# Additional Topics: Usage of bwHPC & HoreKa Clusters

Samuel Braun
KIT, SCC

Baden-Württemberg
Ministerium für Wissenschaft,
Forschung und Kunst

# Outline

- Access and Data Transfer
  - SSH Keys
  - GUI applicatios / Remote Visualization
  - Best Practice: Data Sharing
  - WSL2
- VS Code
  - Remote SSH
  - code-server
  - Code Tunnel
  - Remote SSH to compute node
- Software Topics „Software II“
  - Using Containers: Apptainer/Enroot

# Access and Data Transfer
## SSH Keys

- **It is not possible to self-manage your SSH Keys by adding them to the ~/.ssh/authorized_keys file**
- Deposit and Register Keys
  - UC3: https://bwidm.scc.kit.edu/
  - HoreKa: https://fels.scc.kit.edu/
- Types of Keys:
  - *Interactive*, for login
  - *Command*, for automation
- Use the Key
  - `ssh/scp -i … / rsync -e "ssh -i $HOME/.ssh/somekey" …`
  - Or simply define it here `~/.ssh/config`
- Documentation:
  - https://wiki.bwhpc.de/e/Registration/SSH
  - https://www.nhr.kit.edu/userdocs/horeka/sshkeys/

# Access and Data Transfer
## SSH Key Types

- **Command Key**
  - Valid w/o 2FA login, limited lifetime
  - For automated workflows, e.g. CI/CD
  - Restriction to a **single command**
  - Example: For transfers with rsync, register the following command:
    `/usr/bin/rrsync -ro / -rw /`

- **Interactive Key**
  - Valid for one hour after 2FA login, limited lifetime

- **Interactive Key with FIDO2 Security Key** 😎
  - **No OTP required**, just touch FIDO stick
  - `ssh-keygen -t ed25519-sk …`
  - Supported clusters: bwUniCluster 3.0, NEMO 2, HoreKa

https://wiki.bwhpc.de/e/Registration/SSH#Step_2A:_Register_Interactive_Key
https://wiki.bwhpc.de/e/Registration/SSH#Step_2B:_Register_Command_Key

# Access and Data Transfer
# GUI applicatios / Remote Visualization

- **How to keep data on the cluster?**
  - ... and involve **applications with GUI?**
- Types of remote visualization / working on remote data
  - sshfs-mount
  - X11-Forwarding
  - Virtual Network Computing (VNC)
  - Jupyter
  - Server-Client model
- Post-processing as batch job
  - Example: Paraview

## Access and Data Transfer
## sshfs-mount

- Mount remote file system
- Work locally with GUI application
- `sshfs <user>@uc3.scc.kit.edu:/path/to/workspace /path/to/mountpoint`

- **Pros**
  - Easy to use

- **Cons**
  - **Data traffic**: Bandwidth, **Latency**
  - **Stress** to the **parallel file system**

# Access and Data Transfer
# X11-Forwarding

- Start GUI application on server (login/compute)
- Application window is shown locally
- `ssh -Y <user>@uc3.scc.kit.edu`

- **Pros**
  - Easy to use
  - Works well with e.g. MobaXterm
  - Data remains on server
- **Cons**
  - Data traffic: Bandwidth, Latency
  - Non-Linux: Additional software required
  - X11 → Wayland
    (but successor tools in the works)

## Access and Data Transfer
## Virtual Network Computing (VNC)

- Start remote desktop session on server
- `start_vnc_desktop` starts interactive job, starts VNC session, gives hints for connecting
- Start any application within desktop environment

- **Pros**
  - Entire desktop environment available
  - Data remains on server
- **Cons**
  - VNC viewer required

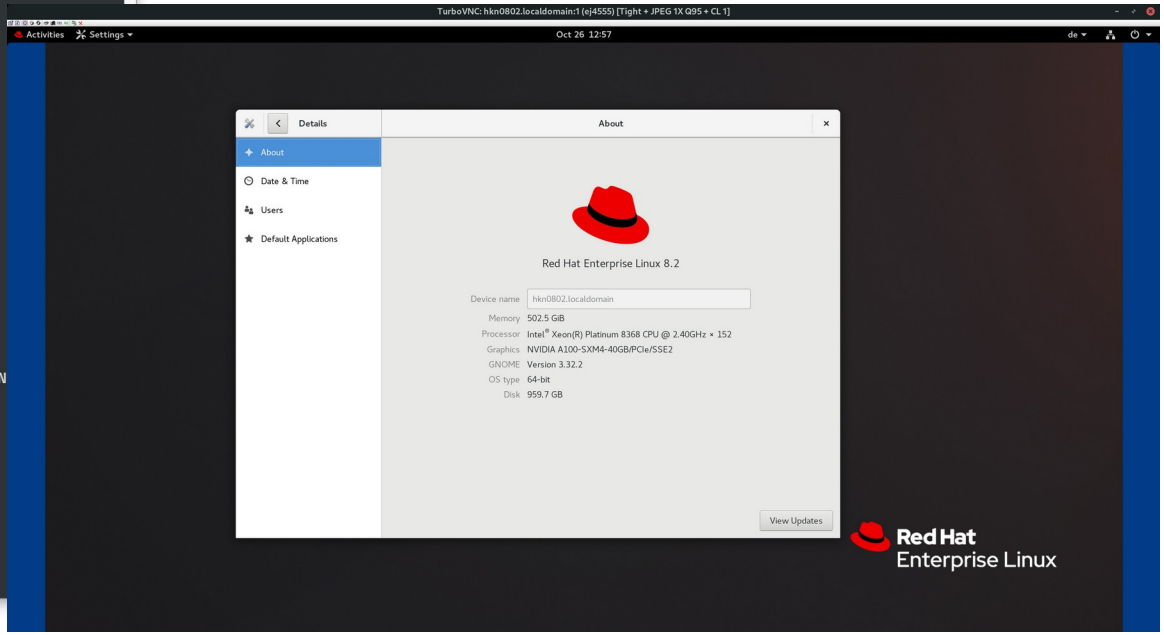
# Access and Data Transfer
## Virtual Network Computing (VNC)

```
hk:~$ start_vnc_desktop --hw-rendering
```

```
local:~$ /opt/TurboVNC/bin/vncviewer
```

ej4555@hkn0802:/hkfs/home/project/hk-project-scs/ej4555

```
[ej4555@hkn1990 ej4555]$ start_vnc_desktop --hw-rendering
 * Allocating 1 node (Use command line option --nodes to change)
 * Allocating 76 processors per node (Use command line option --ppn to change)
 * Allocating 4 GPUs per node (Use command line option --num-gpu to change)
 * Using queue 'accelerated' to allow hardware rendering
 * Using wall clock limit 02:00:00 (Use command line option --walltime to change)
Allocating resources to launch a VNC desktop:
Launching the VNC desktop on the resources granted by the batch system.
salloc: Granted job allocation 1498539
salloc: Waiting for resource configuration
salloc: Nodes hkn0802 are ready for job

Desktop 'TurboVNC: hkn0802.localdomain:1 (ej4555)' started on display hkn0802.localdomain:1

Starting applications specified in vglrun -d :0.0 -c proxy +wm /usr/bin/gnome-session
Log file is /home/hk-project-scs/ej4555/.vnc/hkn0802.localdomain:1.log

Hint for TurboVNC Viewer users (command line):
==============================================
  vncviewer ExtSSH=1 Via=ej4555@hk.scc.kit.edu Server=hkn0802:1 Password=olbmW1+9 SecurityTypes=VN

Hint for TurboVNC Viewer users (GUI)
====================================
  Fill in the following entry field:
  VNC server:   hkn0802:5901

  Click "Options" and choose tab "Security".
  Fill in the following entry fields:

  Gateway (SSH server or UltraVNC repeater)
```



## Access and Data Transfer
## Virtual Network Computing (VNC) via JupyterHub 😎

- Start remote desktop session via JupyterHub*, choose Basemodule `jupyter/extensions`
- Access VNC desktop session in new browser tab
- Start any application within desktop environment

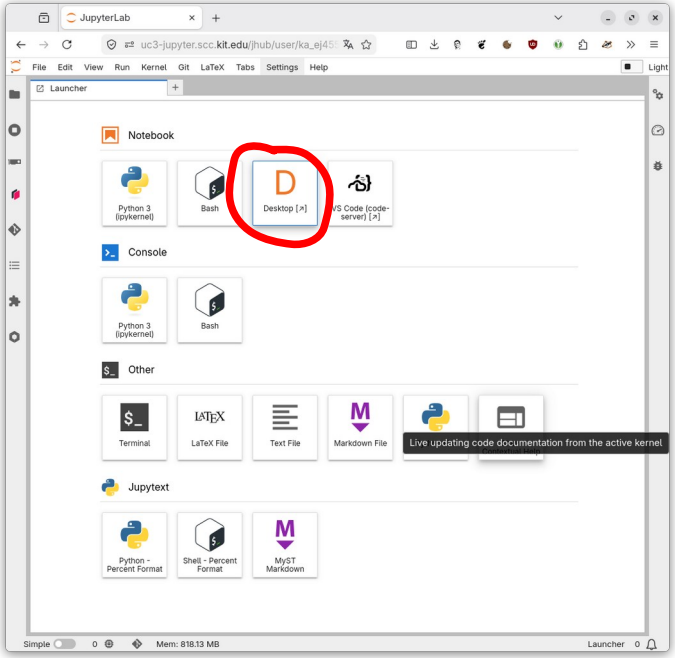
- **Pros**
  - Entire desktop environment available
  - Arbitrary GUI applications on server
  - Data remains on server
  - No additional software required
  - **Trivial to use**

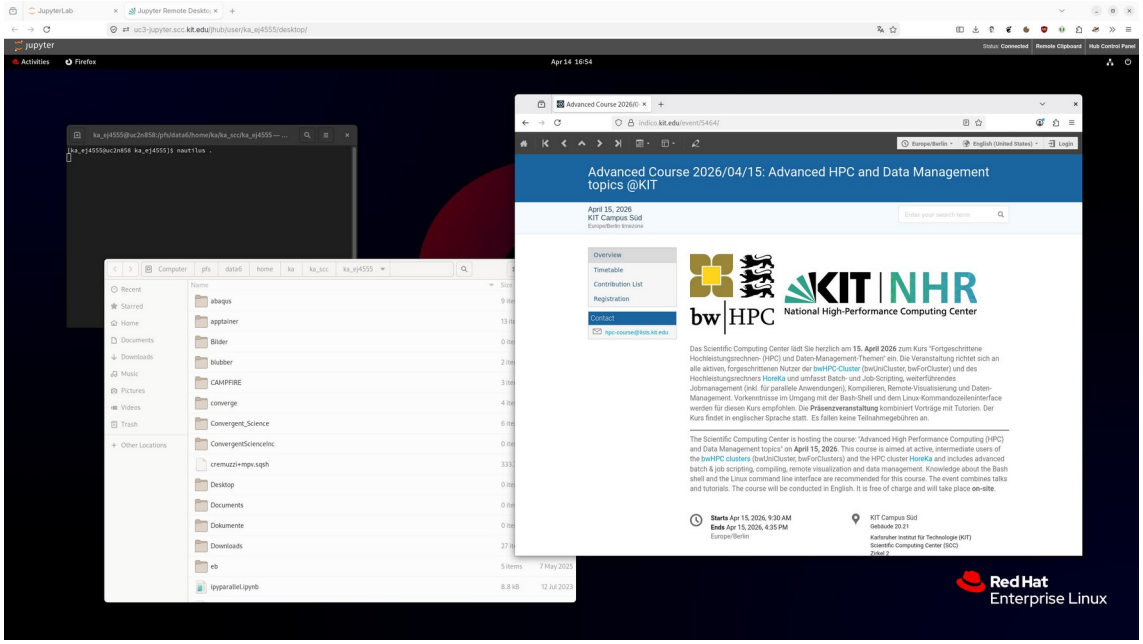
- **Cons**
  - Some users may prefer not to work in a browser and instead run the GUI applications natively.

***) JupyterHub instances**

https://uc3-jupyter.scc.kit.edu
https://hk-jupyter.scc.kit.edu

# Access and Data Transfer
## Virtual Network Computing (VNC) via JupyterHub



# Access and Data Transfer
## Jupyter

- Access server via web-browser
- Manually start JupyterLab or usage via JupyterHub*

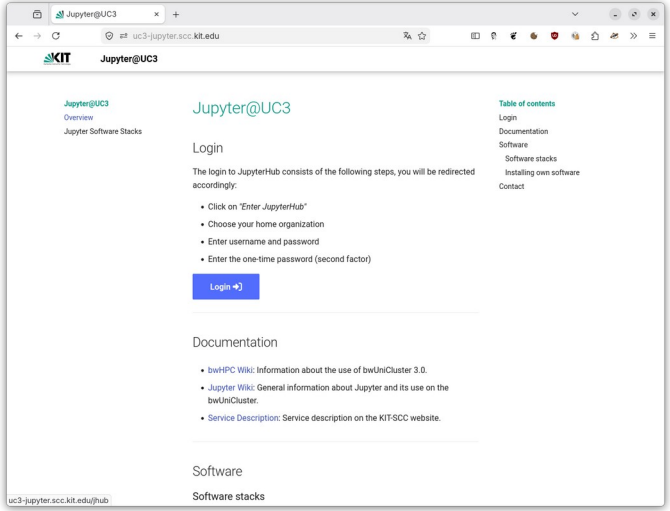
- **Pros**
  - Data remains on server
  - No additional software required
  - Trivial to use
- **Cons**
  - No execution of arbitrary GUI applications

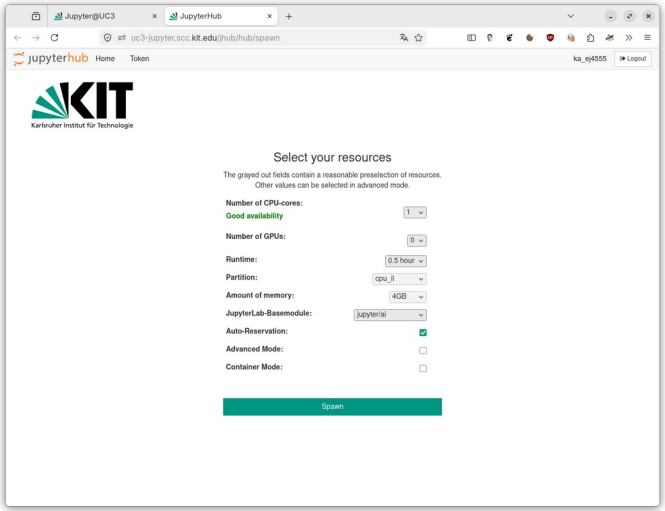
***) JupyterHub instances**

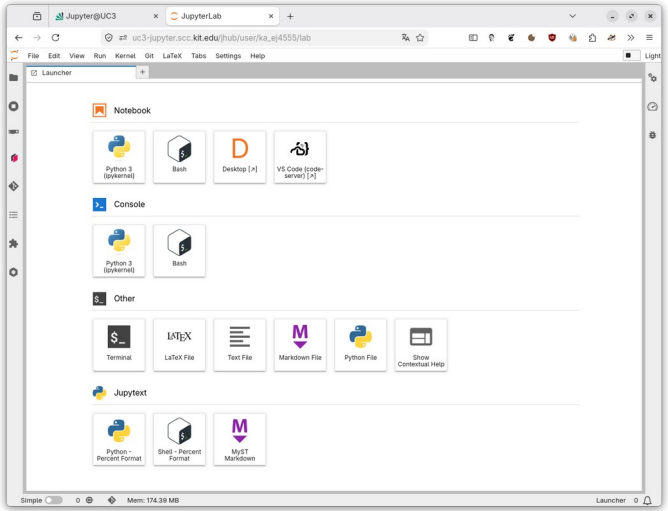
https://uc3-jupyter.scc.kit.edu
https://hk-jupyter.scc.kit.edu

# Access and Data Transfer
## Jupyter

- JupyterHub
  - Browse JupyterHub
  - Login
  - Select resources and modules
  - Spawn



# Access and Data Transfer
## Jupyter

- Manually start JupyterLab
  - Login to login node
  - Start interactive job
  - Load jupyter module (or install in own venv)
  - Start jupyter

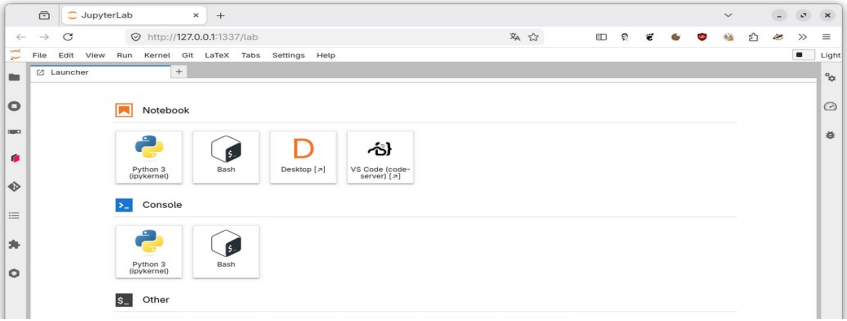
```
$ ssh uc3.scc.kit.edu
$ salloc -p dev_cpu -t 30
$ module load jupyter/extensions
$ jupyter lab --no-browser --port=1337 --ip 0.0.0.0
```

  - Create SSH tunnel: Laptop → login node → compute node

```
# New terminal
ssh uc3.scc.kit.edu -L 1337:<ComputeNodeID>:1337
```

  - Browse https://127.0.0.1:1337

```
Click http-link in terminal
```



## Access and Data Transfer
## Server-Client model

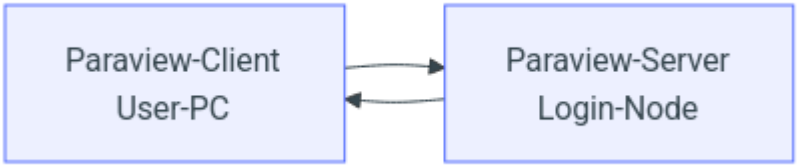
- Many applications feature server-client model
- Start server-side application on login/compute node
- Establish SSH tunnel (only port 22 is open)
- Connect client-side application to server

- **Pros**
  - Data remains on server
- **Cons**
  - GUI application must support server-client mode

# Access and Data Transfer
# Server-Client model

## Example: Paraview (on login node)

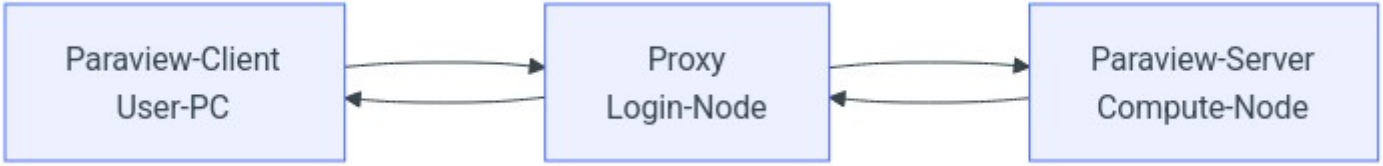
- Login to Cluster
- Start pvserver
- Establish a tunnel
- Connect client (server and client MUST be same software versions)




```
# Connect
ssh <user>@uc3.scc.kit.edu

# Load software
source /opt/bwhpc/common/etc/easybuild/enable_eb_modules
module load ParaView

# Go to workspace
cd $(ws_find <myWorkspace>)

# Start server
pvserver -display :0

# Establish SSH tunnel (from new terminal)
ssh <user>@uc3.scc.kit.edu -L 111111:localhost:111111
```

# Access and Data Transfer
# Server-Client model

- **Example: Paraview (on compute node)**
  - Login to Cluster
  - Allocate computre node
  - Start pvserver
  - Establish a tunnel
  - Connect client (server and client MUST be same software versions)




```
# Connect
ssh <user>@uc3.scc.kit.edu
salloc -p <partition> --gres=gpu:1

# Load software
source /opt/bwhpc/common/etc/easybuild/enable_eb_modules
module load ParaView

# Go to workspace
cd $(ws_find <myWorkspace>)

# Start server
pvserver -display :0

# Establish SSH tunnel (from new terminal)
ssh <user>@uc3.scc.kit.edu -L 111111:<nodeID>:111111
```

## Access and Data Transfer
## Post-processing as batch job

- Reproducible Post-processing: Use scripts/macros!
- For similar/identical post-processing tasks on large number of simulations

- **Pros**
  - Consistent post-processing
  - Saves you work
- Cons
  - High initial effort

## Access and Data Transfer
## Post-processing as batch job

- Python script: `automation.py`
- Assumption:
  Paraview Statefile exists

```
#!/usr/bin/env python

from paraview.simple import *
LoadState("state.pvsm")
myview = GetActiveView()

SaveAnimation('bild.png', myview,
      SuffixFormat = ".%04d",
      CompressionLevel = 0,
      FrameRate = 1)
```

- Execute script via batch job
- `xvfb-run` pretends that graphical output exists (virtual X server)

```
#!/bin/bash

#SBATCH -partition=cpu
#SBATCH -nodes=1
#SBATCH -ntasks-per-node=1
#SBATCH -cpus-per-task=10
#SBATCH -time=48:00:00

source /opt/bwhpc/common/etc/easybuild/enable_eb_modules
module load ParaView

xvfb-run -a -s "-screen 0 3840x2160x24" pvbatch automation.py
```

# Access and Data Transfer
## Best Practice: Data Sharing

- How to share data with another person on the same cluster?

```
$ ws_allocate sharing 30
...
$ ls -ld $(ws_find sharing)
drwx------ 2 ab1234 xyz 4096 Oct  9 00:42 /pfs/work7/workspace/scratch/ab1234-sharing-0
```

→ workspace is private!

- Allow full permission for user xy3456

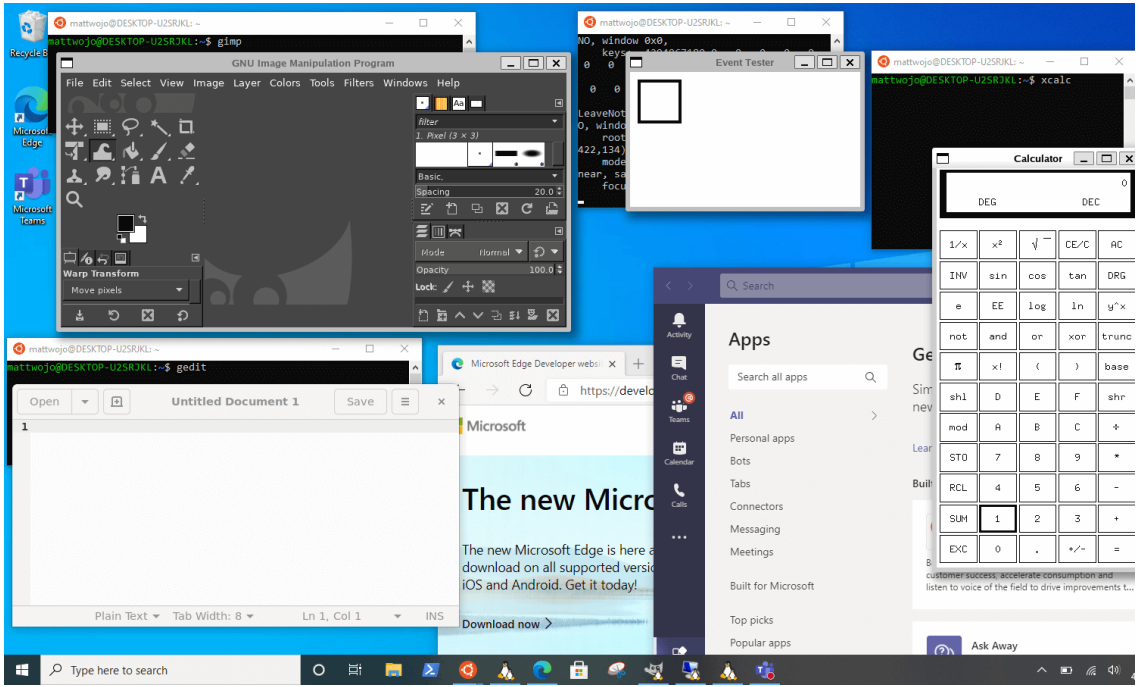
```
$ setfacl -Rm u:xy3456:rwX,d:u:xy3456:rwX,other::-,group::-,d:other::-,d:group::- $(ws_find sharing)
```

→ Access Control Lists (ACLs) are used
→ Default ACLs define permissions for newly created files and directories

- Further information and examples:
https://wiki.bwhpc.de/e/Workspace#Sharing_Workspace_Data_within_your_Workgroup

# Access and Data Transfer
## WSL2

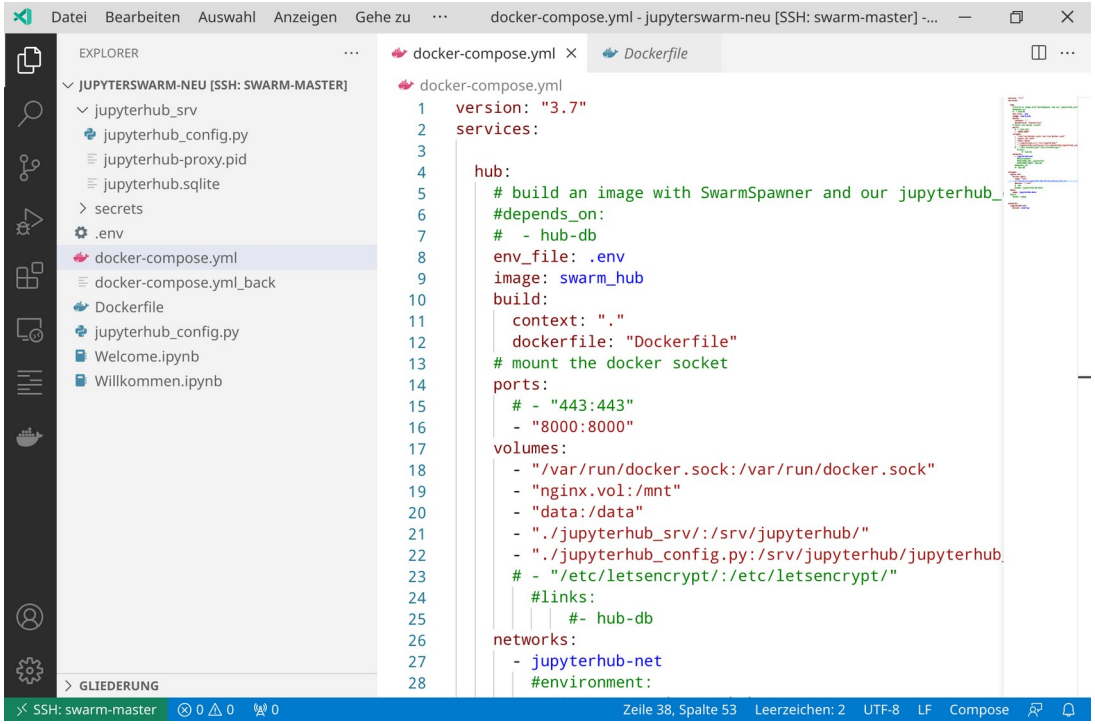
- Linux environment directly on Windows
- Different distros
- Very well integrated VM
  - Access to file systems
  - Windows Terminal
  - GUIs
- All Linux tools at hand:
  - ssh
  - rsync
  - …
- Documentation:
  https://docs.microsoft.com/de-de/windows/wsl/




https://docs.microsoft.com/de-de/windows/wsl/tutorials/gui-apps

# Visual Studio Code Remote SSH

- IDE, code editor
- VS Code has become very popular
- Windows, macOS and Linux
- Extension: Remote SSH
  - SSH to HPC systems
  - Connection to local WSL2
  - Supports 2FA, respects `~/.ssh/config`
- Documentation:
  - https://code.visualstudio.com/docs/remote/remote-overview
  - Remote SSH



# Visual Studio Code code-server

- Server-side component of VS Code
- Access VS Code in web browser
- Development and Debugging on compute nodes!

Usage:

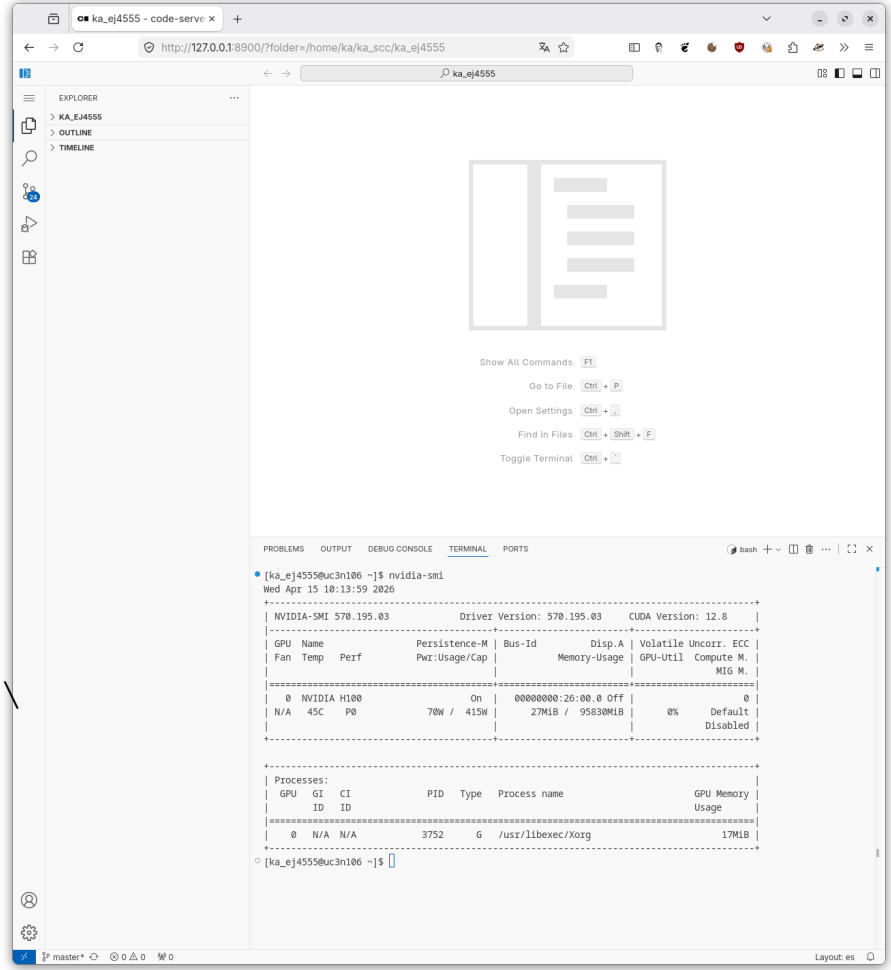
- Start code-server on compute node

```
module load devel/code-server
PASSWORD='EinSicheresPasswort!!1elf' \
     code-server --bind-addr 0.0.0.0:8900 \
     --auth password
```

