HPC Cluster

- OpenOnDemand
  - Open OnDemand User Guide
  - OpenOnDemand Job Templates User Guide
  - Apptainer User Guide for R on VoWa HPC

OpenOnDemand

Open OnDemand User Guide

HPC Cluster @ Vogelwarte

Version 1.0 | Last Updated: December 2025 | Thx to Claude Code

Introduction

What is Open OnDemand?

Open OnDemand (OOD) is a web-based portal that provides easy access to the VoWa HPC cluster. Through your web browser, you can:

- Launch interactive applications (JupyterLab, RStudio, VS Code, Remote Desktop)
- Manage files on the cluster
- Access terminal shells
- Submit and monitor computational jobs
- Work with your research data

Accessing Open OnDemand

Portal URL: https://hpc.vogelwarte.ch

Authentication: Single Sign-On (SSO) via your Vogelwarte Microsoft/Azure AD account

Requirements:

- A modern web browser (Chrome, Firefox, Edge, or Safari)
- Vogelwarte network access (VPN if working remotely)
- Active Vogelwarte account with HPC access permissions

Getting Started

First Login

1. Navigate to https://hpc.vogelwarte.ch
2. Click the login button
3. Authenticate using your Vogelwarte credentials (same as email/Office 365)
4. You'll be redirected to the Open OnDemand dashboard

Dashboard Overview

After logging in, you'll see the OOD dashboard with:

- **Pinned Apps**: Quick access to frequently used applications
- **Files**: File browser for managing your data
- **Jobs**: View and manage your running computational jobs
- **Clusters**: Shell access to cluster nodes
- **Interactive Apps**: Launch graphical applications

Storage Access

Your home directory and shared storage are automatically accessible:

Location	Path on hpc	Path on Windows (Mac)	Purpose
Home Directory	`~/` or `/home/vogelwarte.ch/[username]`	`\\pallidus.vogelwarte.ch\[username]`	Personal files and settings
SciData	`~/SciData`	`Z:\SciData`	Shared scientific data storage (CephFS)
Scratch	`~/scratch`	`Z:\SciData\ORG_Vogelwarte\scratch`	High-performance temporary storage
Data	`/mnt/ceph`	Direct access to CephFS shared storage	

Note: The `SciData` and `scratch` directories are symbolic links created automatically in your home directory for convenient access.

Interactive Applications

Open OnDemand provides several interactive applications that run on compute nodes with dedicated resources.

JupyterLab

Description: Modern web-based interface for Jupyter notebooks, code editing, and data visualization.

Pre-installed Packages:

- Python data science stack: NumPy, Pandas, Scikit-learn
- Visualization: Matplotlib, Seaborn
- JupyterLab, IPython kernel

How to Launch:

1. Click **Interactive Apps → JupyterLab**
2. Configure your session:
   - **Account**: Select your Slurm account (usually `sci_it` or `root`)
   - **Partition**: Choose `compute` for general work
   - **Number of cores**: 1-16 (start with 2)
   - **Memory (GB)**: 1-64 (start with 4)
   - **Hours**: Maximum session time (1-72 hours)
3. Click **Launch**
4. Wait for the job to start (status: Queued → Running)
5. Click **Connect to JupyterLab** when ready

Tips:

- Start small (2 cores, 4GB RAM) and increase if needed
- Save your work frequently
- Your notebooks are saved in your home directory
- Use `~/SciData` for accessing shared datasets

RStudio Server

Description: Full RStudio IDE in your browser for R programming and statistical analysis.

Pre-installed Packages:

- **Core:** tidyverse, ggplot2, dplyr, data.table
- **Spatial:** sf, tmap, rnaturalearth, amt
- **Statistics:** randomForest, ranger, Bayesian tools (NIMBLE, JAGS)
- **Data:** RPostgres, DBI, readr, readxl
- **Visualization:** viridis, bayesplot, kableExtra
- **And many more** (see full list in role configuration)

How to Launch:

1. Click **Interactive Apps → RStudio Server**
2. Configure your session:
   - **Account**: Select your Slurm account
   - **Partition**: Choose `compute`
   - **Number of cores**: 1-16 (start with 2)
   - **Memory (GB)**: 4-64 (R can be memory-intensive, start with 8GB)
   - **Hours**: Session duration
3. Click **Launch**
4. Wait for job allocation
5. Copy `password`because for security reason there is a temprary login
6. Click **Connect to RStudio Server**
7. Enter login `[username]` and copied temporary `password`

Tips:

- RStudio sessions use more memory than JupyterLab (request at least 8GB)
- Install additional packages with `install.packages()` (saved in your home directory)
- Use `renv` for reproducible project environments
- Connect to PostgreSQL databases using `RPostgres` package
- Parallel processing available with `foreach` and `doParallel` packages

VS Code Server (Code Server)

Description: Full-featured Visual Studio Code development environment in your browser.

