Custom Softwares on SubMIT: OpenMPI and Globus

Xuejian(Jacob) Shen
SubMIT project team

SubMIT Workshop, Jan 2026

OpenMPI

The Open MPI Project is an open source Message Passing Interface implementation that is developed and maintained by a consortium of academic, research, and industry partners.

Message Passing Interface (MPI)?

A standardized and portable message-passing system for **parallel computing**. Designed for **distributed-memory architectures** (e.g., clusters, supercomputers).

Indexing and managing processes, sending & receiving messages between processes, collective operations (broadcast, reduction, gather …)

OpenMPI

Centralized Implementation on SubMIT

Open MPI 4.1.1
available through the “module” system
e.g. module load mpi

Example usage

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --nodes=3
#SBATCH --ntasks=12
#SBATCH --cpus-per-task=1
#SBATCH --time=00:20:00
#SBATCH --mem-per-cpu=100

module load mpi
mpirun -n 12 ./hello_c
mpirun -n 12 python hello.py
```

“mpi4py” or equivalent python library needs to be installed (& configured to “know” the existing MPI library) for running python codes

OpenMPI

Centralized Implementation on SubMIT

Open MPI 4.1.1
available through the “module” system
e.g. module load mpi

Example usage

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --nodes=3
#SBATCH --ntasks=12
#SBATCH --cpus-per-task=1
#SBATCH --time=00:20:00
#SBATCH --mem-per-cpu=100

module load mpi
mpirun -n 12 ./hello_c
mpirun -n 12 python hello.py
```

Default MCA parameters

Byte-transfer layer:
btl = ^openib, ofi
(we do not yet have openFabrics interfaces; so disabled by default)

Point-to-point messaging layer:
pml = ucx
(“ucx”, a communication library to support our RoCE networking devices)

OpenMPI

Centralized Implementation on SubMIT

User guide

Table of Contents

Previous topic

Conda and its benefits beyond python

Next topic

OpenMPI on SubMIT

Tags: OpenMPI

The Message Passing Interface (MPI) is a standardized and widely used communication protocol designed for parallel computing in distributed-memory systems. It enables processes running on different nodes of a cluster to exchange data efficiently, making it essential for high-performance computing (HPC) applications. OpenMPI is a popular open-source implementation of the MPI standard, offering high flexibility, scalability, and performance optimizations. Developed collaboratively by the HPC community, OpenMPI supports multiple network interfaces and integrates seamlessly with modern supercomputing environments. Its modular architecture allows users to tailor configurations for specific hardware, making it a preferred choice for researchers and engineers running large-scale simulations, numerical computations, and machine learning workloads.

Here we briefly introduce the OpenMPI installation on SubMIT and provide some examples of running your program with it.

Centralized installation of OpenMPI on SubMIT

We provide a centralized OpenMPI installation through the `module` system. One can load the OpenMPI module using:

```
module load mpi
```

you can check if it is successfully loaded by running:

```
module list
mpirun --version
```

OpenMPI

Centralized Implementation on SubMIT

User guide — Running a simple C/C++ program

Running MPI Programs in C/C++

Here is a simple example of an MPI program in C (*mpi_hello.c*):

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello from process %d out of %d\n", rank, size);
    MPI_Finalize();
    return 0;
}
```

Compile the code using:

```
mpicc -o mpi_hello mpi_hello.c  # For C
mpicxx -o mpi_hello mpi_hello.cpp  # For C++
```

Run the program with:

```
mpirun -np 4 ./mpi_hello
```

OpenMPI

Centralized Implementation on SubMIT

User guide — Running a python program with mpi4py

Running MPI Programs in Python

To use OpenMPI with Python, one can install `mpi4py` or equivalent python packages. We recommended you to install it with `conda` using the following command:

```
conda install -c conda-forge mpi4py openmpi=4.1.*=external_*
```

The last part of the command specifies that the system-provided OpenMPI libraries will be used. Otherwise, `conda` will try to install its own version of OpenMPI or other MPI distributions (which usually will work as well, but there is no guarantee that they are optimized as the system-provide one).

Please also install the `ucx` libary (as the necessary point-to-point messaging layer):

