ilifu Online Training – Resource Allocation

Theophilus Matsepane

System Administrator & User Support, ilifu
University of the Western Cape, March 2026

ilifu

IDIA Inter-University Institute for Data Intensive Astronomy

UNIVERSITY of the WESTERN CAPE

ilifu: a shared resource-limited cluster

1. Supports a diverse range of projects
   - Astronomy and Bioinformatics
   - Varying resource requirements

   e.g.
   CPU's, Memory,
   Running Time, GPU's etc.

2. Shared environment
3. Resource-limited

ilifu: Understanding Compute Resources

RAM (memory)
"Short-term memory"

- High-speed temporary storage
- Data & instructions for active tasks

CPU Cores
Complex logic & moderate parallelism

- Few powerful cores
- Can run parallel software (MPI, OpenMP)
- Best for: Complex logic, task parallelism

GPU Cores
Massive parallel calculations

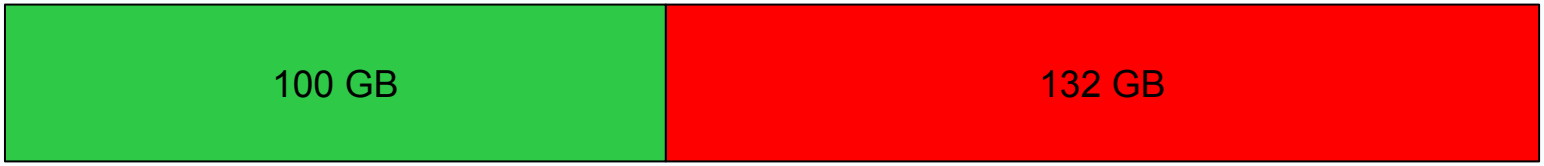
- Thousands of smaller cores
- Best for: Matrix ops, simulations,
- identical parallel calculations

Analogy:
CPU = Team of 10 experts working together on complex problems
GPU = 1000 interns doing the same simple calculation simultaneously

Efficient Usage of Resources

- Resource Allocation: "Picking the right amount of resources for your jobs"

  e.g.
  if a job uses 100 GB of RAM, don't want to request 232 GB




```
Memory Utilized: 1.82 GB
Memory Efficiency: 2.84% of 64.00 GB (64.00 GB/node)
```

- 4 GB can handle this, over 60 GB wasted.
- Best practices: Resource Allocation Guide

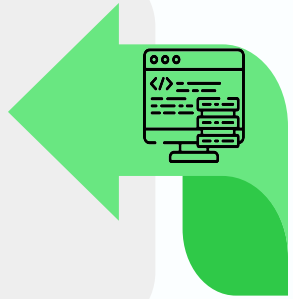
Services and Partitions

Login node

Run SLURM & bash commands
cd, mkdir, ls, etc

Jupyter + JupyterDev + Devel

Development space
New code / workflows / routines
Debugging / testing software

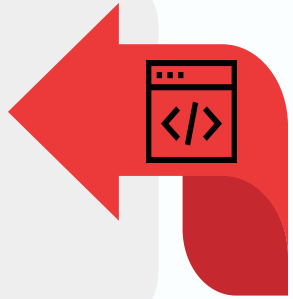


Main partition

Stable, computationally heavy processing

HighMem/GPU

For single-high memory jobs or GPU resources



Services and Partitions: Jupyter

Jupyter (Jupyter.ilifu.ac.za)

