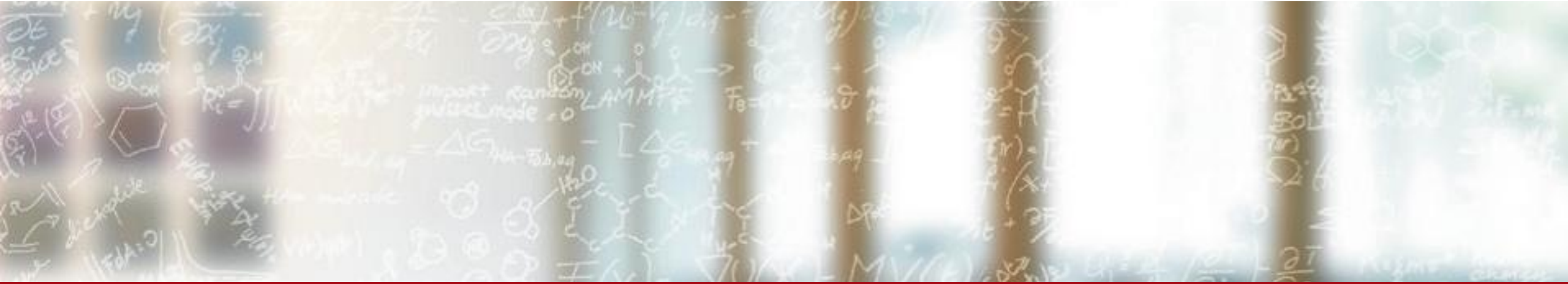




CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Interactive computing on Alps

- **Jupyter** notebooks via JupyterHub
- **Julia** in JupyterHub, with GPU profiling
- Best practices for setting up **VSCode**

27 May 2025

Today's topics

1. Introduction to **JupyterHub** on Alps vClusters
2. Using **Julia** with JupyterHub, including GPU profiling
3. Best practices for setting up **VSCode** on Alps vClusters



Tim Robinson

HPC Platform
Service Manager



Samuel Omlin

Research Software
Engineer



Prashanth Kanduri

Research Software
Engineer

JupyterHub

- Run notebooks on compute nodes on Alps
- Separate deployment per vCluster
 - <https://jupyter-daint.cscs.ch>
 - <https://jupyter-santis.cscs.ch>
 - <https://jupyter-clariden.cscs.ch>
- Documentation is available in [CSCS Docs](#)
- Software environment is provided by [uenv](#)
 - You can use your own virtual environment by [creating a kernel](#)
 - You can load your own uenv (⊕Advanced options)
- Initial support for container images
- Happy to receive your [feedback and requests for enhancement](#)

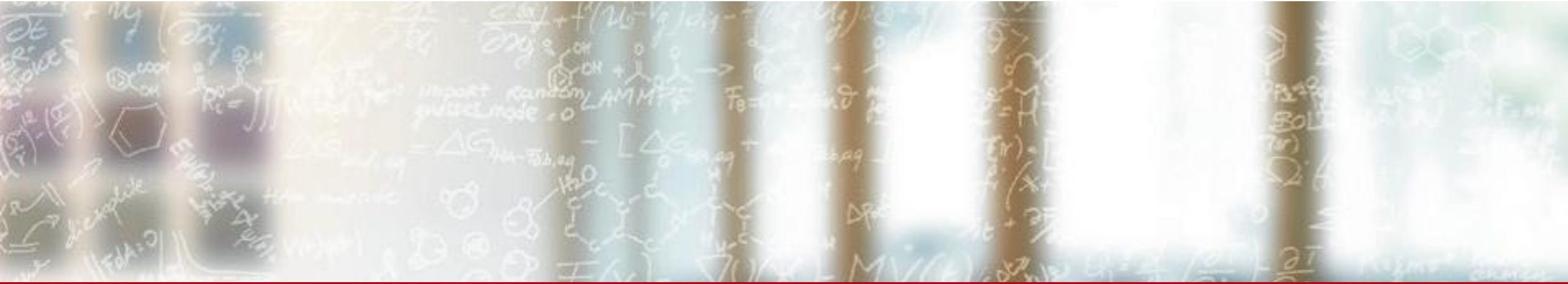




CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Using Julia in JupyterHub

Webinar at the Swiss National Supercomputing Centre (CSCS), ETH Zurich

Dr. Samuel Omlin¹

May 27 2025

¹Swiss National Supercomputing Centre (CSCS), ETH Zurich

Agenda

- Introduction: Julia for HPC
- Setting up Julia in JupyterHub
- GPU profiling using Julia
- Documentation
- Conclusions

Introduction: Julia for HPC

Julia for HPC: low level CUDA kernel

```
function cumsum_dim1!(B, A)
    iy = (blockIdx().y-1) * blockDim().y + threadIdx().y
    iz = (blockIdx().z-1) * blockDim().z + threadIdx().z
    tx = threadIdx().x
    pow2 = Int(log2(blockDim().x))
    cumsum_ix = 0.0
    for ix_offset = 0 : blockDim().x : size(A,1)-1
        ix = threadIdx().x + ix_offset
        @inbounds val = A[ix,iy,iz]           # Read the x-dimension chunk into thread-local registers.
        cumsum_tx = val
        pos = 1
        width = 2
        for i = 1:pow2
            cumsum_lower = shfl_sync(CUDA.FULL_MASK, cumsum_tx, pos, width)
            if ((tx-1) % width) > pos-1
                cumsum_tx += cumsum_lower
            end
            pos *= 2           # pos = 2^(pow2-1)
            width *= 2        # width = 2^pow2
        end
        cumsum_tx += cumsum_ix
        @inbounds B[ix,iy,iz] = cumsum_tx    # Write the x-dimension chunk of results to main memory.
        cumsum_ix = shfl_sync(CUDA.FULL_MASK, cumsum_tx, blockDim().x)
    end
    return nothing
end
```

~ theoretical peak performance
(effective memory throughput)



Julia for HPC: high-level ParallelStencil kernel (architecture-agnostic)

math-close notation and in an architecture-agnostic fashion, using a single equation with second order derivatives:

```
@parallel function diffusion2D_step!(T2, T, Ci, lam, dt, dx, dy)
    @inn(T2) = @inn(T) + dt*(lam*@inn(Ci)*(@d2_xi(T)/dx^2 + @d2_yi(T)/dy^2));
    return
end
```

or using multiple equations with first order derivatives (note that `qx`, `qy` and `dTdt` will be computed on the fly...):

```
@parallel function diffusion3D_step!(T2, T, Ci, lam, dt, dx, dy)
    @all(qx) = -lam*@d_xi(T)/dx # Fourier's law of heat conduction
    @all(qy) = -lam*@d_yi(T)/dy # ...
    @all(dTdt) = @inn(Ci)*(-@d_xa(qx)/dx - @d_ya(qy)/dy) # Conservation of energy
    @inn(T2) = @inn(T) + dt*@all(dTdt) # Update of temperature
    return
end
```

Then the kernel can be called simply as:

```
@parallel diffusion2D_step!(T2, T, Ci, lam, dt, dx, dy)
```

code generation & optimization
=>
near theoretical peak performance
(effective memory throughput)

Setting up Julia in JupyterHub

Start a Julia-enabled server

Queue: Dedicated | Account (optional): | Job Duration: 12 hours

Advanced options

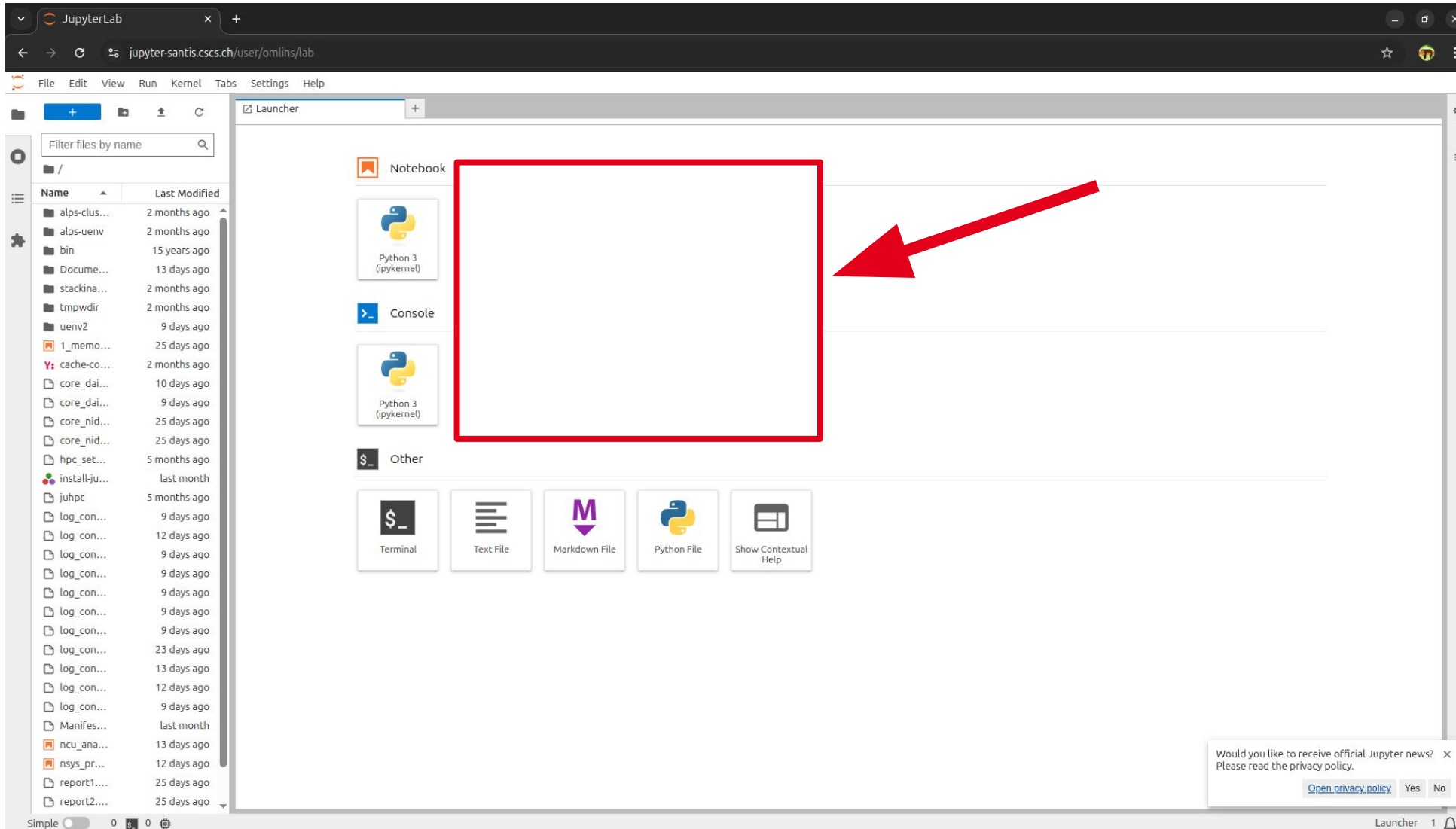
Reservation (optional):