Pre-installed Tools:

Python:

- Development: black, flake8, pylint
- Interactive: IPython, Jupyter
- Data Science: pandas, numpy, matplotlib, seaborn
- Utilities: requests, pytest

JavaScript/TypeScript:

- TypeScript compiler
- ESLint, Prettier
- Node.js and npm

System Tools:

- Git, vim, wget, curl

- Build tools (gcc, make)

How to Launch:

1. Click **Interactive Apps → Code Server**
2. Configure your session:
   - **Account**: Select your Slurm account
   - **Partition**: Choose `compute`
   - **Number of cores**: 1-8 (start with 2)
   - **Memory (GB)**: 2-32 (start with 4)
   - **Hours**: Session duration
3. Click **Launch**
4. Copy temporary `password`
5. Connect when ready
6. Enter temporary `password`

Tips:

- Install VS Code extensions from the marketplace
- Settings and extensions persist in your home directory
- Use integrated terminal for command-line access
- Great for multi-language projects
- Git integration built-in

Remote Desktop (MATE)

Description: Full Linux desktop environment with graphical applications.

Use Cases:

- Running GUI applications (GIS tools, visualization software)
- Using applications not available in other interfaces
- Traditional desktop workflow

How to Launch:

1. Click **Interactive Apps → Desktop**
2. Configure resources (similar to other apps)
3. Choose **MATE** desktop environment
4. Launch and connect
5. Use the desktop like a regular Linux workstation

Tips:

- Requires more resources (start with 4 cores, 8GB RAM)

- Best for applications that require GUI
- Can run multiple terminal windows
- Copy/paste between your local machine and remote desktop

Resource Selection Guidelines

Choosing the right resources helps you get work done efficiently without wasting cluster capacity:

Application	Typical Use	Cores	Memory	Duration
JupyterLab	Data exploration	2	4 GB	2-4 hours
JupyterLab	Data processing	4-8	8-16 GB	4-8 hours
RStudio	Interactive analysis	2-4	8 GB	2-4 hours
RStudio	Large datasets	8-16	32-64 GB	4-8 hours
Code Server	Development	2	4 GB	4-8 hours
Desktop	GUI applications	4	8 GB	2-4 hours

Remember: You can always launch a new session with more resources if needed. Start small and scale up.

File Management

Files App

The built-in file manager lets you:

- Browse your home directory and shared storage
- Upload/download files
- Create, rename, move, and delete files/folders
- Edit text files directly in the browser
- View file permissions

Accessing the File Manager:

1. Click **Files** in the top menu
2. Choose a location:
   - **Home Directory**: Your personal files
   - **SciData**: Shared scientific data

- Any custom path

Common Operations:

- **Upload**: Click **Upload** button, select files
- **Download**: Right-click file → **Download**
- **Create Folder**: Click **New Folder**
- **Edit File**: Click on text file to open editor
- **Move/Copy**: Select files → Use toolbar buttons
- **Change Permissions**: Right-click → **Change Permissions**

Data Transfer

Small Files (<100 MB): Use the web file manager upload/download feature.

Large Files (>100 MB): Use command-line tools via shell access:

```
# From your local machine to cluster
scp large_file.tar.gz username@hpc.vogelwarte.ch:/home/username@vogelwarte.ch/

# Using rsync for efficient transfer
rsync -avzP local_directory/ username@hpc.vogelwarte.ch:~/remote_directory/

# From cluster to local machine
scp username@hpc.vogelwarte.ch:~/results.zip ./
```

Shared Data Storage:

- Use `~/SciData` for data that needs to be shared with collaborators
- Use `~/scratch` for temporary high-performance storage
- Regular backups are performed on home directories, not scratch

---

Shell Access

Cluster Shell Access

Open OnDemand provides web-based terminal access to the cluster.

How to Access:

1. Click **Clusters** in the top menu
2. Select **Shell Access** or your cluster name
3. A terminal window opens in your browser

What You Can Do:

- Run command-line tools
- Submit batch jobs with Slurm
- Check job status
- Compile code
- Manage files with CLI tools

Session Timeouts:

- **Inactive timeout**: 5 minutes (default)
- **Maximum duration**: 1 hour (default)
- Sessions close automatically after timeout for security

Tips:

- Use interactive apps for long-running work
- For persistent sessions, use `tmux` or `screen`
- Shell ping-pong can be enabled (contact admin) for keep-alive

Basic Slurm Commands

If you need to submit batch jobs from the shell:

```
# View partition information
sinfo

# Submit a batch job
sbatch job_script.sh

# Check your job queue
squeue -u $USER

# Cancel a job
scancel <job_id>

# View job details
scontrol show job <job_id>
```

```
# View cluster usage
squeue
```

Note: Most users will use interactive apps and won't need to submit batch jobs directly.

Best Practices

Resource Management

1. **Request Appropriate Resources**
   - Don't over-request cores/memory you won't use
   - Start small and scale up if needed
   - Consider other users sharing the cluster
2. **Session Duration**
   - Choose realistic time limits
   - Terminate sessions when done (don't leave them running)
   - Save your work frequently
3. **Data Storage**
   - Home directory: Personal files, code, small datasets
   - SciData: Shared datasets, collaborative projects
   - Scratch: Temporary high-I/O work (files may be deleted)

Security

1. **Authentication**
   - Never share your credentials
   - Log out when finished
   - Use VPN when accessing remotely
2. **Data Handling**
   - Don't store sensitive data without proper permissions
   - Check file permissions for shared data
   - Follow institutional data policies

Performance

1. **Efficient Computing**
   - Close unused applications to free resources
   - Use appropriate partitions for your work

- Optimize code before requesting large resources

2. **File Operations**
   - Use `rsync` for large transfers
   - Avoid many small file operations
   - Clean up old files and data regularly

Troubleshooting

Common Issues

Issue: Cannot log in

- **Solution**: Verify VPN connection, check credentials, contact IT

Issue: Interactive app won't start (stays in "Queued" state)