```
conda install -c conda-forge ucx
```

Here is a simple Python MPI example (*mpi_example.py*):

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

print(f"Hello from process {rank} out of {size}")
```

Run the script using:

```
mpirun -np 4 python mpi_example.py
```

OpenMPI

Customized OpenMPI in Containers — For HTCondor

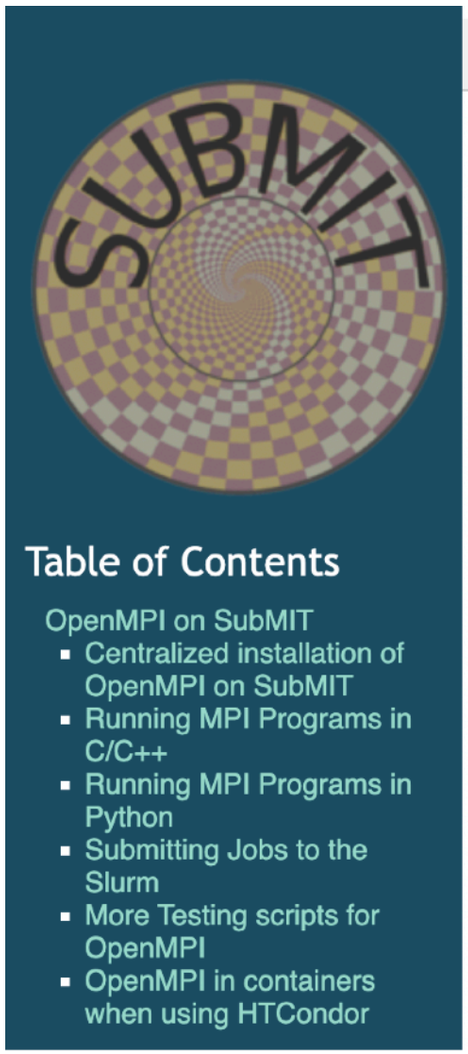


Table of Contents

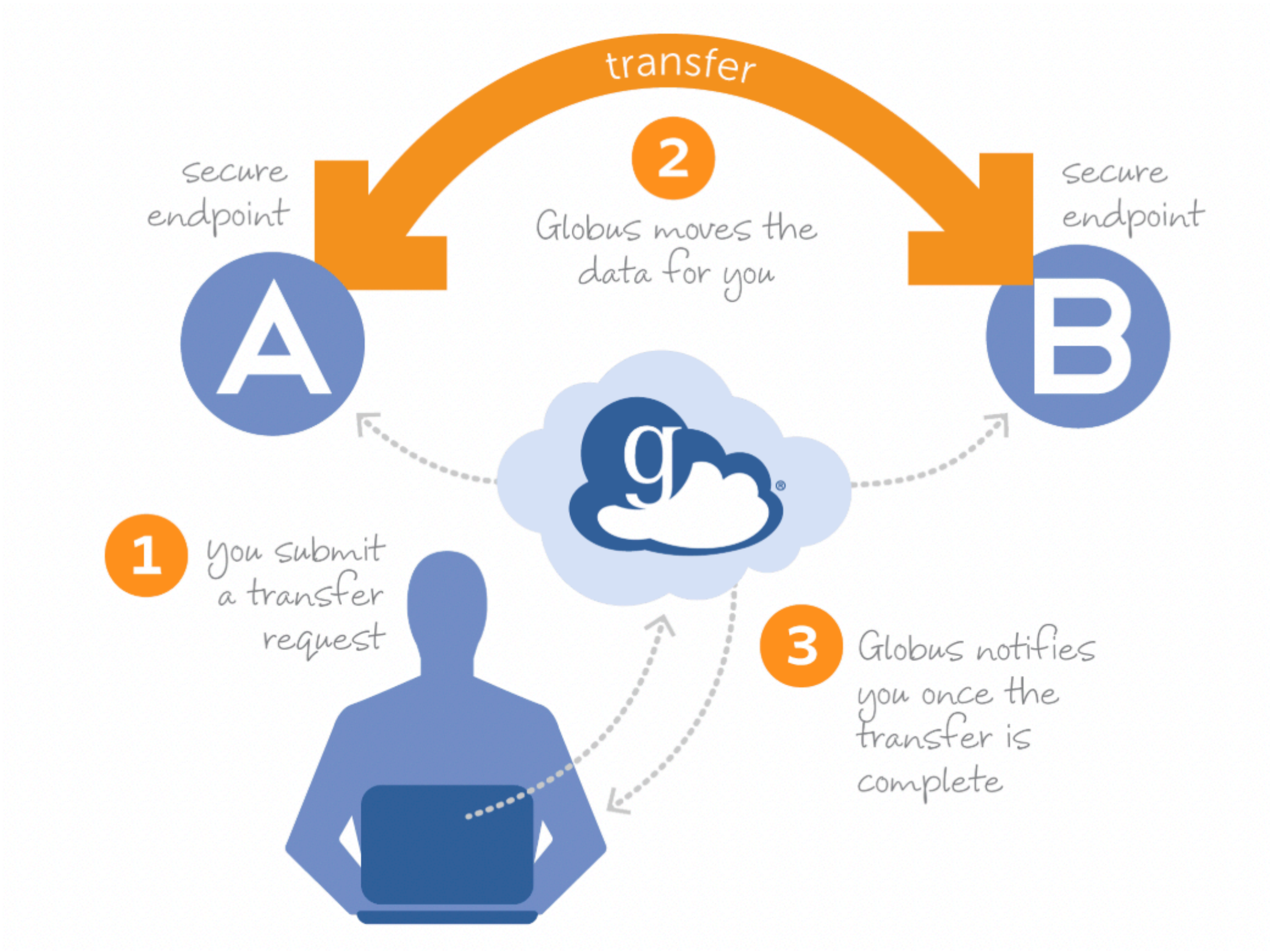
OpenMPI in containers when using HTCondor

This part walks you through running OpenMPI applications on an HTCondor-managed external cluster. Unfortunately, in this case, we cannot use the centralized OpenMPI installation, since that is only available on SubMIT clusters. We will here show how to run some testing scripts as an example. We need to first create a container image that includes OpenMPI and the necessary libraries. Our `mpi.def` contains:

```
Bootstrap: docker
From: centos:7

%post
    # Redirect repos to vault.centos.org
    sed -i 's|^mirrorlist=|#mirrorlist=|g' /etc/yum.repos.d/CentOS-Base.repo
    sed -i 's|^#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g' /etc/yum.repos.d/C
    yum clean all

    yum install -y openmpi openmpi-devel hwloc numactl

    export CFLAGS="-march=core2 -mtune=generic -mno-avx -mno-avx2 -mno-sse4"
    /usr/lib64/openmpi/bin/mpicc $CFLAGS -o /usr/local/bin/hello_c /hello_c.c
    /usr/lib64/openmpi/bin/mpicc $CFLAGS -o /usr/local/bin/ring_c  /ring_c.c

%files
    hello_c.c /hello_c.c
    ring_c.c  /ring_c.c

%environment
    export PATH=/usr/lib64/openmpi/bin:$PATH
    export LD_LIBRARY_PATH=/usr/lib64/openmpi/lib:$LD_LIBRARY_PATH
```