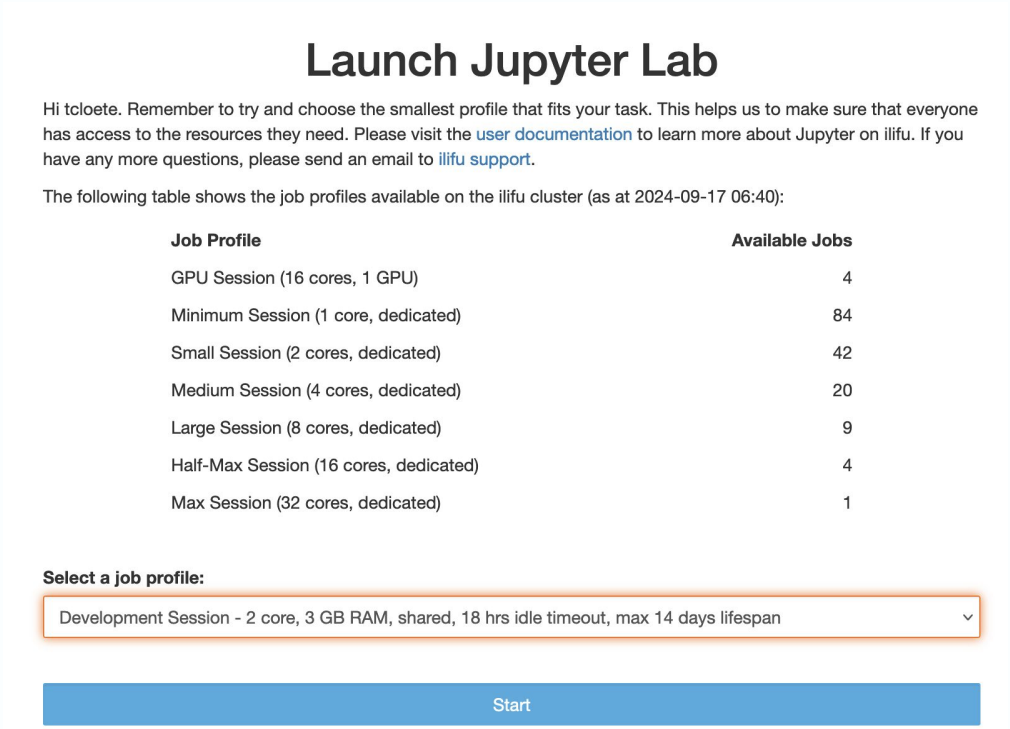
Primary Use: Development & debugging of new code, software, workflows, and routines

Interactive Features:

- Full Jupyter notebook support
- Real-time code execution
- Visual output integration
- Immediate feedback loop

When to Use:

- Writing new code/prototypes
- Testing workflows interactively
- Short analysis routines
- Stable workflows (non-Slurm alternative)



Launch Jupyter Lab

Hi tcloete. Remember to try and choose the smallest profile that fits your task. This helps us to make sure that everyone has access to the resources they need. Please visit the user documentation to learn more about Jupyter on ilifu. If you have any more questions, please send an email to ilifu support.

The following table shows the job profiles available on the ilifu cluster (as at 2024-09-17 06:40):

Job Profile	Available Jobs
GPU Session (16 cores, 1 GPU)	4
Minimum Session (1 core, dedicated)	84
Small Session (2 cores, dedicated)	42
Medium Session (4 cores, dedicated)	20
Large Session (8 cores, dedicated)	9
Half-Max Session (16 cores, dedicated)	4
Max Session (32 cores, dedicated)	1