Uenv (leave empty for `imgenv-gnu/24.11`): `julia/25.5:v1`

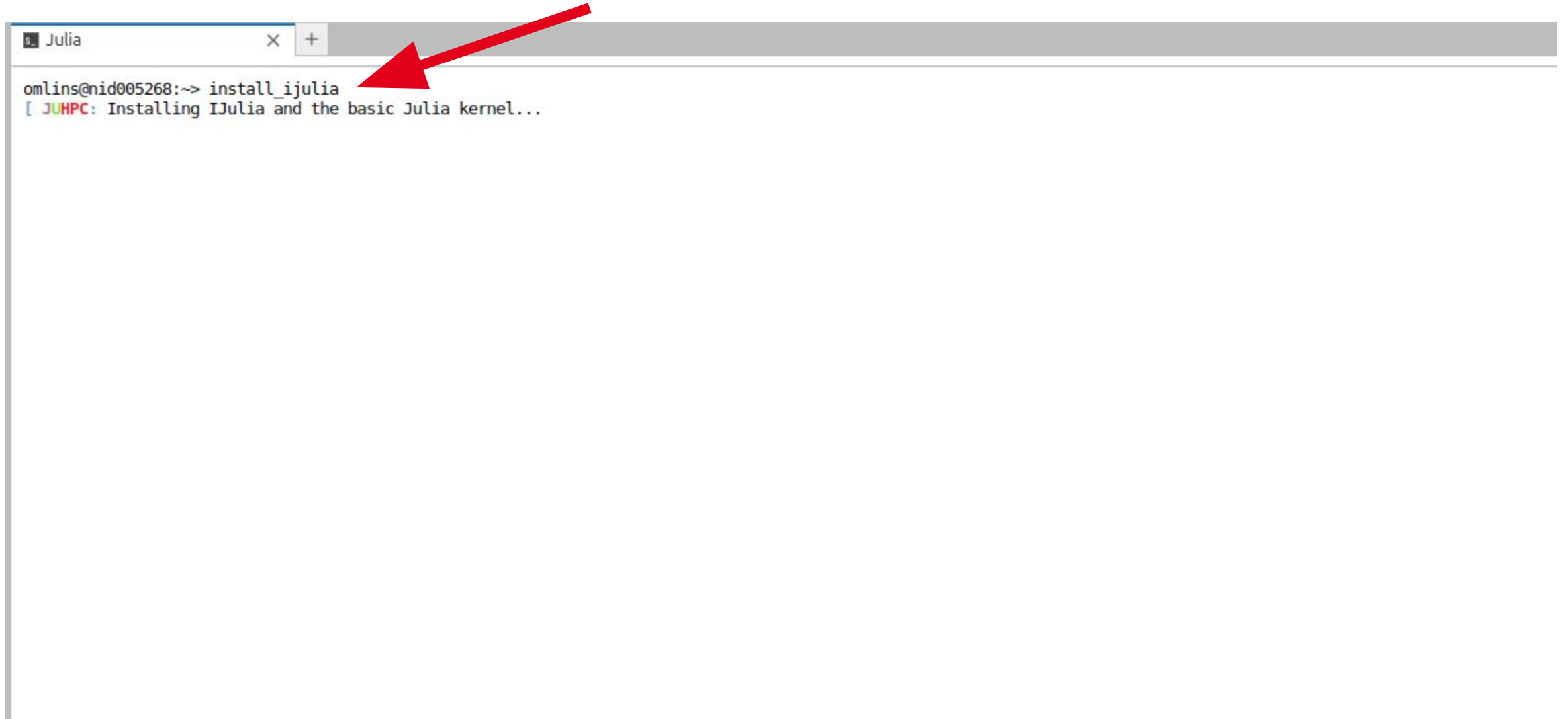
Uenv view (leave empty for default): `jupyter`

Launch JupyterLab

First time IJulia & kernel installation



First time IJulia & kernel installation



A terminal window titled "Julia" with a red arrow pointing to the command prompt. The terminal shows the command `install_ijulia` and the output `[JUHPC: Installing IJulia and the basic Julia kernel...`

```
Julia  
omlins@nid005268:~> install_ijulia  
[ JUHPC: Installing IJulia and the basic Julia kernel...
```

First time IJulia & kernel installation

```
Julia
[a63ad114] + Mmap v1.11.0
[ca575930] + NetworkOptions v1.2.0
[44cfe95a] + Pkg v1.11.0
[de0858da] + Printf v1.11.0
[3fa0cd96] + REPL v1.11.0
[9a3f8284] + Random v1.11.0
[ea8e919c] + SHA v0.7.0
[6462fe0b] + Sockets v1.11.0
[f489334b] + StyledStrings v1.11.0
[fa267f1f] + TOML v1.0.3
[a4e569a6] + Tar v1.10.0
[cf7118a7] + UUIDs v1.11.0
[4ec0a83e] + Unicode v1.11.0
[deac9b47] + LibCURL_jll v8.6.0+0
[e37daf67] + LibGit2_jll v1.7.2+0
[29816b5a] + LibSSH2_jll v1.11.0+1
[c8ffd9c3] + MbedTLS_jll v2.28.6+0
[14a3606d] + MozillaCACerts_jll v2023.12.12
[83775a58] + Zlib_jll v1.2.13+1
[8e850ede] + nhttp2_jll v1.59.0+0
[3f19e933] + p7zip_jll v17.4.0+2
Info Packages marked with x have new versions available but compatibility constraints restrict them from upgrading. To see why use `status --outdated -m`
Building Conda → `/capstor/scratch/cscs/omlins/.julia/gh200/juliaup/depot/scratchspaces/44cfe95a-1eb2-52ea-b672-e2afdf69b78f/b19db3927f0db4151cb86d073689f2428e5f`
Building IJulia → `/capstor/scratch/cscs/omlins/.julia/gh200/juliaup/depot/scratchspaces/44cfe95a-1eb2-52ea-b672-e2afdf69b78f/be30be76e25b0aff2c9a85930ed3ac34c5f1`
Precompiling project...
20 dependencies successfully precompiled in 35 seconds. 21 already precompiled.
[ Info: Installing Julia kernelspec in /users/omlins/.local/share/jupyter/kernels/julia-1.11
[ JUHPC: ... done: installation completed.
[ JUHPC: Installing a kernel for Julia under Nsight Systems...
[ Info: Installing Julia under Nsight Systems kernelspec in /users/omlins/.local/share/jupyter/kernels/julia-under-nsight-systems-1.11
[ JUHPC: ... done: installation completed.
omlins@nid005268:~>
```

First time IJulia & kernel installation: if Julia missing...

```
Julia x +
omlins@nid005278:~> ls /capstor/scratch/cscs/omlins/.julia
ls: cannot access '/capstor/scratch/cscs/omlins/.julia': No such file or directory
omlins@nid005278:~> install_ijulia
[ JUHPC: Installing IJulia and the basic Julia kernel...
[ JUHPC: Julia is not yet installed, calling juliaup to install it...

J UHPC

[ JUHPC: This is the first call to juliaup: installing juliaup and the latest julia...

info: downloading installer
Welcome to Julia!

This will download and install the official Julia Language distribution
and its version manager Juliaup.

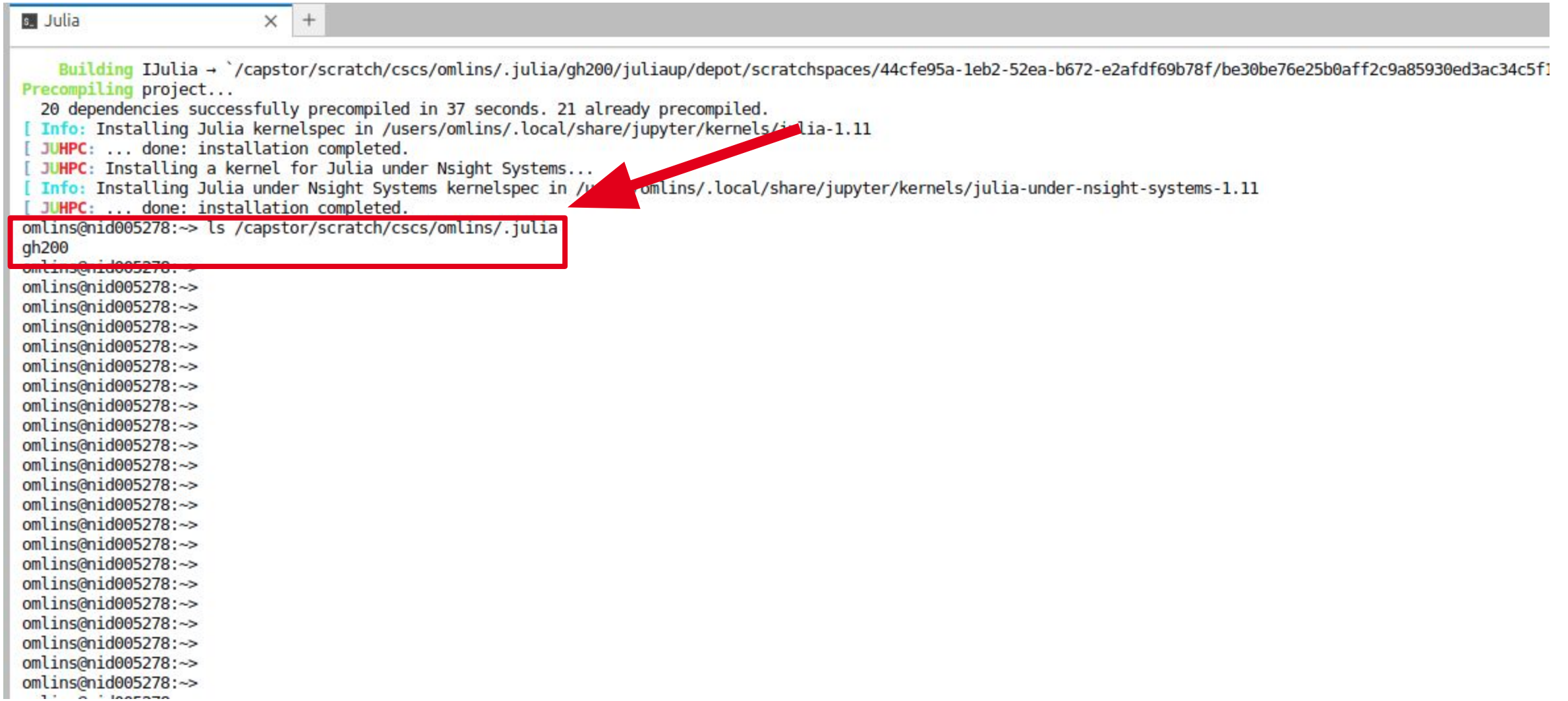
Juliaup will be installed into the Juliaup home directory, located at:

/capstor/scratch/cscs/omlins/.julia/gh200/juliaup

The julia, juliaup and other commands will be added to
Juliaup's bin directory, located at:
```



First time IJulia & kernel installation: if Julia missing...



```
Julia x +
Building IJulia → `~/capstor/scratch/cscs/omlins/.julia/gh200/juliaup/depot/scratchspaces/44cfe95a-1eb2-52ea-b672-e2afdf69b78f/be30be76e25b0aff2c9a85930ed3ac34c5f1
Precompiling project...
20 dependencies successfully precompiled in 37 seconds. 21 already precompiled.
[ Info: Installing Julia kernelspec in /users/omlins/.local/share/jupyter/kernels/julia-1.11
[ JUHPC: ... done: installation completed.
[ JUHPC: Installing a kernel for Julia under Nsight Systems...
[ Info: Installing Julia under Nsight Systems kernelspec in /users/omlins/.local/share/jupyter/kernels/julia-under-nsight-systems-1.11
[ JUHPC: ... done: installation completed.
omlins@nid005278:~> ls /capstor/scratch/cscs/omlins/.julia
gh200
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
omlins@nid005278:~>
```

A red arrow points from the text "Installing a kernel for Julia under Nsight Systems..." to the terminal prompt line "omlins@nid005278:~> ls /capstor/scratch/cscs/omlins/.julia gh200". The prompt and the first two lines of the command output are enclosed in a red box.