- **Possible causes:**
  - Cluster is busy (wait a bit)
  - Requested resources exceed limits
  - Requested partition doesn't exist
- **Solution**: Try reducing resources or contact support

Issue: Session disconnected unexpectedly

- **Possible causes:**
  - Network interruption
  - Session timeout
  - Cluster maintenance
- **Solution**: Reconnect; your work may be saved depending on the application

Issue: Application runs out of memory

- **Solution**: Terminate and relaunch with more memory

Issue: Can't access shared data

- **Possible causes:**
  - Permissions issue
  - Mount point not available
- **Solution**: Check file permissions, contact admin if storage mount is down

Issue: Files don't appear in file manager

- **Solution**: Refresh browser, check path, verify permissions

Getting Help

Before Contacting Support:

1. Note the exact error message
2. Record what you were trying to do
3. Check this guide and FAQs
4. Try basic troubleshooting steps

Session Information: When reporting issues with interactive apps, provide:

- Application name (JupyterLab, RStudio, etc.)
- Session ID (visible in "My Interactive Sessions")
- Time of issue
- Error messages

Support

Documentation

- **This Guide**: Comprehensive user documentation
- **Open OnDemand Official Docs**: https://osc.github.io/ood-documentation/
- **Slurm Documentation**: https://slurm.schedmd.com/documentation.html

Contact

VoWa HPC Support Team

- **Email**: scientific.it@vogelwarte.ch

What to Include in Support Requests:

- Your username
- Description of the issue
- Steps to reproduce
- Error messages (screenshots helpful)

- Application and session information

System Status

Check Cluster Status:

- Dashboard shows current cluster availability
- Maintenance windows announced via email
- Emergency maintenance posted on login page

Appendix

Slurm Accounts

Your jobs run under Slurm accounts for resource tracking:

Account	Description	Typical Use
`sci_it`	IT Science Account	General scientific computing
`root`	Root Account	Administrative or special projects

Check your accounts:

```
sacctmgr show user $USER
```

Partitions

Compute resources are divided into partitions:

Partition	Description	Typical Resources
`normal`	General computing	Standard CPU nodes

Software Environment

Containerized Applications: All interactive apps run in Apptainer (formerly Singularity) containers, providing:

- Consistent software environments
- Pre-configured tool stacks
- Isolation and security
- Reproducibility

Custom Software: Contact support if you need:

- Additional Python/R packages
- Specialized scientific software
- Custom container images
- System-wide installations

Keyboard Shortcuts

In Web Shell:

- `Ctrl+C`: Cancel current command
- `Ctrl+D`: Exit shell
- `Ctrl+L`: Clear screen
- `Tab`: Auto-complete

In File Manager:

- `Ctrl+A`: Select all
- `Delete`: Delete selected
- `F2`: Rename

In Interactive Apps: Depends on the application (JupyterLab, RStudio, VS Code each have their own shortcuts)

Changelog

Version 1.0 (December 2025)

- Initial release
- Covers JupyterLab, RStudio, Code Server, and Desktop apps
- Basic file management and shell access
- Resource management guidelines

Quick Reference Card

URLs

- **Portal**: https://hpc.vogelwarte.ch
- **File Manager**: Click "Files" → "Home Directory"
- **Shell**: Click "Clusters" → "Shell Access"

Getting Help

- Check this guide first
- Contact HPC support via [email/portal]
- Include error messages and session details

Resource Recommendations

- **Light work**: 2 cores, 4 GB, 2-4 hours
- **Medium work**: 4-8 cores, 8-16 GB, 4-8 hours
- **Heavy work**: 8-16 cores, 32-64 GB, 8-24 hours

Storage Paths

- **Home**: `~/` or `/home/vogelwarte.ch/[username]`
- **Shared Data**: `~/SciData` or `/mnt/ceph`
- **Scratch**: `~/scratch`

End of User Guide

This guide is maintained by the SciIT-Team. Suggestions and corrections welcome!

OpenOnDemand Job Templates User Guide

This guide explains how to use job templates in OpenOnDemand to submit SLURM jobs efficiently.

What are Job Templates?

Job templates are pre-configured SLURM job scripts that help you quickly submit common types of jobs without writing scripts from scratch. Each template includes:

- Pre-configured SLURM directives (cores, memory, time limits)
- Example code or workflows
- Documentation and best practices
- Ready-to-run scripts

Accessing the Job Composer

1. **Log in to OpenOnDemand** at your cluster's URL
2. Click on **"Jobs"** in the top navigation menu
3. Select **"Job Composer"** from the dropdown

You'll see the Job Composer interface with:

- List of your existing jobs (left sidebar)
- Job details and files (main panel)
- Action buttons (Submit, Edit, Delete, etc.)

Creating Jobs from Templates

Step 1: Create New Job from Template

1. In the Job Composer, click **"New Job"** button
2. Select **"From Template"**
3. Choose a template from the list (see Available Templates)
4. Click **"Create New Job"**

The template will be copied to your jobs directory with all necessary files.