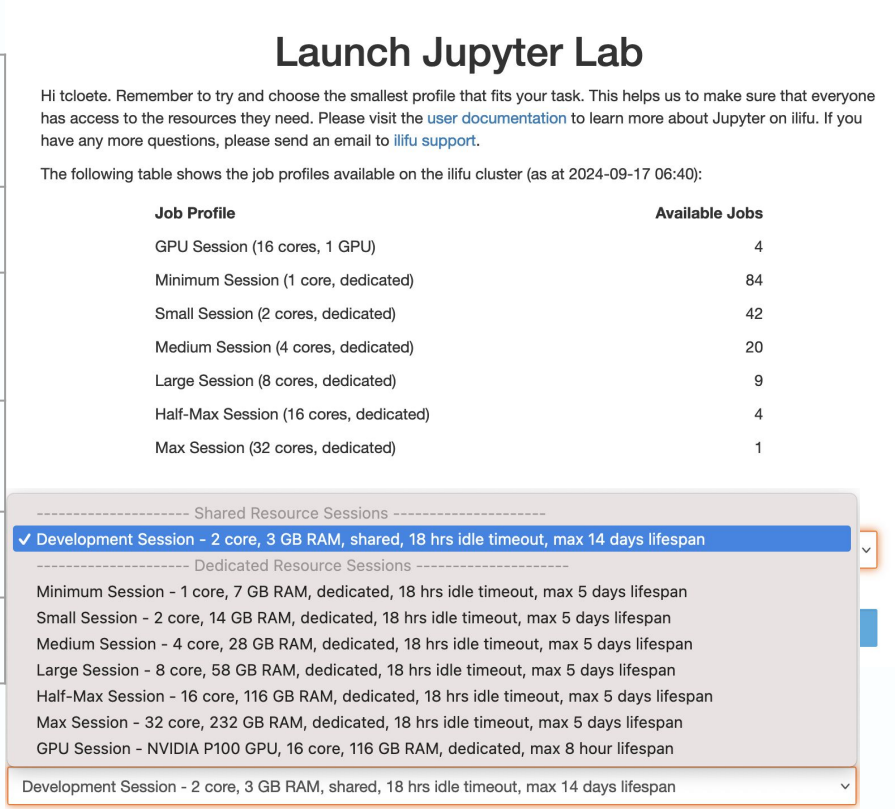
Select a job profile:

Development Session - 2 core, 3 GB RAM, shared, 18 hrs idle timeout, max 14 days lifespan

Start

Jupyter Sessions Comparison

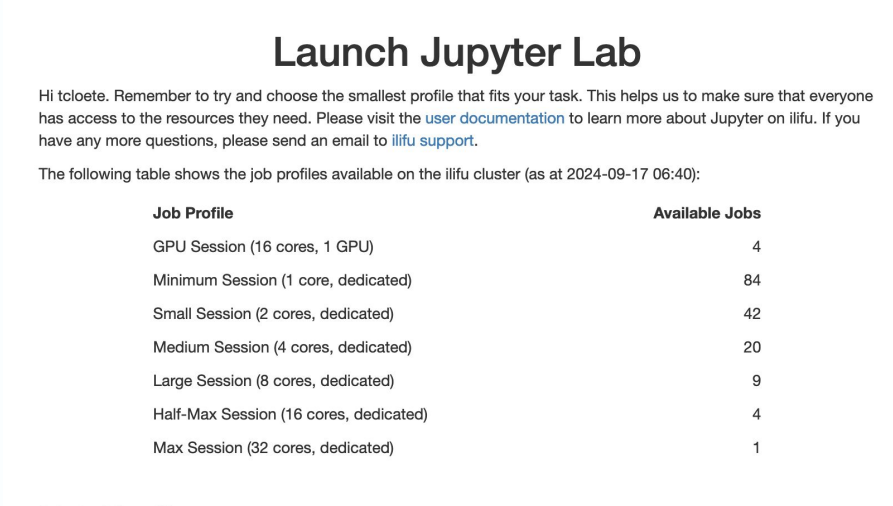
Feature	Shared	Dedicated
Nodes	1 (jupyter-001)	12
Resources	2 cores + 3 GB	1-32 cores + 7-232 GB /GPU
Sharing	With other users	Just for you
Access	Immediate	May queue
Best for	Light dev & test	Heavy work



Note: For heavy work, submit a job to Main nodes using sbatch

Jupyter: Dedicated Resource Allocation

- Dedicated resources: CPU and Memory is allocated to you throughout the job duration
- Memory often most important
- Jupyter shows current **memory usage** at the bottom
- Emailed about usage stats e.g. low memory usage
- Shut down your session