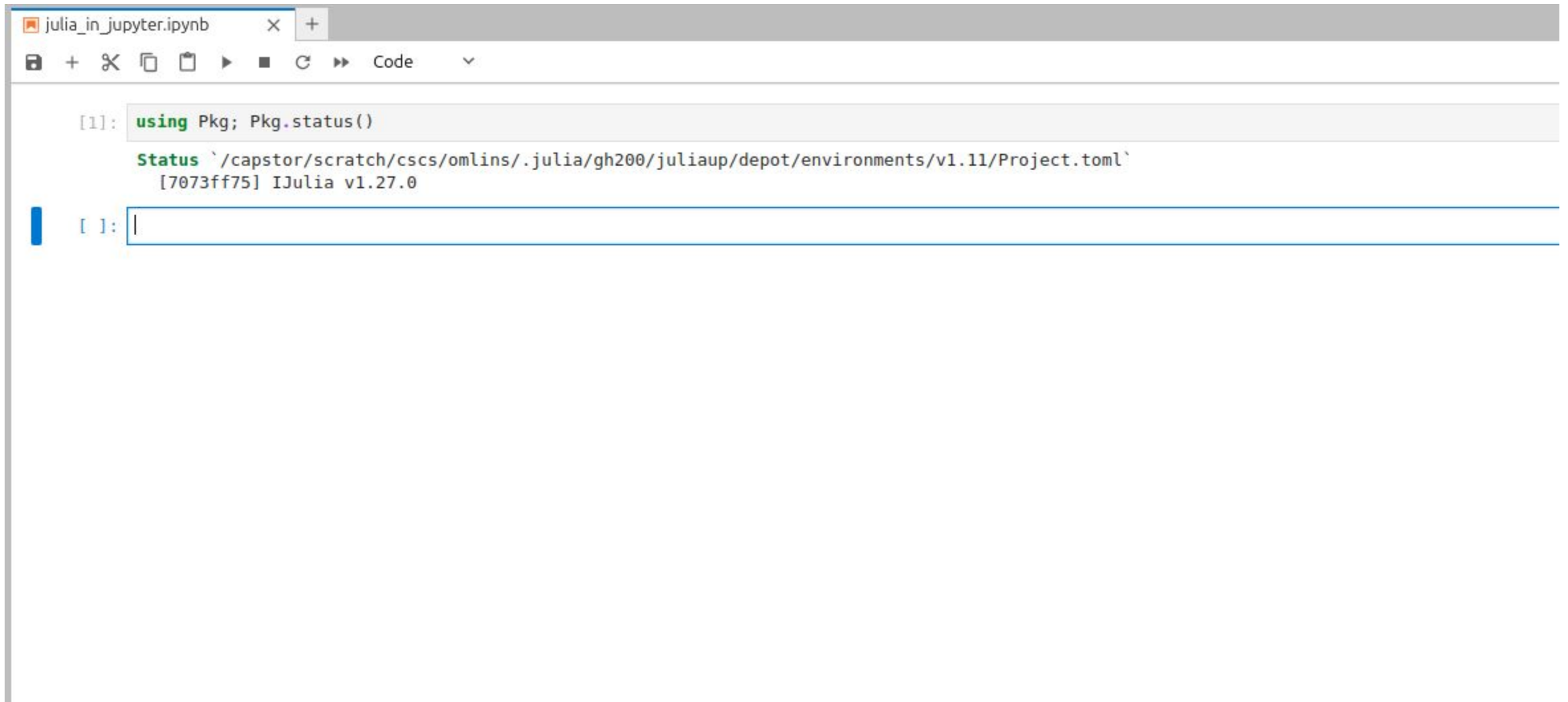
First time IJulia & kernel installation: result



First time IJulia & kernel installation: result

The screenshot displays the JupyterLab interface. On the left, a file browser shows a list of files and folders, with 'julia_in_ju...' selected. The main area is a code editor for 'julia_in_jupyter.ipynb' containing the Julia code: `using Pkg; Pkg.status()`. A 'Select Kernel' dialog box is centered, asking to select a kernel for the file. The dropdown menu shows 'Julia 1.11.5'. There are 'Cancel' and 'Select' buttons, and a checkbox for 'Always start the preferred kernel'. A small notification in the bottom right asks if the user wants to receive official Jupyter news.

First time IJulia & kernel installation: result



The screenshot shows a Jupyter notebook interface with a single code cell. The code cell contains the following Julia code:

```
[1]: using Pkg; Pkg.status()
```

The output of the code cell is:

```
Status `~/capstor/scratch/cscs/omlins/.julia/gh200/juliaup/depot/environments/v1.11/Project.toml`  
[7073ff75] IJulia v1.27.0
```

Below the code cell, there is an empty code cell with a cursor at the beginning of the line.

Package installation: predefined preferences and ENV

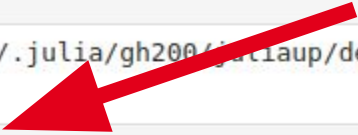
```
julia_in_jupyter.ipynb x +
+ ✂ 📄 ▶ ■ ↻ ▶▶ Code v

[1]: using Pkg; Pkg.status()

Status `~/capstor/scratch/cscs/omlins/.julia/gh200/juliaup/depot/environments/v1.11/Project.toml`
 [7073fff75] IJulia v1.27.0

[2]: Pkg.add(["CUDA", "ADIOS2", "HDF5"]);

Updating registry at `~/capstor/scratch/cscs/omlins/.julia/gh200/juliaup/depot/registries/General.toml`
Resolving package versions...
Installed libaec_jll _____ v1.1.3+0
Installed GPUArraysCore _____ v0.2.0
Installed MPICH_jll _____ v4.3.0+1
Installed Crayons _____ v4.1.1
Installed Adapt _____ v4.3.0
Installed ColorTypes _____ v0.12.1
Installed Scratch _____ v1.2.1
Installed PrettyTables _____ v2.4.0
Installed Zstd_jll _____ v1.5.7+1
Installed demumble_jll _____ v1.3.0+0
Installed TableTraits _____ v1.0.1
Installed CUDA_Driver_jll _____ v0.13.0+0
Installed nlohmann_json_jll _____ v3.11.3+0
Installed GPUCompiler _____ v1.5.1
Installed MPI _____ v0.20.22
Installed OpenSSL_jll _____ v3.5.0+0
Installed Hwloc_jll _____ v2.12.1+0
Installed LLVMLoopInfo _____ v1.0.0
```



For MPI.jl, CUDA.jl, HDF5.jl, and ADIOS2.jl, the julia uenv provides the libraries and presets package preferences and environment variables for automatic optimal installation and usage

Package installation: predefined preferences and ENV

The screenshot shows a JupyterLab interface with a file browser on the left and a code editor on the right. The code editor displays the output of a Julia session. The output is as follows:

```
[3]: using CUDA; CUDA.versioninfo()

CUDA runtime 12.8, local installation
CUDA driver 12.9
NVIDIA driver 550.54.15

CUDA libraries:
- CUBLAS: 12.8.3
- CURAND: 10.3.9
- CUFFT: 11.3.3
- CUSOLVER: 11.7.2
- CUSPARSE: 12.5.7
- CUPTI: 2025.1.0 (API 26.0.0)
- NVML: 12.0.0+550.54.15

Julia packages:
- CUDA: 5.8.1
- CUDA_Driver_jll: 0.13.0+0
- CUDA_Runtime_jll: 0.17.0+0
- CUDA_Runtime_Discovery: 0.3.5

Toolchain:
- Julia: 1.11.5
- LLVM: 16.0.6

Environment:
- JULIA_CUDA_MEMORY_POOL: none

Preferences:
- CUDA_Runtime_jll.version: 12.8
- CUDA_Runtime_jll.local: true

4 devices:
0: NVIDIA GH200 120GB (sm_90, 93.667 GiB / 95.577 GiB available)
1: NVIDIA GH200 120GB (sm_90, 93.663 GiB / 95.573 GiB available)
2: NVIDIA GH200 120GB (sm_90, 93.670 GiB / 95.580 GiB available)
3: NVIDIA GH200 120GB (sm_90, 93.668 GiB / 95.578 GiB available)

[6]: using ADIOS2; @show ADIOS2.libadios2_c;
ADIOS2.libadios2_c = "/user-environment/linux...neoverse_v2/gcc-13.3.0/adios2-2.10.2-uoe2ctr7j34tm7oed7sc4b6qr7g6a2ei/lib64/libadios2_c"

[7]: using HDF5; @show HDF5.has_parallel();
HDF5.has_parallel() = true

[ ]:
```

Red arrows point to the following lines in the output:

- The first line of the output: `[3]: using CUDA; CUDA.versioninfo()`
- The line `ADIOS2.libadios2_c = "/user-environment/linux...neoverse_v2/gcc-13.3.0/adios2-2.10.2-uoe2ctr7j34tm7oed7sc4b6qr7g6a2ei/lib64/libadios2_c"`
- The line `HDF5.has_parallel() = true`

A notification box at the bottom right asks: "Would you like to receive official Jupyter news? Please read the privacy policy." with buttons for "Open privacy policy", "Yes", and "No".