Step 2: Review Job Location

Your new job is created in:

```
~/ondemand/data/sys/myjobs/projects/default/<job-id>/
```

Each job gets a unique directory containing:

- `script.sh` - The SLURM job script
- Template-specific files (e.g., example R scripts, definition files)
- `README.md` - Documentation (if included)

Step 3: Understand the Job Structure

Every job template includes a `script.sh` file with SLURM directives at the top:

```
#!/bin/bash
#SBATCH --job-name=my_job        # Job name
#SBATCH --time=01:00:00          # Time limit (HH:MM:SS)
#SBATCH --partition=normal       # Queue/partition
#SBATCH -n 1                     # Number of tasks
#SBATCH -c 4                     # CPU cores per task
#SBATCH --mem=8G                 # Memory
#SBATCH --output=%x-%j.out       # Output file
#SBATCH --error=%x-%j.err        # Error file
```

Editing Job Scripts

Using the Built-in Editor

1. In Job Composer, select your job from the left sidebar
2. Click on `script.sh` in the file list
3. Click **"Edit"** button
4. Make your changes in the editor
5. Click **"Save"** when done

Common Edits

Change Resource Requirements

Edit the SLURM directives to match your needs:

```
#SBATCH --time=04:00:00          # Increase time limit
#SBATCH -c 8                     # Use more CPU cores
#SBATCH --mem=32G                # Request more memory
#SBATCH --partition=gpu          # Use GPU partition
```

Add Your Data Files

Edit the script to point to your actual data:

```
# Change this:
Rscript hello.R

# To this:
Rscript /path/to/your/analysis.R
```

Modify Job Name and Output

```
#SBATCH --job-name=my_analysis   # Descriptive name
#SBATCH --output=results_%j.out  # Custom output filename
```

Uploading Additional Files

1. In Job Composer, select your job
2. Click **"Open Dir"** to open the job directory in the file browser
3. Use **"Upload"** button to add your data files
4. Update the script to reference your uploaded files

Submitting Jobs

Submit Your Job

1. Select your job in the Job Composer
2. Review the script and ensure all settings are correct
3. Click the **"Submit"** button

You'll see a confirmation message with the job ID (e.g., "Job submitted successfully with ID: 12345")

What Happens Next?

1. **Queued**: Job enters the SLURM queue
2. **Running**: Job starts when resources are available
3. **Completed**: Job finishes (check output files for results)
4. **Failed**: Job encountered an error (check error file)

Monitoring Jobs

View Active Jobs

1. Click **"Jobs"** → **"Active Jobs"** in the top menu
2. You'll see all your running and pending jobs
3. Information displayed:
   - Job ID
   - Job Name
   - Status (Running, Pending, Completed, Failed)
   - Time elapsed
   - Nodes/cores used

Check Job Output

While the job is running or after completion:

1. In Job Composer, select your job
2. Click on the output file (e.g., `my_job-12345.out`)

3. Click **"View"** to see the contents
4. Click **"Refresh"** to update (for running jobs)

View Job Details

For detailed job information:

1. Go to **"Jobs"** → **"Active Jobs"**
2. Click on your job ID
3. View comprehensive details:
   - Start time
   - Resource usage
   - Node assignment
   - Full job parameters

Available Templates

Basic R Serial Job

Template: `rscript`

Purpose: Run R scripts on a single core

Includes:

- `script.sh` - SLURM job script
- `hello.R` - Example R script with system information

Use cases:

- Data analysis
- Statistical computing
- Report generation

How to customize:

1. Replace `hello.R` with your R script or upload your own
2. Edit `script.sh` to reference your script:

   ```
   srun /usr/bin/apptainer exec /data/apps/rstudio.sif Rscript your_script.R
   ```

3. Adjust resources (memory, cores, time) as needed

Build Custom Apptainer Image

Template: `apptainer_builder`

Purpose: Build custom container images based on RStudio

Includes:

- `script.sh` - Build automation script
- `rstudio_custom.def` - Apptainer definition file
- `README.md` - Detailed instructions

Use cases:

- Installing additional R packages
- Adding system dependencies (GDAL, PROJ, etc.)
- Creating reproducible environments
- Custom software stacks

How to customize:

1. Edit `rstudio_custom.def` to add your packages:

```
%post
    apt-get update
    apt-get install -y your-system-packages

    R --slave -e 'install.packages(c("your", "packages"))'
```

2. Submit the job (build takes 1-4 hours)
3. Image is saved to `$HOME/apps/rstudio_custom.sif`
4. Use in future jobs or RStudio sessions

Advanced Usage

Creating Job Arrays

Run the same script multiple times with different parameters:

1. Edit your `script.sh` and add:

```
#SBATCH --array=1-10          # Run 10 instances
```

2. Use `$SLURM_ARRAY_TASK_ID` in your script:

```
Rscript analysis.R $SLURM_ARRAY_TASK_ID
```

Job Dependencies

Run jobs in sequence:

1. Submit first job and note the job ID (e.g., 12345)
2. Create second job with dependency:

```
#SBATCH --dependency=afterok:12345
```

Using Custom Container Images

After building a custom image:

1. Edit your job script
2. Change the container path:

```
# Instead of:
apptainer exec /data/apps/rstudio.sif Rscript script.R

# Use:
apptainer exec $HOME/apps/rstudio_custom.sif Rscript script.R
```

Email Notifications

Get notified when jobs complete:

```
#SBATCH --mail-type=END,FAIL     # Email on end or failure
#SBATCH --mail-user=your.email@example.com
```

Using GPU Resources

For GPU-accelerated jobs:

```
#SBATCH --partition=gpu          # GPU partition
#SBATCH --gres=gpu:1             # Request 1 GPU
#SBATCH --gres=gpu:2             # Or request 2 GPUs
```