Globus

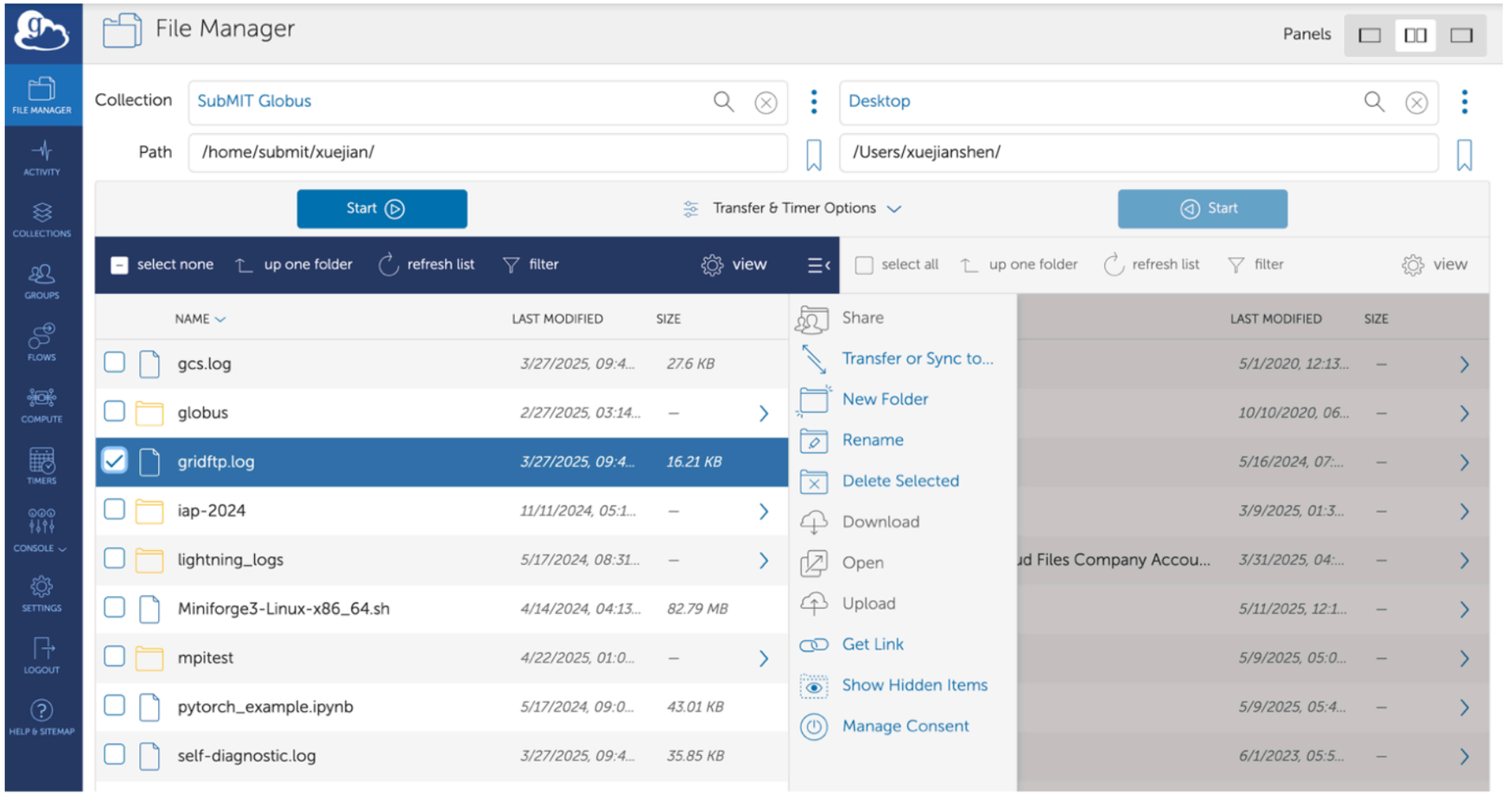
For efficiently, securely, and reliably transferring data directly between systems.