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

GPU profiling using Julia

CUDA profiling - using CUDA.jl profiler

```
julia_in_jupyter.ipynb x CUDA_nsys_profiling.ipynb x CUDA_integrated_profiling.iX +
+ 🔍 📄 ▶ ⏪ ⏩ Code ▾ Julia 1.11.5 ○
```

```
[1]: using CUDA
```

```
[2]: CuArray{Int64}([1]) .+ 1;
```

```
[3]: CUDA.@profile CuArray{Int64}([1]) .+ 1
```

```
[3]: Profiler ran for 227.93 μs, capturing 71 events.
```

Host-side activity: calling CUDA APIs took 106.81 μs (46.86% of the trace)

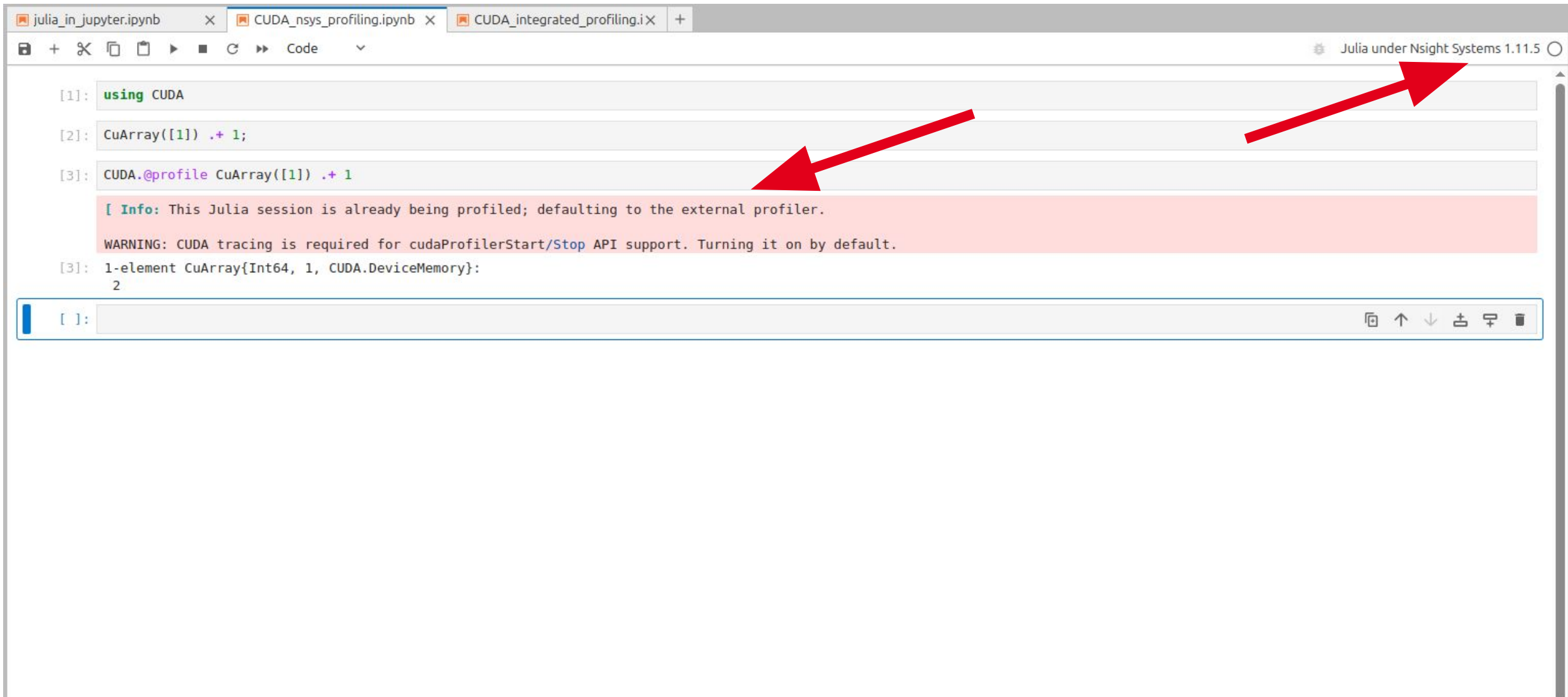
Time (%)	Total time	Calls	Time distribution	Name
36.51%	83.21 μs	1		cuLaunchKernel
4.50%	10.25 μs	2	5.13 μs ± 3.2 (2.86 .. 7.39)	cuMemAlloc
1.78%	4.05 μs	1		cuMemcpyHtoDAsync

Device-side activity: GPU was busy for 2.38 μs (1.05% of the trace)

Time (%)	Total time	Calls	Name
0.73%	1.67 μs	1	gpu_broadcast_kernel_linear(CompilerMetadata{DynamicSize, DynamicCheck, void, CartesianIndices{1, Tuple{OneTo{Int64}}}}, NDRange{1, DynamicSize, DynamicSize, CartesianIndices{1, Tuple{OneTo{Int64}}}}, CartesianIndices{1, Tuple{OneTo{Int64}}}, CuDeviceArray{Int64, 1, 1}, Broadcasted{CuArrayStyle{1, DeviceMemory}, Tuple{OneTo{Int64}}, Tuple{Extruded{CuDeviceArray{Int64, 1, 1}, Tuple{Bool}}, Tuple{Int64}}, Int64})
0.31%	715.26 ns	1	[copy pageable to device memory]



CUDA profiling - using Nvidia Nsight System

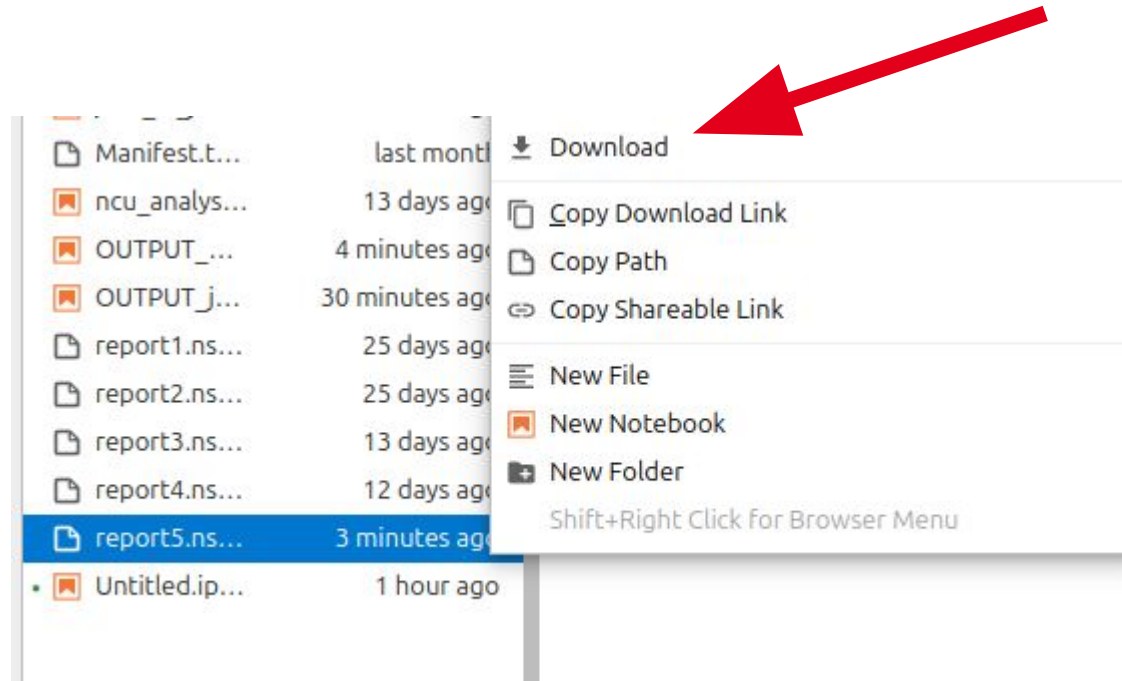


```
[1]: using CUDA
[2]: CuArray{Int64, 1, CUDA.DeviceMemory} .+ 1;
[3]: CUDA.@profile CuArray{Int64, 1, CUDA.DeviceMemory} .+ 1

[ Info: This Julia session is already being profiled; defaulting to the external profiler.
WARNING: CUDA tracing is required for cudaProfilerStart/Stop API support. Turning it on by default.
[3]: 1-element CuArray{Int64, 1, CUDA.DeviceMemory}:
      2
```