Parallel Processing in R

For multi-core R jobs:

```
#SBATCH -c 8                     # Request 8 cores
```

In your R script:

```
library(parallel)
library(doParallel)

# Use all available cores
n_cores <- as.numeric(Sys.getenv("SLURM_CPUS_PER_TASK"))
registerDoParallel(cores = n_cores)

# Your parallel code here
results <- foreach(i = 1:1000) %dopar% {
    # Computation
}
```

Troubleshooting

Job Stays in Pending State

Possible causes:

- Requested resources not available
- Partition full or not accessible
- Time limit too high for requested partition
- Account/QOS limits reached

Solutions:

1. Check active jobs: **"Jobs"** → **"Active Jobs"**

2. Reduce resource requests (cores, memory, time)
3. Try a different partition
4. Contact admin if issue persists

Job Fails Immediately

Check the error file (`*.err`):

1. In Job Composer, select your job
2. Open the `.err` file
3. Look for error messages

Common issues:

- File not found: Check paths are absolute or relative to job directory
- Permission denied: Ensure files are readable
- Module not loaded: Container may be missing dependencies
- Syntax errors: Review script for typos

Out of Memory Errors

Symptoms:

- Job fails with "out of memory" or "killed" message
- Exit code 137

Solutions:

1. Increase memory request:

```
#SBATCH --mem=32G              # Instead of 8G
```

2. Use memory-efficient approaches in your code
3. Split job into smaller chunks

Container Image Not Found

Error: `FATAL: container not found`

Solutions:

1. Check the container path in your script

2. Verify the container exists:

```
ls -l /data/apps/rstudio.sif
```

3. If using custom image, ensure build completed:

```
ls -l $HOME/apps/rstudio_custom.sif
```

Job Takes Too Long

Options:

1. Request more time:

```
#SBATCH --time=12:00:00
```

2. Optimize your code (vectorization, parallel processing)
3. Use more CPU cores if parallelizable
4. Check if you're in the correct partition for long jobs

Can't Edit Files

If editor doesn't work:

1. Click **"Open Dir"** in Job Composer
2. Use the file browser's built-in editor
3. Or download file, edit locally, re-upload

Need to Cancel a Job

1. Go to **"Jobs"** → **"Active Jobs"**
2. Find your job in the list
3. Click **"Delete"** or **"Cancel"** button
4. Confirm the cancellation

Best Practices

Resource Estimation

- **Start small**: Begin with minimal resources and increase if needed
- **Monitor usage**: Check actual resource usage after jobs complete
- **Be realistic**: Don't over-request resources (wastes queue time)

File Organization

- Keep related jobs in same project directory
- Use descriptive job names
- Document your workflow in README files
- Clean up old jobs periodically

Testing

- Test with small datasets first
- Use short time limits for testing
- Verify output before running large batches
- Use interactive sessions for debugging

Reproducibility

- Document software versions
- Use container images for consistent environments
- Save SLURM scripts with your results
- Note date and job ID in your analysis notes

Getting Help

Resources

- **Documentation**: Check README files in templates
- **Active Jobs**: Monitor job status and resource usage
- **Error Logs**: Always check `.err` files for failures

Contact Support

If you encounter persistent issues:

1. Note the job ID
2. Save error messages
3. Document what you've tried
4. Contact your HPC admin

Quick Reference

Common SLURM Directives

```
#SBATCH --job-name=name          # Job name
#SBATCH --partition=normal       # Queue/partition
#SBATCH --time=HH:MM:SS          # Time limit
#SBATCH -n 1                     # Number of tasks
#SBATCH -c 4                     # CPUs per task
#SBATCH --mem=8G                 # Memory per node
#SBATCH --mem-per-cpu=2G         # Memory per CPU
#SBATCH --output=file.out        # Output file
#SBATCH --error=file.err         # Error file
#SBATCH --mail-type=ALL          # Email notifications
#SBATCH --mail-user=email        # Email address
#SBATCH --gres=gpu:1             # GPU request
#SBATCH --array=1-10             # Job array
#SBATCH --dependency=afterok:123 # Job dependency
```

Environment Variables in Jobs

```
$SLURM_JOB_ID                    # Job ID
$SLURM_JOB_NAME                  # Job name
$SLURM_SUBMIT_DIR                # Submission directory
$SLURM_JOB_NODELIST              # Assigned nodes
$SLURM_NTASKS                    # Number of tasks
$SLURM_CPUS_PER_TASK             # CPUs per task
$SLURM_ARRAY_TASK_ID             # Array index
```

Useful Commands (in scripts)

```
cd $SLURM_SUBMIT_DIR             # Go to submission directory
echo "Job ID: $SLURM_JOB_ID"    # Print job info
date                             # Timestamp
hostname                         # Node name
```

Example Workflow

Complete Example: Running an R Analysis

1. **Create job from template**:
   - Jobs → Job Composer → New Job → From Template
   - Select "Basic R Serial Job"
2. **Upload your R script**:
   - Click "Open Dir"
   - Upload your `analysis.R` file
3. **Edit the job script**:

   ```
   #!/bin/bash
   #SBATCH --job-name=my_analysis
   #SBATCH --time=02:00:00
   #SBATCH -c 4
   #SBATCH --mem=16G

   cd $SLURM_SUBMIT_DIR
   srun apptainer exec /data/apps/rstudio.sif Rscript analysis.R
   ```

4. **Submit the job**:
   - Click "Submit"
   - Note the job ID
5. **Monitor progress**:
   - Jobs → Active Jobs
   - Check output file periodically
6. **Review results**:
   - Open `.out` file when job completes
   - Download output files if needed

Last Updated: 2025-12-03

Questions? Contact your SciIT team.