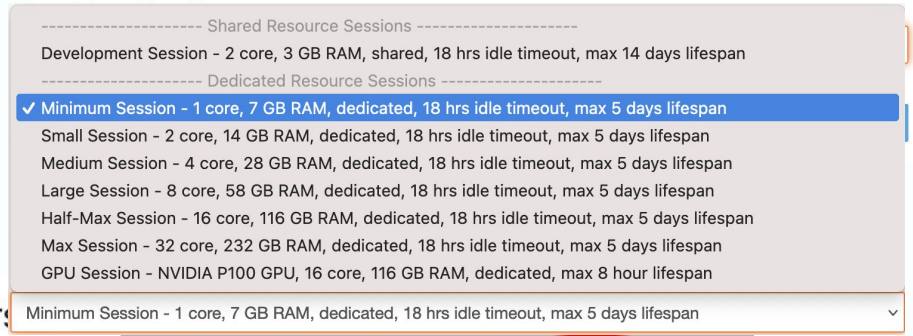


Launch Jupyter Lab

Hi tcloete. Remember to try and choose the smallest profile that fits your task. This helps us to make sure that everyone has access to the resources they need. Please visit the user documentation to learn more about Jupyter on ilifu. If you have any more questions, please send an email to ilifu support.

The following table shows the job profiles available on the ilifu cluster (as at 2024-09-17 06:40):

Job Profile	Available Jobs
GPU Session (16 cores, 1 GPU)	4
Minimum Session (1 core, dedicated)	84
Small Session (2 cores, dedicated)	42
Medium Session (4 cores, dedicated)	20
Large Session (8 cores, dedicated)	9
Half-Max Session (16 cores, dedicated)	4
Max Session (32 cores, dedicated)	1



Services and partitions: Devel

- Devel (--partition=Devel)
  - Rapid Development Environment
  - Key Features:
    - ➢ **Instant job submission** – no queue waiting
    - ➢ **Shared resources** – not dedicated allocation
    - ➢ **Interactive shell access** – direct development
    - ➢ **Workflow testing** – ideal for debugging
- Access: `sinteractive`

```
theo@slurm-login:~$ sinteractive
Starting interactive Slurm session.
srun: job 12966667 queued and waiting for resources
srun: job 12966667 has been allocated resources
theo@compute-001:~$
```

Services and partitions: Main

- **Main** partition
  - Default Slurm partition
  - Generally for stable, computationally-heavy workflows and pipelines
  - Can be used for:
    - Many small jobs OR
    - A few large jobs allocated many resources
  - For large workflows, better to first test on Devel or Jupyter (e.g. subset of data / fewer iterations)

```
theo@slurm-login:~$ sinfo
PARTITION  AVAIL  TIMELIMIT  NODES  STATE NODELIST
Main*         up 14-00:00:0     12   mix- compute-[002,004-005,008,013,015-016,019,204,208,210,251]
Main*         up 14-00:00:0     15    mix compute-[003,011-012,022,101-105,206-207,211,227,252-253]
Main*         up 14-00:00:0     59  alloc compute-[006-007,009-010,014,017-018,020-021,023-026,201-203,205,209,212-226,228-250,254-256]
Jupyter       up   infinite      6    mix jupyter-[002-005,011-012]
Jupyter       up   infinite      7  alloc jupyter-[006-010,013-014]
JupyterGPU    up 14-00:00:0      2    mix gpu-[003-004]
JupyterDev    up   infinite      1  alloc jupyter-001
HighMem       up 14-00:00:0      4    mix highmem-[001,003,007-008]
HighMem       up 14-00:00:0      4  alloc highmem-[002,004-006]
GPU           up 14-00:00:0      5    mix gpu-[001-004,007]
GPU           up 14-00:00:0      2   idle gpu-[005-006]
Devel         up 5-00:00:00      1  alloc compute-001
theo@slurm-login:~$
```

Services and partitions: GPU and HighMem

- **HighMem** partition
  - Single high-memory jobs that can't be split into multiple jobs using MPI
- **GPU** partition
  - Jobs making use of GPUs
  - Not for jobs that only require CPUs (rather use Main)

```
theo@slurm-login:~$ sinfo
PARTITION  AVAIL  TIMELIMIT  NODES  STATE NODELIST
Main*         up 14-00:00:0     12   mix- compute-[002,004-005,008,013,015-016,019,204,208,210,251]
Main*         up 14-00:00:0     15    mix compute-[003,011-012,022,101-105,206-207,211,227,252-253]
Main*         up 14-00:00:0     59  alloc compute-[006-007,009-010,014,017-018,020-021,023-026,201-203,205,209,212-226,228-250,254-256]
Jupyter       up   infinite      6    mix jupyter-[002-005,011-012]
Jupyter       up   infinite      7  alloc jupyter-[006-010,013-014]
JupyterGPU    up 14-00:00:0      2    mix gpu-[003-004]
JupyterDev    up   infinite      1  alloc jupyter-001
HighMem       up 14-00:00:0      4    mix highmem-[001,003,007-008]
HighMem       up 14-00:00:0      4  alloc highmem-[002,004-006]
GPU           up 14-00:00:0      5    mix gpu-[001-004,007]
GPU           up 14-00:00:0      2   idle gpu-[005-006]
Devel         up 5-00:00:00      1  alloc compute-001
theo@slurm-login:~$
```

Primary Resources

http://docs.ilifu.ac.za/#/tech_docs/resource_allocation

- Primary resources
  1. CPU
  2. Memory
  3. Wall-time

- Notes
  - Nodes have 2 CPUs (sockets), each with 16 cores, all of which Slurm calls “CPUs”
  - **CPU** is the **physical processor chip** installed on the motherboard.
  - **Core** is an individual, **independent processing unit** within that chip.

How to Allocate Resources?

- **The Golden Rule**

1. Accurately determine your resource requirements
2. Use what you require – not more, not less

- **Why it matters?**
  - **When you overallocate:**
    - Resources allocated but not used
    - Wasted computing capacity
    - Other users wait longer
    - Clusters efficiency decreases
  - Better fairshare priority for future job submissions.

Determining resource requirements

1. Determine parallelism of software
2. Profiling previous similar jobs
3. Scaling up test jobs

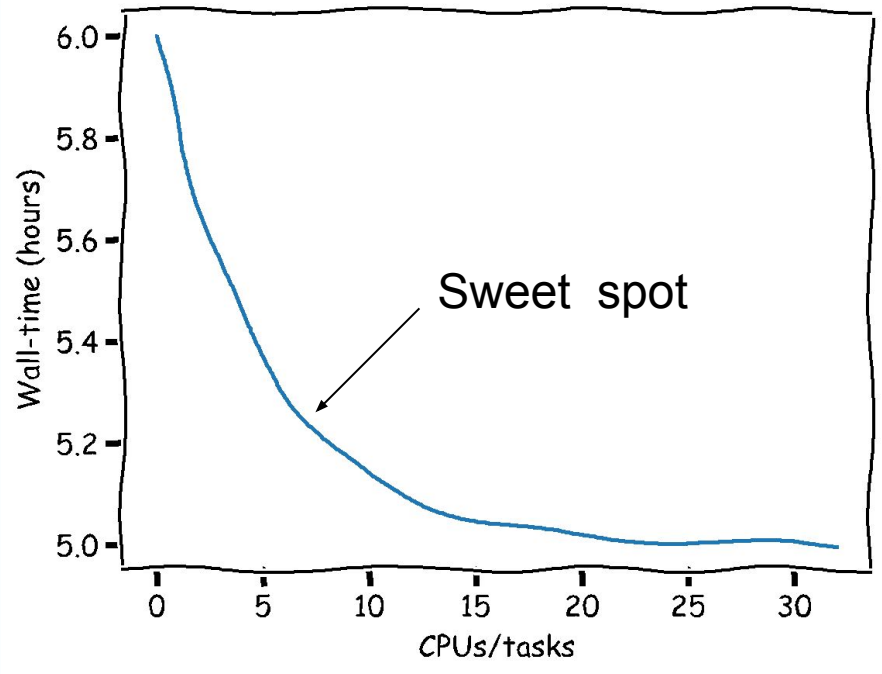
Determining resource requirements

- Determining parallelism of software
  - Many software packages only use 1 CPU
  - CPU-level parallelism: Max 1 Node of CPUs
  - Task-level parallelism: >= 1 Node

Determining resource requirements

- Determining parallelism of software

Myth	Reality
Double cores = ½ the wall time	Most softwares doesn't scale linearly
More cores always faster	Communication overhead grows
Maximum cores = Best	Maximum performance ≠ Most efficient



✅ Your Checklist

- □ Test with small cores first
- □ Find your efficiency sweet spot
- □ Consider many small jobs vs one big job
- □ Don't overallocate "just in case"
- □ Monitor and adjust

Remember:

Being a good cluster citizen = Faster results for everyone!