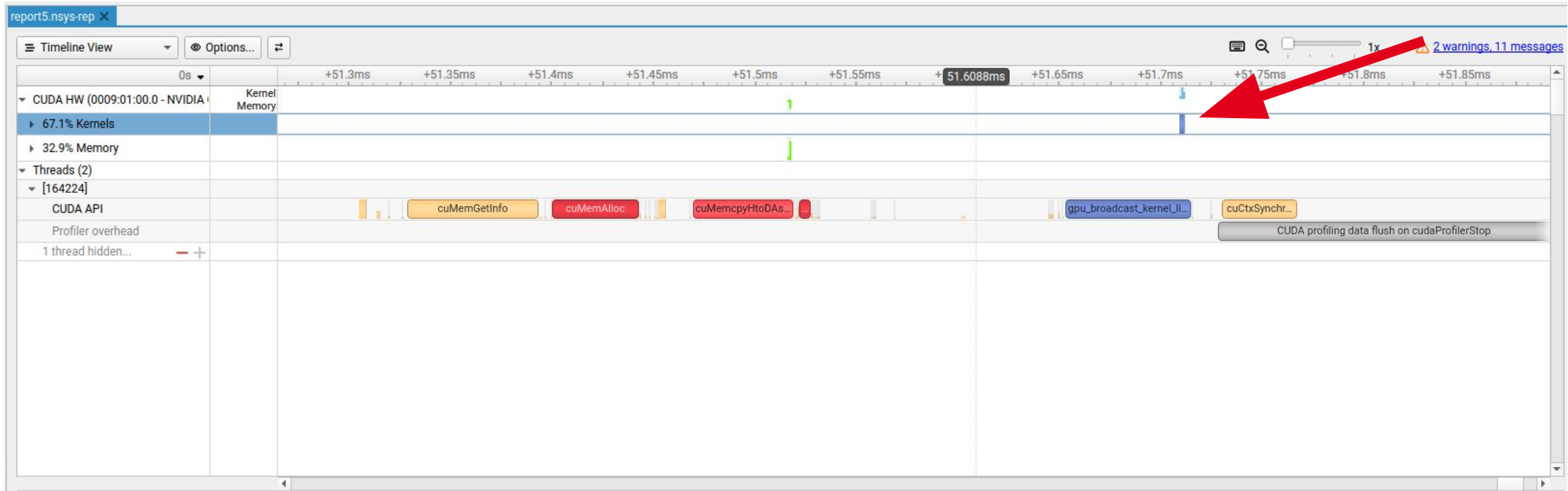
[]: 🏠 ↑ ↓ 🔍 🗑️

CUDA profiling - using Nvidia Nsight System

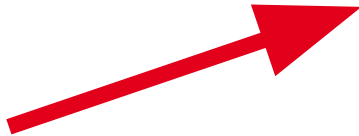
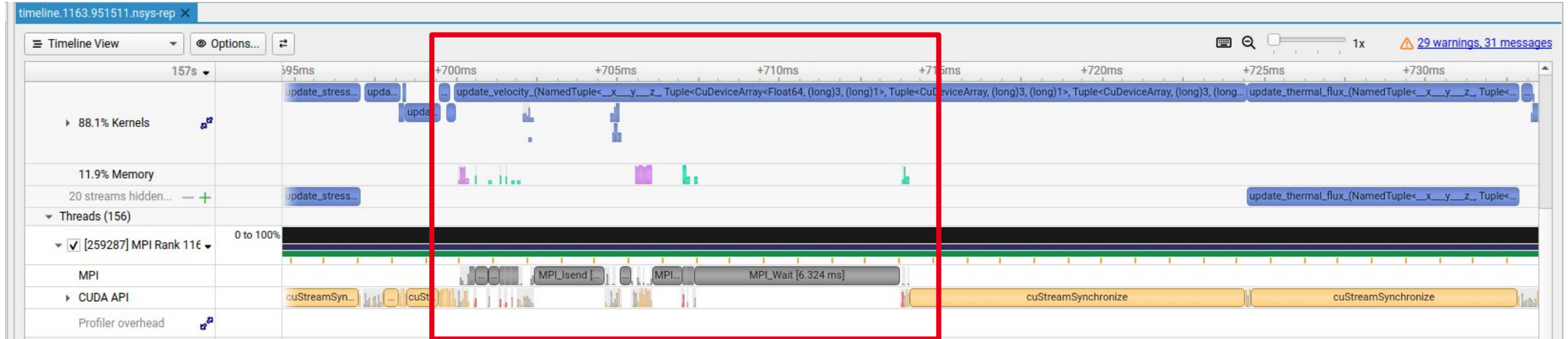


```
omlins@Omlin-Lenovo:~/tmpwdir$ nsys-ui ~/Downloads/report5.nsys-rep
```

CUDA profiling - using Nvidia Nsight System



CUDA profiling - using Nvidia Nsight System (real-world example)



Verification that communication is overlapped with computation (MPI not available in Jupyter)



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Documentation

Documentation: docs.cscs.ch/

The screenshot shows a web browser window displaying the documentation for JupyterLab. The address bar shows the URL `eth-cscs.github.io/cscs-docs/services/jupyterlab/#using-julia-in-jupyterhub`. The page header includes the JupyterLab logo, a search bar, and GitHub repository information (11 stars, 21 forks). The main content area is titled "Using Julia in JupyterHub" and contains the following text:

Each time you start a JupyterHub server, you need to do the following in the JupyterHub Spawner Options form mentioned above:

```
pass a julia uenv and the view jupyter.
```

At first time use of Julia within Jupyter, IJulia and one or more Julia kernel needs to be installed. Type the following command in a shell within JupyterHub to install IJulia, the default Julia kernel and, on systems with Nvidia GPUs, a Julia kernel running under Nvidia Nsight Systems:

```
install_ijulia
```

You can install additional custom Julia kernels by typing the following in a shell:

```
julia
using IJulia
installkernel(<args>) # type `? installkernel` to learn about valid `<args>`
```

⚠ First time use of Julia

If you are using Julia for the first time at all, executing `install_ijulia` will automatically first trigger the installation of `juliaup` and the latest `julia` version (it is also triggered if you execute `juliaup` or `julia`).

The right sidebar contains a "Table of contents" with links to various sections: Access and setup, Debugging, Accessing file systems, Creating Jupyter kernels for Python, Using uenvs in JupyterLab for Python, **Using Julia in JupyterHub**, Ending your interactive session and logging out, MPI in the notebook via IPyParallel and MPI4Py, and Further documentation.

Documentation: docs.cscs.ch/

The screenshot shows a web browser displaying the CSCS Documentation page for Julia HPC setup. The browser address bar shows the URL `eth-cscs.github.io/cscs-docs/software/prgenv/julia/#julia`. The page header includes the CSCS logo and name, a search bar, and a GitHub repository link with 11 stars and 21 forks. The main content area features a dark theme with a sidebar on the left containing a navigation menu. The central content area displays the Julia logo and a section titled "Versioning".

Navigation Menu (Left Sidebar):

- CSCS Documentation
- Welcome
- Alps >
- Accounts and Projects >
- Connecting to Alps >
- Software >
 - Scientific Applications >
 - Programming Environments >
 - prgenv-gnu
 - prgenv-nvfortran
 - linalg
 - julia**
 - Cray modules (CPE)
- Machine Learning >
- Communication Libraries >
- Building and Installing software >
- Debugging and Performance Analysis >
- uenv

Main Content Area:

julia

The `julia` uenv provides a complete HPC setup for running Julia efficiently at scale, using the supercomputer hardware optimally. Unlike in traditional approaches, this Julia HPC setup enables you to update Julia yourself using the included preconfigured community tool `juliaup`. It also does not preinstall any packages site-wide. Instead, for HPC key packages that benefit from using locally built libraries (`MPI.jl`, `CUDA.jl`, `AMDGPU.jl`, `HDF5.jl`, `ADIOS2.jl`, etc.), this uenv provides the libraries and presets package preferences and environment variables for an automatic optimal installation and usage of these packages using these local libraries. As a result, you only need to type, e.g., `] add CUDA` in the Julia REPL, in order to install `CUDA.jl` optimally. The `julia` uenv internally relies on the community scripting project `JUHPC` to achieve this.

Versioning

The naming scheme is `julia/<version>`, where `<version>` has the `YY.M[M]` format, for example September 2024 is `24.9`, and May 2025 would be `25.5`. The release schedule is not fixed; new versions will be released, when there is a compelling reason to update.

Table of Contents (Right Sidebar):

- Table of contents
- Versioning
- How to use
- Background on Julia for HPC
- References



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Conclusions

Conclusions

- Setting up Julia in JupyterHub is trivial
- Two ways for GPU profiling using Julia
- Julia is a great language for HPC



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



VS Code for Interactive Computing

Productivity and Development Tips

**Prashanth Kanduri
& Lukas Drescher**

Why use an IDE?

- Intuitive/familiar user experience
- Ease of managing files, visual previews
- Syntax highlighting
- Handy extensions for languages, frameworks and debugging
- Development aids like codebase awareness, code completion, etc
- Easy to setup!



```
1 import argparse
2 import pathlib
3
4 from .orphans import report_orphan_attachments, report_notes_not_in_structure
5 from .link_graph import build_link_graph
6
7
8 def main():
9     options = parse_args()
10
11     vault = options.vault
12
13     link_graph = build_link_graph(vault)
14     report_orphan_attachments(vault, link_graph)
15     report_notes_not_in_structure(vault, link_graph)
16
17
18 def parse_args() -> argparse.Namespace:
19     parser = argparse.ArgumentParser()
20     parser.add_argument("vault", type=pathlib.Path)
21     parser.add_argument_group("options")
22     return parser.parse_args()
23
24 if __name__ == "__main__":
25     main()
```

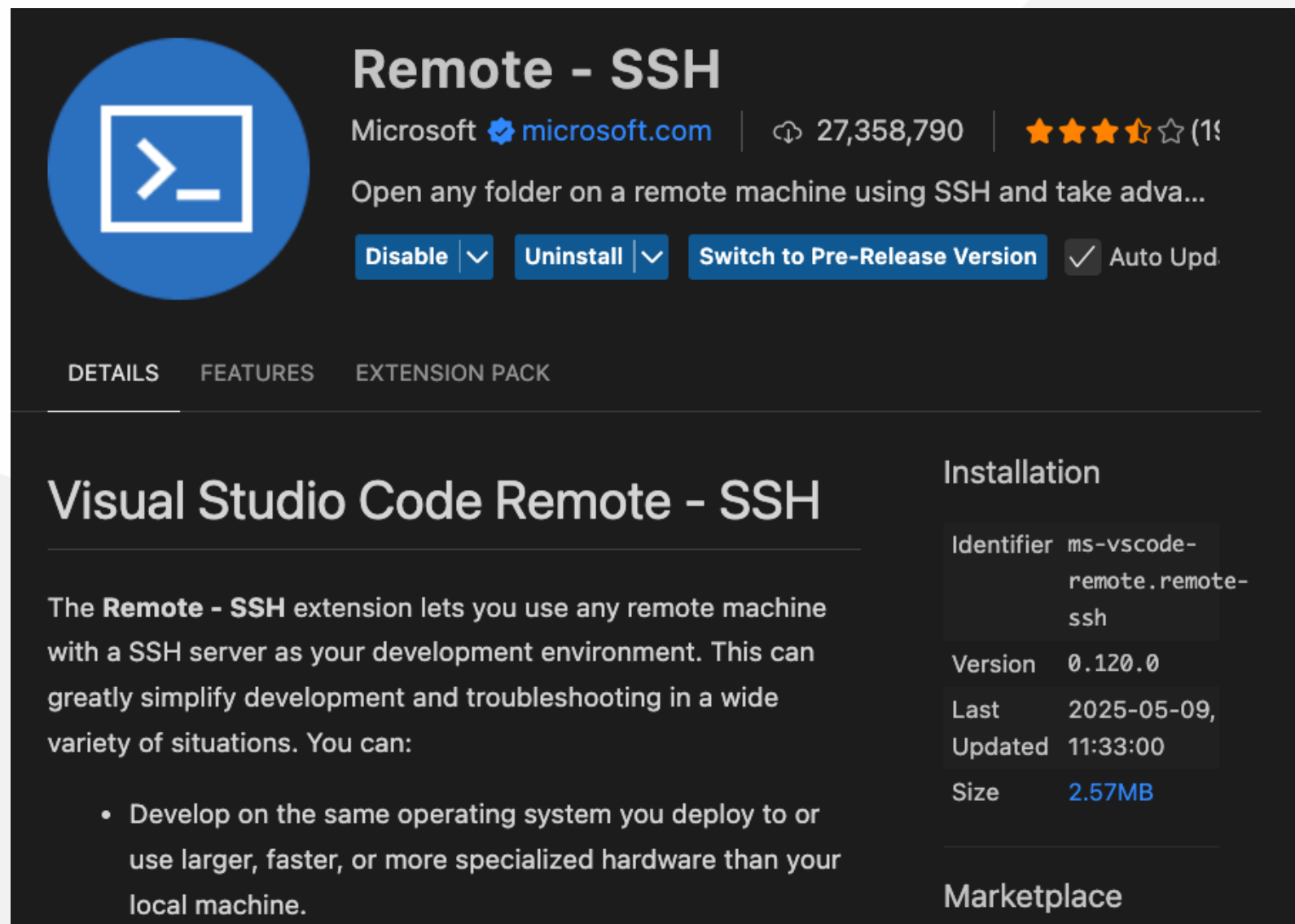
Basic Setup

- Ensure that SSH config and 2FA works and SSH keys are refreshed
- `ssh <clustertype>` on the local should get you to the shell of a cluster

SSH config	Password-Less Login
<pre>Host ela HostName ela.cscs.ch User <username> IdentitiesOnly yes IdentityFile ~/.ssh/cscs-key Host santis User <username> IdentitiesOnly yes IdentityFile ~/.ssh/cscs-key ProxyJump ela</pre>	<pre>→ ssh santis Last login: Mon May 26 18:47:17 2025 from ela6.cscs.ch ===== IMPORTANT NOTICE FOR USERS of Alps (Santis) Documentation: Alps User Guide - https://confluence.cscs.ch/x/mQ_KMg Request support: Service Desk at https://support.cscs.ch =====</pre>

VS Code Setup

Install the Remote-SSH extension by Microsoft from the marketplace.



Remote - SSH
Microsoft [microsoft.com](#) | 27,358,790 | ★★★★★ (1)

Open any folder on a remote machine using SSH and take adva...

[Disable](#) [Uninstall](#) [Switch to Pre-Release Version](#) Auto Upd.

DETAILS FEATURES EXTENSION PACK

Visual Studio Code Remote - SSH

The **Remote - SSH** extension lets you use any remote machine with a SSH server as your development environment. This can greatly simplify development and troubleshooting in a wide variety of situations. You can:

- Develop on the same operating system you deploy to or use larger, faster, or more specialized hardware than your local machine.

Installation

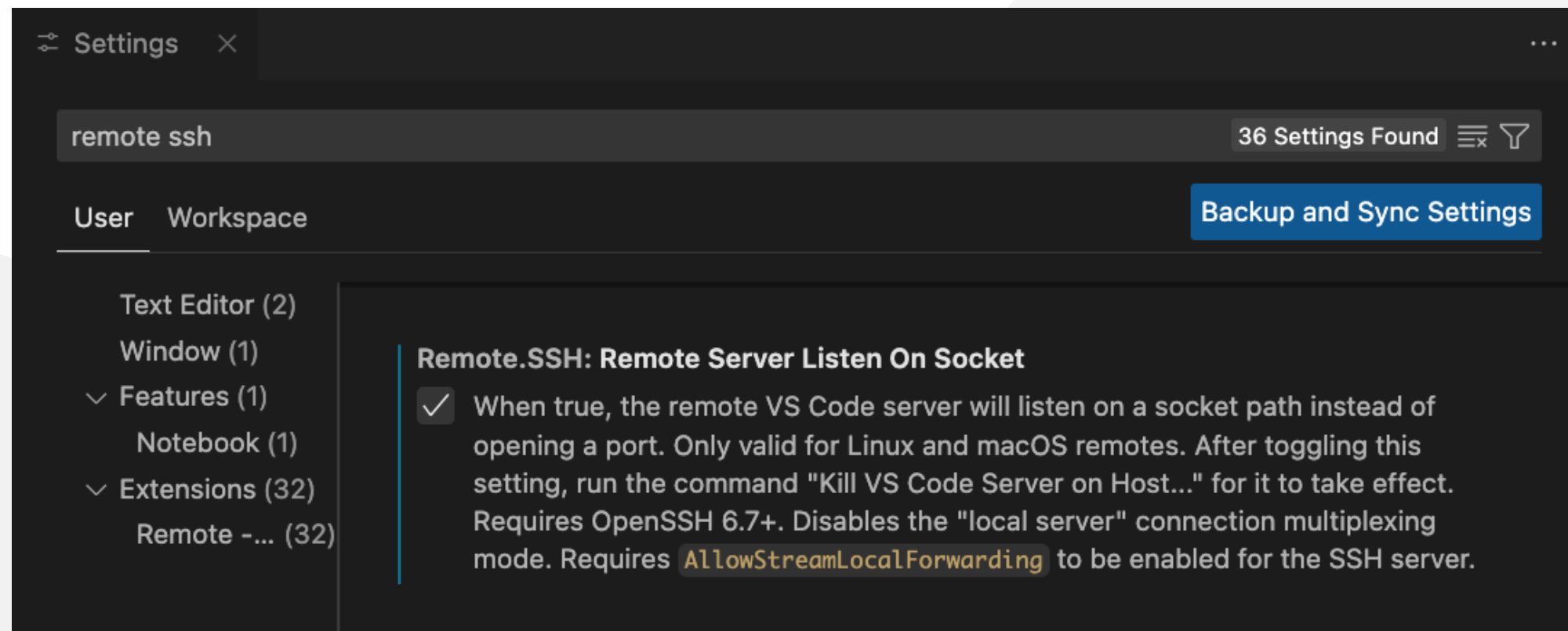
Identifier	ms-vscode-remote.remote-ssh
Version	0.120.0
Last Updated	2025-05-09, 11:33:00
Size	2.57MB

Marketplace

VS Code Setup cont...

After installation, in the settings, scroll down to the extensions and find the section concerning the one above.

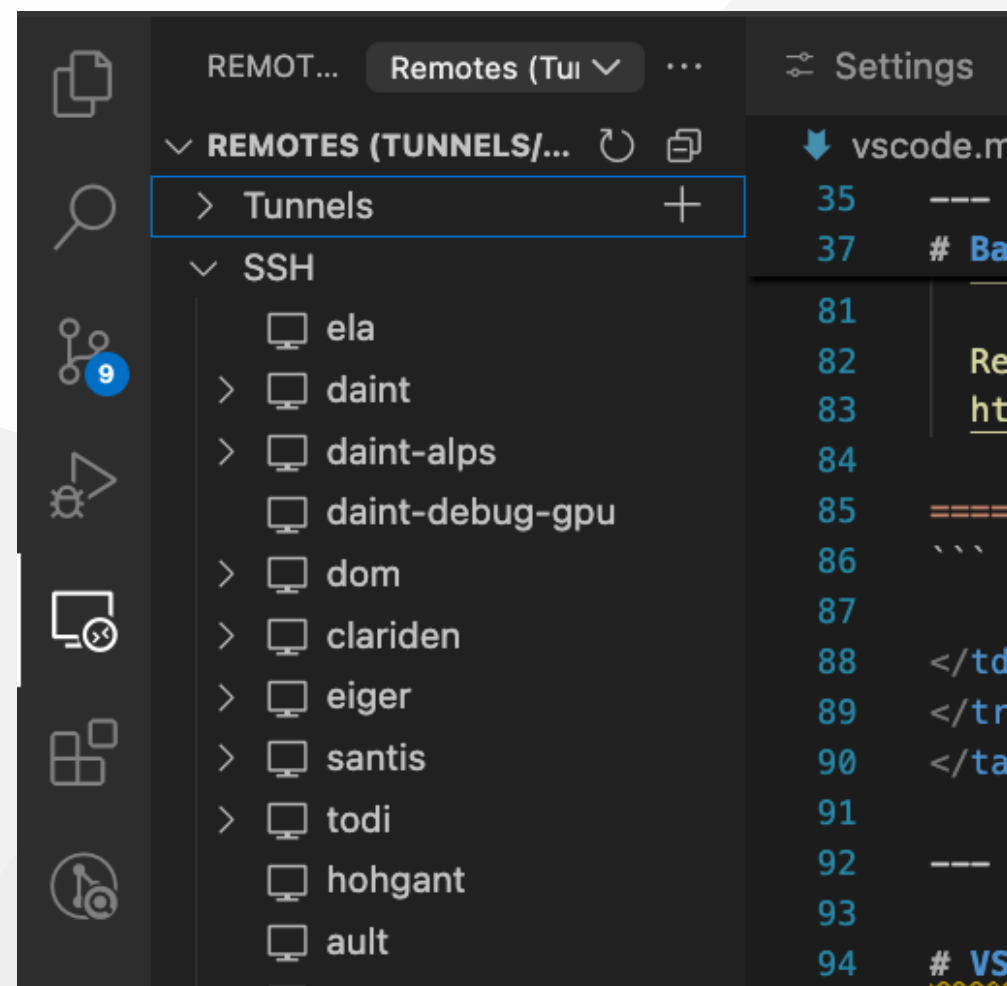
There activate the **Remote Server Listen On Socket** setting.



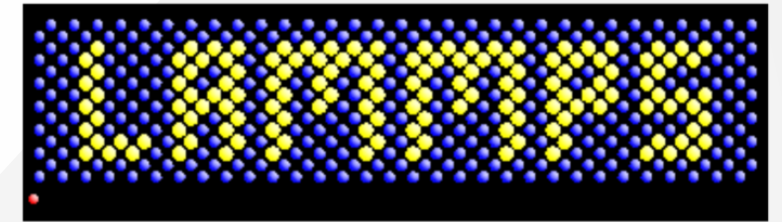
Connect to a Cluster

With these settings, restart VS Code and on the left-hand side, find the **Remote Explorer** with the hostnames setup in the `~/.ssh/config` file.

Select the cluster and the window will reload to login. One can select a folder to open there.