Apptainer User Guide for R on VoWa HPC

This guide explains how to use Apptainer (formerly Singularity) containers to run R scripts and build custom R environments on the VoWa HPC cluster.

What is Apptainer?

Apptainer (formerly Singularity) is a container platform designed for HPC environments. It allows you to:

- Run software in isolated, reproducible environments
- Use pre-built images with all dependencies included
- Build custom images with your specific f. e. R packages
- Share computational environments with collaborators
- Ensure reproducibility across different systems

Why Use Containers?

- **Reproducibility**: Same environment every time
- **Isolation**: No conflicts with system packages
- **Portability**: Works on any system with Apptainer
- **Dependency Management**: All packages bundled together
- **Version Control**: Pin specific software versions

Quick Start: Running R Scripts

Running R Scripts in a Container

The basic command to run an R script using the RStudio container:

```
apptainer exec /data/apps/rstudio.sif Rscript your_script.R
```

Example: Run a simple R script

```
# Create a test script
cat > hello.R << 'EOF'
print("Hello from Apptainer!")
print(paste("R version:", R.version.string))
print(paste("Working directory:", getwd()))
EOF

# Run it with Apptainer
apptainer exec /data/apps/rstudio.sif Rscript hello.R
```

Running R Scripts with Arguments

Pass arguments to your R script:

```
apptainer exec /data/apps/rstudio.sif Rscript analysis.R arg1 arg2 arg3
```

In your R script (`analysis.R`):

```
# Read command line arguments
args <- commandArgs(trailingOnly = TRUE)
print(paste("Argument 1:", args[1]))
print(paste("Argument 2:", args[2]))
```

Running in a SLURM Job

Create a job script (`job.sh`):

```
#!/bin/bash
#SBATCH --job-name=r_analysis
#SBATCH --time=01:00:00
#SBATCH -c 4
#SBATCH --mem=8G

# Navigate to your working directory
cd $SLURM_SUBMIT_DIR
```

```
# Run R script in container
apptainer exec /data/apps/rstudio.sif Rscript my_analysis.R
```

Submit the job:

```
sbatch job.sh
```

Using Apptainer Interactively

Start an Interactive R Session

Launch R inside the container:

```
apptainer exec /data/apps/rstudio.sif R
```

You'll get an R prompt where you can work interactively:

```
R version 4.x.x ...
> library(tidyverse)
> data <- read.csv("mydata.csv")
> summary(data)
> quit()
```

Interactive Shell Inside Container

Start a shell inside the container to explore:

```
apptainer shell /data/apps/rstudio.sif
```

Now you're inside the container and can run any command:

```
Apptainer> which R
Apptainer> R --version
Apptainer> ls /usr/local/lib/R/site-library/
Apptainer> exit
```

Interactive Session on Compute Node

For longer interactive work, request a compute node first:

```
# Request interactive node
srun --pty --mem=16G --cpus-per-task=4 --time=02:00:00 bash

# Once on the compute node, start R
apptainer exec /data/apps/rstudio.sif R
```

Available Container Images

System-Wide Images

Located in `/data/apps/`:

RStudio Container

```
/data/apps/rstudio.sif
```

Includes:

- Latest R version
- RStudio Server
- tidyverse packages
- Common R packages
- Development tools

Usage:

```
apptainer exec /data/apps/rstudio.sif Rscript script.R
apptainer exec /data/apps/rstudio.sif R
```

Your Custom Images

Located in `$HOME/apps/`:

After building custom images (see Building Custom Images), they're saved to:

```
$HOME/apps/rstudio_custom.sif
$HOME/apps/my_analysis.sif
```

Usage:

```
apptainer exec $HOME/apps/rstudio_custom.sif Rscript script.R
```

Checking What's in a Container

List installed R packages:

```
apptainer exec /data/apps/rstudio.sif R -e 'installed.packages()[,c("Package","Version")]'
```

Check R version:

```
apptainer exec /data/apps/rstudio.sif R --version
```

Check installed system packages:

```
apptainer exec /data/apps/rstudio.sif dpkg -l | grep -i gdal
```

Building Custom Images

Why Build Custom Images?

- Install additional R packages not in the base image
- Add system dependencies (GDAL, PROJ, JAGS, etc.)
- Create reproducible environments for specific projects
- Share your environment with collaborators

Using the Job Template

The easiest way to build custom images is using the OpenOnDemand job template:

1. **Access Job Composer**:
   - Log in to OpenOnDemand
   - Go to **Jobs** → **Job Composer**

2. **Create job from template**:
   - Click **New Job → From Template**
   - Select "**Build Custom Apptainer Image**"
3. **Edit the definition file** (`rstudio_custom.def`):

```
Bootstrap: docker
From: rocker/rstudio:latest

%post
    # Install system dependencies
    apt-get update
    apt-get install -y libgdal-dev libproj-dev

    # Install R packages
    R --slave -e 'install.packages(c(
        "sf",
        "terra",
        "raster",
        "your-package-here"
    ), repos="https://cloud.r-project.org/")'
```

