aiida_intro_NSCmarch2017

March 30, 2017

1 Hands-on introduction to AiiDA

NSC workshop on electronic structure calculations, March 29 2017
Thor Wikfeldt
kthw@kth.se
PDC Center for HPC, Stockholm

> This material has been adopted from an AiiDA tutorial from http://www.aiida.net/tutorials/ (January 2017 version)

1.1 Interacting with AiiDA via verdi

verdi, with its subcommands, enables a variety of operations such as inspecting the status of ongoing or terminated calculations, showing the details of calculations, computers, codes, or data structures, access the input and the output of a calculation, etc

> Note that verdi commands are typically typed in a terminal

1.1.1 Listing stuff

```
In [ ]: !verdi user list
```

Codes and computers

```
In [ ]: !verdi code list -o -A
```

```
In [ ]: !verdi code show 1088
```

If you have set up remote clusters, they will show up here

```
In [ ]: !verdi computer list -a
```

```
In [ ]: !verdi computer show daint
```

```
In [ ]: !verdi code list
```

```
In [ ]: !verdi code show 4681
```

Ongoing calculations

```
In [ ]: !verdi calculation list
```

All finished calculations

```
In [ ]: !verdi calculation list --states FINISHED
```

Show one particular calculation

```
In [ ]: !verdi calculation show 4079
```

1.1.2 Graphs and nodes

Generate graph for one particular node

```
In [ ]: !verdi graph generate 4079
```

Convert to pdf using `dot` utility

```
In [ ]: !dot -Tpdf -o 4079.pdf 4079.dot
```

```
In [ ]: !evince 4079.pdf
```

Show ParameterData node

```
In [ ]: !verdi data parameter show 4080
```

Can be compared with raw input file for Quantum Espresso generated by AiiDA (run this on `calculation` node)

```
In [ ]: !verdi calculation inputcat 4079
```

List all files used as input to calculation

```
In [ ]: !verdi calculation inputls 4079
```

```
In [ ]: #!verdi data structure show --format ase 4123
```

```
In [ ]: !verdi data structure export --format xyz 4123 > 4123.xyz
```

Calculation results

```
In [ ]: !verdi calculation res 4079
```

```
In [ ]: !verdi calculation outputls 4079
```

```
In [ ]: !verdi calculation outputcat 4079
```

1.1.3 Groups of calculations

Calculations can be organized in groups, which are particularly useful to assign a set of calculations to a common project

```
In [ ]: !verdi group list
```

```
In [ ]: !verdi group show 1
```

1.2 Interacting with AiiDA objects

To inspect calculations and launch new calculations, the verdi shell is useful.
Run either in Jupyter Notebook (as here), or ipython session by typing `verdi shell` in terminal.

Custom magic command to load AiiDA environment (this is automatically loaded in `verdi shell`)

```
In [ ]: %aiida
```

```
In [ ]: node = load_node(4079)
```

```
In [ ]: node.res.energy
```

Use tab autocompletion to list all possible output results of calculation

```
In [ ]: node.res.
```

1.2.1 Pseudopotentials

Fetch and upload pseudopotential family

```
In [ ]: !wget http://www.materialscloud.ch/sssp/pseudos/SSSP_eff_PBESOL.tar.gz
        !tar -zxvf SSSP_eff_PBESOL.tar.gz
        !verdi data upf uploadfamily SSSP_eff_PBESOL 'SSSP' 'SSSP pseudopotential l
```

```
In [ ]: !verdi data upf listfamilies
```

Inspect pseudopotentials

```
In [ ]: upf = load_node(4187)
```

```
In [ ]: upf.element
```

1.2.2 K-points

Inspect k-points

```
In [ ]: kpoints = load_node(4370)
```

```
In [ ]: kpoints.get_kpoints_mesh()
```

```
In [ ]: kpoints.get_kpoints_mesh(print_list=True)
```

Create a k-point instance to use in a calculation later on
One can import KpointsData using explicit location of module, but `DataFactory` function simplifies this (returns the class itself, not class instance)

```
In [ ]: #from aiida.orm.data.array.kpoints import KpointsData
        KpointsData = DataFactory("array.kpoints")
        kpoints = KpointsData()
        kpoints_mesh = 2
        kpoints.set_kpoints_mesh([kpoints_mesh,kpoints_mesh,kpoints_mesh])
        kpoints.store()
```

