

Onboarding

Julia on HLRS Cluster/Laptops

HLRS Training Cluster

The cluster training.hlr.de has two types of nodes.

CPU nodes

- “skl”
- 2x Intel Skylake
- 40 cores total

GPU nodes

- “clx-ai”
- 2x Intel Cascade Lake
- 36 cores total
- 8x NVIDIA V100

Jobs are scheduled with PBS Pro.

- Submit a job:
 - `qsub job_script.sh`
- Check on your queued/running jobs:
 - `qstat -nw`

VS Code → HLRS Cluster

Run VS Code on a cluster node via SSH.

Login node

- Works fine, just connect to
 - `accountname@training.hlrs.de`

Compute node

- At HLRS, possible but inconvenient
 - `SetEnv PBS_JOBID=...`
 - `SSH ProxyJump`

We will stay on Login nodes for the course.

To get Julia, load the necessary system modules.

- Modules on the HLRS training cluster
 - `module use julia`
 - `module use nvidia/nvhpc` # MPI+CUDA
 - `module use compiler/nvidia` # MPI+CUDA
- Outside of the course: If there is no (working) system module, use standard binaries provided by [juliaup](#).

Comment: Julia depot is on the parallel file system.

- Julia depot = where Julia stores stuff
 - packages
 - binary dependencies
 - ...
- Environment variable: JULIA_DEPOT_PATH
- Why not \$HOME?
 - Quotas
 - Read-only from compute jobs (sometimes)

Julia VS Code integration via extension.

The screenshot displays the VS Code interface with the Julia extension. The Explorer sidebar on the left shows the project structure, including folders like `.github`, `.vscode`, `benchmark`, `bin`, `deps`, `docs`, `src`, `test`, `.gitignore`, `.travis.yml`, `appveyor.yml`, and `LICENSE.md`. The Julia Explorer sidebar shows the workspace structure, including folders for `Core`, `InteractiveUtils`, `PlotThemes`, `Plots`, and a `Julia REPL`. The main editor shows a Julia script with the following code:

```
9      ...end
10      ...return maxiter
11  end mandel (generic function with 1 method)
12
13  for i in 1:10
14      ...println(i)
15  end
16
17  map
```

A tooltip for the `map` function is visible, showing its signature `map(f, c...) -> collection` and a description: "Transform collection `c` by applying `f` to each element. For multiple collection arguments, apply `f` elementwise." It also includes examples:

```
julia> map(x -> x * 2, [1, 2, 3])
3-element Array{Int64,1}:
 2
```

The bottom panel shows the Julia REPL with the output of the script:

```
4
5
6
7
8
9
10
```

The status bar at the bottom indicates the current file is `sp/isdeflocals*`, the environment is `Julia env: v1`, and the cursor is at `Ln 17, Col 4`.

On the cluster, the extension requires a wrapper.

- [Julia: Executable Path](#) should point to a wrapper script, like this one:

```
#!/bin/bash
[...]  
  
# Load modules  
module load julia  
module load nvidia/nvhpc  
module load compiler/nvidia  
  
# Act like Julia (i.e. pass on all arguments)  
exec julia "${@}"
```

Let's do it!

→ `exercises/Day1/1_cluster_onboarding`

HLRS Laptops

Most of the course can be completed on the laptops.

- Equipped with NVIDIA GPU
- Jupyter + VS Code
 - jupyter lab
 - code
- Course materials
 - [\\$HOME/JuliaHLRS24](#)

Let's start Jupyter!

→ `cd JuliaHLRS24`

→ `jupyter lab`

→ `notebooks/Day1/1_julia_fundamentals.ipynb`