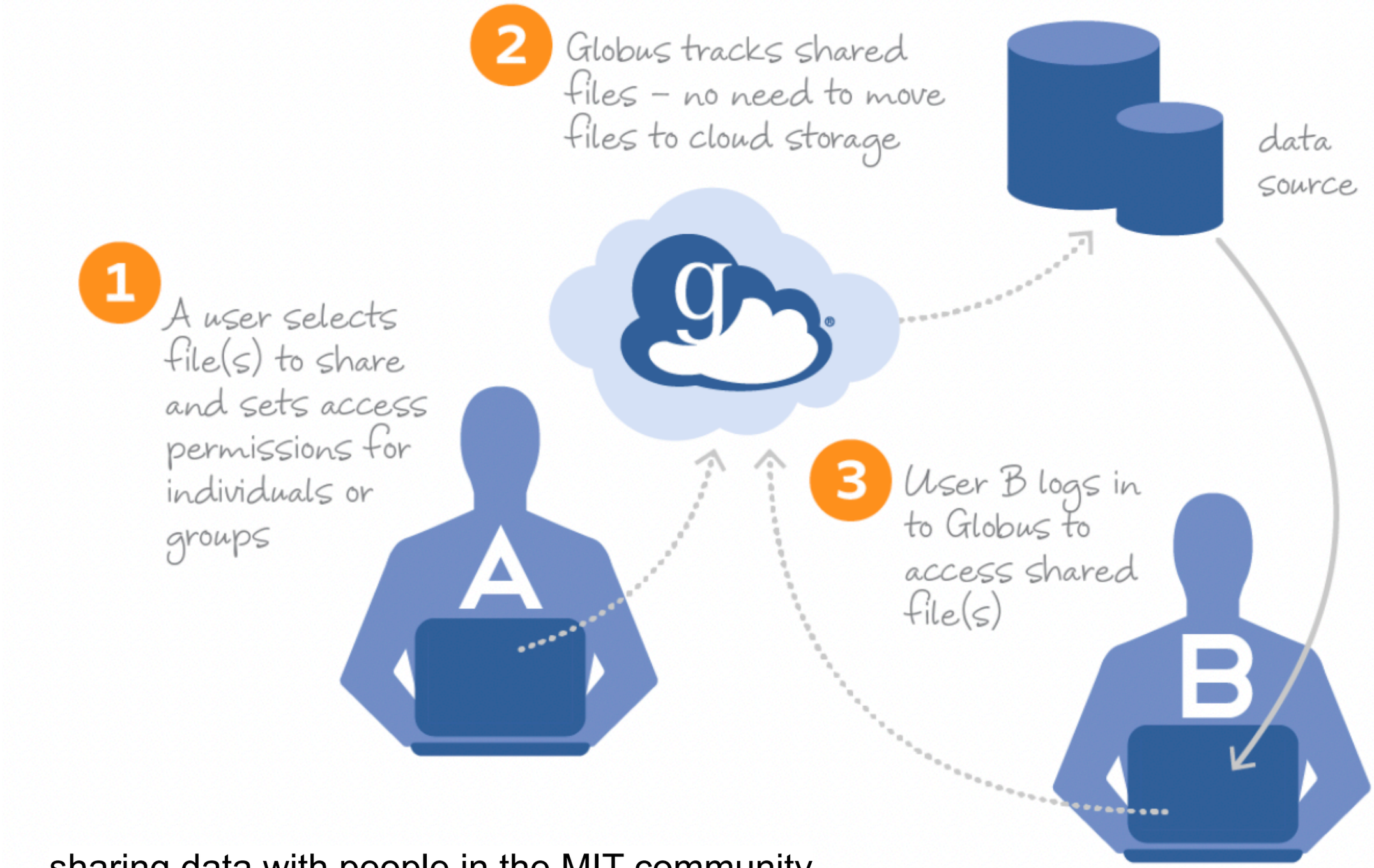
Globus

Why Use Globus?

- Globus is a fast, secure, and reliable way to move MB's, TB's and even PB's of data.
- The basic Globus Transfer service is free to use for non-profit organizations, and subscriptions are reasonably priced to enable access to premium features.



Globus



sharing data with people in the MIT community & guests (through guest collections)

Globus

Why Use Globus to Share Data?