Finding Job Information

Finding Job IDs & Resource Usage

- Where to Find Job IDs?
  - Shown when you submit a job
  - Can search for historical jobs

- Search for jobs by name and date range:
  - `sacct -X --name=my-job --starttime=YYYY-MM-DD --endtime=YYYY-MM-DD`
    - Or (Shorter command version)
  - `Sacct -X –name=my-job -S=YYYY-MM-DD -E=YYYY-MM-DD`
    - Omit job name or end time to show all jobs

- Once you have job ID, you can search for specific information about resource usage

Tracking & Optimizing Memory Usage

Memory usage: Track, Learn, Optimize

- Find MaxRSS (Maximum Resident Memory)
  - MaxRSS = Peak memory usage (sampled every 20s)
  - Compare actual vs requested for job 123456:
  - `sacct -j 123456 --unit=G -o JobID,JobName,MaxRSS,ReqMem`

- Once you know actual memory usage:

  ✅ Add **~10-20% buffer:**
  - E.g. Peak 7.25 GB → Request 8.5-9 GB

  ✅ Benefits:
  - Avoids out-of-memory (OOM) crashes
  - Don't waste memory
  - Better resource utilization

⚠️ Too little = Crash!

⚠️ Too much = Waste!

```
theo@slurm-login:~$ sacct -j 12966133_19 --unit=G
-o JobID,JobName,MaxRSS,ReqMem
JobID           JobName     MaxRSS     ReqMem
------------ ---------- ---------- ----------
12966133_19  myarrayjob                 3.02G
12966133_19+      batch      0.02G
12966133_19+     extern
theo@slurm-login:~$
```

CPU & Memory Efficiency

CPU & Memory Usage: Are You Wasting Resources?

- Quick Efficiency Check with `seff`
  - Show Slurm resource efficiency (seff) for job ID 12966133_19 / 12971939

```
theo@slurm-login:~$ seff 12966133_19
Job ID: 12966152
Array Job ID: 12966133_19
Cluster: ilifu-slurm2021
User/Group: theo/idia-group
State: COMPLETED (exit code 0)
Cores: 1
CPU Utilized: 00:00:00
CPU Efficiency: 0.00% of 00:00:11 core-walltime
Job Wall-clock time: 00:00:11
Memory Utilized: 21.66 MB
Memory Efficiency: 0.70% of 3.02 GB (3.02 GB/core)
theo@slurm-login:~$
```

```
theo@slurm-login:~$ seff 12971939
Job ID: 12971939
Cluster: ilifu-slurm2021
User/Group: theo/idia-group
State: COMPLETED (exit code 0)
Cores: 1
CPU Utilized: 00:01:22
CPU Efficiency: 97.62% of 00:01:24 core-walltime
Job Wall-clock time: 00:01:24
Memory Utilized: 1.21 GB
Memory Efficiency: 60.56% of 2.00 GB (2.00 GB/node)
```

📊 CPU Efficiency:

- < 50% → Too many cores requested
- > 90% → Good utilization

📊 Memory Efficiency:

- < 50% → Too much memory requested
- > 95% → Risk of OOM crash

💡 Sweet Spot: 70-90% efficiency

Wall-Time Usage - Right-Sizing Your Job

Wall-Time: Estimate Accurately, Wait Less!

