only parts of the development setup are included below.

2.2 Running the tests

The following will discover and run all unit tests:

```
pip install -e .[testing]
pytest
```

2.3 Automatic coding style checks

Enable enable automatic checks of code sanity and coding style:

```
pip install -e .[pre_commit]
pre-commit install
```

After this, the yapf formatter, the pylint linter, the prospector code analyzer, and the mypy static type checker will run at every commit.

If you ever need to skip these pre-commit hooks, just use:

```
git commit -n
```

2.4 Continuous integration

`aiida-testing` comes with a `ci.yml` file for continuous integration tests on every commit using GitHub Actions. It will:

1. run all tests
2. build the documentation
3. check coding style and version number

2.5 Online documentation

The documentation of `aiida-testing` is continuously being built on ReadTheDocs, and the result is shown on https://aiida-testing.readthedocs.org/.

If you have a ReadTheDocs account, you can also enable it on your own fork for testing, but you will have to use a different name.

2.6 Local documentation

Of course, you can also build the documentation locally:

```
pip install -e .[docs]
cd docs
make
```

2.7 PyPI release

The process for creating a distribution and uploading it to PyPI is:

```
pip install twine
python setup.py sdist
twine upload dist/*
```

This can only be done by people who are registered as `aiida-testing` maintainers on PyPI. After this, you (and everyone else) should be able to:

```
pip install aiida-testing
```

CHAPTER

THREE

AIIDA_TESTING PACKAGE

3.1 Subpackages

3.1.1 aiida_testing.export_cache package

Module contents

Defines fixtures for automatically creating / loading an AiiDA DB export, to enable AiiDA - level caching.

3.1.2 aiida_testing.mock_code package

Module contents

Defines fixtures for mocking AiiDA codes, with caching at the level of the executable.

aiida_testing.mock_code.**mock_code_factory**(*aiida_localhost*, *testing_config*, *testing_config_action*, *mock_regenerate_test_data*, *mock_fail_on_missing*, *request: FixtureRequest*, *tmp_path: Path*)

Fixture to create a mock AiiDA Code.

testing_config_action :
Read config file if present ('read'), require config file ('require') or generate new config file ('generate').

aiida_testing.mock_code.**mock_regenerate_test_data**(*request*)

Read whether to regenerate test data from command line option.

aiida_testing.mock_code.**pytest_addoption**(*parser*)

Add pytest command line options.

aiida_testing.mock_code.**testing_config**(*testing_config_action*)

Get content of .aiida-testing-config.yml

testing_config_action :
Read config file if present ('read'), require config file ('require') or generate new config file ('generate').

aiida_testing.mock_code.**testing_config_action**(*request*)

Read action for testing configuration from command line option.

3.2 Module contents

A pytest plugin for testing AiiDA plugins.

If you use AiiDA for your research, please cite the following work:

> Giovanni Pizzi, Andrea Cepellotti, Riccardo Sabatini, Nicola Marzari, and Boris Kozinsky, *AiiDA: automated interactive infrastructure and database for computational science*, Comp. Mat. Sci 111, 218-230 (2016); https://doi.org/10.1016/j.commatsci.2015.09.013; http://www.aiida.net.

`aiida-testing` is released under the Apache license.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX