Spider user documentation

Release 1.0

SURFsara support team

Jun 17, 2024

CONTENTS

Welcome to the SPIDER (Symbiotic Platform(s) for Interoperable Data Extraction and Redistribution) user guide! SPIDER is a feature-rich platform based at SURF and tailored for data processing and collaboration.

This guide aims to help new users getting started on SPIDER, while it also serves a useful reference for existing SPIDER members. Whether you seek information for SPIDER in general, or how to access and use the available features, or best practices for the efficient use of the resources, read on!

Table of contents:

CHAPTER

ONE

ABOUT

Tip: We welcome you to our new platform! In this page you will learn:

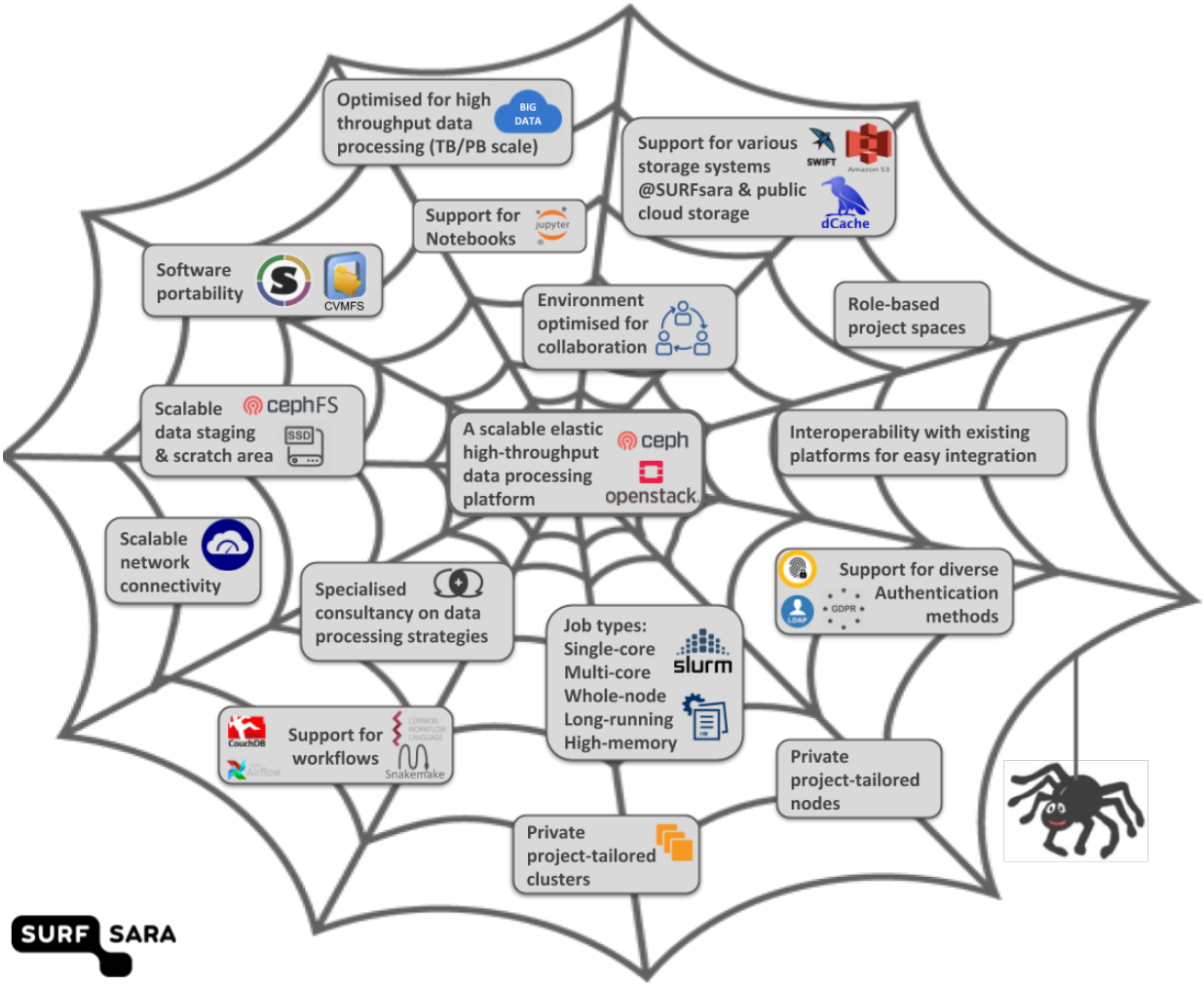
- what SPIDER is all about
- whether it is suitable for your research project
- what are the collaboration options
- how to obtain access and work within a SPIDER project

1.1 Spider at a glance

SPIDER is a versatile high-throughput data-processing platform aimed at processing large structured data sets. It runs on top of our in-house elastic Cloud. This allows for processing on SPIDER to scale from many terabytes to even petabytes. Utilizing many hundreds of cores simultaneously SPIDER can process these datasets in exceedingly short timespans. Superb network throughput ensures connectivity to external data storage systems.

Apart from scaling and capacity, SPIDER is aimed at interoperability with other platforms. This interoperability allows for a high degree of integration and customization into the user domain. SPIDER is further enhanced by specific features supporting collaboration, data (re)distribution, private resources or even private SPIDER instances.

Have a glance here of the main features offered by SPIDER:



1.2 Platform components

Spider is a feature-rich platform under continuous development. We keep adding new features to the platform in order to meet the needs of researcher projects working with massive datasets.

Spider is built on powerful infrastructure and designed with the following components:

- Batch processing cluster (based on Slurm) for generic data processing applications
- Batch partitions to enable Single-core, Multi-core, Whole-node, High-memory and Long-running jobs
- Large CephFs data staging area (POSIX-compliant filesystem) scales to PBs without loss of performance or stability
- Large and fast scratch area's (NVMe SSDs) on the worker nodes
- Fast network uplink (1200 Gbit/s) allowing for scalable parallel data transfers from other SURFsara based storage systems (e.g. dCache, SWIFT), or from external storage systems
- Role-based project spaces tailored for data-centric projects
- Scientific catalogs for cross-project collaboration
- Web access over HTTPS for public data distribution and sharing with external collaborators

- Singularity containers for software portability
- CVMFS/Softdrive support for software distribution
- Jupyter Notebooks
- Interactive jobs and direct visualization from within jobs
- Specific tooling for data-processing workflows
- Workflow management support
- Diverse authentication methods
- Private resources for special purposes (reservations, private nodes, private clusters)

1.3 Best suited cases

The best-suited cases for SPIDER are scientific projects with a requirement to process relatively large data sets. For example research projects suitable for SPIDER that deal with massive datasets are commonly in: Genomics, Proteomics, Earth observation, Astronomical observation, Climate modeling, Engineering or Physics experiments.

You would be eligible for SPIDER if your project reflects some of the following needs:

- Processing of large amount of data of many terabytes to petabytes in short time spans
- Processing of large amount of independent simulations and workflows
- Interactive processing with user-friendly interfaces for efficient data handling
- Industry standard interfaces and other interoperability features
- Co-working with your collaborators on the same project-based workspace
- Accessing external storage facilities with fast connectivity

Also SPIDER is a viable alternative for current and potential Grid users who are looking to use a more customizable system. It is a low-threshold platform, as opposed to highly complex Grid platforms that take many months of specialist development before they can start. Being built upon the exact same physical data-processing infrastructure and sharing the same scalable network connectivity as the Grid-based processing environments, SPIDER offers the same data-parallel processing capabilities as the most powerful Grid platforms.

Note though that while it's great for data-intensive applications, SPIDER is *not* really aimed at:

- HPC applications where operations per second are critical
- Processing of simulations that require multi-node execution
- Applications that cannot be ported onto Linux-based system

1.4 Collaboration

SPIDER is designed for Big Science which requires collaboration. SPIDER supports several ways to collaborate, either within your project, across projects, or to external sources.

1.4.1 Project space

Project spaces on SPIDER are shared workspaces given to team members that enable collaboration through sharing data, software and workflows. Within your project space there are four folders:

- Data: Housing source data from data managers
- Share: For sharing between project members
- Public: For sharing publicly through webviews
- Software: Scripts, libraries and tools

SPIDER enables collaboration for your project with granular access control to your project space through project roles, enabling collaboration for any team structure:

- *data manager* role: designated data dissemination manager; responsible for the management of project-owned data
- *software manager* role: designated software manager; responsible to install and maintain the project-owned software
- *normal user* role: scientific users who focus on their data analysis

1.4.2 Scientific catalog

Collaboration is also possible across different SPIDER projects. These are cases where different user groups work on projects with different scope and goals but need to (partly) share read-only data (such as observations or biobank data). SPIDER offers a place for multiple project teams to collaborate by sharing data sets or tools. This workspace is called *scientific catalog* and it is *not* offered by default to a project.

The scientific catalog data can be either *open* to everyone on the platform or *private* to selected SPIDER project groups.

The scientific catalog has only one (but important) role:

- *scientific catalog manager*: designated data dissemination SC (scientific catalog) manager; responsible for populating the catalog and deciding which SPIDER project groups have read access to that catalog.

1.4.3 Interoperability hotspot

In contrast to many of the processing platforms already available, typically offering an all-inclusive solution within the boundaries of the their environment, SPIDER is exactly the opposite. It aims to be a connecting platform in a world that has already a lot to offer in terms of storage systems, data distribution and collaboration frameworks, software management and portability systems, and pilot job and task management frameworks. The SPIDER platform can hook them all together as an interoperability hotspot to support a variety of data processing and data collaboration use cases.

For all external services supported, even services owned by the users themselves, SPIDER offers optimized configurations and practical guidelines how to connect to these services together into a practical processing environment tailored specifically to each project.

1.5 Project lifecycle

If you decided that SPIDER sounds suitable for your research project, then you can apply to obtain access and start your project or join an existing one.

1.5.1 Starting a project

For information about the granting routes on SPIDER please see our Proposals Page.

Before applying for a new project on SPIDER we suggest you to contact *our helpdesk* to discuss your project.

1.5.2 Extending a project

You can apply for a time or resource capacity extension for an existing project on SPIDER by requesting extra resources. Please see our Proposals Page or contact *our helpdesk*.

1.5.3 Joining an existing project

If you are interested to join an existing project please contact our *our helpdesk*. Upon your request we will verify with the project PI whether we can give you access to the project and what your project role would be.

1.5.4 Ending a project

Once your project ends, all the relevant data and accounts will be removed according to the Usage Agreement terms and conditions.

See also:

Still need help? Contact *our helpdesk*

CHAPTER

TWO

GETTING STARTED

Tip: This is a quickstart on the platform. In this page you will learn:

- How to login Spider
- Browsing your project space
- Submitting simple jobs

2.1 Setting up your account

Access to the cluster is provided via SSH (Secure Shell) Public key authentication only. For the highest security of your data and the platform, we do not allow username/password authentication.

To use this method you will need first to configure your SSH public key on a portal provided by SURF. Then you can connect and authenticate to Spider with your SSH keys without supplying your username or password at each visit.

Please note that it takes 1 hour from the time you receive your account credentials for your accounts to be able to run jobs on Spider

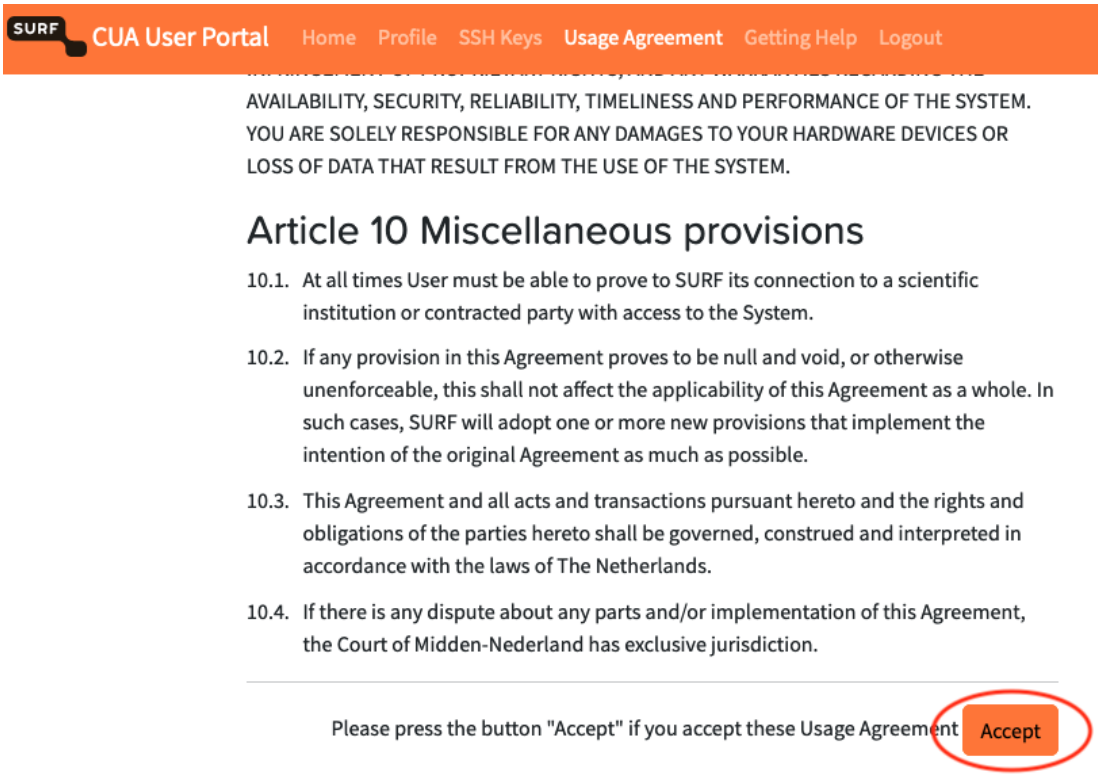
Please follow these steps to access Spider:

- **Step 1**: Login to the SURF CUA portal with your SURF user account

As a member of a Spider project you shall have received a SURF user account. Please use the username and password sent to you and login to the SURF CUA portal .

- **Step 2**: Accept the Usage Agreement in the portal

Once you login to the portal please agree to our usage terms and conditions to be able to gain access to Spider. You can perform this action on the "Usage Agreement" tab as shown in the image below. Please note that you will be denied access to Spider if you do not accept this agreement.



- **Step 3**: Upload your SSH public key to the portal

In order to access SPIDER you need to have a file on your local computer (say, your laptop) with a private SSH key, and you need to upload its matching public SSH key on the SURF CUA portal. Then, when you are going to connect to SPIDER from your laptop, the SSH private and public keys will be compared and, if they successfully relate to one-another, your connection will be established. Note that uploading your key to the portal is an *one time* task.

If you already have an SSH key-pair you can proceed to upload it.

NOTE: Please give your keys and account about 45 minutes to sync after uploading your public key in the portal

Else you have to generate a key-pair in your laptop or other machine that you use to connect to SPIDER. If you need help to generate an SSH key-pair, see:

2.1.1 Generate a private/public key pair

Open a terminal and type the following command:

```
ssh-keygen
```

While interacting with *ssh-keygen* in the terminal, you will see something similar to:

```
Generating public/private rsa key pair.
Enter file in which to save the key (~/.ssh/id_rsa): ### see note 1 below
Enter passphrase (empty for no passphrase): ### see note 2 below
Enter same passphrase again:
Your identification has been saved in ~/.ssh/id_rsa
Your public key has been saved in ~/.ssh/id_rsa.pub
The key fingerprint is:
40:1f:33:78:32:51:b5:c4:51:56:99:b6:6a:3d:18:8b user@computer.surfsara.nl
```

(continues on next page)

(continued from previous page)

```
The key s randomart image is:
+---[RSA 2048]----+
|                 |
|                ..|
|      .       .o.|
|       + +.o  o+ |
|      + S.B.o.+.o|
|       E =++oooo+|
|      . =oo= ++.=|
|       *....Bo = |
|      o.o..+o..  |
+----[SHA256]-----+
```

Some notes:

[1]. You can leave the output file name blank (simply hit enter) for the default file name, or type a variation of *~/.ssh/my_chosen_name*.

[2]. You can leave the passphrase field empty but we strongly recommend you to choose and remember an easy but long passphrase. If you forget the passphrase, you need to generate a new key pair and replace the old public keys you installed on remote hosts.

The ssh-agent (see below) will assist you so that you only need to type the passphrase once per session.

Using an SSH agent

SSH-agent is a service on your computer to remember your ssh passphrase during your local session (that is, until you log out). This way, you do not have to type in that loooong passphrase every time you need to unlock your private key.