4. **Submit the job**:
   - Click **Submit**
   - Wait 1-4 hours for build to complete
5. **Use your custom image**:

```
apptainer exec $HOME/apps/rstudio_custom.sif Rscript script.R
```

Definition File Examples

Example 1: Spatial Analysis Environment

```
Bootstrap: docker
From: rocker/rstudio:latest

%post
    # Spatial dependencies
    apt-get update
    apt-get install -y \
        libgdal-dev \
```

```
        libproj-dev \
        libudunits2-dev \
        libgeos-dev

    # Spatial R packages
    R --slave -e 'install.packages(c(
        "sf",
        "terra",
        "raster",
        "rgdal",
        "sp",
        "lwgeom",
        "stars"
    ), repos="https://cloud.r-project.org/")'
```

Example 2: Bayesian Statistics Environment

```
Bootstrap: docker
From: rocker/rstudio:latest

%post
    # Install JAGS for Bayesian modeling
    apt-get update
    apt-get install -y jags

    # Bayesian R packages
    R --slave -e 'install.packages(c(
        "rjags",
        "jagsUI",
        "R2jags",
        "coda",
        "MCMCvis",
        "nimble",
        "brms",
        "rstanarm"
    ), repos="https://cloud.r-project.org/")'
```

Example 3: Bioinformatics Environment

```
Bootstrap: docker
From: rocker/rstudio:latest

%post
    apt-get update
    apt-get install -y \
        libcurl4-openssl-dev \
        libssl-dev \
        libxml2-dev

    # Install Bioconductor
    R --slave -e '
        if (!requireNamespace("BiocManager", quietly = TRUE))
            install.packages("BiocManager")
        BiocManager::install(c(
            "GenomicRanges",
            "DESeq2",
            "edgeR",
            "limma",
            "GenomicFeatures"
        ))
    '
```

Building Multiple Images

You can build different images for different projects:

1. Create separate definition files:
   - `spatial_analysis.def`
   - `bayesian_stats.def`
   - `bioinformatics.def`
2. Modify `script.sh` to set output name:

   ```
   OUTPUT_IMAGE="${HOME}/apps/spatial_analysis.sif"
   ```

3. Submit separate build jobs for each image

Best Practices

File Access in Containers

Apptainer automatically mounts your home directory and current working directory.

Accessible locations:

- `$HOME` - Your home directory
- Current working directory
- `/data` - Data storage (if available)
- `/scratch` - Temporary storage (if available)

Example:

```
# These work automatically
cd /home/username/project
apptainer exec /data/apps/rstudio.sif Rscript analysis.R

# Reads/writes files in /home/username/project
```

Binding Additional Directories

If you need access to other directories:

```
apptainer exec --bind /scratch:/scratch /data/apps/rstudio.sif Rscript script.R
```

Or set environment variable:

```
export APPTAINER_BIND="/scratch:/scratch,/data:/data"
apptainer exec /data/apps/rstudio.sif Rscript script.R
```

Working Directory

Always change to your data directory before running:

```
cd /path/to/your/data
apptainer exec /data/apps/rstudio.sif Rscript analysis.R
```

Or in a SLURM job:

```
cd $SLURM_SUBMIT_DIR
apptainer exec /data/apps/rstudio.sif Rscript analysis.R
```

Reproducibility Tips

1. **Document your environment**:
   ```
   # Save package versions
   apptainer exec /data/apps/rstudio.sif R -e 'sessionInfo()' > environment.txt
   ```
2. **Use specific image versions**:
   ```
   # Instead of "latest", use specific tags in definition files
   From: rocker/rstudio:4.3.0
   ```
3. **Version control your definition files**:
   ```
   git add rstudio_custom.def
   git commit -m "Added spatial analysis packages"
   ```
4. **Share your images**:
   - Export definition files to collaborators
   - Or share the `.sif` image file directly

Performance Considerations

Use appropriate resources:

```
# For memory-intensive tasks
#SBATCH --mem=32G

# For parallel processing
#SBATCH -c 8
```

Parallel R inside container:

```
library(parallel)
ncores <- as.numeric(Sys.getenv("SLURM_CPUS_PER_TASK", "1"))
cl <- makeCluster(ncores)
# Your parallel code here
stopCluster(cl)
```

Advanced Usage

Running Specific R Versions

Pull a specific R version:

```
Bootstrap: docker
From: rocker/r-ver:4.2.0

%post
    R --slave -e 'install.packages("your-packages")'
```

Python + R Environment

Create an environment with both Python and R:

```
Bootstrap: docker
From: rocker/rstudio:latest

%post
    # Install Python
    apt-get update
    apt-get install -y python3-pip python3-dev

    # Install Python packages
    pip3 install numpy pandas scikit-learn

    # Install reticulate for R-Python interface
    R --slave -e 'install.packages("reticulate")'
```

Use in R:

```
library(reticulate)
np <- import("numpy")
arr <- np$array(c(1, 2, 3, 4))
```

Using RStudio Server from Container

For interactive analysis through web browser (configured in OpenOnDemand):

```
# This is handled automatically by OpenOnDemand RStudio app
apptainer exec /data/apps/rstudio.sif rserver --www-port=8787
```

Installing Packages at Runtime

For quick testing (not persistent):

```
apptainer exec /data/apps/rstudio.sif R -e 'install.packages("newpackage", lib="/tmp/Rlibs")'
apptainer exec /data/apps/rstudio.sif R -e '.libPaths("/tmp/Rlibs"); library(newpackage)'
```

Note: This is temporary. For permanent installation, build a custom image.