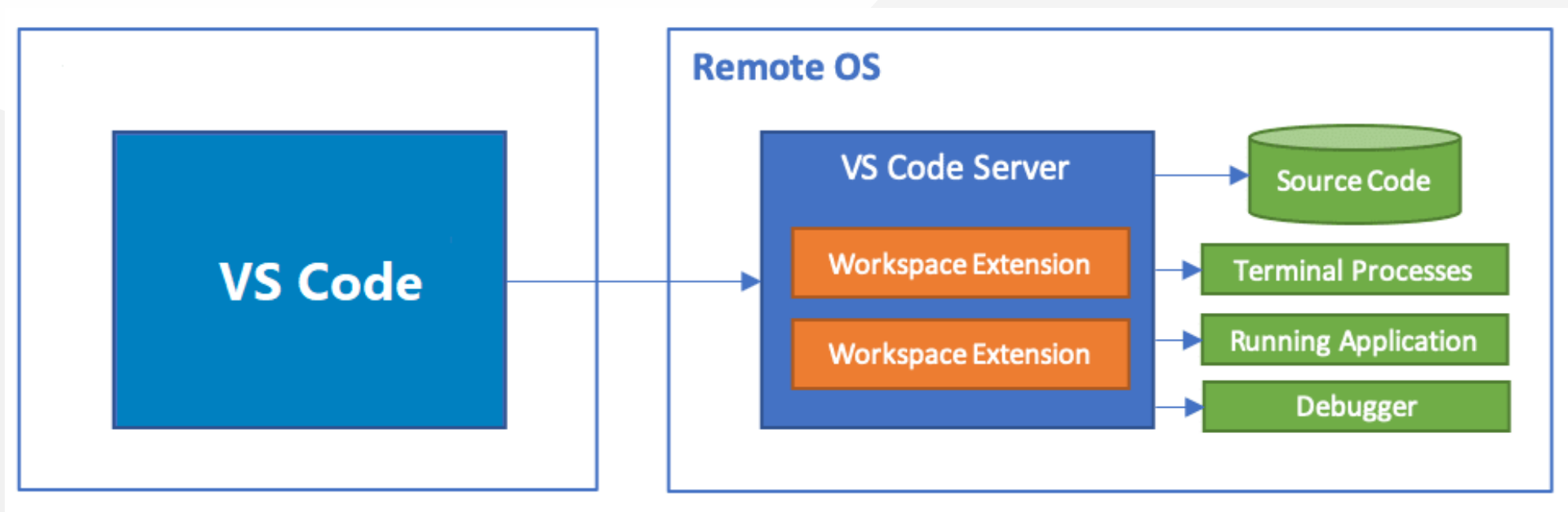


Loading Development Environments



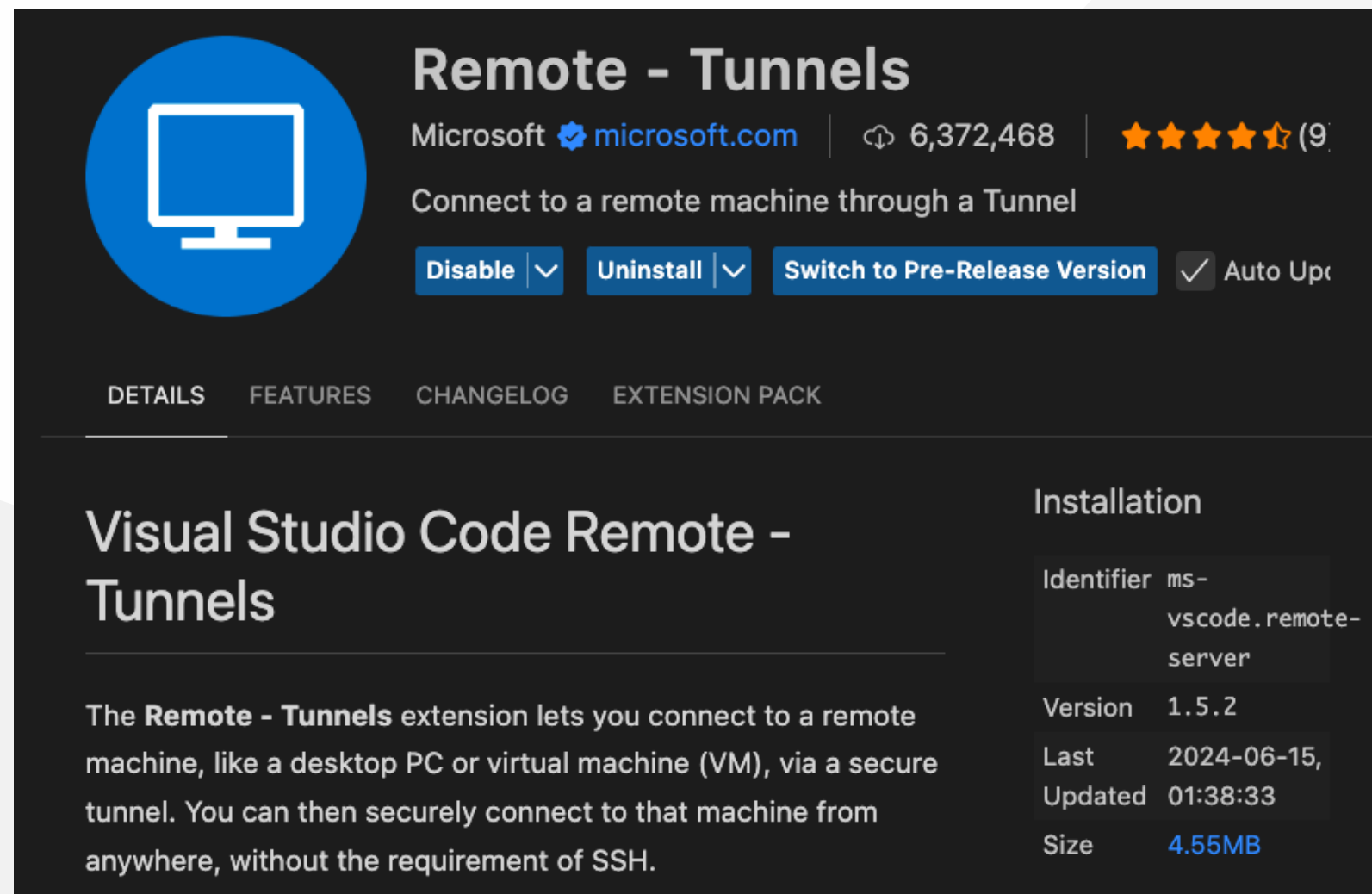
Working with Remote Tunnels

“ The Visual Studio Code **Remote - Tunnels** extension lets you connect to a remote machine, like a desktop PC or virtual machine (VM), via a secure tunnel. You can connect to that machine from a VS Code client anywhere, without the requirement of SSH. ”



Setting up Locally

Install the **Remote Tunnels** extension from Microsoft on VS Code.



The screenshot shows the Visual Studio Code extension marketplace page for the 'Remote - Tunnels' extension by Microsoft. The extension is currently installed, as indicated by the 'Uninstall' button. The page includes a description of the extension's functionality, a list of installation details, and navigation tabs for details, features, changelog, and extension pack.

Remote - Tunnels
Microsoft [microsoft.com](#) | 6,372,468 | ★★★★★ (9)

Connect to a remote machine through a Tunnel

[Disable](#) | [Uninstall](#) | [Switch to Pre-Release Version](#) | Auto Update

[DETAILS](#) | [FEATURES](#) | [CHANGELOG](#) | [EXTENSION PACK](#)

Visual Studio Code Remote - Tunnels

The **Remote - Tunnels** extension lets you connect to a remote machine, like a desktop PC or virtual machine (VM), via a secure tunnel. You can then securely connect to that machine from anywhere, without the requirement of SSH.

Installation	
Identifier	ms-vscode.remote-server
Version	1.5.2
Last Updated	2024-06-15, 01:38:33
Size	4.55MB

Setting up on Alps

Download the VS Code CLI tool `code`, which CSCS provides for easy download. There are two executables, one for using on systems with x86 or ARM CPUs respectively.

ARM64 → `daint`, `clariden`, `santis`

x86-64 → `eiger`, `bristen`

Download with the following commands (for ARM64) on the cluster:

```
wget https://jfrog.svc.cscs.ch/artifactory/uenv-sources/vscode/vscode_cli_alpine_arm64_cli.tar.gz
tar -xf vscode_cli_alpine_arm64_cli.tar.gz
```

We will now have an executable named `code` which we shall use. We can make it accessible from anywhere by adding its absolute path to the `PATH` environment variable in our `.bashrc` file.

```
export PATH=/location/of/code/executable:$PATH
```

Tunnel with UENV

On the shell of a cluster, start a uenv session and then a tunnel. For a `uenv` already loaded, one can use the following command.

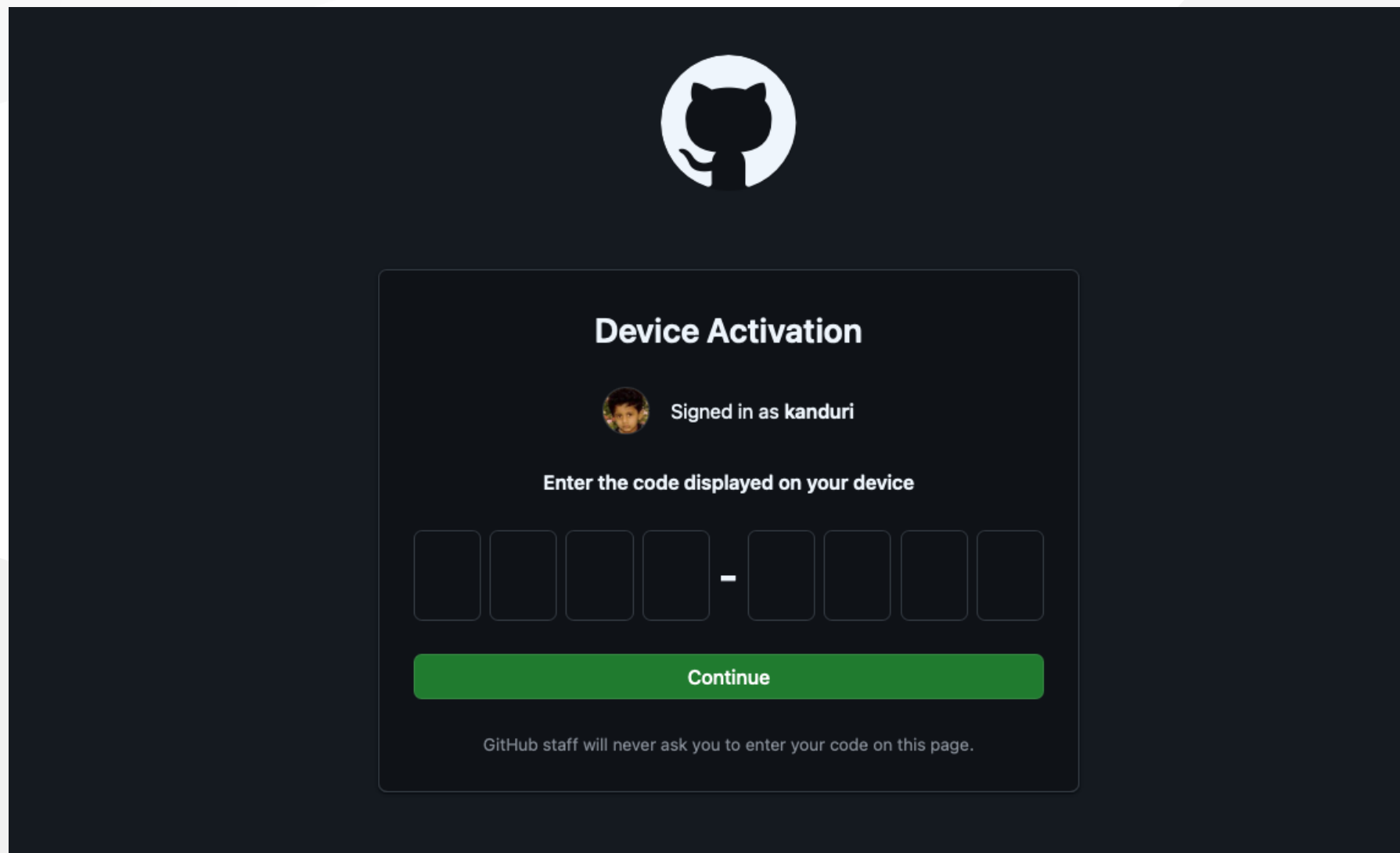
```
# start a uenv session on the login node
uenv start --view=default prgenv-gnu/24.11:v1
# then start the tunnel
code tunnel --name=$CLUSTER_NAME-tunnel
```

It will ask the user to sign in using Microsoft or GitHub account. We have reliably tested it with GitHub.