- Why Wall-Time Matters ⏱️?
  - Accurate estimation = Better slurm scheduler efficiency
  - Better efficiency = Reduce job wait time
- Check Used vs. requested wall-time for job ID 12971939
  - `sacct -j 12971939 -o jobID,jobName,Elapsed,TimeLimit`

```
theo@slurm-login:~$ sacct -j 12971939 -o jobID,jobName,Elapsed,TimeLimit
JobID           JobName    Elapsed  Timelimit
------------ ---------- ---------- ----------
12971939     combined_+   00:01:24   00:06:00
12971939.ba+      batch   00:01:24
12971939.ex+     extern   00:01:24
```

  - Once you know actual runtime:
    - Add **~20-30% buffer**
      - Avoids job timeout crashes
    - Avoid excessive wall-time

⚠️ Too little = Job killed!

– Need longer time? Contact us!

📧 support@ilifu.ac.za

How to Estimate Wall-Time Accurately?

- **The Problem**:
  - Accurately estimating wall-time is difficult – but essential for efficient scheduling
- **The Solution: Profile with Test Jobs**
  - Step 1: Start small
    - Run test with subset of data (1-10%), 1-2 cores
    - Measure wall-time & memory
    - If it crashes → No problem! (small investment)
  - Step 2: Scale & Learn
    - Increase data size → How does time grow?
    - Add more cores → Find efficiency sweet spot

Test	Data %	Core	Wall-Time (min)	Memory (GB)
1	1	1	2	0.5
2	10	1	5	4.3
3	25	2	12	10.1

  - Step 3: Project + Buffer
    - Project for full data: e.g., 10% → 5 min = 100% → 50 min
    - ADD 20-30% BUFFER: Request 65-70 minutes
    - Why? Cluster load varies, data differences, safety!
- **Golden Rule:** Test small → Measure → Scale → Add buffer → Submit

Scaling tests on running jobs

- Get MaxRSS for running job
  - `sstat -j 123456 -o MaxRSS`
  - Given in kB units. Divide by 1024^2 for GB

- Display real time stats on dashboard (`top` / `htop`)
  - For sbatch:
    First ssh into the login node using authentication forwarding.
    `ssh -A <username>@slurm.ilifu.ac.za`

    It's required to have a job running on a worker node.
    You can then ssh into that worker node (e.g. node 102)
    `ssh compute-102`
  - For Jupyter: can simply open a new terminal.
  - Now Run: `htop -u $USER`

- Can monitor real-time usage

Maximum Resources

- If using **all** CPUs or memory, node becomes fully allocated
  - Any remaining CPUs / memory unavailable to other jobs (incl. your own)

E.g.
Typical worker node: 32 CPU and 232 GB RAM

Job Requesting: 2 CPU and 232 GB RAM == Full Node
30 CPU **not accessible to other jobs**

If possible to split into two smaller jobs, if they ran on different nodes then:

1 CPU and 116 GB
31 CPUs accessible

1 CPU and 116 GB
31 CPUs accessible

```
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 32
CPU Utilized: 02:51:38
CPU Efficiency: 96.35% of 02:58:08 core-walltime
Job Wall-clock time: 00:05:34
Memory Utilized: 10.04 GB
Memory Efficiency: 33.47% of 30.00 GB (30.00 GB/node)
```

1 node is fully occupied by 300 GB is not used and accessible.