Using renv for Package Management

Create reproducible R environments:

```
# In your project directory
library(renv)
renv::init()

# Install packages
install.packages(c("dplyr", "ggplot2"))

# Save state
renv::snapshot()

# Share renv.lock with collaborators
```

Run with container:

```
apptainer exec /data/apps/rstudio.sif R -e 'renv::restore()'
apptainer exec /data/apps/rstudio.sif Rscript analysis.R
```

Troubleshooting

Container Not Found

Error: `FATAL: container not found`

Solution:

```
# Check if image exists
ls -lh /data/apps/rstudio.sif

# For custom images
ls -lh $HOME/apps/rstudio_custom.sif

# Use full path
apptainer exec /data/apps/rstudio.sif Rscript script.R
```

Permission Denied

Error: `Permission denied` when reading/writing files

Solution:

- Ensure files are in accessible locations (home directory, current directory)
- Check file permissions: `ls -la yourfile.csv`
- Make files readable: `chmod 644 yourfile.csv`

Package Not Found

Error: `Error: package 'xyz' not found`

Solution:

1. Check if package is installed:

   ```
   apptainer exec /data/apps/rstudio.sif R -e 'installed.packages()[,"Package"]' | grep xyz
   ```

2. If not installed, build custom image with the package

3. Or install to user library (temporary):

```
install.packages("xyz", lib="~/R/library")
.libPaths("~/R/library")
```

Out of Memory

Error: `cannot allocate vector of size...`

Solution:

- Request more memory in SLURM:

```
#SBATCH --mem=64G
```

- Use memory-efficient R packages (`data.table` instead of `dplyr`)
- Process data in chunks
- Use `gc()` to free memory

Build Fails

Error: Build fails with various errors

Common solutions:

1. **Network issues**: Try again later
2. **Disk space**: Clean up old files in `~/.apptainer/cache`
3. **xattr errors**: Already fixed in the template script
4. **Package dependencies**: Add required system packages to definition file

Slow Performance

Causes:

- Not enough resources allocated
- Working with data on slow storage
- Inefficient R code

Solutions:

- Increase CPU/memory in SLURM directives
- Copy data to `/scratch` for faster I/O

- Profile and optimize your R code
- Use vectorized operations in R

Cannot Access Files

Problem: Files in other directories not accessible

Solution:

Bind additional directories:

```
export APPTAINER_BIND="/project:/project,/data:/data"
apptainer exec /data/apps/rstudio.sif Rscript script.R
```

Quick Reference

Common Commands

```
# Run R script
apptainer exec /data/apps/rstudio.sif Rscript script.R

# Interactive R
apptainer exec /data/apps/rstudio.sif R

# Shell inside container
apptainer shell /data/apps/rstudio.sif

# Check R version
apptainer exec /data/apps/rstudio.sif R --version

# List installed packages
apptainer exec /data/apps/rstudio.sif R -e 'installed.packages()[,1]'

# Run with custom image
apptainer exec $HOME/apps/rstudio_custom.sif Rscript script.R

# Bind additional directories
```

```
apptainer exec --bind /scratch:/scratch /data/apps/rstudio.sif Rscript script.R
```

SLURM Job Template

```
#!/bin/bash
#SBATCH --job-name=r_analysis
#SBATCH --time=02:00:00
#SBATCH --partition=normal
#SBATCH -c 4
#SBATCH --mem=16G
#SBATCH --output=%x-%j.out
#SBATCH --error=%x-%j.err

# Load modules if needed
# module load apptainer

# Navigate to working directory
cd $SLURM_SUBMIT_DIR

# Run R script with container
apptainer exec /data/apps/rstudio.sif Rscript analysis.R

# Or with custom image
# apptainer exec $HOME/apps/rstudio_custom.sif Rscript analysis.R
```

Definition File Template

```
Bootstrap: docker
From: rocker/rstudio:latest

%labels
    Author Your Name
    Version 1.0

%post
    # System packages
    apt-get update
```

```
    apt-get install -y package1 package2

    # R packages
    R --slave -e 'install.packages(c(
        "package1",
        "package2"
    ), repos="https://cloud.r-project.org/")'

%environment
    export LC_ALL=C
```

Resources

Documentation

- Apptainer Documentation: https://apptainer.org/docs/
- Rocker Project: https://rocker-project.org/
- R Package Documentation: https://cran.r-project.org/

Getting Help

- **Check installed packages**: `apptainer exec /data/apps/rstudio.sif R -e 'installed.packages()'`
- **Test interactively**: Use `apptainer shell` or `apptainer exec R` for testing
- **Review job logs**: Check `.out` and `.err` files in your job directory
- **Contact support**: Provide job ID and error messages

Example Workflow

1. **Develop interactively**:

   ```
   srun --pty --mem=8G -c 4 --time=01:00:00 bash
   apptainer exec /data/apps/rstudio.sif R
   # Test your code
   ```

2. **Create job script**:

```
vim my_job.sh
# Add SLURM directives and commands
```

3. **Submit and monitor**:

```
sbatch my_job.sh
squeue -u $USER
tail -f my_job-<jobid>.out
```

4. **Review results**:

```
less my_job-<jobid>.out
# Analyze output files
```

Last Updated: 2025-12-03

Questions? Contact VoWa HPC support team.