- Add the key to the local ssh-agent

Type the following on your terminal:

```
ssh-add ~/.ssh/id_rsa ### or the file name you provided to ssh-keygen
```

- While interacting with *ssh-add* in the terminal, you will see something similar to:

```
Enter passphrase for ~/.ssh/id_rsa: ### type it in
Identity added: ~/.ssh/id_rsa
```

If this fails because "Could not open a connection to your authentication agent", you need to start the ssh-agent daemon before you run *ssh-add*:

```
eval `ssh-agent -s`
```

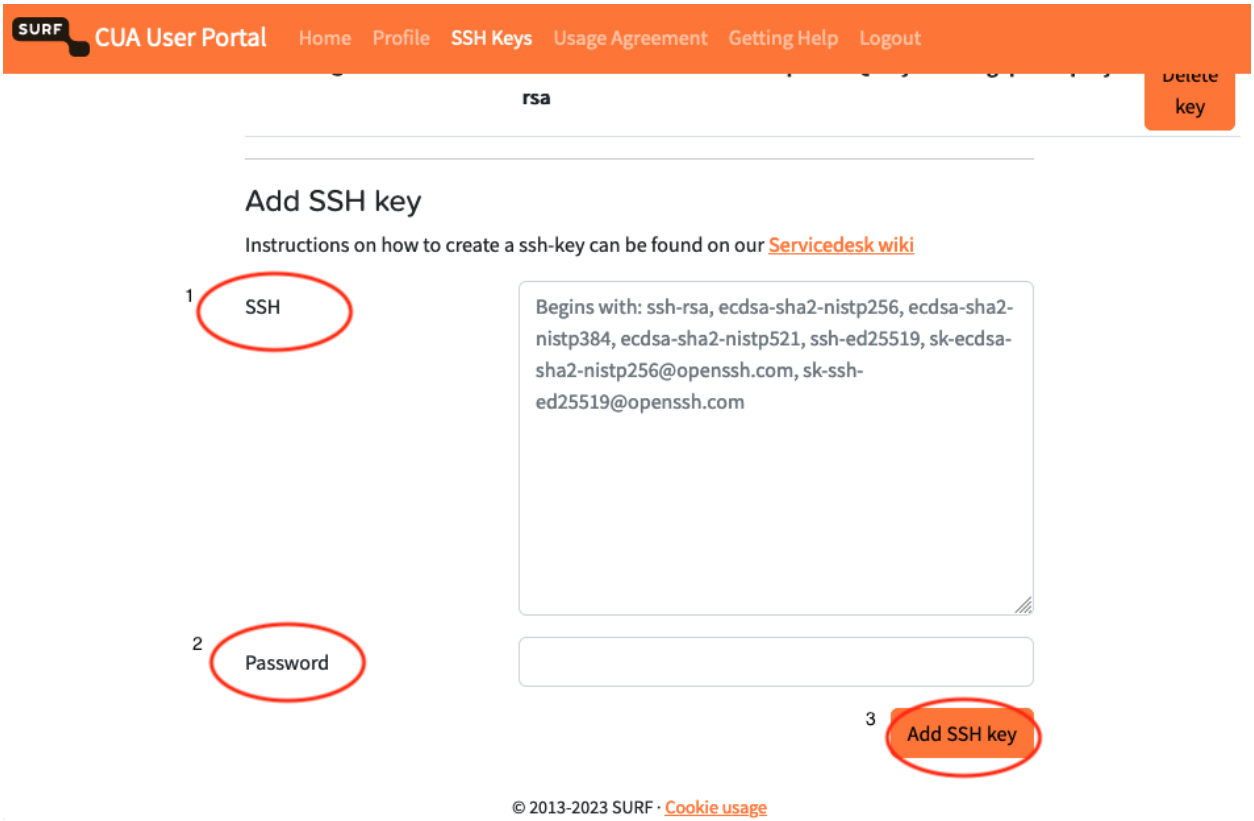
- To list the keys loaded in the *ssh-agent* type the following on your terminal:

```
ssh-add -l
```

The output will be one line for each key stored in the ssh-agent, similar to:

```
2048 SHA256:ajAxT3T3ZKl2rALBGGmMqufU0n6XAU15lj+fObZEvrI ~/.ssh/id_rsa (RSA)
```

Once you have generated your SSH key-pair, upload your public key to our SURF CUA portal. Click on the tab "Public ssh keys" on the left pane of the portal and add your public key by copying the contents of your public key file (e.g. `cat ~/.ssh/id_rsa.pub`) as shown below:



Field [1] SSH key: here you paste your public key

Field [2] Password: here you enter your password for your account

Field [3] Add sshkey: press the key for the changes to take effect

From now on you can login to Spider with your SSH keys from your laptop (or other computer where your SSH key was generated/transferred). See next, *how to login*.

2.2 Logging in

The login node is your entry and access point to Spider. From this node you can submit jobs, transfer data and prototype your application. It has a software environment very similar to the worker nodes where your submitted jobs will run.

In order to login to Spider you must have already uploaded your SSH public key on the SURF CUA portal (see *Setting up your account*)

If you already completed this step once, you are ready to login!

- Login to Spider via a terminal with the following command:

```
ssh [USERNAME]@[Spider UI HOSTNAME]
```

- For example, the user *homer* will login as:

```
ssh homer@spider.surfsara.nl
#[homer@htp-ui ~]$  # this is the first prompt upon login
```

Congrats! You've just logged in to Spider.

Note: In case that you have multiple keys in your `.ssh/` folder, you would need to specify the key that matches the .pub file you uploaded on the SURF CUA portal, i.e. `ssh -i ~/.ssh/surfsarakey homer@spider.surfsara.nl`

Note: The first time you login to Spider, you need to accept Spider's SSH key fingerprint. Public key fingerprints can be used to validate a connection to a remote server. Spider's public key fingerprint is: `SHA256:HO8Cz3Fns+DoiK+VFlILbTGYkAOy5i/izzFYc005z+s (ECDSA)`

2.3 Getting around

As a user on Spider you are a member of a project, and each project member gets access to the following directories:

2.3.1 Home directory

- `/home/$USER`: each project member in a project has her/his personal home space. Only the account owner can read and write data in this directory

2.3.2 Project spaces directories

Project space is a POSIX storage place allocated to each Spider project. It includes the following shares:

- `/project/[PROJECTNAME]/Data`: any project-specific data. Any member of the project can read data in this directory, but only the data manager(s) can write data
- `/project/[PROJECTNAME]/Software`: any project-specific software. Any member of the project can read/execute software in this directory, but only the software manager(s) can install software
- `/project/[PROJECTNAME]/Share`: any data to be shared among the project members. Any member of the project can read and write data in this directory
- `/project/[PROJECTNAME]/Public`: Any member of the project can write in this directory. Any data stored here will be read-only by all users on Spider and exposed publicly via http (see *how*)

The summary table below gives a quick overview of your project space permissions ('r'-read/'w'-write/'x'-execute):

Directories vs. Access Roles	/project/[PROJECTNAME]/Data	/project/[PROJECTNAME]/Software	/project/[PROJECTNAME]/Share	/project/[PROJECTNAME]/Public
Project Data manager(s)	rwx	r-x	rwx	rwx
Project Software manager(s)	r-x	rwx	rwx	rwx
Project normal user(s)	r-x	r-x	rwx	rwx
Other Spider project user	—	—	—	r–
Outside Spider user	—	—	—	r– (via the *web views*)

2.4 Submitting a job

On SPIDER you will use Slurm to schedule, manage and execute your jobs. Slurm (short for Simple Linux Utility for Resource Management) is an open source, fault-tolerant, highly scalable, cluster management and job scheduling system for Linux clusters. Further information can be found at the Slurm documentation page. You can see the currently installed version of Slurm by typing `sinfo --version` on the command line.

Let's run our first job on SPIDER. Download the sample job script to your `home` account on the SPIDER login and inspect the file before submitting it to the cluster:

```
wget https://raw.githubusercontent.com/sara-nl/spiderdocs/master/source/scripts/welcome-
↪to-spider.sh
chmod u+x welcome-to-spider.sh
```

- Submit your job to the cluster:

```
sbatch welcome-to-spider.sh
#Submitted batch job [JOBID]
```

- Check the status of your submitted and not completed job(s):

```
squeue --job [JOBID] # status of [JOBID]
squeue -u $USER # status of all my jobs
scontrol show jobid [JobID] # detailed info of [JOBID]
```

- Check your job output:

```
cat slurm-[JOBID].out
```

- Once your job has completed, you can get job statistics and accounting:

```
sacct -j [JOBID] --format=JobID,JobName,AveCPU,MaxRSS,Elapsed
```

More examples of how to use SPIDER Slurm can be found in section *Compute on Spider* and more generic info can be found at the Slurm documentation page.

2.5 Common commands

Slurm has many commands with many options, here you have a list with the most common ones. For more information please checkout the Slurm documentation page.

Command	What it does
`sinfo`	displays the nodes information
`sbatch`	submits a job to the batch system
`squeue`	displays the state of all submitted jobs
`scancel`	cancels a submitted job
`scontrol`	shows detailed job, node information (useful for debugging), e.g. `scontrol show job [jobID]`, `scontrol show node [worker node]`
`sacct`	shows detailed accounting information for jobs

See also:

Still need help? Contact *our helpdesk*

CHAPTER

THREE

STORAGE ON SPIDER

Tip: Spider is meant for processing of big data, thus it supports several storage backends. In this page you will learn:

- which internal and external storage systems we support
- best practices to manage your data

3.1 Internal storage

The available filesystems on SPIDER are CephFS and SSDs. Home and project spaces are mounted on CephFS, while the batch worker nodes have large scratch areas on local SSD.

CephFS is a distibuted parallel filesystem which stores files as objects and it is suitable for workloads that deal with comparably large files. Please note that conda/pip packages handling lots of small files can slow down the system response. For high I/O performance, we recommend the local scratch of the worker nodes on SSDs.

3.1.1 Transfers within Spider

To transfer data between directories located within SPIDER we advise you to use the unix commands `cp` and `rsync`. Other options may be available, but these are currently not supported by us.

Help on these commands can be found by (i) typing `man cp` or `man rsync` on the command line after logging into the system, or (ii) by contacting *our helpdesk*.

Warning: When copying data on the local filesystem and accross different project space folders we suggest you to copy your data and then remove the source files instead of moving the data. This will ensure that the new copy inherits the permissions of the destination project folder.

3.1.2 Spider filesystems

Using Home

SPIDER provides to each user with a globally mounted home directory that is listed as `/home/[USERNAME]`. This directory is accessible from all nodes. This is also the directory that you as a user will find yourself in *upon first login* into this system. The data stored in the home folder will remain available for the duration of your project.

Using project spaces

Similarly to home folders Spider's project spaces are also available on all worked nodes, the following paths are available on your SPIDER UI:

- `/project/[Project Name]/Data`
- `/project/[Project Name]/Public`
- `/project/[Project Name]/Share`
- `/project/[Project Name]/Software`

This allows you to easily access your software, data and output from the worker nodes from the project spaces. See below for an example of a command that could be executed from a script on a worker node:

```
sh /project/[Project Name]/Software/[script].sh /project/[Project Name]/Data/[input␣
↪file(s)] /home/[USER]/[output]
```

Using scientific catalogs

Scientific catalogs allow for you to share software and data repositories accross projects. For example if you would like to share a large biobank of data with other research projects you could request access to upload to the scientific catalogue. Then it will be accessible from the worker nodes similarly to the `/home` and `/project` folders.

To request access to add a shared catalogue please reachout to *our helpdesk*.

Using scratch

Each of SPIDER worker nodes has a large scratch area on local SSD. These scratch directories enable particularly efficient data I/O for large data processing pipelines that can be split up into many parallel independent jobs.

Please note that you should only use the scratch space to temporarily store and process data for the duration of the submitted job. The scratch space is cleaned regularly in an automatic fashion and hence can not used for long term storage.

For more information about how to use scratch during your compute jobs, please refer to using local scratch.

> **Warning:** The local directories in the Spider, such as `/tmp` and `/var/tmp`, should not be used by the users. They are slow and small to be used for any tasks. Furthermore, the local directories in either login nodes or worker nodes are needed by the operating system itself and is cleaned up sometimes, for example when the system is rebooted.

In addition, if a user fill up `/tmp` on a node, the operating system will experience serious problems due to lack of space. Eventually the jobs submitted by you and other users who share the same node will also experience issues. It is strongly advised to calculate the temporary space needed by the software in advance, and request enough cores for your jobs to avoid filling up the `/tmp` of a node.

Querying internal storage usage

As a mounted filesystem spider storage can be queried with local linux commands, but for optimal performance we recommend querying some preconfigured *fattr* tags instead of *du* commands that slow down the system.

The total usage of local spider storage is the total usage of projct home folders and project space together.

Please note that this will show your current usage, not the max, or average for the month.

Example

```
# Project folder
getfattr -n ceph.dir.rbytes --absolute-names /project/[PROJECT]/

# Home folder
getfattr -n ceph.dir.rbytes --absolute-names /home/[PROJECT]-[USER]
```

3.2 External storage

3.2.1 Transfers from own system

If you are logged in as a user on SPIDER then we support `scp`, `rsync`, `curl` or `wget` to transfer data between SPIDER and your own Unix-based system. Other options may be available, but these are currently not supported by us.

- Example of transferring data from SPIDER to your own system:

```
# Using scp
scp [spider-username]@spider.surfsara.nl:[path-to-your-spider-folder]/transferdata.tar.
↪gz [path-to-your-local-folder]/

# Using rsync
rsync -a -W [spider-username]@spider.surfsara.nl:[path-to-your-spider-folder]/
↪transferdata.tar.gz [path-to-your-local-folder]/
```

- Example of transferring data from your own system to SPIDER:

```
# Using scp
scp [path-to-your-local-folder]/transferdata.tar.gz [spider-username]@spider.surfsara.
↪nl:[path-to-your-spider-folder]/

# Using rsync
rsync -a -W [path-to-your-local-folder]/transferdata.tar.gz [spider-username]@spider.
↪surfsara.nl:[path-to-your-spider-folder]/
```

3.2.2 SURF grid storage / dCache

SURF grid storage / dCache is our large scalable storage system for quickly processing huge volumes of data. The system runs on dCache software, that is designed for managing scientific data. You can use grid storage for disk or tape, or address both types of storage under a single virtual filesystem tree. Our grid storage / dCache service is a remote storage with an extremely fast network link to Spider. You may use the storage if your data does not fit within the storage allocation on SPIDER project space or if your application is I/O intensive.

There are several protocols and storage clients to interact with grid storage. On SPIDER we support two main methods to use grid storage, ADA and Grid interfaces:

ADA interface

Our ADA (Advanced dCache API) interface is based on the dCache API and the webdav protocol to access and process your data on dCache from any platform and with various authentication methods.

rclone is a webdav client that supports by default 4 parallel streams of data, and is installed on the spider platform.

macaroons are a token based authentication method supported by dCache. Macaroons can be used to give access to dCache data in a very granular way. This enables data managers autonomously share their data in dCache without having to reach out to SURFsara to request access.

A quick start up guide for ADA is captured in the video below:

Browser view

dCache storage can be viewed both through the Ada tools or through your browser using the web client, this is just one additional way you can explore the storage space.

As a Data manager you have direct credentials on dCache and it is possible to access the browser view using you Spider credentials [project-username] in the following link:

https://webdav-secure.grid.surfsara.nl/pnfs/grid.sara.nl/data/{[}PROJECT{]}/

Note: You may be asked for a browser certificate, just select cancel and you will be asked for your credentials

Using ADA

ADA is a wrapper of tools created by SURFsara to simplify your interactions with dCache. Rclone can support uploading and downloading data but other operations such as listing or deleting files and directories can be performed directly on the dCache API. ADA wraps all of this functionality into one clean package saving you the hassle of having to download and troubleshoot multiple packages and dependencies. ADA is installed on Spider.

This section provides examples and the steps to start using ADA to interact with your dCache storage.

