Setting up the environment on Cedar:

0. ssh-connect to cedar.computecanada.ca with your credentials;

1. Setting up the environment:

Install **miniconda** via EasyBuild, following https://docs.computecanada.ca/wiki/Anaconda/en

or just

```
$ eb Miniconda3-4.3.27.eb
```

the installed module should be loaded every time you log in to the machine

```
$ module load miniconda3
```

(!) I recommend adding this line to `~/.bashrc`

After creating and activating your conda environment, e.g.:

```
$ conda create -n py36 python=3.6
$ source activate py36
```

install required packages:

```
(py36) $ pip install numpy
(py36) $ pip install mpi4py
```

MPI 4 Python will be utilized by sasktran scheduler to efficiently run jobs on whole nodes; the sasktran itself can so far efficiently utilize only whole sockets (half node)

And store your env. name to a system variable PYENV:

```
export PYENV=py36
```

(!) I recommend adding this line to `~/.bashrc`

2. Installing sasktran:

clear the PYTHONPATH (for some reason manylinux wheels are ignored by default):

```
(py36) $ export PYTHONPATH=
```

install the sasktran

```
(py36) $ pip install sasktran -f https://arg.usask.ca/wheels/
```

if you use discrete ordinates radiative transfer model, it is highly recommended to update sasktran Disco with the locally built version; you can build it yourself or use prebuilt package (version 1.10 for python 3.6):

```
(py36) $ pip install --upgrade /project/6007007/shared/ARG/SasktranDO/whl/sktran_disco-1.10+0.gb64bab1.dirty-cp36-cp36m-linux_x86_64.whl
```

To check whether installation was successful, try:

```
 (py36) $ python -c "import sasktran"

------- Sasktran Discrete-Ordinates ISKEngine: DLL REGISTRATION DETAILS -------------

/SasktranIF/Engines/DO:DLLName =
/home/eld439/.local/easybuild/software/2017/Core/miniconda3/4.3.27/envs/py36/lib/python3.6
/site-packages/sktran_disco/libisk_disco.so
```

3. Testing the basic installation

Before running the test example, you should take care of the following:

- module ***miniconda3*** must be loaded (you can check it via `module list`);
- PYENV variable should contain the name of your environment (`echo $PYENV`);

Now you can run a simple test

(!) It is recommended to use SCRATCH file system for code execution

```
(py36) $ source deactivate
       $ cd $SCRATCH
       $ mkdir test0; cd test0
       $ cp /project/6007007/shared/ARG/test/*.* .
```

test example consists of simple python program and a SLURM job submission script;

to submit the job, do:

```
       $ sbatch sub.sh
Submitted batch job <jobid>
```

to check whether the job is queued, running or completed, do:

```
       $ squeue -u <your_username>
```

or

```
       $ squeue -j <jobid>
```

if the queue is empty, your job is completed;

you can check accounting data for the job (see `sacct -e` for available info fields):

```
       $ sacct -j 5282440
   Account      User        JobID               Start                End  AllocCPUS    Elapsed
AllocTRES    CPUTime     AveRSS     MaxRSS MaxRSSTask MaxRSSNode        NodeList
---------- --------- ------------ ------------------ ------------------ ---------- ---------- ----------------
-------------- ---------- ---------- ---------- ---------- ---------- ---------------
def-dad23+    eld439 5282440      2018-02-20T10:47:57 2018-02-20T10:48:28          8   00:00:31
cpu=8,mem=32000M,node=1,billi+   00:04:08                                                    cdr409
def-dad23+           5282440.bat+ 2018-02-20T10:47:57 2018-02-20T10:48:28          8   00:00:31
cpu=8,mem=32000M,node=1   00:04:08     11888K     11888K           0     cdr409          cdr409
def-dad23+           5282440.ext+ 2018-02-20T10:47:57 2018-02-20T10:48:28          8   00:00:31
cpu=8,mem=32000M,energy=18446+   00:04:08        97K        97K           0     cdr409          cdr409
```

you can also check SLURM stderr and stdout for results:

```
       $ cat errfile-<jobid>.out
       $ cat slurm-<jobid>.out
```

Note: If you see neither of these files, while the job is pending or completed, your job has most likely failed due to SLURM issues. It happens on Cedar from time to time. Just resubmit the job later.

4. Using HITRAN atmospheric parameters

4.1 Create local config

Create directory `sasktran` in `$HOME/.config/`

Create file `config.yml` in `$HOME/.config/sasktran/`

Add the following line to the config file:

```
       hitran_directory:  /project/6007007/shared/ARG/hitran
```

4.2 Run a test, thus updating global config

Now you can run a simple test

```
$ cd $SCRATCH
$ mkdir test-hitran; cd test-hitran
$ cp /project/6007007/shared/ARG/test-hitran/*.* .
```

test example consists of simple python program and a SLURM job submission script;

to submit the job, do:

```
$ sbatch sub.sh
```

to check whether the job is queued, running or completed, do:

```
$ squeue -u <your_username>
```

if the queue is empty, your job is completed; check SLURM stderr and stdout:

```
$ cat errfile-<jobid>.out
$ cat slurm-<jobid>.out
```

if the execution was successful, the error file is empty, while the out file will contain output for radiation and weighting functions.