- SSH-Tunnel

```
ssh -L 8900:<uc3nXXX>:8900 \
     ka_ab1234@uc3.scc.kit.edu
```

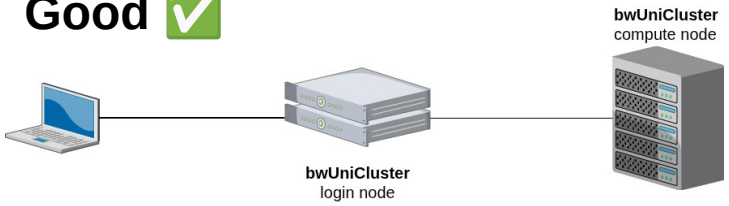
- Open web browser at http://127.0.0.1:8081
- Documentation:
  - code-server
  - https://github.com/cdr/code-server/tree/main/docs



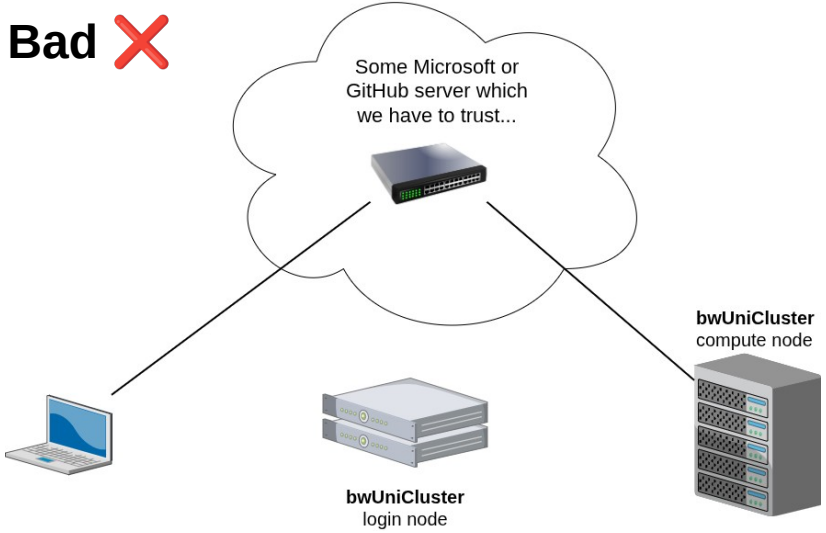
## Visual Studio Code Code Tunnels

- Connects server-side `code tunnel` via {GitHub,Microsoft}-proxy to client side VS Code
- **Allows access to compute nodes**
- **Bypasses intended security measures and access policies**
- The need for countermeasures is currently being evaluated
- Documentation: https://code.visualstudio.com/docs/remote/tunnels

**Good** ✅




**Bad** ❌



# Visual Studio Code
# Remote SSH to compute node

- Use default Remote SSH plugin

```
[ka_ab1234@uc3n991 ~]$ salloc -p gpu_h100_short --gres=gpu:1 -t 30
```

- Run your own SSH-server on compute node

```
[ka_ab1234@uc3n106 ~]$ /usr/sbin/sshd -d -f ~/.ssh/sshd_config
```

  Establish SSH tunnel to compute node

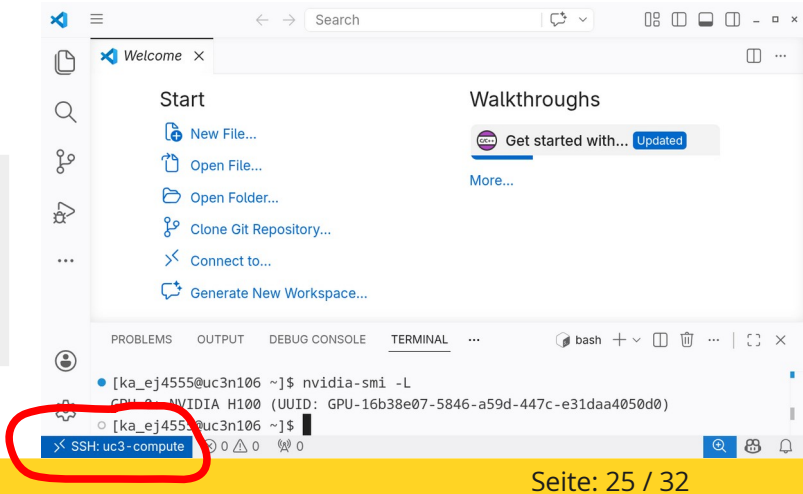
```
[samuel@laptop ~]$ ssh uc3-login1.scc.kit.edu -L 2222:uc3n106:2222
```