Create a macaroon

- Requirements: credential to dCache
  - username/pwd or
  - x509 proxy
- Spider role: Data manager
- Action: Create a macaroon
- Output: rclone tokenfile *[PROJECT_tokenfile].conf*. You can share this file with any member in the project in next step.
- Description: the DM creates a macaroon for a shared directory (including the sub-directories & files). In the next step he will share the macaroon with the project team in a non-public space, either user's home directories, or the 'shared' or 'data' project space directories.
- Example:

```
get-macaroon \
    --url https://webdav.grid.surfsara.nl:2880/pnfs/grid.sara.nl/data/[PROJECT] \
    --duration P7D \
    --chroot \
    --user [PROJECT]-[USER] \
    --permissions DOWNLOAD,UPLOAD,DELETE,MANAGE,LIST,READ_METADATA,UPDATE_METADATA \
    --ip [IP RANGE] \
    --output rclone [PROJECT_tokenfile]
```

These permissions can be given comma separated upon creation of the macaroon:

Permission	Function
DOWNLOAD	Read a file
UPLOAD	Write a file
DELETE	Delete a file or directory
MANAGE	Rename or move a file or directory
LIST	List objects in a directory
READ_METADATA	Read file status
UPDATE_METADATA	Stage/unstage a file, change QoS

Share macaroons

The config file generated in the step above can be shared with project members and collaborators for them to access their data. The holder of this config file can operate on the dCache project data directly and thus, the config file should be shared with the project team in a non-public space, for example user's home directories, or the 'Shared' or 'Data' project space directories on Spider.

- Requirements: the rclone tokenfile *[PROJECT_tokenfile].conf*
- Spider role: Data manager
- Actions: Share [PROJECT_tokenfile].conf in a project space that can be read by other project users
- Output: the tokenfile *tokenfile.conf* is stored in a shared space
- Example:

```
cp [PROJECT_tokenfile].conf /project/[PROJECT]/Data
```

Inspect the macaroon

- Requirements: the rclone tokenfile *[PROJECT_tokenfile].conf*
- Spider role: Normal user
- Actions: View macaroon
- Output: the list activities and directories that you can use on dCache
- Example:

```
# Your macaroon is the value of 'bearer_token'
$ cat [PROJECT_tokenfile].conf
[tokenfile]
type = webdav
bearer_token = MDAxY2xvY2F0aWXXXXXXXXXXXXXXXX
url = https://webdav.grid.surfsara.nl:2880/
vendor = other
user =
password =

#View the macaroon details
$ view-macaroon [PROJECT_tokenfile].conf
location Optional.empty
identifier NDFXzXXX
cid iid:03FXXX//
cid id:39147;35932,30013;[Data Manager Name]
cid before:2020-02-05T11:01:11.577Z
cid home:/[Project folder]
cid root:/[Project folder]
cid activity:DOWNLOAD,UPLOAD,MANAGE,LIST
signature fefef25a4973e59b10ad464054dXXXXXXX
```

Use the macaroon

This section describes how to work with your files.

- Requirements: the rclone tokenfile *[PROJECT_tokenfile].conf*
- Spider role: Normal user

Tip: If you want to use an environment variable to set the token file, rather than having to pass it on the command line every time then you can do: `$export ada_tokenfile=/path-to-mytoken/[PROJECT_tokenfile].conf` and then you can omit the option '–tokenfile' from all of the ada commands

---

Tip: You can get extra information about the submitted command and the rest call details by using the *–debug* option in your ada command.

Check your access to the system

–whoami

- Action: request authentication details
- Output: information about the token owner and permissions
- Example:

```
ada --tokenfile [PROJECT_tokenfile].conf --whoami
```

```
{
"status": "AUTHENTICATED",
"uid": 515XX,
"gids": [
    511XX
],
"username": "[Data Manager name]",
"rootDirectory": "/pnfs/grid.sara.nl/data/[Project]/disk",
"homeDirectory": "/"
}
```

Listing files

–list <directory>

–longlist <file|directory>

–longlist –from-file <file-list>

- Action: List files or directories
- Output: List or long list of the files from the directory that the macaroon allows permission
- Example:

```
ada --tokenfile [PROJECT_tokenfile].conf --longlist /[DIRECTORY]
```

Get file or directory details

–stat <file|directory>

- Action: Show all details of a file or directory
- Output: metadata information
- Example:

```
ada --tokenfile [PROJECT_tokenfile].conf --stat /[FILE or DIRECTORY]
```

Create a directory on dCache

–mkdir <directory>

- Action: Create directories
- Output: New directory created
- Example:

```
ada --tokenfile [PROJECT_tokenfile].conf --mkdir /[DIRECTORY]
```

Moving or renaming files

–mv <file|directory> <destination>

- Action: Move file or directory. This can be used as an option also to rename a directory if the move is done in the same directory. Specify the full path and name to the source and target directory
- Output: File or Directory moved to a different dCache location or renamed
- Example:

```
ada --tokenfile [PROJECT_tokenfile].conf --mv /[SOURCE] /[DESTINATION]
```

Recursively remove folders

–delete <file|directory> [–recursive [–force]]

- Action: Delete files or directories
- Output: File or Directory is deleted
- Recursive deletion: To recursively delete a directory and ALL of its contents, add –recursive. You will need to confirm deletion of each subdir, unless you add –force.
- Alternative: *rclone purge*
- Example:

```
ada --tokenfile [PROJECT_tokenfile].conf --delete /[FILE or DIRECTORY]
ada --tokenfile [PROJECT_tokenfile].conf --delete /[FILE or DIRECTORY] --recursive
ada --tokenfile [PROJECT_tokenfile].conf --delete /[DIRECTORY] --recursive --force
# alternative
$ rclone --config=[PROJECT_tokenfile].conf purge PROJECT_tokenfile]:/disk/rec-delete/
```

Checksum

–checksum <file>

–checksum <directory>

–checksum –from-file <file-list>

- Action: Get the checksum of a files or files inside a directory or list of files
- Output: Show MD5/Adler32 checksums for files

- Example:

```
ada --tokenfile [PROJECT_tokenfile].conf --checksum /[FILE or DIRECTORY]
# create a filelist and get checksums for files in it
ada --tokenfile [PROJECT_tokenfile].conf --list /disk/mydir > files-to-checksum
sed -i -e 's/^/\/disk\/mydir\//' files-to-checksum
ada --tokenfile [PROJECT_tokenfile].conf --checksum --from-file files-to-checksum
#/disk/file1  ADLER32=80690001
#/disk/file2  ADLER32=80690001
#/disk/file3  ADLER32=80690001
```

View your usage

- Action: get your storage usage with Rclone
- Example:

```
rclone --config=[PROJECT_tokenfile].conf size [PROJECT_tokenfile]:/
```

Staging

The dCache storage at SURFsara consists of magnetic tape storage and hard disk storage. If your quota allocation includes tape storage, then the data stored on magnetic tape has to be copied to a hard drive before it can be used. This action is called Staging files or 'bringing a file online'.

Your macaroon needs to be created with UPDATE_METADATA permissions to allow for staging operations.

–stage <file>

–stage <directory>

–stage –from-file <file-list>

- Action: Stage a file from tape or files in directory or a list of files (restore, bring it online)
- Output: the file or list of files comes online on disk
- Example:

```
#list files to get the status
ada --tokenfile [PROJECT_tokenfile].conf --longlist /[PROJECT_tape_dir]
#file1  1186443  2020-02-13 16:27 UTC  tape  NEARLINE
#file2  1635     2018-10-24 15:34 UTC  tape  NEARLINE

#stage a single file
ada --tokenfile [PROJECT_tokenfile].conf --stage /[PROJECT_tape_dir]/file1

#stage a list of files
ada --tokenfile [PROJECT_tokenfile].conf --stage --from-file files-to-unstage
```

Unstaging

Your macaroon needs to be created with UPDATE_METADATA permissions to allow for unstaging operations.

–unstage <file>

–unstage <directory>

–unstage –from-file <file-list>

- Action: Unstage/Release a file from tape or files in directory or a list of files
- Output: the file or list of files is unstaged and may be removed for the disk any time so dCache may purge its online replica.

```
#unstage a single file
ada --tokenfile [PROJECT_tokenfile].conf --unstage /[PROJECT_tape_dir]/file1

#unstage a list of files
ada --tokenfile [PROJECT_tokenfile].conf  --list /tape > files-to-unstage
sed -i -e 's/^/\/tape\//' files-to-unstage
ada --tokenfile [PROJECT_tokenfile].conf  --unstage --from-file files-to-unstage
```

Transfer Data

In order to transfer files from/to dCache we use the same [PROJECT_tokenfile].conf and the rclone client to trigger webdav transfers as shown below.

Copy data from dCache

```
rclone --config=[PROJECT_tokenfile].conf copy [PROJECT_tokenfile]:/[SOURCE] ./
↪[DESTINATION] -P
```

Example, copy an existing test folder to Spider:

```
rclone --config=[PROJECT_tokenfile].conf copy [PROJECT_tokenfile]:/tests/ ./tests/ -P
```

Write data to dCache

```
rclone --config=[PROJECT_tokenfile].conf copy ./[SOURCE]/ [PROJECT_
↪tokenfile]:[DESTINATION] -P
```

Notes on data transfers:

- The rclone `copy` mode will just copy new/changed files. The rclone `sync` (one way) mode will create a directory identical to the source so be careful because this can cause data loss. We suggest you to test first with the `-dry-run` flag to see exactly what would be copied and deleted.
- You can increase the number of parallel transfers with the `--transfers [Number]` option.
- When copying a small number of files into a large destination you can add the `--no-traverse option` in the rclone copy command for controlling whether rclone lists the destination directory or not. This can speed transfers up greatly.

- If you are certain that none of the destination files exists you can add the `--no-check-dest option` in the rclone copy command to speed up the transfers.
- For very large files it is important to set the `-timeout` option high enough. As a rule of thumb, set it to 10 minutes for every GB of the biggest file in a collection. This may look ridiculously large, but it provides a safe margin to avoid problems with timeout issues
- Using `--multi-thread-streams 1` increases the performance for large files copied to dCache.

```
#example command to upload a big file
rclone --timeout=240m  --multi-thread-streams 1 --config=[PROJECT_tokenfile].conf copy ./
↪[SOURCE]/ [PROJECT_tokenfile]:[DESTINATION] -P
```

Event-driven processing

Events are useful when you want to know something you're interested in happened in your dCache project space, such as when new data is available or when files are staged from tape, etc.

- Subscribe to changes in a given directory:

```
ada --tokenfile [PROJECT_tokenfile].conf --events changes-in-dir /[PROJECT_directory] --
↪recursive
```

- Check the available channels listening to events:

```
ada --tokenfile [PROJECT_tokenfile].conf --channels
```

- Report staging events

When you start this channel, all files in the scope will be listed, including their locality and QoS. This allows your event handler to take actions, like starting jobs to process the files that are online. When all files have been listed, the command will keep listening and reporting all locality and QoS changes.

```
ada --tokenfile [PROJECT_tokenfile].conf --report-staged staging-in-tape-dir /[PROJECT_
↪directory] --recursive
```

Authentication

In this page we gave an extended example on using ada with macaroons authentication. Ada can be used with multiple authentication options.

Authentication	ADA commands	When to use
Macaroon	`ada --tokenfile <filename>`	You don't have direct access on dCache but you have a token from the project data manager that allows you certain permissions on the data
Username/passwo	`ada --netrc [filename]`	You have direct usr/pwd access credentials on dCache
X509 Certificate	`ada --proxy [filename]`	You have direct VO membership access on dCache

Here is an example of a .netrc file that you can create in your home to use username/password authentication:

```
$ cat ~/.netrc:
machine webdav.grid.surfsara.nl
login [your-ui-username]
password [your-ui-password]
machine dcacheview.grid.surfsara.nl
login [your-ui-username]
password [your-ui-password]
```

Run ADA anywhere

In this page we gave an extended example on using ada on the Spider platform. Ada is portable and can be used on any platform. On the SURFsara UIs ADA is already on board. If you want to interact with the dCache API and transfer files from your own machine then you need to install the following prerequisites:

- `jq`: the only dependency for executing ada commands
- `rclone`: the client to perform transfers (MacOS: brew install rclone)

As a Data manager if you wish to create macaroons from any platform, e.g. your local machine, then you need to install the following *get-macaroon* and *view-macaroon* scripts:

- `wget https://raw.githubusercontent.com/sara-nl/GridScripts/master/get-macaroon`
- `wget https://raw.githubusercontent.com/sara-nl/GridScripts/master/view-macaroon`
- And their dependencies: `pymacaroons, python3-html2text`

Ada configuration files

The user specific configuration files are written in ~/.ada/

1) The URL to query the API is stored in */etc/ada.conf* (system default) or *~/.ada/ada.conf* (user specific, optional)
2) The bearer tokens information based on a tokenfile is stored in *~/.ada/headers/*. The authorization_header is created for security to prevent from reading the token as argument and be displayed in 'ps' info. This way the token is read from a hidden file in the user home dir
3) The Events information such as the last eventID is stored in *~/.ada/channels/*

Our ADA (Advanced dCache API) interface is based on the dCache API and the webdav protocol to access and process your data on dCache from any platform and with various authentication methods.

Grid interface

The Grid interface is recommended in cases that your project data and/or processing is tied to the Grid authentication and authorisation. To use the supported Grid clients on SPIDER you need to have an X509 Grid certificate installed into your local directory and be a part of a Virtual Organisation (VO). Please refer to our Grid documentation page for instructions on how to get a certificate and join a (VO).

Grid authentication

To be able to interact with dCache using a storage client, you need to create a proxy. A proxy is a short-lived certificate/private key combination which is used to perform actions on your behalf without using passwords.

Create a proxy with the following command:

```
voms-proxy-init --voms [YOUR_VO]
```

On Spider your proxy is generated by default in your `$HOME/.proxy` location such that it is accessible from anywhere on Spider. You can check this with `echo X509_USER_PROXY`.

The default lifetime of a proxy is 12h. If your application runs longer then you can create a proxy that is valid up to 7 days with the following command:

```
voms-proxy-init --voms [YOUR_VO] --valid 168:00
```

Grid clients

There are many Grid clients to interact with dCache. On Spider we support `gfal`.

In the examples below, a user who is a member of the VO e.g., lsgrid, has the certificate installed on to the Spider login node and will copy data from dCache to/from your home directory on Spider.

gfal client

Note: The `gfal` commands fail on our centos 8 worker nodes due to the security setup. The workaround is to run the commands below with a centos 7 container, i.e. `singularity run -B /etc/grid-security/certificates /cvmfs/atlas.cern.ch/repo/containers/fs/singularity/x86_64-centos7 gfal-ls -l [gsiftp://path-to-file]`

Please note that you need a valid proxy to run the following commands.

- Listing directories on dCache:

```
gfal-ls -l gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/
```

- Create directory on dCache:

```
gfal-mkdir gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/path-to-your-
↪data/newdir/
```

- Copy file from dCache to Spider:

```
gfal-copy \
    gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/path-to-your-data/
↪your-data.tar \
    file:///`pwd`/your-data.tar
```

- Copy file from Spider to dCache:

```
gfal-copy \
    file:///$HOME/your-data.tar \
    gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/path-to-your-data/
↪your-data.tar
```

- Remove a file from dCache:

```
gfal-rm gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/path-to-your-
↪data/your-data.tar
```

- Remove a whole (non empty) directory from dCache:

```
gfal-rm -r gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/path-to-your-
↪data/
```

Recursive transfer of files (transferring a directory) is not supported with the gfal-copy command.

Tip: Need more examples? See gfal Grid documentation

Grid data processing

Below we show an example for I/O intensive applications. In this example you submit a job on SPIDER that performs the following steps:

- Creates a runtime directory on local `scratch` (or `$TMPDIR`)
- Retrieves the input data from dCache
- Runs the analysis
- Stores the output data on dCache

Here is a job script template for local `scratch` usage;

```
#!/bin/bash
#SBATCH -N 1      #request 1 node
#SBATCH -c 1      #request 1 core and 8GB RAM
#SBATCH -t 5:00   #request 5 minutes jobs slot

mkdir "$TMPDIR"/myanalysis
cd "$TMPDIR"/myanalysis
gfal-copy gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/path-to-your-data/
↪your-data.tar file:///`pwd`/your-data.tar

# = Run you analysis here =

#when done, copy the output to dCache
tar cf output.tar output/
gfal-copy file:///`pwd`/output.tar gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/
↪data/path-to-your-data/output.tar
echo "SUCCESS"
exit 0
```

Please note that in the above example, it is assumed that the data is present on the disk storage on dCache. If the data is stored on Tape, it may need to be copied to disk first (called as staging).

Sharing data with macaroons

Macaroons are bearer tokens that authorize someone to access certain directories or files. With this technique, you can share (some of) your data with anyone else. The other person does not need to have a user account or a certificate; only a WebDAV client that supports bearer tokens. Clients that support this are Curl, Rclone and (read only) ordinary browsers such as Firefox.

You can get a Macaroon with X509 authentication. Please note, that port 2883 is used for this. The lifetime of your proxy does not limit the lifetime of the macaroon.

```
[xxxx@ui-01 ~]$ voms-proxy-init -voms lsgrid:/lsgrid
Enter GRID pass phrase for this identity:
....
Your proxy is valid until Fri Jul 06 01:37:31 CEST 2018

[xxxx@ui-01 ~]$ get-macaroon --url https://webdav.grid.surfsara.nl:2883/pnfs/grid.sara.
↪nl/data/lsgrid/homer/Shared --proxy --chroot --duration PT1H --permissions DOWNLOAD,
↪LIST
https://webdav.grid.surfsara.nl:2883/?
↪authz=MDAxY2xvY2F0aW9uIE9wdGlvbmFsLmVtcHR5CjAwMThpZGVudGlmaWVyIGNOMDBnRHRSCjAwMzZjaWQgaWQ6MzY0OTQ7MzE4
↪zh2OGkwo
```

For more information, see the dCache User Guide.

Our Grid interface is based on the Grid computing technology and the gridftp protocol to access and process your data on dCache from Grid compliant platforms and with X509 certificate authentication.

3.2.3 SURFsara Central archive

For long-term preservation of precious data SURFsara offers the Data Archive. Data ingested into the Data Archive is kept in two different tape libraries at two different locations in The Netherlands. The Data Archive is connected to all compute infrastructures, including Spider.

Access on Data Archive is *not* provided by default to the Spider projects. To request for Data Archive access, please contact *our helpdesk*.

If you already have access on Data Archive, then you can use it directly from Spider by using `scp` and `rsync` to transfer data between Spider and Data Archive:

- Transfer data from Spider to Data Archive:

```
# Using scp
scp /home/[USERNAME]/transferdata.tar.gz [ARCHIVE_USERNAME]@archive.surfsara.nl:/home/
↪[ARCHIVE_USERNAME]/

# Using rsync
rsync -a -W /home/[USERNAME]/transferdata.tar.gz [ARCHIVE_USERNAME]@archive.surfsara.nl:/
↪home/[ARCHIVE_USERNAME]/
```

- Transfer data from Data Archive to Spider:

```
# Using scp
scp [ARCHIVE_USERNAME]@archive.surfsara.nl:/home/[ARCHIVE_USERNAME]/transferdata.tar.gz /
↪home/[USERNAME]/
```

(continues on next page)

(continued from previous page)

```
# Using rsync
rsync -a -W [ARCHIVE_USERNAME]@archive.surfsara.nl:/home/[ARCHIVE_USERNAME]/transferdata.
↪tar.gz /home/[USERNAME]/
```

In case that the file to be retrieved from Data Archive to Spider is not directly available on disk then the scp/rsync command will hang until the file is moved from tape to disk. Data Archive users can query the state of their files by logging into the Data Archive user interface and performing a `dmls -l` on the files of interest. Here the state of the file is either on disk (REG) or on tape (OFL). The Data Archive user interface is accessible via `ssh` from anywhere for users that have a login account and an example is given below:

```
ssh [ARCHIVE_USERNAME]@archive.surfsara.nl
      touch test.txt
      dmls  -l test.txt
      -rw-r--r--  1 homer    homer    0 2019-04-25 15:24 (REG) test.txt
```

Best practices for the usage of Data Archive are described on the Data Archive page.

3.3 Quota policy

Each Spider is granted specific compute and storage resources in the context of a project. For these resources there is currently **no hard quotas**. However, we monitor both the core-hour consumption and storage usage to prevent that users exceed their granted allocation.

3.4 Backup policy

The data stored on CephFS (home and project spaces) is disk only, replicated three times for redundancy. For disk-only data there is **no backup**. If you cannot afford to lose this data, we advise you to copy it elsewhere as well.

3.5 Data Ownership Policy

The data stored in the /project folder is owned by the grant's signing authority. If data is owned by a user who has left the project in the /project folder we ask that you request that user change the ownership to an active project member before leaving.

The data stored in the /home folders is owned by individual users of those folders and can not be transferred to another user without their consent. We are also obligated to remove a users data no more than 6 months after they have left the project.

See also:

Still need help? Contact *our helpdesk*

CHAPTER

FOUR

SOFTWARE ON SPIDER

Tip: There are several ways to setup and use your software on Spider. In this page you will learn:

- what is our policy for system software installations
- how to install your software on the local filesystem
- using containerization for making your software portable across different platforms
- distributing your software on different platforms
- installing software using EasyBuild

4.1 System software

There are cases in which a user or project may need extra software not included on SPIDER system. It may require the installation of a new software tool (i.e. emacs editor) or an specific version of a software component that is already on SPIDER (i.e. gcc 9). As a user of this platform you are free to submit a request to *our helpdesk* to ask us install a software required for your project system wide. The requests will be evaluated case-by-case, but in general the following policy applies:

- Stand-alone applications easily available through the official RPM repositories (CentOS, EPEL, ...) are suitable to be installed system wide. Some examples are emacs, joe, jq, ...
- Alternative versions of core tools (i.e. Python 3.6, gcc 9, ...) will have to be evaluated case by case. We will accept requests of software that can be deployed using well defined and automated procedures.

The standard supported login shell on SPIDER is bash. The standard supported software setup is identical on all nodes. Basic unix functionality is installed system-wide:

- software compilers (e.g., gcc, g++, f95)
- editors (e.g., vi, vim, emacs, nano and edit).
- graphical tools are supported via X11 ssh forwarding on the login node.
- operating system (OS on Spider is CentOS 8) on login and worker nodes.

4.2 User installed software

Software that does not require root privileges can be setup in user-space. The software that is installed on the local CephFS will be made available on all nodes for your jobs because the local directories are globally mounted.

4.2.1 Software on home

Home can be used to install software that you don't want to share with other members in your SPIDER project. This can be placed in the location `/home/$USER`.

4.2.2 Software on project space

For software that you want to install and share with other project members, you can use the `/project/[PROJECTNAME]/Software` in your *Project spaces directories*.

Only dedicated software managers have permissions to write in this directory, and all members in the project have read and execute permissions in this space.

The project members can use the software installed by the software manager, simply by exporting the right path in `$HOME/.bashrc`.

Note: If you wish to install conda environments or other software that handles many small files, we suggest you to use one of the other software installation methods below, because the home or project spaces installations lie on CephFS and loading such software can be very slow.

4.3 Apptainer containers

4.3.1 What is Apptainer

Apptainer (formerly Singularity) is a container technology specifically designed for HPC/HTC systems. As such it properly controls the permissions of the container during build- and runtime, while allowing access to host components when needed. Apptainer allows putting whole software stacks into a single container file, which can then execute code that depend on this software. Examples include GPU software stacks such as Nvidia/CUDA and AMD/ROCm or entire programs such as MATLAB.

4.3.2 When to use apptainer

Apptainer works best in this scenario if you have a large software stack does not change. For example using a static version of GPU drivers together with static python modules that stay at one version works best. This is because upgrading components is relatively hard and it is advised to completely rebuild the container when one updates a component. As the build can take up to 20 minutes, this may work for some, especially for students that only need reliable software: a static container that does not change during their project works best.

A single image file is created containing everything in the container, resulting in faster execution times and lower load on the system. Moreover, these "image"-files are portable to machines with the same architecture, so the built file can be moved to different systems running the same Linux flavour.

Large software packages with many files (such as conda) will run relatively slow on distributed file systems, which is used on Spider. So if you have a **large software stack** that **does not change**, using **apptainer** instead of running directly from the disks is preferred.

4.3.3 Caveats to apptainer

The stability of the software stack is important, as build-times can go up to 20 minutes for a single container. If you have multiple programs, they should live in their own containers and not be merged into a single container. Apptainer requires some training, as you need to run, mount and bind paths with containers to get the full potential of the technology.

4.3.4 Upload your image

Your Apptainer image can be viewed as a single file containing all the necessary software for your purpose. When compared to traditionally compiled software it is similar to a binary file containing the executable software. The image can be placed anywhere on Spider, as long as the location is accessible to your processing jobs.

4.3.5 Example code

Here is a job script template for `apptainer` usage. It assumes the container is already built and ready to be used. The `analysis.py` script takes arguments `filename.in parameter` and writes output into `[filename]_[parameter].out`. The Slurm JobArray goes over values 24 to 40 in steps of 2: we do a parameter sweep over these values and feed the values to the script.

```
#!/bin/bash
#SBATCH -N 1            #request 1 node
#SBATCH -c 1            #request 1 core and 8000 MB RAM
#SBATCH -t 5:00         #request 5 minutes jobs slot
#SBATCH --array=24-40:2 #go over parameters 24-40 in steps of 2

# the array goes over 24-40 in steps of 2, save the value in PARAM for clarity
PARAM=$((SLURM_ARRAY_TASK_ID))

# copy the input data to scratch
mkdir "$TMPDIR"/myanalysis
cp -r $HOME/mydata "$TMPDIR"/myanalysis
cd "$TMPDIR"/myanalysis

# mount the analysis folder into the container at /mnt and run the analysis on a file␣
↪using 'exec'
apptainer exec --bind $TMPDIR/myanalysis:/mnt python analysis.py /mnt/file1.in $PARAM

# copy the output back as TMPDIR is cleaned after the job
cp $TMPDIR/file1_{24..40..2}.out $HOME/myoutput

echo "SUCCESS"
exit 0
```

This example uses many options simultaneously to show the power of combining containers, slurm job arrays and scratch space for an analysis.

Please note that it is possible to bind several directories by providing a comma separated list to the `--bind` option, e.g. `--bind /cvmfs,/project`. Additional information can be found in the Sylabs documentation.

4.4 LUMI Container Wrapper

4.4.1 What is the LUMI Container Wrapper

The LUMI Container Wrapper (LCW) is a tool that wraps containers such that you can install conda and pip environments in a container and allows running the binaries in the container easily for the user. By writing the whole software stack into an external file and mounting this file into the container, you can update the software without rebuilding the base container. Allowing for faster load- and run-times on distributed file systems (such as Spider), while maintaining the ability to update software stored in the external file. For more information, see the full LCW documentation.

4.4.2 When to use LCW

When using conda- and/or pip-based virtual environments, consider using LCW instead of an installation on disk.

4.4.3 Caveats to LCW

You can only run a single apptainer container simultaneously, so if you have LCW running in your terminal, you can not run a second container in the same terminal. Recursive containerization is also disallowed in apptainer. When using very specific **large** containers, such as GPU containers (Nvidia, AMD, Intel), use the container directly instead of user LCW, as you have to build on top of the container contents.

4.4.4 Example code

Clone the code-base at github and set up the Spider environment. You can do this by adding spider.yaml to the `hpc-container-wrapper/configs` folder of the cloned repository.

Run the following commands:

```
cd hpc-container-wrapper
bash install.sh spider
```

The spider in the second command refers to the `spider.yaml` file in `hpc-container-wrapper/configs`. Once the base installation is setup, you can create a wrapper with:

```
mkdir /path/to/install_dir/
conda-containerize new --prefix /path/to/install_dir/ conda.yaml
```

where `conda.yaml` contains your installation, for example:

```
channels:
  - conda-forge
dependencies:
  - python=3.8.8
  - scipy
  - nglview
```

Once the wrapper is created you need to add it to your path to run, and all relevant binaries (such as `python`) will be called from the container wrapper: `export PATH="/path/to/install_dir/bin:$PATH"`. You can put the export in your `.bash_rc` or set it by hand each time you want to use the container wrapper.

Tip: There are more options that can be set in the `spider.yaml` file and while building / updating the wrapper. See the documentation and repository for more information:

LUMI Documentation

GitHub repository

4.5 Softdrive

4.5.1 What is Softdrive

Softdrive is a software distribution service based on CVMFS, which has been developed at CERN, and is being used extensively in production environments since several years. CVMFS is a network file system based on HTTP. The CVMFS software repositories are publicly available and can be mounted read-only on multiple compute clusters, including Spider.

Simply put, systems with the CVMFS installed have instant access to the Softdrive software repositories via the command line. This is very handy when you work on multiple platforms to solve the problem of installing and maintaining the software in different places. It is also very efficient when your software handles many smalls files, e.g. conda environments.

Access on Softdrive is *not* provided by default to the Spider projects. To request for Softdrive access, please contact our *our helpdesk*.

4.5.2 Access

If you already have access on Softdrive, then you can use it directly from Spider, simply by exporting the `/cvmfs/softdrive.nl/$USER` software paths into your Spider scripts or your `.bashrc` file.

On Spider nodes, your Softdrive files will be available under:

```
/cvmfs/softdrive.nl/[SOFTDRIVE_USERNAME]/
```

Please note that your [SOFTDRIVE_USERNAME] can be different than your [SPIDER_USERNAME].

4.5.3 Installation your software

1 Once access has been arranged, you can log in on the software distribution node, using your Softdrive username and password:

```
ssh username@softdrive.ms4.surfsara.nl
```

2. Prepare your software somewhere in your Softdrive home directory. Compile your software tree in your home directory. When you want to run your workflows over multiple system types, it may be worthwhile and good practice to build your software independent of local libraries as much as possible. Try to build static binaries whenever you can.

3. When satisfied, install your software under /cvmfs/softdrive.nl/$USER

4. Then trigger publication by executing the following command:

```
publish-my-softdrive
```

After a couple of minutes your new software becomes available on :abbr:`Spider (Symbiotic Platform(s) for Interoperable Data.

Note: Another possible method for the user Software installatuon is EasyBuild. EasyBuild is offered on multiple HPC systems. If you are familiar with EasyBuild or wish to try it on Spider, follow our instuctions here.

See also:

Still need help? Contact *our helpdesk*

CHAPTER

FIVE

COMPUTE ON SPIDER

Tip: This is a quickstart on the platform. In this page you will learn:

- how to prepare and run your workloads
- about job types, partitions and Slurm constraints

The table below lists the available* Spider node types.

Number of nodes	Node name	CPU SKU	CPU cores per node	Total memory per node (per core)	Other characteristics	Included in the partition
25	wn-ca-[01-25]	AMD Rome 64 Cores/Socket	64	480 GB (7,5 GB)	Local scratch 6TB SSD	normal, infinite, short, interactive
18	wn-dc-[01-18]	AMD Rome 64 Cores/Socket	64	960 GB (15 GB)	Local scratch 12TB SSD	normal
5	wn-ha-[01-05]	AMD Rome 64 Cores/Socket	64	950 GB (14,85 GB)	Local scratch 12TB SSD	normal, infinite
5	wn-hb-[01-05]	AMD Naples 64 Cores/Socket	64	950 GB (14,85 GB)	Local scratch 12TB SSD	normal
2	wn-gp-[01,02]	Intel Cascade Lake 22 Cores/Socket	22	720 GB (32,77 GB)	Local scratch 6TB SSD	gpu_v100
3	wn-ga-[01-03]	AMD Rome (2x) 7 Cores/Socket	14	229 GB (16,38 GB)	Local scratch 6TB SSD	gpu_a100_7c
5	wn-gb-[01-05]	Intel Ice Lake(2x) 22 Cores/Socket	44	353 GB (8 GB)	Local scratch 6TB SSD	gpu_a100_22c

- Updated on the 29th November, 2023.

5.1 Prepare your workloads

When you submit jobs to the batch system, you create a job script where you specify the resources that your programs need from the system to execute successfully.

Before submitting your jobs, it is a good practice to run a few tests of your programs locally (on the login node or other system) and observe:

i) the time that your programs take to execute

ii) the amount of cores that your software needs to execute these tasks

iii) the maximum memory used by the programs during execution

We suggest you, where possible, first debug your job template on the login node. In doing so, please take into account that the login node is a shared resource and hence any job testing should consume the least demanding set of resources. If you have high resource demands please contact *our helpdesk* for support in testing your jobs.

Once you get a rough estimate of the resources above, you are set to go. Create your job script to request from the scheduler the estimated resources.

In the current setup of Slurm on SPIDER, we ask you to specify at least the following attributes:

SBATCH directive	Functionality	Usage example
`-N <number>`	the number of nodes	`#SBATCH -N 1` (the job will run on a single node)
`-c <number>`	the number of cores	`#SBATCH -c 2` (the job will use 2 cores couple to 16000 MB memory)
`-t HH:MM:SS`	the wall-clock time	`#SBATCH -t 1:00:00` (the job will run max for 1 hour)
`-p <partition>`	partition selection	`#SBATCH -p normal` (the job will run max for 120 hours)
`-p <partition>`	partition selection	`#SBATCH -p infinite` (the job will run max for 720 hours)
`-p <partition>`	partition selection	`#SBATCH -p short` (the job will run max for 12 hours)
`-p <partition>`	partition selection	`#SBATCH -p interactive` (the job will run max for 12 hours)
`-p <partition>`	partition selection	`#SBATCH -p gpu_v100` (the job will run on V100 nodes with a max of 120 hours)
`-p <partition>`	partition selection	`#SBATCH -p gpu_a100_22c` (the job will run on A100 nodes with a max of 120 hours, with max 22 cores per GPU)
`-p <partition>`	partition selection	`#SBATCH -p gpu_a100_7c` (the job will run on A100 nodes with a max of 120 hours, with max 7 cores per GPU)

The specifics of each partition can be found with `scontrol show partitions`, the information per machine can be found with `scontrol show node NAME`, where NAME is the name of the worker node and for a simple overview use `sinfo`.

5.2 Run your jobs

5.2.1 Running a local Job with *srun*

The `srun` command creates an allocation and executes an application on a cluster managed by Slurm. It comes with a great deal of options for which help is available by typing `srun --help` on the login node. Alternatively, you can also get help at the Slurm documentation page.

The `srun` command when used on the command line is executed locally by Slurm, an example of this is given below. A python script, `hello_world.py`, has the following content;

```
#!/usr/bin/env python
print("Hello World")
```

This python script can be locally executed as;

```
srun python hello_world.py
#Hello World
```

Typically `srun` should only be used with a job script that is submitted with `sbatch` to the Slurm managed job queue.

5.2.2 Running an interactive Job with *srun*

You can start an interactive session on a worker node. This helps when you want to debug your pipeline or compile some software directly on the node. You will have direct access to your home and project space files from within your interactive session.

The interactive jobs will also be 'scheduled' along with batch jobs for resources so they may not always start immediately.

The example below shows how to start an interactive session on a normal partition worker node with maximum time of one hour, one core and one task per node;

```
srun --partition=normal --time=00:60:00 -c 1 --ntasks-per-node=1 --pty bash -i -l
```

To stop your session and return to the login node, type `exit`.

The example below shows how to start an interactive session on a single core of a specific worker node;

```
srun -c 1 --time=01:00:00 --nodelist=wn-db-02 --x11 --pty bash -i -l
```

5.2.3 Submitting a Job Script with *sbatch*

The `sbatch` command submits `batch script` or `job description script` with 1 or more `srun` commands to the batch queue. This script is written in bash, and requires SBATCH header lines that define all of your jobs global parameters. Slurm then manages this queue and schedules the individual `srun` jobs for execution on the available worker nodes. Slurm takes into account the global options specified with `#SBATCH <options>` in the job description script as well as any local options specified for individual `srun <options>` jobs.

Below we provide an example for `sbatch` job submission with options. Here we submit and execute the above mentioned `hello_world.py` script to the queue via `sbatch` and provide options `-N 1` to request only 1 node, `-c 1` to request for 1 core and 8000 MB memory (coupled) and `-t 1:00` to request a maximum run time of 1 minute. The job script, `hello_world.sh`, is an executable bash script with the following code;

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -c 1
#SBATCH -t 1:00
srun python /home/[USERNAME]/[path-to-script]/hello_world.py
```

You can submit this job script to the Slurm managed job queue as;

```
sbatch hello_world.sh
#Submitted batch job 808
```

The job is scheduled in the queue with `jobid 808` and the stdout output of the job is saved in the ascii file `slurm-808.out`.

```
more slurm-808.out
#Hello World
```

More information on `sbatch` can be found at the Slurm documentation page.

5.2.4 Using local `scratch`

If you run jobs that require intensive IO processes, we advise you to use `scratch` because it is the local SSD on every compute node of the the Spider. This is a temporary storage that can be used only during the execution of your job and will be arbitrarily removed at any point once your job has finished running.

In order to access the `scratch` filesystem within your jobs, you should use the `$TMPDIR` variable in your job script. We advise you the following job workflow:

- At the start of your job, copy the necessary input files to `$TMPDIR`
- Run your analysis and produce your intermediate/output files on `$TMPDIR`
- Copy the output files at the end of the job from `$TMPDIR` to your home directory

`TMPDIR` is `/tmp` which is a 'bind mount' from `/scratch/slurm.<JOBID>` so you will only see your own job files in `/tmp` and all files will be removed after the job finishes.

Tip: TMPDIR variable can only be used within the SLURM jobs. It can not be used nor tested on the UI because there is no scratch space.

Here is a job script template for `$TMPDIR` usage;

```
#!/bin/bash
#SBATCH -N 1      #request 1 node
#SBATCH -c 1      #request 1 core and 8000 MB RAM
#SBATCH -t 5:00   #request 5 minutes jobs slot

mkdir "$TMPDIR"/myanalysis
cp -r $HOME/mydata "$TMPDIR"/myanalysis
cd "$TMPDIR"/myanalysis

# = Run your analysis here =

#when done, copy the output to your /home storage
```

(continues on next page)

(continued from previous page)

```
tar cf output.tar output/
cp "$TMPDIR"/myanalysis/output.tar $HOME/
echo "SUCCESS"
exit 0
```

5.3 Job types

5.3.1 CPU jobs

- For regular jobs we advise to always only use 1 node per job script i.e., `-N 1`. If you need multi-node job execution, consider better an HPC facility.
- On Spider we provide **8000 MB RAM per core**.
  - This means that your memory requirements can be specified via the number of cores *without* an extra directive for memory
  - For example, by specifying `-c 4` you request 4 cores and 32000 MB RAM
- On Spider we provide **80 GB scratch disk per core**.
  - This means that your scratch disk requirements can be specified via the number of cores *without* an extra directive for storage
  - For example, by specifying `-c 2` you request 2 cores and 160 GB scratch disk
  - When you target specifically our fat nodes with 12TB available scratch, the provided scratch disk per requested core is 200 GB

5.3.2 GPU jobs

- For more information on using GPUs on Spider, see the *dedicated section*.
- For jobs that require GPU resources a specific partition is available (see *partitions* for all the different partitions).
- Access to the GPU paritions needs to be requested and received.

5.4 Slurm partitions

We have configured four CPU and two GPU partitions on Spider as shown in the *table above*:

- If no partition is specified, the jobs will be scheduled on the normal partition which has a maximum walltime of 120 hours and can run on any worker nodes.
- Infinite partition jobs have a maximum walltime of 720 hours. Please note that you should run on this partition at your own risk. Jobs running on this partition can be killed without warning for system maintenances and we will not be responsible for data loss or loss of compute hours.
- Short partition is meant for testing jobs. It allows for 2 jobs per user with 8 cores max per job and 12 hours max walltime.
- Interactive partition is meant for testing jobs and has 12 hours maximum walltime.
- GPU V100 contains 1 Nvidia V100 (32GB) card per node.

- GPU A100 contains 2 Nvidia A100 (40GB) cards per node.

5.5 Slurm constraints

5.5.1 Regular constraints

The Slurm scheduler will schedule your job on any compute node that can fulfil the constraints that you provide with your `sbatch` command upon job submission.

The minimum constraints that we ask you to provide with your job are given in the example above.

Many other constraints can also be provided with your job submission. However, by adding more constraints it may become more difficult to schedule and execute your job. See the Slurm manual (https://slurm.schedmd.com) for more information and please note that not all constraint options are implemented on SPIDER. In case you are in doubt then please contact *our helpdesk*.

5.5.2 Spider-specific constraints

In addition to the regular `sbatch` constraints, we also have introduced a number of Spider-specific constraints that are tailored to the hardware of our compute nodes for the SPIDER platform.

These specific constraints need to be specified via constraint labels to `sbatch` on job submission via the option `--constraint=<constraint-label-1>,<constraint-label-2>,...,<constraint-label-n>`

Here a comma separated list implies that all constraints in the list must be fulfilled before the job can be executed.

In terms of Spider-specific constraints, we support the following constraints to select specific hardware:

SBATCH directive	Functionality	Worker Node
`--constraint=skylake`	cpu architecture	`wn-db-[01-06]`
`--constraint=napels`	cpu architecture	`wn-hb-[01-05]`
`--constraint=rome`	cpu architecture	`wn-ca-[01-25], wn-ha-[01-05]`
`--constraint=ssd`	local scratch	`all nodes`
`--constraint=amd`	cpu family	`wn-ca-[01-25], wn-ha-[01-05], wn-hb-[01-05]`
`--constraint=intel`	cpu family	`wn-db-[01-06], wn-gb-[01-04], wn-gp-[01-02]`

As an example we provide below a bash shell script `hello_world.sh` that executes a compiled C script called 'hello'. In this script the #SBATCH line specifies that this script may only be executed on a node with 2 cpu-cores where the node must have a skylake cpu-architecture and ssd (solid state drive) local scratch disk space.

```
#!/bin/bash
#SBATCH -c 2 --constraint=skylake,ssd
echo "start hello script"
/home/[USERNAME]/[path-to-script]/hello
echo "end hello script"
```

From the command line interface the above script may be submitted to Slurm via:

```
sbatch hello_world.sh
```

Please note that not all combinations will be supported. In case you submit a combination that is not available you will receive the following error message:

> 'sbatch: error: Batch job submission failed: Requested node configuration is not available'

5.6 Querying compute usage

5.6.1 Overview

`sacct` and `sreport` are slurm tools that allows users to query their usage from the slurm database. The accounting tools `sacct` and `sreport` are both documented on the Slurm documentation page.

These slurm queries result in a users total usage for a user. The sum of Raw CPU times / 3600 gives total core usage for the defined period. *-d Produces delimited results for easier exporting / reporting*

5.6.2 Examples

```
# look into the details of your usage by job
sacct \
   -X #sum\
   -S2020-07-01 -E2020-07-30 \
   --format=jobid,jobname,cputimeraw,user,alloccpus,state,partition,account,exitcode
```

```
#view the spexone project usage and your user's usage
sreport \
   -t second \
   -T cpu cluster \
   AccountUtilizationByUser \
   Start="2020-07-01" \
   End="2020-07-30"
```

See also:

Still need help? Contact *our helpdesk*

CHAPTER

SIX

GPUS ON SPIDER

Tip: This is a quickstart for using GPUs. In this page you will learn:

- how to access GPU nodes
- how to build and run a singularity container that:
  - uses GPUs
  - runs CUDA code
  - runs python code
  - runs jupyter notebooks

6.1 Using GPU nodes

To run your program on GPU nodes some guidelines for the user have to be taken into account. Firstly, GPUs and their drivers are only available on the GPU nodes `wn-gp-[01,02]` and `wn-ga-[01,02]` and **not** on the UI nodes.

To *interactively* log in to a GPU node run:

```
srun --partition=gpu_v100 --time=00:60:00 --gpus v100:1 --pty bash -i -l
```

This will open a bash session on a machine in the `gpu_v100` partition for 60 minutes.

Tip: Asking for more GPUs than the total available on a node does not give an error, your jobs will run on the maximum number.

All GPU nodes run Nvidia hardware and the CUDA drivers are available on the GPU nodes. Other GPU software needs to be obtained and deployed by the user. We suggest users to *create* or make use of pre-build Singularity containers. The Spider team can provide assistance to users who are not familiar with container. In this case please submit your request for assistance via our *our helpdesk*.

In case the version number of the drivers has to be known, the user can find the version number and other information on the GPU hardware with:

```
srun -p GPU_PARTITION --gpus GPU:N_GPUS nvidia-smi
```

where the GPU_PARTITION is either `gpu_v100`, `gpu_a100_22c` or `gpu_a100_7c` depending on which one you are planning to use. The `gpu_a100_7c` partition has an AMD CPU with 7 cores, while the `gpu_a100_22c` has an Intel with 22 cores per GPU. Both partitions have the exact same A100 cards. The `--gpus` flag specifies which type of GPU

you want to use and how many, you will get `N_GPUS` up to the maximum in the cluster of type `GPU` which can be `v100` or `a100`.

The compilation and running of code is recommended to be done inside of a singularity container, so start by building a singularity image. More information on singularity on Spider can be found at *singularity containers*. Once the container is available, the program can be run.

Next, some short examples for building and running commands are shown. A more in-depth container build procedure is shown *here*.

6.1.1 Simple building example

Building can be done as follows:

```
singularity build ubuntu.sif docker://ubuntu
```

In this example, the latest stable version of ubuntu is used (found here). For running libraries like tensorflow or pytorch or CUDA tools, please create or obtain appropriately compiled containers. A few links to more resources are given *here*.

After the singularity image has been sucessfully built, the user can enter a shell in the container with:

```
singularity shell --nv ubuntu.sif
```

In the shell, commands can be run which are executed in the container environment. You can also run a command directly in the container and get the output using `exec`.

```
singularity exec --nv ubuntu.sif echo "hello world"
```

Warning: The `--nv` flag is necessary to expose the GPUs on the host to the container.

Here follows an example for running the container in batch mode with a shell script. Start by making a file called `script.sh` containing:

```
#!/bin/bash

#SBATCH -p gpu_v100
#SBATCH -G v100:1
#SBATCH -e slurm-%j.out
#SBATCH -o slurm-%j.out

singularity exec --nv ubuntu.sif echo "hello world"
```

The flags `-e` and `-o` instruct SLURM in which files to write respectively *stderr* and *stdout* of the job. In this case they are both sent to the same file, this is done for comparison in the next step. If you now run this shell script on the `ui-[01-02]` nodes with `bash script.sh`, it will result in:

```
INFO:    Could not find any nv files on this host!
hello world
```

as the UI nodes do not have access to GPUs and thus do not have an nv file to point the container to the required libraries. Running the script in batch mode with `sbatch script.sh`, the `-p` flag is used, and the job ends up on a GPU node. The output becomes:

```
hello world
```

Of course, this ubuntu image does not have any of the tools needed to build GPU-native code or libraries that can run on the GPU. Refer to *this section* for more resources and *this section* for an example.

Tip: While you do not get the warning about finding the nv file when using the `--nv` flag, you also have to specify the name of the GPU to use, otherwise none are allocated to you! This can be done with the `--gpus` or `-G` flag, as can be seen in the example shell script.

Now you are ready to build on top of a base container and run your code on a GPU!

6.1.2 Accounting of GPU usage

Currently the usage of GPU nodes is accounted for in GPU hours. This means that even though multiple cores are used simultaneously, one hour of use of a GPU node is billed as 1 GPU-hour. By default, half the CPU cores of the node (22) are used when you use half of the available GPUs. When using GPUs the CPU cores are not counted and fall under the GPU hours. In contrast to generic CPU use in a 'regular' job, where one hour of multi-core usage is billed as multiple CPU hours, depending on the number of cores.

6.2 Building and running a singularity container

In this section we show how to build a singularity container use it to run code in its environment. There is extensive documentation from singularity itself here.

The steps in this section are done on GPU nodes, to ensure availability of the drivers, which may be needed in some compilation steps.

6.2.1 Building directly from dockerhub

There are multiple ways to build a container. To build directly from docker hub, for example the latest version of tensorflow, one can invoke:

```
singularity build --nv tf_latest.sif docker://tensorflow/tensorflow:latest
```

and the image `tf_latest.sif` from dockerhub will be built, containing the contents of the latest `tensorflow` image from the makers of tensorflow. The docker image is converted by singularity to a singularity container. You can also get an image from a different source, such as the Nvidia container repository:

```
singularity build --nv nvidia-tf.sif docker://nvcr.io/nvidia/tensorflow:22.07-tf2-py3
```

An Nvidia image contains all the necessary prerequisites to run on Nvidia GPUs, which is preferable on Spider. The tag on the docker image in this case refers to the build release date, the tensorflow version and the python version: july 2022, TF v2, python3.

To directly run the container in memory without writing an image to disk invoke:

```
singularity run --nv docker://nvcr.io/nvidia/tensorflow:22.07-tf2-py3
```

In the examples below, the base images are taken from the internet and expanded upon using *definition* files, to build custom singularity containers. The singularity documentation on definition files can be found here.

6.2.2 Running CUDA code

Here, we show the method of using a *definition file*, as opposed to above, where directly building from a repository is shown. A definition file contains the steps that are followed during the building of the container and steps that are performed when, for example, `singularity run` is called. The contents of the definition file are shown before these contents are explained. Start by making the file called `cuda_example.def` and add all the steps we want to take to make a container:

```
Bootstrap: docker
From: nvidia/cuda:11.7.0-devel-centos7

%post
#This section is run inside the container
yum -y install git make
mkdir /test_repo
cd /test_repo
git clone https://github.com/NVIDIA/cuda-samples.git
cd /test_repo/cuda-samples/Samples/2_Concepts_and_Techniques/eigenvalues/
make