Account allocation

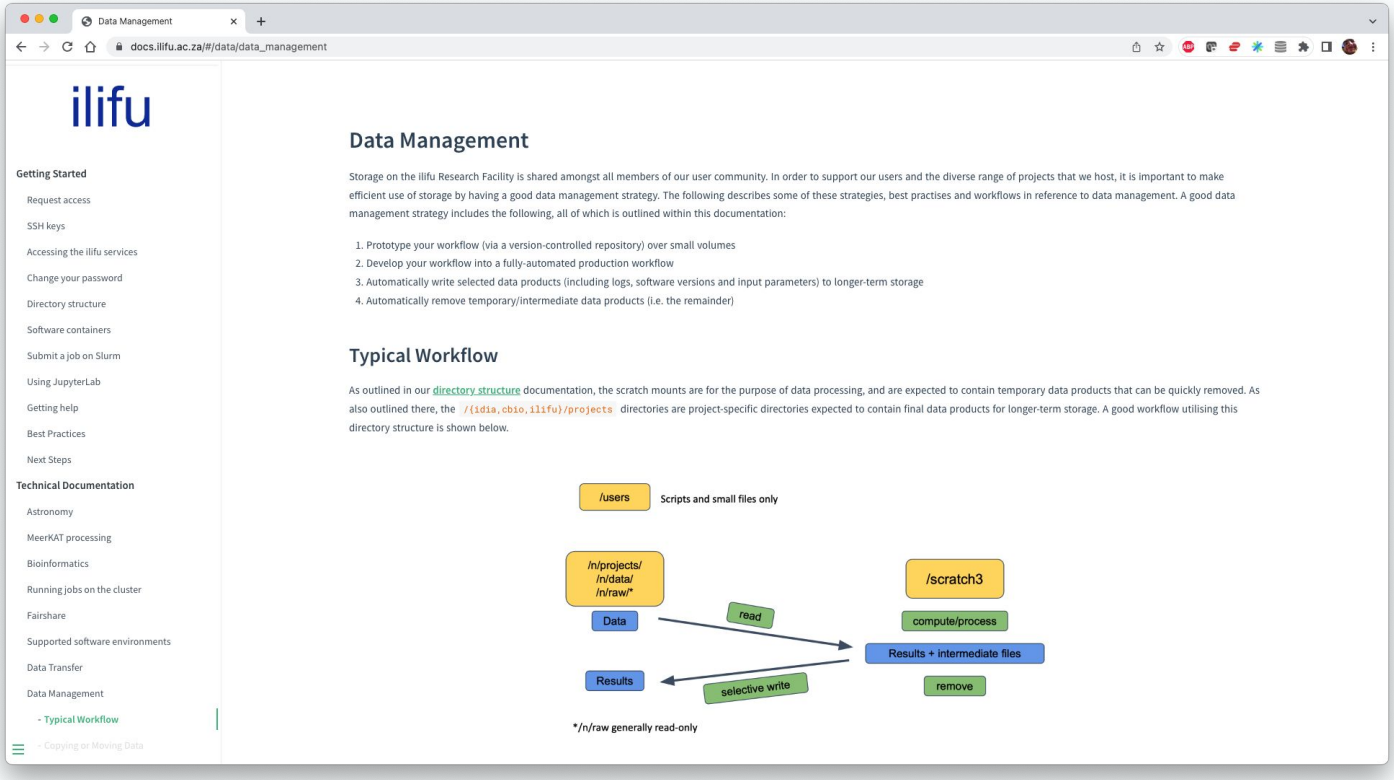
- Each ilifu project has a Slurm account
- Resource usage charged against account (affects fairshare)
- View your accounts:
  - `shelp`
- View your default account
  - `sacctmgr show user $USER`
- Change default
  - `sacctmgr modify user name=${USER} set DefaultAccount=<account>`
- Set account (after #SBATCH for sbatch jobs)
  - `--account=b05-pipelines-ag`

Resource Allocation Guide

DEMO TIME!

Data Management Guidelines

- Hot off the press!
- https://docs.ilifu.ac.za/#/data/data_management



Best practices

- Don’t run software / heavy processes / scp on the login node
  - Only submit jobs and run SLURM commands (sbatch, srun, squeue, etc)
  - Use transfer.ilifu.ac.za to transfer data (external/internal), not login node
- Before running a large job, identify the available resources
  - Use sinfo. Don’t hog the cluster. Reduce your allocation if possible
  - Increase likelihood of jobs running with less memory and less walltime
- Use sbatch (srun / screen / tmux / mosh are volatile)
- Cleanup files that aren’t needed
  - Old raw data, temporary products, /scratch data, etc
- Don’t place large files in your home directory (/users)
- Use Singularity (you cannot install software on the nodes)

Thank you!

Remember our support channels!

support@ilifu.ac.za
https://docs.ilifu.ac.za