Note: If you see neither of these files, while the job is pending or completed, your job has most likely failed due to SLURM issues. It happens on Cedar from time to time. Just resubmit the job later.

If the job completed successfully, check whether global config file was updated:

```
$ tail $HOME/.local/easybuild/software/2017/Core/miniconda3/4.3.27/envs/$PYENV/share/usask-arg/registry/sasktranif/globalkey.yaml | grep hitran
hitran: {basedirectory: /project/6007007/shared/ARG/hitran}
```

4.3 Delete local config to avoid problem described below

Remove (or rename) local config, e.g.

```
$ rm $HOME/.config/sasktran/config.yml
```

(!) There is a known bug related to sasktran config:
if you have two jobs starting simultaneously, they both try to rewrite the global config at

```
/home/<username>/.local/easybuild/software/2017/Core/miniconda3/4.3.27/envs/<pyenv>/share/usask-arg/registry/sasktranif/globalkey.yaml
```

merging it with your local config from

```
/home/<username>/.config/sasktran/config.yml
```

which causes error.

(!) Note that global config is unique for each environment. If you are setting up NEW environment, you should repeat steps 4.1 to 4.3

5. Installing Osiris Level 1 Services

```
$ source activate py36
```

clear the PYTHONPATH, if you haven't done so yet:

```
(py36) $ export PYTHONPATH=
```

install the osl1 and add the path to the installed libraries to LD_LIBRARY_PATH system variable

```
(py36) $ pip install osl1 -f https://arg.usask.ca/wheels/
(py36) $ export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/.local/easybuild/software/2017/Core/miniconda3/4.3.
27/envs/$PYENV/lib/python3.6/site-packages/osl1/
```

To check whether installation was successful, try:

```
(py36) $ conda list
(py36) $ python
    >>> import sasktran
```

```
export JPLEPH=       C:\Program Files\USask-ARG\OsirisLevel1\JPLEPH
export ODINCDBDIR=
/project/6007007/shared/data/OsirisDataProducts/OsirisConfigurationDatabase/cdb
export
ODINFLIGHTDIR=/project/6007007/shared/data/OsirisDataProducts/OsirisConfigurationDatabase/
flightduration
export ODINORBITDIR=/project/6007007/shared/data/OsirisDataProducts/Level1/orbit;
/project/6007007/shared/data/OsirisDataProducts/Level0/orbit
```

Appendix 1 – Tips and Tricks

There are three data spaces you can use on Cedar:

HOME – is recommended for long-term data, such as local installations and data required for computation

SCRATCH – is recommended for temporary files, e.g. for computations; *scratch* is not backed up, and all files older than 60 days are subject to purging, so you should take care of downloading your data timely.

PROJECT – is recommended to use for sharing data with other group members. Disk space per user is very limited, however, if required, you can utilize shared group disk space. Use it wisely, this space is allocated to the whole group.

You can check the available disk space and the current disk utilization for the *project*, *home* and *scratch* file systems with the command line utility:

```
$ diskusage_report
                  Description                Space          # of files
           Home (user eld439)            9468M/50G            77k/500k
         Scratch (user eld439)            3522M/20T          8880/1000k
        Project (group eld439)            31k/2048k             1/5000k
Project (group rrg-dad230-ac)              40k/10T            11/5000k
   Project (group def-dad230)             93G/1000G            52k/5000k
```

Read more about data storage at https://docs.computecanada.ca/wiki/Storage_and_file_management

To see the priority of your jobs:

```
$ sshare | head -n 1; sshare | grep eld439
  def-dad230_cpu        eld439         1    0.125000       18854      0.000517   0.038160
  def-dad230_gpu        eld439         1    0.125000           0      0.000000   1.000000
  rrg-dad230-ac_cpu     eld439         1    0.111111       78919      0.003354   0.168314
```

Disk quota exceeded error on /project filesystems

By design, new files and directories in /project should normally be created belonging to a project group. The two main reasons why files may be associated with the wrong group are that

- files were moved from /home to /project with the `mv` command; to avoid this, use `cp` instead;
- files were transfered from another cluster using `rsync` or `scp` with an option to preserve the original group ownership.

To guarantee correct ownership, user can change default group to `<def-gname>` using `newgrp` command, before working with /project data space. To check the current default group, use `groups` command:

```
$ groups
      eld439 rrg-dad230-ac rrg-dad230 rrg-dad230-ab def-dad230
$ newgrp def-dad230
$ groups
      def-dad230 eld439 rrg-dad230-ac rrg-dad230 rrg-dad230-ab
```