%runscript
#Executes when the "singularity run" command is used
#Useful when you want the container to run as an executable
cd /test_repo/cuda-samples/Samples/2_Concepts_and_Techniques/eigenvalues/
./eigenvalues

%help
This is a demo container to show how to build and run a CUDA application
on a GPU node
```

This container will take a base image from docker-hub and use a pre-built nvidia/cuda container of a specific version. This container also contains the necessary CUDA tools to compile binaries that run on GPUs. After starting from this base-image, in the next steps some tools are installed, directories are created and filled with a git repository. From this repository a single example of a CUDA applictation is compiled. When running the container on the command line, this application is run automatically.

Now that we have the definition file, we can build the singularity image with:

```
singularity build --fakeroot --nv --sandbox cuda_example.sif cuda_example.def
```

In this command some flags are used, these and more are explained in the table below.

Flag	Functionality
`--fakeroot`	raises permissions inside the container to `sudo`, necessary for installing packages
`--nv`	exposes the nvidia drivers of the host to the container (makes them available)
`--sandbox`	allows the final container to be changed in *write-mode*, should only be used for debugging!
`--writable`	allows writing into a sandboxed container when invoking `singularity shell`

`--fakeroot` was needed for installing `git` and `make` in the container, as of 2024 it is not strictly necessary. `--nv` is necessary to access the GPU from within the container, and `--sandbox` is used to allow the user after running this example to go into the container and make changes to folders, files or run other commands that change the state of the container. If container `--fakeroot` building permissions are not enabled for you on the GPU nodes, please contact us at *our helpdesk*.

Once the container is built - which can take a few minutes as multiple base containers have to be retrieved from the internet - you can run it using

```
singularity run --nv cuda_example.sif
```

which will output the result of the *eigenvalues-test*, as was instructed in the definition file under `%runscript`. To run commands from within a shell in the container that allow for making changes, do

```
singularity shell --nv --writable cuda_example.sif
```

The container was exposed to the GPU at build-time, and at run-time it also has to be exposed with `--nv`, otherwise it can not find the drivers! In case the container is still under development and needs debugging, use the `--writable` flag so that missing packages/libs can be added to the container at runtime. These packages have to be added in the definition file for the final singularity build.

Tip: Only use `--sandbox` and `--writable` when developing the image. Once the build is settled, create the container with a definition file and distribute it as-is for maximum stability.

There is also a full HPC development image made available by Nvidia, called "HPC SDK", which is the software development kit that contains all the compilers, libraries and tools necessary to build efficient code that runs on GPUs. This image can be found here.

6.2.3 Running python

Popular python interfaces for modelling are tensorflow, keras, pytorch, and more. An example for using tensorflow in singularity is provided below, but some warnings have to be taken into account, due to the default behaviour of singularity with the host machine.

Starting on a machine in the GPU partition, we create a definition file `nv-tf-22.07.def` containing:

```
Bootstrap: docker
From: nvcr.io/nvidia/tensorflow:22.07-tf2-py3

%post
cd /tmp
git clone https://github.com/tensorflow/docs

%runscript
cd /tmp/docs/site/en/tutorials/keras
python

%help
This is a demo container to show how to run tensorflow in python
```

and build the container using the usual

```
singularity build --nv nv-tf-22.07.sif nv-tf-22.07.def
```

In this definition file, the tensorflow docs and tutorials are installed as an example to show how to do it.

Warning: Running `pip` inside the container using `singularity shell` when it is in `--writable` mode will write the python libraries to the default **mounted** location. This location is the `$HOME`-folder of `$USER`. As such,

pip packages will end up on the host machine and not in the container. To avoid this behaviour, only run `pip` during the building of the image in the definition file, or change the mounting behaviour of singularity when entering the shell. For example, mount the local path of your project as working directory as the `$HOME` in the container.

For information on this, read `man singularity-shell` and bind mounts.

Warning: As the home folder is mounted by default in singularity, and python searches certain folders by default, it is possible that inside the container packages from the host machine are called, instead of what is inside the container. For example, the `~/.local` folder on the host machine can have precedence over site-packages in the container. To avoid errors from mounting or binding at all, use the flags `--no-home` or `--no-mount=[]`. If errors appear relating to CUDA `.so` files, or versions of packages are mismatching, ensure that the user-space is not accidentally providing libraries to the container.

Tip: Use singularity only to control the versioning of the environment and encapsulate your libraries in the container and thus control their versioning. Code and data files can be fed to singularity, so keep such files external to the container.

The example we are about to execute in the container comes from the tensorflow library: classifying pieces of clothing. Now create a file to run `fashion.py`, set it to executable with `chmod 755 fashion.py` and add the following:

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

# TensorFlow and tf.keras
import tensorflow as tf

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)

fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

train_images = train_images / 255.0
test_images = test_images / 255.0

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
```

(continues on next page)

(continued from previous page)

```
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10)

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print('\nTest accuracy:', test_acc)

probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])

predictions = probability_model.predict(test_images)
print(predictions[0])
```

This example will create a model that recognizes the clothes in a picture, and a prediction of a set of test images is done at the end. The result can be compared to the official example. The matplotlib output is omitted in this example for simplicity. This output can be seen in the section on *jupyter notebooks*.

Now this code can be run on a GPU node with:

```
singularity exec --nv nv-tf-22.07.sif ./fashion.py
```

Or run it interactively on a GPU node in the container line-by-line with:

```
singularity shell --nv nv-tf-22.07.sif
```

If there is an output in the terminal running the python code similar to:

```
2022-07-29 11:53:24.017428: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1532]␣
↪Created device /job:localhost/replica:0/task:0/device:GPU:0 with 30987 MB memory:  ->␣
↪device: 0, name: Tesla V100-PCIE-32GB, pci bus id: 0000:00:06.0, compute capability: 7.
↪0
```

this means the GPU is being used for your computations.

Also, by wrapping the singularity command in a shell script called `fashion.sh` and adding the appropriate `#SBATCH` commands at the top, the script can be submitted to the batch system with `sbatch fashion.sh`. The script would look like:

```
#!/bin/bash

#SBATCH -p gpu_v100
#SBATCH -G v100:1

singularity exec --nv nv-tf-22.07.sif ./fashion.py
```

6.2.4 Running jupyter notebooks

Many users prefer working in interactive notebooks during development of their models. Here an example is shown of running tensorflow in a jupyter notebook. There is also a more general section in this documentation on jupyter notebooks *here*.

Tip: Make sure you use the GPU version and not the CPU version of your software in the container.

We start with the image from the *previous subsection*, the tensorflow container from the Nvidia repository with the added examples: `nv-tf-22.07.sif`. This image also contains jupyter by default.

```
ssh USERNAME@spider.surfsara.nl
srun --partition=gpu_v100 --gpus v100:1 --time=12:00:00 --x11 --pty bash -i -l
singularity shell --nv nv-tf-22.07.sif
```

where USERNAME is your username and the partition is a GPU partition, like `gpu_v100`, `gpu_a100_7c` or `gpu_a100_amd_22c` depending on your project. The `singularity shell` command is needed to start jupyter from the command inside the container. The tutorials were cloned during the building of the image. The container is read-only, and some of the examples will require to download and store some files. To have writing functionality available for the examples, build the image with `--sandbox` and run it with `--writable`, as mentioned in *this section*.

Start the notebook with:

```
cd /tmp/docs/site/en/tutorials/keras
jupyter notebook --ip=0.0.0.0
```

The python output will return an address like `http://127.0.0.1:8888/?token=abc123`. Opening this address in your browser will give you access to the notebook, but only if there is a tunnel that forwards the jupyter kernel to your machine. Now, we have open a tunnel to forward the port on which the python kernel communicates to the local machine where the user works. In this way, the notebook can be openened in the browser:

```
ssh -NL 8888:wn-gp-01:8888 USERNAME@spider.surfsara.nl
```

where USERNAME is your username and `wn-gp-01` should changed to the node on which the python kernel is running. This tunneling command has to be running in **a separate terminal**, and ensures the communication from port 8888 (right hand side) on the remote machine is forwarded to port 8888 (left hand side) on the local machine. The port that is given when you start the jupyter notebook defaults to 8888, but if it is already in use, the value will be different. The used value can be seen in the jupyter output in the terminal.

Now you can run an example from the `keras` folder by going to the http-address provided by jupyter.

Warning: Some jupyter instances provide a link of that contains `hostname:8888`. Replace `hostname` with `localhost` or `127.0.0.1` to properly fetch the notebook.

The terminal will now have CUDA output, while the notebook contains all the python and graphical output. Again, if there is an output in the terminal running the notebook similar to:

```
2022-07-29 11:53:24.017428: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1532]␣
↪Created device /job:localhost/replica:0/task:0/device:GPU:0 with 30987 MB memory:  ->␣
↪device: 0, name: Tesla V100-PCIE-32GB, pci bus id: 0000:00:06.0, compute capability: 7.
↪0
```

this means the GPU is being used for your computations. Now you can run the classification (fashion) notebook and compare with the output of the repository to see if you get similar results.

6.2.5 Advanced GPU querying

Some of the GPU nodes in spider have multiple GPUs installed. This opens up the avenue where multiple users use the same node simultaneously. Here are some more advanced commands to explore a few options.

To get one GPU and leave the other GPU on the node available for other users, do:

```
srun -p gpu_a100_22c --gpus=a100:1 --pty bash
```

To run on 2 GPUs simultaneously and have no other users on the nodes do:

```
srun -p gpu_a100_22c --nodes=1 --exclusive --gpus=a100:2 --pty bash
```

Warning: Do not request multiple GPUs unless you are sure your code can run on multiple GPUs. If you need exclusive acces to the node, use the `--exclusive` flag.

By default, half the cores of the node (22) are used when you use 1 out of 2 GPUs. To use only a single CPU core while using GPU do:

```
srun -p gpu_a100_22c --cpus-per-task=1 --gpus=a100:1 --pty bash
```

For more information read the man-pages of SLURM.

6.2.6 Resources on singularity and containers

https://docs.sylabs.io/guides/latest/user-guide/
https://hub.docker.com/r/nvidia/cuda
https://catalog.ngc.nvidia.com/
https://gpucomputing.shef.ac.uk/education/creating_gpu_singularity

See also:

Still need help? Contact *our helpdesk*

CHAPTER

SEVEN

WORKFLOWS

Tip: This is a quickstart on the platform. In this page you will learn:

- creating a job dependency with Slurm
- using Picas pilot framework from Spider

7.1 Slurm job dependencies

A job can be given the constraint that it only starts after another job has finished. Lets say that you have two Slurm jobs, A and B. You want job B to start after job A has successfully completed. Here are the steps:

- Submit job A and keep the returned job ID:

```
sbatch <jobA.sh>
```

- Submit job B with a condition to start after job A, by providing its assigned job ID. This way job B only starts after Job A has successfully completed:

```
sbatch --dependency=afterok:<jobID_A> jobB.sh
```

- We can also tell Slurm to run job B, even if job A fails:

```
sbatch --dependency=afterany:<jobID_A> jobB.sh
```

- If you want job B to start after several other jobs have completed, use the delimiter ':' as:

```
sbatch --dependency=afterok:<jobID_A:jobID_C:jobID_D> jobB.sh
```

7.1.1 Job dependency example

Now we show an example of how to use the dependencies. In this example it is shown how to run a job that depends on another job to finish before it starts.

First make two files, `hello.sh` which contains

```
#!/bin/bash

date
```

(continues on next page)

(continued from previous page)

```
sleep 30
date
echo "hello world"
```

and `bye.sh` which contains

```
#!/bin/bash

date
echo "bye!"
```

And make the scripts executable with `chmod +x hello.sh` and `chmod +x bye.sh`.

The `sleep 30` command stops the code for 30 seconds and then the `date` command shows you the time, so you can see that the second job waits for the dependent to finish before starting.

Now you can submit the jobs with:

```
jobid=$(sbatch --parsable hello.sh)
sbatch --dependency=afterok:$jobid bye.sh

Submitted batch job 2560644
```

The first command submits a job and returns only the JobID value with `--parsable`, and saves this value to a variable called `jobid`. The second command only starts after the job with jobid has finished in a success state with `afterok`. For more information on the dependency flag, see the SLURM man-pages.

Now you can see your jobs in the queue with:

```
squeue -u homer
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           2560644    normal   bye.sh    homer PD       0:00      1 (Dependency)
           2560643    normal hello.sh    homer  R       0:11      1 wn-hb-04
```

The `bye.sh` script is pending (PD), waiting until `hello.sh` is finished, which is running (R). A bit later the jobs are finished:

```
squeue -u homer
            JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
```

And now we can see the output of the log files:

```
cat slurm-2560643.out

Wed Aug 17 11:27:25 CEST 2022
Wed Aug 17 11:27:55 CEST 2022
hello world
```

```
cat slurm-2560644.out

Wed Aug 17 11:27:56 CEST 2022
bye!
```

And we see that the first job slept for 30 seconds and the second job waited until the first was finished!

See also:

For more information on job dependencies, see also the `-d, --dependency` section in the man page of the sbatch command.

7.2 PiCaS

When you run many jobs on Spider it can be difficult to keep track of the state of these jobs, especially when you start running hundreds to thousands of jobs. Although Slurm offers some functionality for tracking the status of the jobs, via the Slurm job ID, in many cases a Pilot job framework, such as PiCaS, is necessary for this purpose.

Access on PiCaS is *not* provided by default to the Spider projects. To request for PiCaS access, please contact our *our helpdesk*.

If you already have access on PiCaS, then you can use it directly from Spider, i.e. you can establish a connection to your CouchDB database and use the python PiCaS client either from the login node or the worker nodes.

To connect with your PiCaS database you need to provide your credentials (username, password, database name). It is possible to specify the password on the command line, however for security reasons this should be avoided on shared systems (like the login node) because it can allow other local users to read the password (e.g. with the `ps` command). Also to avoid having to type these credentials every time your client connects to your database or using them within your jobs, we advice you to authenticate to PiCaS with the steps below.

- Create a PiCaS configuration directory in your home directory. Here we will call this directory `picas_cfg`, but you are free to give it any other name.

```
mkdir /home/[USERNAME]/picas_cfg
chmod go-rwx /home/[USERNAME]/picas_cfg
```

- Check the settings of your directory with `ls -la`. The output should be similar to:

```
ls -la /home/homer/picas_cfg
drwx------ 1 homer homer  3  May  7 08:33 picas_cfg
```

- Create a new file called `picasconfig.py` inside the `picas_cfg` directory:

```
cd /home/[USERNAME]]/picas_cfg
touch picasconfig.py
```

- Add the following lines to the `picasconfig.py` file:

```
PICAS_HOST_URL="https://picas.surfsara.nl:6984"
PICAS_DATABASE="[YOUR_DATABASE_NAME]"
PICAS_USERNAME="[YOUR_USERNAME]"
PICAS_PASSWORD="[YOUR_PASSWORD]"
```

- Storing cleartext passwords in any medium is dangerous, so we need to make sure it is not readable by others. Save the `picasconfig.py` file and for additional security set it to read-write (rw) access for you only:

```
chmod go-rw /home/[USERNAME]/picas_cfg/picasconfig.py
```

- Check the permissions of your `picasconfig.py` file with `ls -la`. The output should be similar to:

```
ls -la /home/homer/picas_cfg/picasconfig.py
-rw------- 1 homer homer  126 May  7 08:33 picasconfig.py
```

- Finally, add the `picas_cfg` directory to your PYTHONPATH environment variable so that python can locate it. We recommend that you set this variable in your /home/[USERNAME]]/.bashrc file by adding the following lines to it:

```
PYTHONPATH=/home/[USERNAME]/picas_cfg:$PYTHONPATH
export PYTHONPATH
```

You are now ready to start using your PiCaS credentials without having to type them each time you or your jobs need to connect to the PiCaS server. Good practices to build worflows with PiCaS can be found in PiCaS example.

See also:

Still need help? Contact *our helpdesk*

7.3 Snakemake

Snakemake is a workflow management tool that uses makefiles to define analyses and provide reproducable and scalable results. The workflow can be developed locally and then moved to server, cluster, cloud or grid to scale up the data size and computational needs.

Snakemake has integrations with Slurm, PBS and SGE for deploying to compute clusters, see the documentation here.

For a showcase that demonstrates the power of snakemake, we advise you to go through the tutorial found here.

You can find the full documentation located at the read the docs page.

7.4 Dask

Dask is not a workflow management tool, but it does help in automating some parts of your analysis when you want to *scale up*. Dask helps you split up work and scale to an arbitrary amount of machines. It helps you manage data that is too large for a single machine and run your code in parallel on multiple machines.

