# Scientific Computing on the Keck Cluster with Anaconda and Python 3

2019-06-09  
jakelly

version .02

### Introduction

This document covers:

1. Connecting and authenticating to the Keck Cluster.
2. Using the slurm scheduler to run Python workloads in both single-node interactive sessions (testing/debugging) and distributed batch jobs (production).
3. Running Python 3 workloads.
4. Accessing common scientific Python packages through **conda** environments**.**
5. Installing additional **conda** packages into user **conda** environments, if required.

Anaconda is a powerful and flexible set of tools for managing scientific Python computing environments. This document barely scratches the surface. Users are encouraged to study the (excellent) Conda user guide at: <https://conda.io/docs/user-guide/index.html>

TensorFlow-CPU is available on all CPU-only nodes, using slurm queue --partition=defq.

TensorFlow-GPU is available on all GPU-equipped nodes, using slurm queue --partition=gpuq.

**Network Connectivity Notes:**

On campus, you will need to be connected through a physical ethernet cable or EduRoam wireless.

Off campus, or on-campus but connected to ChapmanOpen wireless, you will need to have VPN access and be connected through the campus VPN server, vpn.chapman.edu.

**Connecting to the cluster:**

Log into the headnode (keckcluster.chapman.edu) using your full Chapman email as your username:

ssh -l [emailname@chapman.edu](mailto:emailname@chapman.edu) keckcluster.chapman.edu ***<== parameter is lowercase L***

*or*

ssh emailname@chapman.edu@keckcluster.chapman.edu

**SLURM Job Scheduler Notes:**

There are many resources on the web for learning the syntax of SLURM scripts.

Here is one to start with: (ctrl+click to follow the link)  
<https://support.ceci-hpc.be/doc/_contents/QuickStart/SubmittingJobs/SlurmTutorial.html>

Here is the official documentation for sbatch and its script syntax:   
<https://slurm.schedmd.com/sbatch.html>

**Keck Cluster Support and Questions:**

For further questions or assistance, please contact researchcomputing@chapman.edu.

**CPU-only Python workloads**:

1. **Interactive access to CPU nodes:**

Using the slurm job scheduler, access cluster resources and open an interactive bash session.

The below srun command requests:  
 a CPU-only session (--partition defq),   
on one node (--ntasks=1),   
with two CPU cores (--cpus-per-task=2), and   
two megabytes of memory per CPU core (--mem-per-cpu=2048), for a total of 4gb of memory.

[user@chapman.edu@keckcluster:~]$ srun --pty --partition defq --ntasks=1 --cpus-per-task=2 --mem-per-cpu=2048 bash

The slurm job scheduler selects a node with enough free resources, and runs the program specified at the end of the request (bash) on that node.

When the session opens, the prompt changes to the name of that node.

[user@chapman.edu@node001:~]$ <== node001

In the new session, identify the current (default) Python settings and version:

[user@chapman.edu@node001:~]$ module list

Currently Loaded Modulefiles:  
1) gcc/7.2.0 2) slurm/17.11.8

[user@chapman.edu@node001:~]$ which python

/usr/bin/python

[user@chapman.edu@node001:~]$ python --version

Python 2.7.5 <== default system python

Load the environment-settings for Anaconda+Python 3.6, using the module command:

[user@chapman.edu@node001:~]$ module load anaconda3/current

The shell session is modified to use Anaconda's Python3 instead of the system default.

Verify the altered Python settings and version:

[user@chapman.edu@node001:~]$module list

Currently Loaded Modulefiles:

1) gcc/7.2.0 2) slurm/17.11.8 3) anaconda3/current <== anaconda3 module loaded

[user@chapman.edu@node001:~]$which python

/cm/shared/apps/anaconda3/bin/python

[user@chapman.edu@node001:~]$python –-version

Python 3.6.6 :: Anaconda custom (64-bit) <== python version changed

Create and/or activate an isolated conda environment for your project/workload:

[user@chapman.edu@node001:~]$conda info --envs

# conda environments:

#

base \* /cm/shared/apps/anaconda3 <== base/root system conda environment (read-only)

[user@chapman.edu@node001:~]$conda create --name your\_project\_name

/ Solving environment: done

## Package Plan ##

environment location: /home/user@chapman.edu/.conda/envs/your\_project\_name

Proceed ([y]/n)? y

Preparing transaction: done

Verifying transaction: done

Executing transaction: done

#

# To activate this environment, use:

# > source activate your\_project\_name

#

# To deactivate an active environment, use:

# > source deactivate

#

[user@chapman.edu@node001:~]$conda info --envs

# conda environments:

#

base \* /cm/shared/apps/anaconda3

your\_project\_name /home/user@chapman.edu/.conda/envs/your\_project\_name <== new user environment

[user@chapman.edu@node001:~]$ source activate your\_project\_name <== make user environment active environment

(your\_project\_name)

[user@chapman.edu@node001:~]$ <== prompt prefixed with active environment name

The list of currently loadable Python packages is the sum of the active user-environment’s packages and the base system-environment’s packages:

(your\_project\_name)

[user@chapman.edu@node001:~]$ **conda list** <== list packages installed in active/user environment

# packages in environment at /home/user@chapman.edu/.conda/envs/your\_project\_name:

#

# Name Version Build Channel

...list of packages installed by user...