- Connect VS Code Remote SSH to localhost

😎😎😎

```
# SSHD configuration (bwUniCluster)

[ka_ab1234@uc3n991 ~]$ cat ~/.ssh/sshd_config
Port 2222
AddressFamily any
ListenAddress 0.0.0.0
PermitRootLogin no
PasswordAuthentication no
PubkeyAuthentication yes
HostKey ~/.ssh/keys/ssh_host_rsa_key
```

```
# SSH configuration (laptop)

[samuel@laptop ~]$ cat ~/.ssh/config
Host uc3-compute
    HostName localhost
    User ka_ab1234
    Port 2222
    IdentityFile ~/.ssh/myKey
```



## Visual Studio Code
## Remote SSH to compute node

- `start-vscode-remote-ssh.sh` 😎
- Takes any other SLURM arguments (partition, number of cores, ...)
- Default: 1 CPU core on special „agent“ partition, 1 hour, reservation 8-20h, 3 days wall clock time
- Just run it and follow instructions
- Currently only available on HoreKa 2

- **Terminal 1**
  - Login to login node
  - Start `start-vscode-remote-ssh.sh`
  - Follow (**all**) instructions
- **Terminal 2**
  - Login to login node and establish the SSH tunnel (→ hint how to do that in Terminal 1)
- **VS Code Remote SSH** or **Terminal 3**
  - Directly connect to compute node: `ssh vscode-hk2`

# Software Topics
# Using Containers: Apptainer/Enroot

## Container

- Lightweight, isolated runtime environment for applications → OS-level virtualization
- Packages the application + its dependencies into a single image
- Consistent/reproducible/portable environment
  → attractive for scientific HPC-workloads!

## Docker

- Released 2013
- First widely popular container solution
- Ecosystem
  - Tools: build, run, and manage containers
  - Dockerhub: central public registry
- Not suited for HPC systems
  - (Root) deamon required
  - Security concerns
  - Isolation

```
# Get and run Python 3.15
docker run -it --rm -v $(pwd):/mnt python:3.15 bash
```

# Software Topics
# Using Containers: Apptainer/Enroot

## Containers for HPC

- Dedicated HPC container runtime
- Compatibility with file systems
- Unprivileged, no daemon

## Enroot

- Origin: NVIDIA
- https://github.com/NVIDIA/enroot
- Minimal, rootless container tool for OCI images
- GPU support

## Apptainer

- Origin: Lawrence Berkeley National Laboratory
  Now: community-led
- https://github.com/apptainer/apptainer
- Most popular (?) HPC container solution

❤️ **Recommended tool on bwUniCluster & HoreKa**

## Software Topics
## Using Containers: Apptainer/Enroot

- Using Apptainer
  - Build a container image:
    `apptainer build`
  - Run shell/command in container image:
    `apptainer shell / run`
  - Run an application inside a container:
    `apptainer exec`

```
[samuel@laptop ~]$ cat /etc/os-release | grep PRETTY
PRETTY_NAME="Manjaro Linux"
[samuel@laptop ~]$ apptainer exec --no-home docker://ubuntu cat /etc/os-release | grep PRETTY
INFO:    Using cached SIF image
PRETTY_NAME="Ubuntu 24.04.4 LTS"
```

- Use `--no-home` to avoid loss of your history ;)
- Documentation:
  - https://www.nhr.kit.edu/userdocs/horeka/containers/
  - https://apptainer.org/docs/user/latest/

# Weitere Informationen

Die Initiative „Baden-Württemberg High Performance Computing (bwHPC)“ bietet Wissenschaftler*innen in Baden-Württemberg Zugang zu verschiedenen Hochleistungsrechnern – ideal für datenintensive Forschung, komplexe Berechnungen oder KI-Entwicklung. Auch Studierende, Promovierende und Forschende aus Baden-Württemberg können diese landesweite Infrastruktur und die zugehörigen Support- und Schulungsangebote nutzen.
bwHPC betreibt mehrere spezialisierte Clusterrechner, die in Fachbereiche aufgeteilt sind:

- **bwUniCluster 3.0** (Karlsruher Institut fürTechnologie): Grundversorgung für alle Disziplinen
- **JUSTUS 2** (Universität Ulm): Theoretische Chemie, Quanten- und Festkörperphysik
- **NEMO 2** (Universität Freiburg): Neurowissenschaften, Elementarteilchenphysik, Mikrosystemtechnik, Materialwissenschaften
- **Helix** (Universität Heidelberg): Strukturelle Biologie und Systembiologie, Medizinwissenschaft, weiche Materie, Computational Humanities, Mathematik und Informatik
- **BinAC 2** (Universität Tübingen): Bio- und Medizininformatik, Pharmazie, Astrophysik und Geowissenschaften

Die Registrierung erfolgt clusterspezifisch und setzt ein sogenanntes „Entitlement“ sowie ein genehmigtes Rechenvorhaben voraus. Eine Übersicht zum Anmeldeprozess finden Sie im bwHPC-Wiki.

## Webseite

https://www.bwhpc.de

## Wiki

https://wiki.bwhpc.de

## Mail

office@bwhpc.de