The basic idea is that you define your analysis steps first. Dask takes the data shapes and operations, and prepares a schema of steps that are to be performed by your cluster. When all is ready, you submit your calculation and dask deploys it on a cluster. After the calculation is done, the data and output are aggregated into files that can be handled for post-processing.

Dask has excellent tutorials, which can be found here. The full documentation can be found here.

7.4.1 RS-DAT

RS-DAT or Remote Sensing Data Analysis Tools, created by the NLeSC, integrates Dask, Jupyter and the dCache storage system of SURF into one contained framework, which is then run from your local machine while deploying to a cluster.

The package can be found here and the installation instruction can be found here. Note that RS-DAT is designed around the Spider data processing platform and the Snellius supercomputer, but can be run on any Slurm cluster. The instructions for running on a generic cluster are available in the project documentation.

The examples are available here, but be aware that only the first of the three examples works without an access token to the data stored on dCache. The example, however, does give a good impression of the power or RS-DAT.

7.5 Cron jobs

If you need to automate (part of) your workflow it is possible to set up cronjobs on Spider. Please note that cronjobs on Spider can be used for testing purposes *only* and we do not offer this functionality as part of our service. If you wish to use cron jobs for production workflows please contact *our helpdesk*.

There are some restrictions when setting up a cronjob on Spider. The Spider login `spider.surfsara.nl` is automatically directed to two different login nodes `ui-01.spider.surfsara.nl` or `ui-02.spider.surfsara.nl` and cronjobs will be linked to the UI where they where created. If you would like to make changes in your cronjob you need to login directly to the login node (ui-01 or ui-02) where it was created (tip: to check which node you are on, you can type the command `hostname`). This may also affect your workflows in case of maintenance on the login node you run your cronjobs.

For a more sustainable option for automated jobs see *the next subsection*.

7.5.1 Recurring jobs

Cron is tied to a single machine, which in the case of Spider is a user-interface (ui) node or a worker node (wn). If a cron job is setup on one of these machines and this machine is not available, your job will not run.

For more robust recurring jobs, there is `scrontab`, a SLURM integration of cron. `scrontab` has identical syntax to `cron`. However, the job you submit is added to the SLURM database and run on the defined schedule when the resources are available. The job will only start once the resources are available, so your job may not always run at the exact same time (unlike `cron`). Your job will be scheduled to run on a worker node, regardless of where it was submitted.

Therefore we advice users to use `scrontab` instead of `crontab` when they want to set up recurring jobs. The `scrontab` documentation can be found here.

Warning: Jobs submitted with scrontab are treated as regular jobs in the SLURM queue and will keep running until they finished or killed. To avoid filling up the queue with test jobs that run long, please use short lived jobs like `hello world` when testing.

CHAPTER

EIGHT

WEB PUBLIC VIEW

Tip: Web public views are meant for open data distribution. In this page you will learn:

- how to distribute public data through a web view
- about risks when publishing data

SPIDER allows for public data sharing in a fast and easy way, which is often a huge problem in other platforms. In a web-based front-end, the so called *web public view*, you can give direct access to external collaborators in selected directories containing designated data products.

But how does this work?

Let's say that you have run some analysis on SPIDER and you want to redistribute some scientific products or intermediate results to collaborators that are not members of your SPIDER project, thus they don't have access to your *project space*. Here is what you need to do:

- Login to SPIDER, select the files that you want to publish and copy them to your project space `/project/[PROJECTNAME]/Public` directory, e.g.:

```
#the user [USERNAME: surfadvisors-homer] belongs to the project [PROJECTNAME:␣
↪surfadvisors] and wants to publish the file 'hello-world.sh'
ssh surfadvisors-homer@spider.surfsara.nl
cp $HOME/hello-world.sh /project/surfadvisors/Public/
ls -l /project/surfadvisors/Public/hello-world.sh
#-rwxrwxrwx  1  surfadvisors-homer  surfadvisors-homer  192 Jul  1 08:53 hello-world.sh
```

- That's it! Any data located in your `/project/[PROJECTNAME]/Public` directory is exposed to the web under the domain: `https://public.spider.surfsara.nl/project/[PROJECTNAME]/`. This means that the example file can be downloaded by anyone from link below:

https://public.spider.surfsara.nl/project/surfadvisors/hello-world.sh

Warning: Be careful with the data you place under your project space `/project/[PROJECTNAME]/Public` directory. This data is automatically exposed to the web and can be downloaded by anyone, even if they don't know the URL explicitly: the URL can be found through webcrawling!

See also:

Still need help? Contact *our helpdesk*

CHAPTER

NINE

JUPYTER NOTEBOOKS

Tip: Jupyter notebooks are very popular in science for interactive work. In this page you will learn:

- how to use Jupyter notebooks on Spider
- which available flavors to choose

Two methods of running jupyter notebooks in jupyter lab are discussed here: with virtual environments and with singularity containers. Some of this has also been covered in *the compute section*.

9.1 Where to run notebooks

Interactive notebooks should be run on the worker nodes mentioned in *Prepare your workloads* and **not** on the UI machines. Building the environments and containers can be done on the UI, but once you start to run your code, please connect to a machine interactively with:

```
srun --partition=short --time=12:00:00 --pty bash -i -l
```

Which will open an interactive session on a machine in the short partition for 12 hours. In this way, the other users on the UI machines will not be disadvantaged by resources being used up by the notebook users.

Warning: If all resources (worker nodes in the selected partition) are in use, the `srun` command will hang until the resource becomes available.

9.2 Virtual environment

Starting with python *virtual environments* called a `venv`, these are a contained python environment you can create and load that has all the python modules and packages installed that the user needs. This ensures no componentes leak into the system environment.

You can create a virtual environment (or venv) at a path by doing:

```
python3.9 -m venv test_venv/
```

This will create a folder called `test_venv` which contains the entire python environment. You can also use other python versions if you prefer. To load this environment run:

```
source test_venv/bin/activate
```

This will show in some shells a `(test_venv)` next to your command line. In the environment you can now install packages using pip:

```
pip install jupyterlab pandas docopt
```

To start a jupyter session, run

```
jupyter lab --ip="*" --no-browser
```

Where the `ip` flag and the `no-browser` respectively ensure that the session is forwarded through the network and that no browser is opened in an `X11` session that may be running through your `ssh` connection.

To properly forward the lab session to your local machine, a second terminal has to be opened running:

```
ssh -NL 8888:wn-db-06:8888 spider
```

where the machine name has to match where the kernel is running (`wn-db-06` has to match) and the forwarded port (in this example `8888`) has to match the port given by the jupyter-lab instance. Again, **do not run notebooks on UI machines**. Now that the tunnel is opened and should forward the connection to your browser, open the link provided by jupyter in your favorite browser. The link has the shape `http://localhost:8888/lab?token=abc123`.

Once you are done with the virtual environment and want to go back to the inital user environment type:

```
deactivate
```

and the python environment is unloaded. To reload the environment again do:

```
source test_venv/bin/activate
```

> **Warning:** Some jupyter instances provide a link of that contains `hostname:8888`. Replace `hostname` with `localhost` or `127.0.0.1` to properly fetch the notebook.

9.3 Singularity container

9.3.1 Pre-built container

To run a notebook in a singularity container, we have to fetch or build the container first. A tutorial on containers can be found in *Building and running a singularity container*, but note that this particular example focuses on using GPUs. A more general introduction is provided here.

First we start by fetching a container:

```
singularity build jupyter.sif docker://jupyter/scipy-notebook:latest
```

This will pull one of the official jupyter containers from docker hub, and build a singularity container from it. This container encapsulates the entire environment and can be entered to start a notebook session. Supported jupyter containers can be found here, and more docker images in general can be found at docker hub.

After the build procedure is complete, you can start the jupyter instance on a worker node (**not** a UI) with

```
singularity run jupyter.sif
```

which will automatically start the instance. Alternatively, you can start an interactive shell session in the container and start it manually:

```
singularity shell jupyter.sif
jupyter lab
```

To receive the notebook locally in your browser, as mentioned above, a tunnel has to be opened in a new terminal, with:

```
ssh -NL 8888:wn-db-01:8888 spider
```

Where, again, the machine name and port name have to match where you are running the job and the port chosen by jupyter, respectively. Now you can open the link provided by jupyter, which has the shape of `http://localhost:8888/lab?token=abc123`.

If the forwarding or other steps do not work, please contact *our helpdesk*.

9.3.2 Custom image

Singularity images can be customised to suit your needs, by adding extra steps during the build process. This is done with so-called 'definition' files. These are plaintext files with instructions for the singularity build. For a full overview, see the singularity documentation. Here is a small example of a custom image that can be expanded. This example also has *docopt* installed during installation, and calling the `singularity run` command opens the container and starts the notebook instance for you. Make a file called `jup-custom.def` and fill it with:

```
Bootstrap: docker
From: jupyter/scipy-notebook:latest

%post
  pip install docopt

%runscript
  jupyter lab --ip=0.0.0.0

%help
  This is a demo container to show how to run jupyter lab
```

You can build this with:

```
singularity build jup-custom.sif jup-custom.def
```

and once it is finished building, you can enter the *sif* file with the `singularity shell` command, or start jupyter directly with `singularity run`. You still have to forward the connection as described above before you can open the notebook in a browser. To save your notebook, in the browser you can use *Save As* from the menu. For more information on running jupyter lab and notebooks, see the official jupyter documentation.

To get a full overview of what is possible during building in terms of installing packages, raising permissions, setting paths, mounting local folders and more, see the official singularity documentation.

9.3.3 Notebook resources

A few resources on prebuilt images and documentation:

https://hub.docker.com
https://docs.sylabs.io/guides/latest/user-guide/

See also:

Still need help? Contact *our helpdesk*

CHAPTER

TEN

BEST PRACTICES

Tip: This is a best practices section to help you achieve maximum performance for your jobs on Spider. In this page you will learn which options are most efficient for:

- your software installation on Spider
- storing your data on the platform
- managing your data on Spider
- running a large amount of jobs on Spider

10.1 Background

Spider is continuously expanding as a unique data processing and collaboration platform. The growing demand for storage space, the diversity in applications and the various ways the system is used, all bring some technical challenges. For example, if you're doing a lot of IO-operations (reading and writing files) in your workflows, you should be mindful on which systems these operations are performed as some options can significantly affect the performance of your jobs and the system load.

Our local file system on Spider is CephFS and is suitable as a staging area for your data before or after analysing it. CephFS hosts both your home and project directories on Spider. It is designed for efficient IO of large files, but when dealing with many small files the file system performance can be degraded. This is because CephFS relies on a parallel distributed system that involves many disks to store the data itself and metadata servers to store the files metadata. As a result, when you operate on many small files e.g. run code from python environments, the system response can become slow for you and other users on the platform.

In order to improve the performance of scientific workflows on Spider's storage, we have prepared this best practices guide that includes several tips to install, store and analyse your data efficiently.

10.1.1 Software installation practices

If your software loads a large number of small files upon execution in your jobs then you may see poor I/O performance even if the total software size is not that big. There are ways to mitigate potential bottlenecks, such as the use of CVMFS or containers technology.

Here is an overview of the features and suitability of some of the software installation options supported on Spider:

Feature	*CephFS*	*Apptainer*	*Lumi Container Wrapper*	*Softdrive*
Software with many files (eg. Conda, pip env)	No	Yes	Yes	Yes
Fast execution times & Low load on the system	No	Yes	Yes	Yes
Extensively used in production	Yes	Yes	No	Yes
Easy to setup	Yes	Moderate	Moderate	Yes
Portability	No	Moderate	Moderate	Yes
Frequent software updates	No	No	Yes	Moderate
Software access can be restricted	Yes	Yes	Yes	No (repos are public)

Which software installation practice should I use?

When your application contains python environments or you seek for a portable way to use your software by multiple (SURF) services we suggest you to install your software on Sofdrive. Softdrive relies on the CVMFS thechnology and uses a caching mechanism on the worker nodes that helps launching your software in shorter times. Here you can find instructions for using *Softdrive*.

An alternative is using Apptainer. Apptainer relies on the container technology that provides an isolated software environment for each application. Apptainer works best when you have a large software stack does not change often. The image can be placed anywhere on Spider, as long as the location is accessible to your processing jobs. Examples can be found in our *Apptainer* section.

For software that changes frequently we suggest you the LUMI Container Wrapper (LCW). This is similar to Apptainer but you can update the software without rebuilding the base container. More about LCW can be found in our *LUMI* section.

In cases that you have to install your software locally on Spider and it loads a limited number of files, it is possible to use CephFS on home or project space locations, but take into account its limitations such as slow execution times.

10.1.2 Data storage practices

When you work with large volume of data or your application writes/reads a large number of files then you may encounter performance bottlenecks depending on where you have stored your data.

Here is an overview of the features and suitability of some of the data storage options supported on Spider:

Feature	*CephFS*	*dCache*	*Scratch*
High throughput & low load on the system	No	Yes	Yes
Large volumes of data	No	Yes	Moderate
Data available after jobs end	Yes	Yes	No
Data available outside Spider	No*	Yes	No
Granular access control	Yes	Yes	No
Supports disk	Yes	Yes	Yes
Supports tape	No	Yes	No
Available through an API	No	Yes	No

- *unless explicitly placed in public folder*

Which data storage practice should I use?

For bulk data storage we recommend dCache. dCache is highly connected to Spider worker nodes and is designed for high-throughput processing of data. This storage system is also available outside of Spider, and has highly granular access controls, making data releases, or data uploader roles self-service. dCache is available through a number of interfaces, meaning that it can be used out of the box with WebDAV clients or through a REST API, allowing for future data portals to be developed. Another reason to use dCache is that it supports both disk and tape, meaning that it can easily scale to much more data. Here you can find instructions for using the *dCache remote storage*.

We also advice you to use the scratch file systems as fast temporary storage while running a job. Each of the Spider worker nodes has a large scratch area on local SSD. Any data that you wish to keep should be written to other storage backends such as dCache before the end of the job. The scratch areas are ideal for retrieving the input of a job from dCache during execution or for applications that generate lots of intermediate files that are consumed by other parts of the processing or for generating the job output before copying it back to dCache. More about how to use the temporary disk space can be found in our section *Using scratch*.

In cases that you have multiple jobs that need to access a single set of files that is too large to copy over to scratch, it is possible to use CephFS on home or project space locations for temporarily storing your data, but take into account its limitations compared to dCache in terms of throughput and capacity. It is highly recommended that you do not store more than *10,000* files in a single directory on CephFS. In terms of file sizes, CephFS is most efficient when you deal with files that are larger than *4MB*. Files that are less than *32KB* can be very inefficient.

10.1.3 Managing data practices

There are several data management options for all stages of your project lifecycle. Here we focus on the data managing options for transferring and parsing your data on Spider.

An overview of the features and suitability of some of the managing data options supported on Spider is presented below.

Feature	*Rclone*	Shared memory	mpifileutils
High speed & low load on the system	Moderate	Yes	Yes
Support for parallel operations	Yes	Yes	Yes
Easy setup	Yes	Yes	No
Supports many backends (Object Store, dCache)	Yes	No	No

Which practice for managing data should I use?

When transferring data from/to Spider your experience will vary depending on the client, protocol and parameters you choose. For efficient data transfers we suggest you to use Rclone. Rclone is a command line tool that works on many platforms and it can talk to many storage systems, including dCache. Some advantages of Rclone are that it can sync directories, like rsync does, and it uses parallel transfers, 4 by default, to get a better performance when copying directories. More information about using Rclone, for example with dCache, can be found in our *ADA interface* section.

When you need to tar or zip many small files on Spider, this can be very slow on the local CephFS filesystem and can take several hours. In such cases it may be better to copy the files temporarily in memory (RAM) first and then use tar/zip, as it will speed up these operations remarkably. The files are copied from CephFS into memory in a parallel way, while tar operates on files one by one. Once the files are in the shared memory of the node, the tar process is a lot faster. When using this option please keep in mind that memory is limited and shared with other processes and that it is temporary. An example for using the shared memory to tar and process a file can be found in Shared memory.

For advanced users, who are familiar with MPI operations, we also offer an a MPI-based tool for managing datasets such as copying files across the different home and project space folders on the local file system. The MPI-based tool is

much faster and efficient than the common *cp* operations. Example usage for parallel copying of files using this method can be found in the mpifileutils section.