1.2.3 Input parameters

Load a input parameter node and change a parameter

```
In [ ]: params = load_node(4080)
```

Get a dictionary with input parameters

```
In [ ]: params.get_dict()
```

```
In [ ]: params_dict=params.get_dict()
```

```
In [ ]: params_dict['SYSTEM']['ecutwfc']=20
```

Create new `ParameterData` instance

```
In [ ]: ParameterData = DataFactory('parameter')
```

```
In [ ]: new_params = ParameterData(dict=params_dict)
```

```
In [ ]: new_params.get_dict()
```

Store input parameters in database

```
In [ ]: new_params.store()
```

1.2.4 Structures

Inspecting structures

```
In [ ]: structure = load_node(285)
```

```
In [ ]: structure.get_formula()
```

```
In [ ]: structure.sites
```

```
In [ ]: structure.get_ase()
```

Define a new structure
We go for silicon crystal

```
In [ ]: alat = 5.4
        the_cell = [[alat/2,alat/2,0.],[alat/2,0.,alat/2],[0.,alat/2,alat/2]]
```

Create new `StructureData` instance

```
In [ ]: StructureData = DataFactory('structure')
```

```
In [ ]: structure = StructureData(cell=the_cell)
        structure.cell
```

```
In [ ]: structure.append_atom(position=(alat/4.,alat/4.,alat/4.),symbols="Si")
        structure.sites
```

```
In [ ]: structure.store()
```

Accessing inputs and outputs

```
In [ ]: calc = load_node(4079)
```

```
In [ ]: calc.inp.
```

1.3 Submit, monitor and debug calculations

1.3.1 The AiiDA daemon

The AiiDA daemon is a program running all the time in the background, checking if new calculations appear and need to be submitted to the scheduler.
It also takes care of all the necessary operations before the calculation submission, and after the calculation has completed on the cluster.

```
In [ ]: !verdi daemon status
```

1.3.2 Creating a new calculation

For a Quantum Espresso calculation with AiiDA, we need: 1. pseudopotentials 2. a structure 3. the k-points 4. the input parameters

We can use the input parameters, structure and k-points defined above to submit interactively

```
In [ ]: code = Code.get_from_string('pw-5.1@localhost')
        # AiiDA calculations are instances of the class Calculation (i.e. one of it
        calc = code.new_calc()
        calc.label="PW test"
        calc.description="My first AiiDA calc with Quantum ESPRESSO on BaTiO3"

        # set number of compute nodes and maximum time allowed
        calc.set_resources({"num_machines": 1})
        calc.set_max_wallclock_seconds(30*60)

        # use input parameters defined above
        calc.use_parameters(new_params)

        # use structure defined above
        calc.use_structure(structure)

        # use kpoints defined above
        calc.use_kpoints(kpoints)

        # use pseudopotentials from given family
        calc.use_pseudos_from_family('SSSP')

        calc.store_all()
```

Do a test submit to see how AiiDA creates input files etc.
Run directory is in folder `~/submit_test/`

```
In [ ]: calc.submit_test()
```

Store and submit calculation
So far everything has been held in memory. Now we store it in database

```
In [ ]: calc.store_all()
```