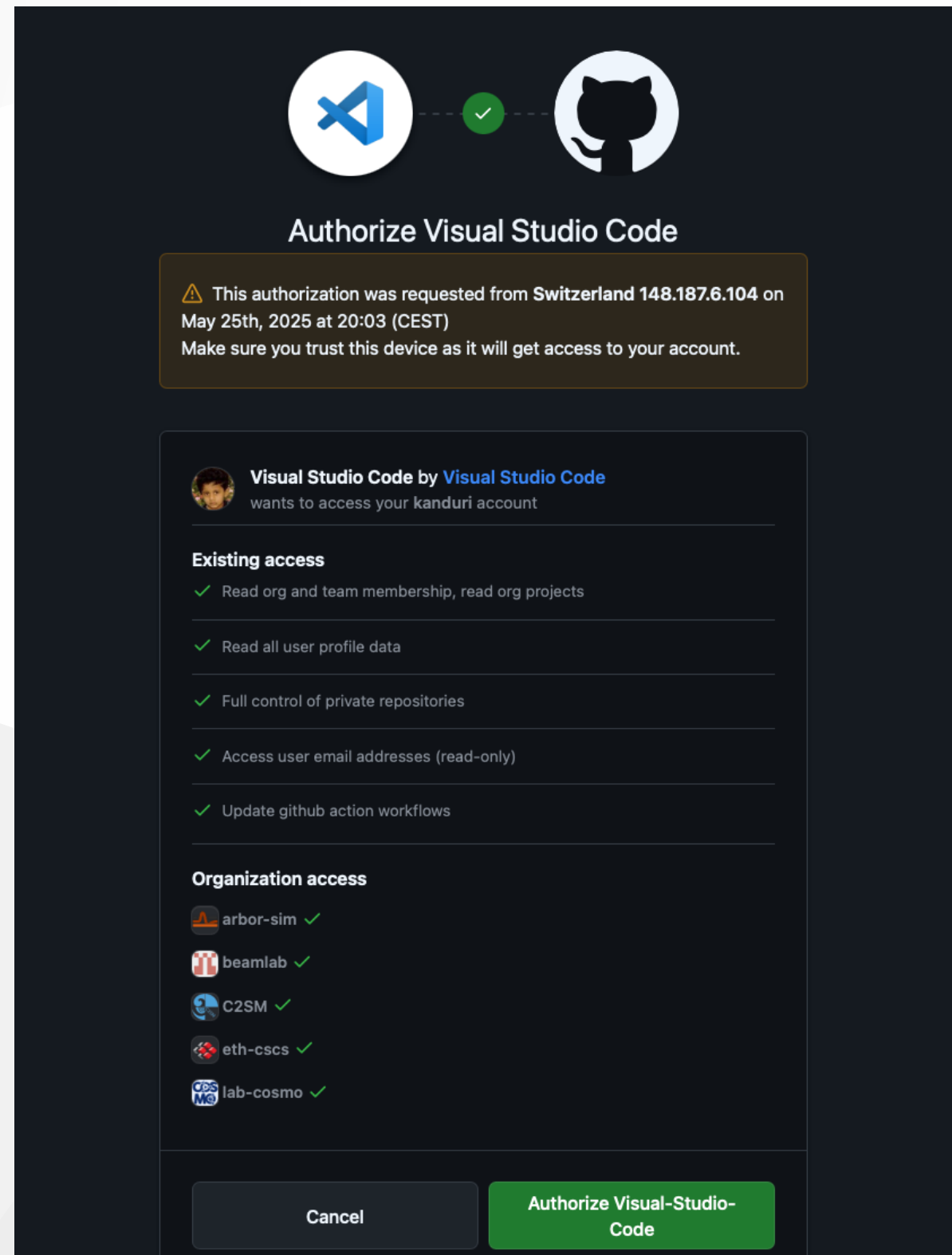
```
> code tunnel --name=$CLUSTER_NAME-tunnel
...
? How would you like to log in to Visual Studio Code? ›
  Microsoft Account
  > GitHub Account
```

This will begin the first time setup process. It will ask to open a URL on the browser.

First-Time Setup



First-Time Setup cont...



The screenshot shows a dark-themed GitHub authorization dialog. At the top, there are icons for Visual Studio Code and GitHub, connected by a dashed line with a green checkmark in the middle. Below this is the title "Authorize Visual Studio Code". A warning box contains the text: "This authorization was requested from Switzerland 148.187.6.104 on May 25th, 2025 at 20:03 (CEST). Make sure you trust this device as it will get access to your account." The main section shows the user profile for "Visual Studio Code by Visual Studio Code" and lists the permissions it wants to access. These are divided into "Existing access" and "Organization access".

Authorize Visual Studio Code

⚠ This authorization was requested from Switzerland 148.187.6.104 on May 25th, 2025 at 20:03 (CEST)
Make sure you trust this device as it will get access to your account.

Visual Studio Code by Visual Studio Code
wants to access your kanduri account

Existing access

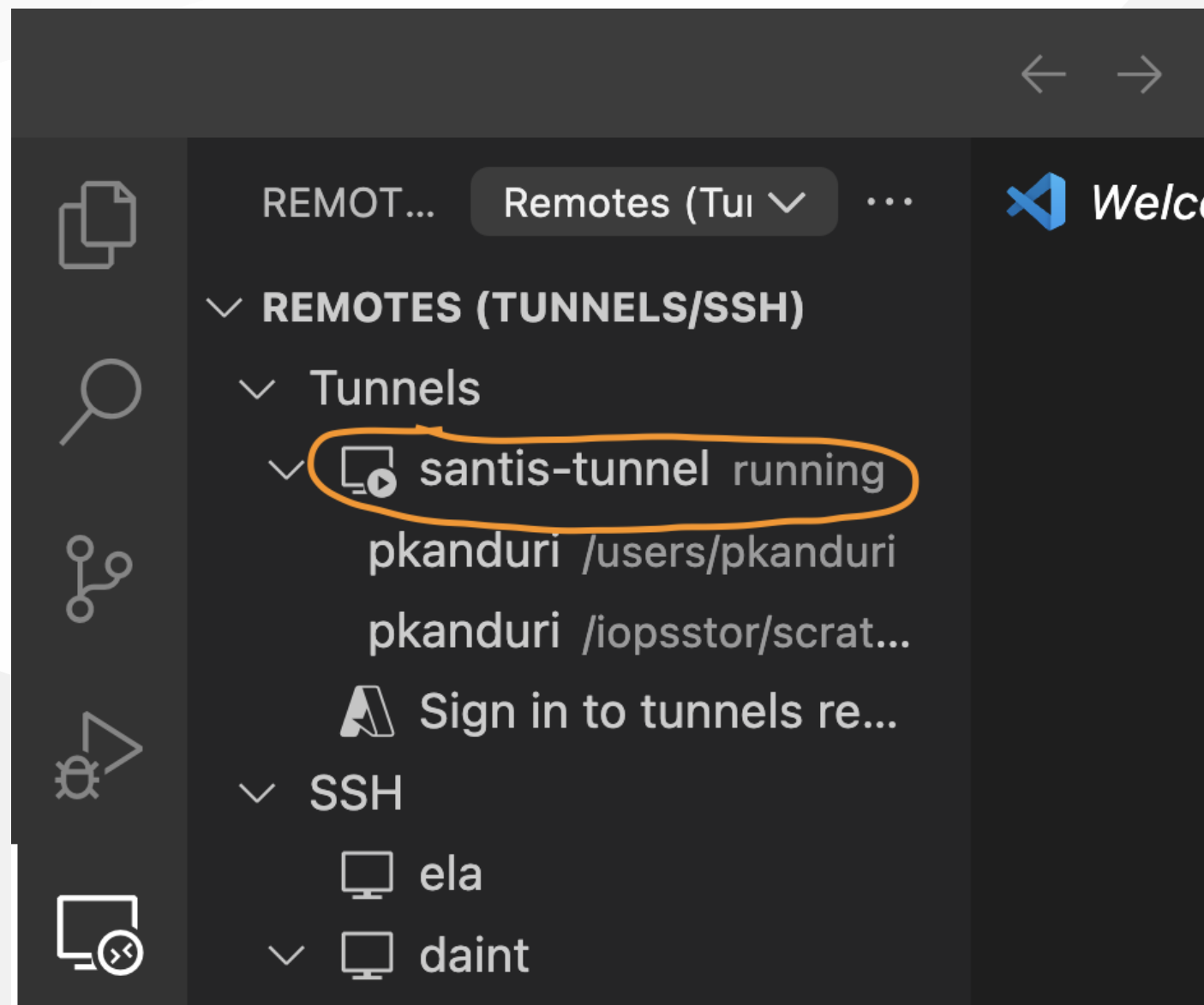
- ✓ Read org and team membership, read org projects
- ✓ Read all user profile data
- ✓ Full control of private repositories
- ✓ Access user email addresses (read-only)
- ✓ Update github action workflows

Organization access

- arbor-sim ✓
- beamlab ✓
- C2SM ✓
- eth-cscs ✓
- lab-cosmo ✓

Cancel Authorize Visual-Studio-Code

Access from VS Code



Interactively Connect to a Compute Node

One can log into a compute node and use the same `code tunnel` command.

```
# log into daint  
ssh daint  
  
# start an interactive shell session  
srun -t 120 -N 1 --pty bash  
  
# set up the environment before starting the tunnel  
uenv start prgenv-gnu/24.11:v1 --view=default  
code tunnel --name=$CLUSTER_NAME-tunnel
```

- `-t 120` requests a 2 hour (120 minute) reservation
- `-N 1` requests a single rank - only one rank/process is required for VSCode
- `--pty` allows forwarding of terminal I/O, for `bash` to work interactively

Connect VS Code with a Container

This will use CSCS's custom **Container Engine** which can easily pull a container from a registry like DockerHub. Same setup process as earlier with GitHub.

TOML File with Image and Mount Paths

```
image = "nvcr.io#nvidia/pytorch:24.01-py3"
writable = true
mounts = ["/iopsstor/scratch/cscs/username",
          "/users/username/.local/aarch64/bin",
          "/users/username/.bash_history"]
workdir = "/iopsstor/scratch/cscs/username"
```

Launch Container & Tunnel

```
# launch container on compute node
srun -N 1 --environment=/path/to/vscode-pytorch.toml --pty bash
# start tunnel
cd /path/to/code/executable
./code tunnel --name=$CLUSTER_NAME-tunnel
```

Using `code` on Different Architectures

Given the `code` application is architecture specific, and the Home Directory is shared across clusters, we can use a trick to streamline our workflow.

On both `x86` and `ARM` clusters, we can download the respective clusters and store them in specific folders using the `uname -m` command.

```
mkdir -p $HOME/.local/$(uname -m)/bin  
cp ./code $HOME/.local/$(uname -m)/bin
```

Then in the `.bashrc` file, we can add the following line which will ensure that the correct executable is available for the architecture.

```
export PATH=$HOME/.local/$(uname -m)/bin:$PATH
```