As a researcher:

- ☑ Share data with anyone in your institution, or in other institutions
- ☑ No local access is required to share data with a collaborator
- ☑ You no longer need to upload and move data to the cloud
- ☑ Send an email to collaborators and let them know there is data they can retrieve

Globus

User guide

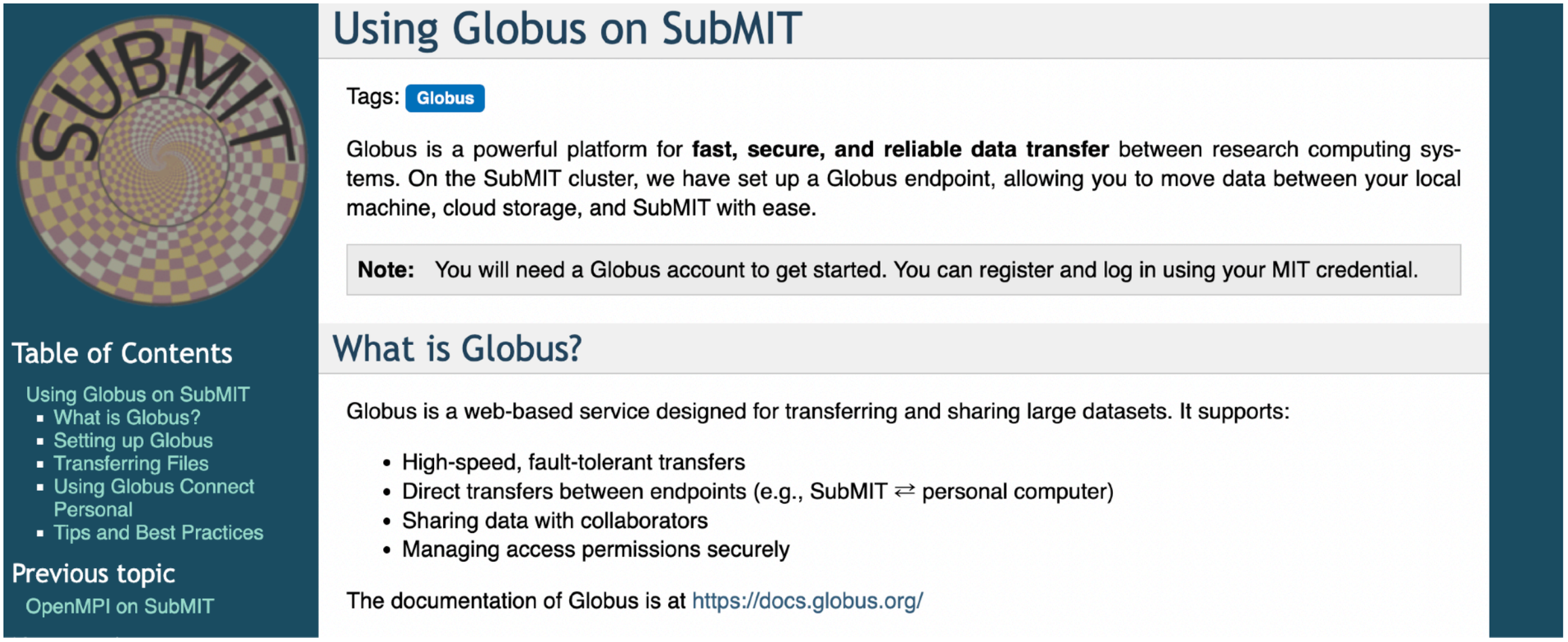


Table of Contents

Using Globus on SubMIT

Tags: Globus

Globus is a powerful platform for **fast, secure, and reliable data transfer** between research computing systems. On the SubMIT cluster, we have set up a Globus endpoint, allowing you to move data between your local machine, cloud storage, and SubMIT with ease.

Note: You will need a Globus account to get started. You can register and log in using your MIT credential.

What is Globus?

Globus is a web-based service designed for transferring and sharing large datasets. It supports:

- High-speed, fault-tolerant transfers
- Direct transfers between endpoints (e.g., SubMIT ⇄ personal computer)
- Sharing data with collaborators
- Managing access permissions securely

The documentation of Globus is at https://docs.globus.org/

Summary

OpenMPI

— Centralized implementation
— Customized version through containers

Globus

— Fast, secure, reliable data transfer
— Data sharing