. . . and submit

```
In [ ]: calc.submit()
```

But it's actually more common to submit new jobs via `verdi run` command, so we write to a file

```
In [ ]: %%writefile test_pw.py
        code = Code.get_from_string('pw-5.1@localhost')
        # AiiDA calculations are instances of the class Calculation (i.e. one of it
        calc = code.new_calc()
        calc.label="PW test"
        calc.description="My first AiiDA calc with Quantum ESPRESSO on BaTiO3"

        # set number of compute nodes and maximum time allowed
        calc.set_resources({"num_machines": 1})
        calc.set_max_wallclock_seconds(30*60)

        #create structure
        alat = 5.4
        the_cell = [[alat/2,alat/2,0.],[alat/2,0.,alat/2],[0.,alat/2,alat/2]]
        StructureData = DataFactory('structure')
        structure = StructureData(cell=the_cell)
        structure.append_atom(position=(alat/4.,alat/4.,alat/4.),symbols="Si")
        structure.store()
        calc.use_structure(structure)

        #create k-points
        KpointsData = DataFactory("array.kpoints")
        kpoints = KpointsData()
        kpoints_mesh = 2
        kpoints.set_kpoints_mesh([kpoints_mesh,kpoints_mesh,kpoints_mesh])
        kpoints.store()
        calc.use_kpoints(kpoints)


        # specify pseudopotential
        calc.use_pseudos_from_family('SSSP')

        # create input parameter dictionary from scratch
        parameters_dict = {'CONTROL': {'calculation': 'scf',
        'tstress': True,
        'tprnfor': True,
        },
        'SYSTEM': {'ecutwfc': 30.,
        'ecutrho': 200.,
        },
        'ELECTRONS': {'conv_thr': 1.e-8,
        },
        }
        ParameterData = DataFactory("parameter")
        parameters = ParameterData(dict=parameters_dict)
        calc.use_parameters(parameters)

        # store in database, and submit
        calc.store_all()
        calc.submit()
```

```
In [ ]: # run this in terminal
        # !verdi run test_pw.py
        # !watch verdi calculation list -a -p1
```

Troubleshooting a calculation

```
In [ ]: !verdi calculation outputcat 4824
```

```
In [ ]: !verdi calculation logshow 4824
```

1.4 Queries in AiiDA

Asking questions to the database

```
In [ ]: from aiida.orm.querybuilder import QueryBuilder
```

```
In [ ]: qb = QueryBuilder()
```

Append all nodes to the query

```
In [ ]: qb.append(Node)
```

Show the current contents of the query (this will be huge since no filtering performed yet)

```
In [ ]: qb.all()
```

```
In [ ]: qb.count()
```

To retrieve a subclass of a node, append that specific subclass instead of Node

```
In [ ]: StructureData = DataFactory('structure')
        qb = QueryBuilder() # Creating a new QueryBuilder instance
        qb.append(StructureData) # Telling the QueryBuilder instance that I want st
        qb.all() # Asking for all the results!
        #qb.count()
```

To find the underlying SQL command:

```
In [ ]: str(qb)
In [ ]: # tab to view all options
        qb.
```

A high-throughput example

```
In [ ]: !verdi group list
```

```
In [ ]: qb = QueryBuilder()
```

Append all three groups to the query, and "project" (extract out) their respective names. Also add tag

```
In [ ]: qb.append(Group, tag="my_groups", project="name",
                  filters={"name": {"in": ["tutorial_pbesol","tutorial_lda","tutori
```

Extend query to all PwCalculation nodes that belong to the groups

```
In [ ]: qb.append(JobCalculation, member_of="my_groups", tag="my_jobcalc")
```

Extend query to return chemical formulas (using projection)

```
In [ ]: qb.append(StructureData,input_of="my_jobcalc",project=["extras.formula"])
```

Extend query to return also a few `ParameterData` outputs (energy smearing, magnetization), along with their units

```
In [ ]: # if you forget names of the key-value pairs, check the attributes like thi
        c1=load_node(4469)
        c1.get_attrs()
```

```
In [ ]: qb.append(ParameterData,output_of="my_jobcalc",
                  project=["attributes.energy_smearing", "attributes.energy_smearin
                           "attributes.total_magnetization","attributes.total_magn
```

```
In [ ]: qb.all()
```

Run the query and return results as dictionary

```
In [ ]: res = qb.dict()
        res
```

Write results to file

```
In [ ]: with open("res.txt","w") as f:
            for i in res:
                line = i["StructureData"]["extras.formula"]+",  "+i["my_groups"]["r
                line += str(i["ParameterData"]["attributes.energy_smearing"])+",  '
                line += str(i["ParameterData"]["attributes.total_magnetization"])+'
                f.write(line+"\n")
```

Now use ready-made plotting script to visualize results

```
In [ ]: %cd tutorial_scripts/
        %pwd
```

```
In [ ]: %pylab inline
        import plot_calculation_results
        plot_calculation_results.plot_results('../res.txt')
```

```
In [ ]: # %load /home/aiida/tutorial_scripts/plot_calculation_results.py
```