10.1.4 Running a large amount of jobs

High-throughput workflows that execute a specific application for many different parameter combinations, often requires the submission of many jobs. When running a large amount of jobs it can be difficult to keep track of the status of these jobs or resume failed jobs that were prematurely canceled (e.g. due to time limit). Another challenge is reducing the large scheduling overhead and waiting times in the queue.

An overview of the features and suitability of some of the options for running a large amount of jobs on Spider is presented below.

Feature	Slurm job arrays	*PiCaS*	*Snakemake*
High speed & low load on the system	No	Yes	Moderate
Scales to hundreds, thousands of jobs and more	No	Yes	Moderate
Transcends spider	No	Yes	No
Easy setup	Yes	Moderate	Moderate
Handles easily dependencies between tasks	No	Moderate	Yes
Error recovery	No	Yes	Moderate

Which practice for running a large amount of jobs should I use?

The first option to check when running a large amount of jobs is whether the software you're using comes with a built-in option for managing your workloads on a Slurm-based cluster. Alternatively, an easy way to submit several independent jobs with one command is the use of Slurm job arrays. Job arrays, however do not scale well for more than a few hundreds of jobs. In this case, you can use external tools for managing your workloads, such as PiCaS or Snakemake.

PiCaS works as a queue, providing a mechanism to step through the work one task at a time. It is also a pilot job system, indicating that the client communicates with the PiCaS server to fetch work, instead of having that work specified in a job (or similar) file. As every application needs different parameters, PiCaS has a flexible data structure that allows users to save different types of data. PiCaS can handle thousands or millions of tasks, it has an easy query mechanism to search among your tasks and is accessible from any platform via a Restful HTTP API. Here you can find instructions for using *PiCaS*.

When your application involves several steps connected in a workflow that each need to be submitted as independent tasks, you may consider using *Snakemake*. Snakemake is a python-based workflow managment tool for defining, managing and executing workflows with multiple steps and complex dependencies. There are possibilities to combine PiCaS and Snakemake to enable workflow automation and run many jobs and subtasks efficiently and fast. Please contact our *our helpdesk* if you need help with automating your workloads on Spider.

CHAPTER

ELEVEN

MONITORING SPIDER

There are two types of monitoring dashboards for Spider, a public section visible for everyone and a section visible only to authenticated Spider users. Both are shortly described below. First some general properties and settings are covered.

At the top of each of these dashboards is a time selection drop down to change the time range of the graph on the x-axis, along with a refresh button and refresh timer. Below these buttons are the actual properties of Spider being shown, which is either a single number of a property, for example current RAM usage, or a time-dependent graph of a property, for example CPU usage over time. If a panel has a legend, a single legend entry can be selected by clicking on it, which filters out the rest. It can be deselected by pressing it again.

11.1 Public dashboard

The public dashboard can be found here and shows 4 views of running jobs, pending jobs, queue times and reserved number of cores for the cluster for different projects.

11.2 User dashboards

The user dashboards show information at different levels, and can be found here. The user dashboards include:

- Spider Cluster overview
- Spider Node exporter
- Spider SLURM dashboard

and are described below.

11.2.1 Spider Cluster Overview

The cluster overview shows the general status of the nodes in Spider and for each of the machines it can be seen what the load and network traffic is over time.

11.2.2 Spider Node exporter

On this extensive dashboard you can select at the top each machine in Spider under "host" and get a general overview at the top of the page. Below, in a variety of panels (note that these can be folded) additional detailed system information is available for the selected node. This information ranges from CPU and memory usage to storage IO and disk usage to network traffic statistics.

11.2.3 Spider SLURM dashboard

This board shows at the top the general job submissions and pending jobs on SLURM, node status and CPU allocation. Next, the running and pending jobs are visible per project and per user. Finally, a section is shown containing per machine, the load, memory usage, disk usage and network IO traffic.

See also:

Still need help? Contact *our helpdesk*

CHAPTER

TWELVE

MAINTENANCES

Tip: To keep the system healthy we need sometimes to interrupt the platform for maintenance. In this page you will find:

- our service notices logging changes or incidents on the platform
- the current status of the platform & the maintenances log
- our downtime policy

12.1 Service notices

To make it easier for users to see what notable changes have been made in the system between maintenances and releases of SPIDER we keep a chronologically ordered list of notable changes and incidents. The service notices can be found in service-notices-list.

12.2 Maintenance windows

12.2.1 Current status

The current status of Spider can be found in our service status page here.

12.2.2 dcache maintenances

The dCache instance of SURF falls under the Grid components, thus dCache maintenances are not announced via the Spider notifications. If you want to receive notifications and be prepared for upcoming downtimes and maintenances of dCache, you can create personal subscriptions in the EGI operations portal by following the instructions in the Grid page here.

12.3 Policy

All SPIDER users are notified in advance for the scheduled maintenance windows. Exceptions to this policy are made in the following cases:

- Urgent changes are needed for the continuity of the platform
- Patches for high-risk security issues

In these cases, the changes will be applied as soon as needed, even outside maintenance windows.

See also:

Still need help? Contact *our helpdesk*

CHAPTER

THIRTEEN

GETTING HELP

Tip: Perhaps you are puzzled with something and seek for help on SPIDER. In this page you will find:

- our Helpdesk details
- how to get expert advice for project challenges
- the most commonly asked questions and workarounds for known issues
- some tutorial as prerequisites to use the platform

13.1 Helpdesk

If you run into problems and cannot find an answer in our *Spider User Guide* or the *FAQ*, we encourage you to contact our experts who are there to support the SPIDER users. Send us an email with your question at: helpdesk@surfsara.nl

Find more about SURFsara support availability and response times here.

13.2 Expertise

Our support team is specialized in advising on data processing strategies and collaborative data analysis. Over the years the team has gained experience in many unique projects, all requiring a unique combination of generic components to implement their ideal data processing setup. Our team is available to support a range of proven solutions for each problem:

- off-the-shelf data staging processes from a variety of (external) storage systems
- off-the-shelf processing orchestration processes
- specialized consultancy for automated production pipes

13.3 FAQ

13.3.1 I cannot login after adding/changing my ssh keys

If you added or changed your SSH key on the SURFsara portal, access to the login node may not be immediately possible. It can take *up to 15 minutes* to be able to login to SPIDER. In case that it takes longer time, please contact us at *helpdesk*.

13.3.2 The FACL permissions on my file do not match the folder the files are stored in

This type permissions mismatch can occur when `mv` is used, as the move command keeps the original ACL entries from the original file. This behaviour is part of `mv` and can not be changed. The permission mismatch occurs when a file with no extra ACL entries is moved to a folder that has some non-trivial ACL entries set up. Using the `cp` command will allow the user to preserve the original ACL entries or accept the ACL entries from the destination folder to be enforced onto the copied files. This will ensure that no such permission mismatch occurs. Similarly, the `rsync` command will also allow the user to control what kind of ACL is applied to the files in the destination.

13.3.3 My apptainer container is failing to build on a worker node

This is a known bug triggered when a user is allowed to use `--fakeroot` and uses fakeroot in a Slurm job to build the container. The error you expect in this case is the container is allowed to write in `/tmp` during building. To avoid this issue, build the container on the UI machines instead.

13.4 Tutorials

- Tutorial for Introduction to Unix
- Tutorial for Introduction to batch scheduling systems
- Tutorial for Slurm Quickstart

See also:

Still need help? Contact *our helpdesk*

The Frequently Asked Questions can be found at Section 13.3.

Other info:

CHAPTER

FOURTEEN

DOCUMENTATION HOW-TO

Tip: Do you have ideas for corrections and improvements of our user documentation? In this page you will learn how to:

- contribute to our documentation
- edit the docs with Sphinx language
- build the docs on your laptop

Our documentation is hosted on Github and is written in Sphinx *restructed text*. Behind the scenes we use ReadTheDocs to publish it automatically. You can contribute either directly to our Spider Github repo or send an email to *our helpdesk* with your remarks and we will change the documentation ourselves. Any contribution is welcome!

The rest of this page explains how to to submit your changes directly through Github.

14.1 Contribute through GitHub

In case that you have a GitHub account and a little knowledge of git (see GitHub's git cheat sheet), you can try submitting your changes directly to our repository. Here is what you have to do:

1. Fork our Spider Github repo
2. Git pull your fork
3. *Edit with Sphinx language* the files with your changes
4. *Build the documentation locally* to preview the changes
5. Commit and push your changes back to your fork
6. Create a pull request to inform us of your changes
7. After we've reviewed and accepted your work, we will merge your commits and the documentation will be updated automatically

14.2 Edit with Sphinx language

When you contribute directly to our Github repo we ask you to write the changes in Sphinx language. The philosophy of Sphinx documentation is that content is stored in files that can be easily read *and* edited by humans, in a format called *restructured text*, with the file extension `.rst`. Using a simple grammar, text can be styled. The document is structured using special tags; using these tags, documentation can be split into multiple files, and you can cross-reference between files and build indexes.

Although Sphinx is quite intuitive, we have created a simple Sphinx cheatsheet to help you use the Sphinx syntax:

14.2.1 Cheatsheet

Look at the raw version of this file to compare the source and build version: `sphinx_cheatsheet.rst`

The first lines of this page will also be explained later on in this document (see *Links* and *Titles*)

Moderate emphasis

Strong emphasis

`This is text in Courier`

This is plain text.

- Use `literal markup` for:
  - pieces of code
  - commands
  - arguments
  - file names, hostnames
  - a specific term, when emphasis is on its name
  - configuration file statements and values
- Acronyms:

In general: avoid acronyms. When you want to use them, the first occurrence on a page should explain them: CephFS (Ceph Filesystem). Sphinx supports a :abbr: tag, see http://www.sphinx-doc.org/en/stable/markup/inline.html#other-semantic-markup. Here's a test: CephFS (Ceph Filesystem)

- This is a bulleted list.
- …

1. This is a numbered list.
2. It has two items.

- this is
- a list
  - with a nested list
  - and some subitems
- and here the parent list continues

Python

or

SURFsara website (see bottom of the document; that is were we tell Sphinx were SURFsara website should point to)

This is an implicit link to title:

Sample H2

Internal wiki link:

Reference tag: place above a title: .. _my-reference-label:

Then refer to it from another page as. For example, for this cheatsheet: *Cheatsheet* or ref:*other label <cheatsheet>*

14.2.2 H1: document title

Sample H2

Sample H3

Sample H4

Sample H5

Sample H6

And some text.

Header 1	Header 2	Header 3
body row 1	column 2	column 3
body row 2	Cells may span columns.	
body row 3	Cells may span rows.	• Cells • contain
body row 4		• blocks.

or

Column1	Column2
value1	40
value2	41
value3	42

Warning: This is a **warning** box.

Note: This is a **note** box.

Tip: This is a **tip** box.

Error: This is an **error** box.

See also:

This is a simple **seealso** note.

Your Topic Title

Subsequent indented lines comprise the body of the topic, and are interpreted as body elements.

Sidebar Title

Optional Sidebar Subtitle

Subsequent indented lines comprise the body of the sidebar, and are interpreted as body elements.

`path/to/myfile.txt`

`A file for download`

- When you want to display commands, output and comments, use `.. code-block:: bash`. Prefix commands with a $ and prefix comments with ##, otherwise they are marked up as a command. Example:

```
$echo 'Hello World!'
Hello World!
## Comments should be prefixed with a double ``#``.
```

- When you want to display commands and comments, use `.. code-block:: bash`. Don't prefix commands. Example:

```
# [homer@htp-ui ~]$ is the first prompt upon login
ssh [USERNAME]@[Spider HOSTNAME]
```

- You can add line numbers to code examples with the :linenos: parameter.

```
1 # [homer@htp-ui ~]$ is the first prompt upon login
2 ssh [USERNAME]@[Spider HOSTNAME]
```

- To display the contents of a shell script, use `.. code-block:: bash`.
- To display configuration files, use `.. code-block:: cfg`.

14.3 Build the documentation locally

Because the syntax of the files is human readable, you can edit the files using your favorite text editor. Once you are done editing, you can generate documentation in various formats, such as HTML or epub. While you can edit the pages on virtually any system, it is recommended to preview your changes before publishing them.

There are different ways to generate the HTML documentation from source and review your changes:

- *Docker image*
- *Sphinx local installation*
- *GitHub edit/preview*

Note that you only need to use one of the options mentioned above. Using Docker is the preferred way, as this mimics the ReadTheDocs build system closest. GitHub edit/preview on the other hand is good enough for minor, textual changes, but is otherwise the least preferred option.

Below you will find information for each of the methods.

14.3.1 Docker image

This is the preferred option to build and test your changes. It tries to build the documentation the same way as readthedocs.org.

Setup the readthedocs Docker image

Preparation

Before you start with the Docker image, make sure that:

- You have Docker Engine installed
- Docker is running on your host

Build image

The official Docker image can be found in the readthedocs repository: https://hub.docker.com/r/readthedocs/build/

To pull the image to your laptop, run:

```
docker pull readthedocs/build
```

- Once the Docker image is ready, find the following script inside your Github fork and run it to build your documentation:

  ```
  ./build.sh
  ```

Optionally you can provide an output location (default: ./build) and the Docker image name (default: readthedocs/build):

```
./build.sh /alternative/output/path/ docker_image_alternative_name
```

Note: For Mac OS X, use `./build_mac.sh` instead.

14.3.2 Sphinx local installation

For the Sphinx documentation setup locally you will need to:

Install Sphinx

Spider documentation itself is written in *restructured text*. This page explains how to install the Sphinx software on your local machine.

Linux

On recent CentOS distributions, *Sphinx* is included in epel. Fedora has it included in the standard distribution. Install *sphinx* with:

```
sudo yum install python-sphinx
```

On Ubuntu, install with:

```
sudo apt-get install python-sphinx
```

Mac OS X

On Mac OS X, you will need macports, so get that if you have not done so.

Next, follow the instructions on the sphinx installation page.

Warning: the standard terminal in Mac OS X has locale settings incompatible with sphinx; to change it once for a terminal session, type:

```
export -n LC_CTYPE
```

To change it permanently:

1. open the terminal preferences
2. go to the tab 'advanced'
3. uncheck "Set locale environment variables on startup"

- To generate HTML documentation, use the command:

```
make html
```

which will generate static pages in the `build`-directory as long as you have the software Sphinx installed locally.

14.3.3 Github edit/preview

For small changes you can edit a page directly from your GitHub fork webview. The *preview* button does not give a fully compatible *rst* overview, but is sufficient for textual changes.

See also:

Still need help? Contact *our helpdesk*

CHAPTER

FIFTEEN

COOKIEBELEID

15.1 Wat is een cookie?

Wij maken op deze website gebruik van cookies. Een cookie is een eenvoudig klein bestandje dat met pagina's van deze website wordt meegestuurd en door uw browser op uw harde schrijf van uw computer wordt opgeslagen. De daarin opgeslagen informatie kan bij een volgend bezoek weer naar onze servers teruggestuurd worden.

15.2 Google Analytics

Via onze website wordt een cookie geplaatst van het Amerikaanse bedrijf Google, als deel van de "Analytics"-dienst. Wij gebruiken deze dienst om bij te houden en rapportages te krijgen over hoe bezoekers de website gebruiken. Google kan deze informatie aan derden verschaffen indien Google hiertoe wettelijk wordt verplicht, of voor zover derden de informatie namens Google verwerken. Wij hebben hier geen invloed op. Wij hebben Google niet toegestaan de verkregen analytics informatie te gebruiken voor andere Google diensten. De informatie die Google verzamelt wordt zo veel mogelijk geanonimiseerd. Uw IP-adres wordt nadrukkelijk niet meegegeven. De informatie wordt overgebracht naar en door Google opgeslagen op servers in de Verenigde Staten. Google stelt zich te houden aan de Safe Harbor principles en is aangesloten bij het Safe Harbor-programma van het Amerikaanse Ministerie van Handel. Dit houdt in dat er sprake is van een passend beschermingsniveau voor de verwerking van eventuele persoonsgegevens.

15.3 In- en uitschakelen van cookies en verwijdering daarvan

Meer informatie omtrent het in- en uitschakelen en het verwijderen van cookies kan je vinden in de instructies en/of met behulp van de Help-functie van jouw browser.

15.4 Meer informatie over cookies?

Op de volgende websites kan je meer informatie over cookies vinden:

- Cookierecht.nl
- Consumentenbond: "Wat zijn cookies?"
- Consumentenbond: "Waarvoor dienen cookies?"
- Consumentenbond: "Cookies verwijderen"
- Consumentenbond: "Cookies uitschakelen"
- Your Online Choices: "A guide to online behavioural advertising"

Happy Spidering!..

Tip: Need help? Contact *our helpdesk*