DFT-toolkit

DFT-toolkit is a Python package for automating DFT calculations across multiple DFT software packages. It allows users to create, submit, and monitor jobs in the SLURM queue, as well as build multi-step calculation pipelines with defined dependencies between steps. DFT-toolkit is designed to work with any DFT software and can be run on any HPC cluster using SLURM.

Installation

To use DFT-toolkit you need to clone its repository:
`git clone https://github.com/mszyszko-uw/vasp-processing.git`
and install requirements:

- `Python >= 3.6`
- `pip ≥ 19.0`
- `simple_slurm-0.3.6`
- `PyYAML==6.0.2`

by
`pip install -r .\requirements.txt`
We strongly recommend using a separate Python environment for this installation. Use a virtual environment or any conda environment.

Quickstart

DTF_toolkit can be run with Python by:
`python dft_toolkit.py [options] action`
a list of available options and parameters can be obtained by:
`python dft_toolkit.py --help`

Available actions

- `freenodes` - list all available node on a machine witch free CPUs and memory,
- `waiting` - create a rapport about waiting time in SLURM queue,
- `print`- print SLURM scripts,
- `create` - create SLURM scripts,
- `submit` - create and submit SLURM scripts,
- `array` - run job array from all subdirectories,
- `checkqueue` - list all user jobs,
- `canceljob` - cancel a job,
- `jobinfo` - print a job details from SLURM,
- `print_avalable_steps` - print defined jobs.

Available options

- `-h, --help` show this help message and exit

- `--path PATH` set a path to the working directory or resource,
- `--config CONFIGURE_FILE` set a configuration file,
- `--id JOB_ID` define a SLURM Job ID (for details or cancellation),
- `--steps STEPS` set up a list of jobs to run,
- `--array` run job array based on step and path
- `--dependency_step DEPENDENCY_STEP` A step that blocks rest of steps

Configurations files

DFT-Toolkit uses two configuration files:

1. Machine file - for definite a cluster details in yaml or json format,
2. step file (`steps.yaml`) - where pipeline's steps are defined.

Machine file

The machine file is used for defining computer cluster details like SLURM partitions, module names and python environment details. It has 3 sections:

- `slurm` where you can define a cluster details using SLURM keywords (like, nodes, partitions)
- `script` where you can set up exact module name. All modules listed in this section will be loaded in queue job,
- `env` where you can specify what type (`type`) of python environment you are using (virtual environment or conda) and path (`path`) to your environment.

You should have a separate machine file for each computer cluster you use and select the right one with --`config` options.

Example of `config.yaml`

```
slurm:
        nodes: 1
        partition: short
        time: "1:00:00"
script:
        module: vasp/22
env:
        type: venv #venv or conda
        path: /temp/dft-toolkit/environments/venv
```

Steps definition

`steps.yaml` is used for the definition of pipeline's steps. The file has a separate section for each step. Each step has two parts: `slurm` and `cmd`. In the `slurm` section you can define SLURM job parameters with SLURM keywords. The `cmd` section is a body of SLURM script and all calculations, preprocessing and postprocessing commands are listed (Use `cmd: |` for multiline script).

Example of `steps.yaml`

```
scf:                    #step 1
  slurm:
    time: "0:05:00"             #max wall time
```

```
    nodes: 1                    #number of nodes
    ntasks: 28                  #total number of CPUs
    mem_per_cpu: "2GB"          #memory per CPU - 56G in total
  cmd: |        #commands to run at step: scf
    ulimit -s unlimited

    START_DIR="$(pwd)"
    cd "$output"
    mpiexec vasp_std > log
postprocessing:      #step 2
  slurm:
    time: "0:20:00"
    nodes: 1
    ntasks: 1
    mem_per_cpu: "1GB"
  cmd: |
    cd OUTPUTS
    python ../postprocessing_plot.py
```

Examples

In this example you're going to learn how to create a working pipeline based on an existing SLURM script. Let's assume that you have a SLURM script:

```
#!/bin/bash -l
#SBATCH --job-name=s01e01scf
#SBATCH --time=0:05:00
#SBATCH --account=plg2dmagsem-cpu
#SBATCH --partition=plgrid
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=28
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=1GB
#SBATCH --output=out_step01
#SBATCH --error=err_step01

ulimit -s unlimited
module load  intel-compilers/2023.2.1  impi/2021.10.0
VASP/6.5.1-Dsingle_prec_bse
module load Python/3.11.5 matplotlib/3.8.2 h5py/3.11.0

START_DIR="$(pwd)"
output=$(python3 toolkit.py --step step01 --part dry)
cd "$output"
mpiexec vasp_std > log

cd "$START_DIR"
output=$(python3 toolkit.py --step step01 --part scf)
```

```
cd "$output"
mpiexec vasp_std > log
```

You need to follow steps:

1. create virtual environment
   ```
   python -m venv ./tollbox_env
   ```
2. clone repository
   ```
   git clone https://github.com/mszyszko-uw/vasp-processing.git
   ```
3. install all dependences
   ```
   ./tollbox_env/bin/activate
   cd vasp-processing
   pip install -r requirements.txt
   ```
4. create machine file and fill __PARTITION_ and __PATH__
   ```
   #config.yaml
   slurm:
           nodes: 1
           partition: __PARTITION__
           time: "1:00:00"
   script:
           module: vasp/22
   env:
           type: venv #venv or conda
           path: __PATH__/tollbox_env
   ```
5. create steps file
   ```
   #steps.yaml
   scf:
     slurm:
       time: "0:05:00"
       nodes: 1
       ntasks: 28
       mem_per_cpu: "1GB"
     cmd: |
       ulimit -s unlimited
       module load  intel-compilers/2023.2.1  impi/2021.10.0
   VASP/6.5.1-Dsingle_prec_bse
       module load Python/3.11.5 matplotlib/3.8.2 h5py/3.11.0

       START_DIR="$(pwd)"
       output=$(python3 toolkit.py --step step01 --part dry)
       cd "$output"
       mpiexec vasp_std > log

       cd "$START_DIR"
       output=$(python3 toolkit.py --step step01 --part scf)
       cd "$output"
       mpiexec vasp_std > log
   ```
6. examin SLURM script
   ```
   python dft_toolkit.py --steps scf print
   ```

7. run calculations

```
python dft_toolkit.py --steps scf submit
```