(your\_project\_name)

[user@chapman.edu@node001:~]$conda list -n base <== list packages installed in base/root environment

# packages in environment at /cm/shared/apps/anaconda3:

#

# Name Version Build Channel

...list of packages installed by root...

To install additional packages into the current environment:

(your\_project\_name)

[user@chapman.edu@node001:~]$conda search package\_name <== verify installable package exists

Loading channels: done

# Name Version Build Channel

package\_name 0.11.0 np15py26\_2 pkgs/free

package\_name 0.12.0 np15py26\_3 pkgs/free

package\_name 0.13.0 np15py37\_ce1 pkgs/free <== newer versions last

...list of all available package versions...

(your\_project\_name)

[user@chapman.edu@node001:~]$conda install package\_name<== installs latest version plus dependencies

# Solving environment: done into active environment location (user homedir)

## Package Plan ##

environment location: /home/user@chapman.edu/.conda/envs/your\_project\_name

added / updated specs:

- package\_name

The following packages will be downloaded:

package | build

---------------------------|-----------------

mkl\_fft-1.0.6 | py37h7dd41cf\_0 150 KB

intel-openmp-2019.1 | 144 885 KB

...installation output...

(your\_project\_name)

[user@chapman.edu@node001:~]$conda install package1 package2 package3<== install all required packages at

once for best dependency resolution

Once your user conda environment is prepared, test/debug/execute your Python workload.

To deactivate your user conda environment:

(your\_project\_name) <== prefixed environment name

[user@chapman.edu@node001:~]$source deactivate

[user@chapman.edu@node001:~]$ <== removed prefixed environment name

To end the SLURM interactive session, exit the shell:

[user@chapman.edu@node001:~]$ exit

The prompt changes back to the hostname for the headnode:

[user@chapman.edu@keckcluster:~]$

1. **Batch access:**

When submitting production jobs to the cluster, include the syntax to load the environment for Anaconda/Python3 in the slurm script.

module load anaconda3/current

Example slurm sbatch script (equivalent to interactive session above):

[user@chapman.edu@keckcluster:~/tf-test]$ cat anaconda3-python-cpu-sbatch.sh

#!/bin/env bash

#SBATCH --partition=defq

#SBATCH --ntasks=1

#SBATCH --cpus-per-task=2

#SBATCH –-mem-per-cpu=2048

#SBATCH --time=01:00:00

#SBATCH --job-name=anaconda3-python3-cpu-test-job-1

module load anaconda3/current

conda create --name your\_project\_name <== skips quickly if already exists on node

source activate your\_project\_name

conda install package1 package2 package3 <== skips quickly if already installed on node

srun -l '~/anaconda3-test/python3-scicomp-cpu.py' <== script/command/shell-pipe to be run distributedly

Run the job by submitting the sbatch script to slurm:

[user@chapman.edu@keckcluster:~/tf-test]$ sbatch anaconda3-python-cpu-sbatch.sh

Review the slurm output file.  
(By default named slurm-NNN.out; contains stdout and stderr for all nodes; NNN = slurm jobID.)

[user@chapman.edu@keckcluster:~/tf-test]$ cat slurm-NNN.out

See all running slurm jobs using the command:

[user@chapman.edu@keckcluster:~/tf-test]$ squeue

**GPU-enabled Python workloads:**

1. **Interactive GPU-enabled session:**

Using the slurm job scheduler, access cluster resources and open an interactive bash session.

The only difference when requesting GPU nodes is the partition requested (--partition gpuq), and the inclusion of the number of GPU cards required, --gres=gpu:2.

This srun command requests:  
a GPU-enabled session (--partition gpuq),  
on one node (--ntasks=1),   
using two CPU cores (--cpus-per-task=2),  
eight megabytes of memory per CPU core (--mem-per-cpu=8192), for a total of 16gb of memory, and  
two GPU cards (--gres=gpu:2).

[user@keckcluster:~]$ srun --pty --partition gpuq --ntasks=1 --cpus-per-task=2 --mem-per-cpu=8192 --gres=gpu:2 bash

The slurm job scheduler selects a node with enough free resources, and runs the program specified at the end of the request (bash) on that node.

When the session opens, the prompt changes to the name of the selected node.

[user@chapman.edu@node009:~]$ <== node009

The remainder of the process is identical to the CPU-workload process above.

1. **Batch access:**

The only differences when requesting GPU nodes are the partition requested (--partition gpuq), and the inclusion of the number of GPU cards required, --gres=gpu:2.

Example sbatch script (equivalent to interactive session above):

[user@chapman.edu@keckcluster:~/anaconda3-test]$ cat anaconda3-python3-cpu-sbatch.sh

#!/bin/env bash

#SBATCH --partition=gpuq

#SBATCH --ntasks=1

#SBATCH --cpus-per-task=2

#SBATCH –-mem-per-cpu=8192

#SBATCH --gres=gpu:2

#SBATCH --time=01:00:00

#SBATCH --job-name= run-anaconda-python3-gpu-test-job-2

module load anaconda3/current

conda create --name your\_project\_name <== skips quickly if already exists on node

source activate your\_project\_name

conda install package1 package2 package3 <== skips quickly if already installed on node

srun -l '~/anaconda3-test/python3-scicomp-gpu.py' <== script/command/shell-pipe to be run distributedly

The remainder of the process is identical to the CPU-workload batch process above.