# Globus Compute Overview

eResearch Australia 2025

THE UNIVERSITY OF CHICAGO

# Globus in a nutshell

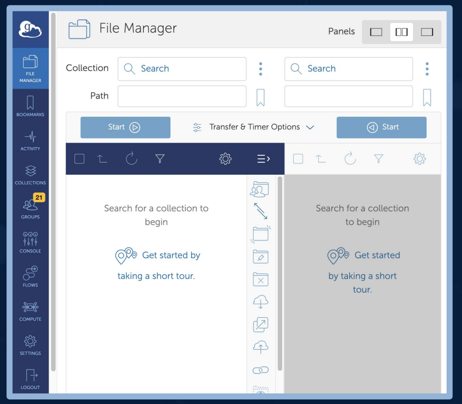

## Managed transfer & sync


## Publication & discovery

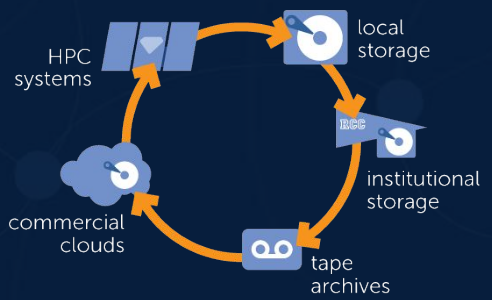

## Platform-as-a-Service


## Collaborative data sharing


## Managed remote execution



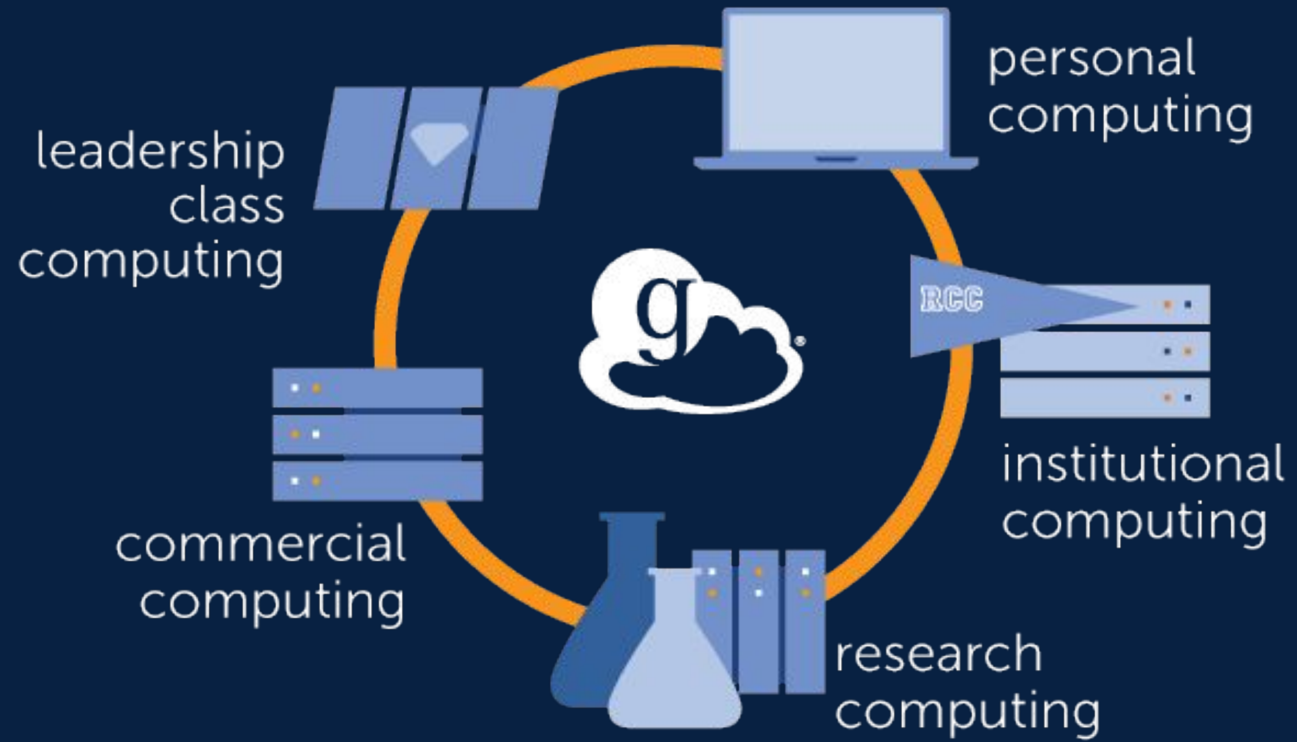

## Software-as-a-Service




## Unified data access


## Reliable automation

# Globus Compute



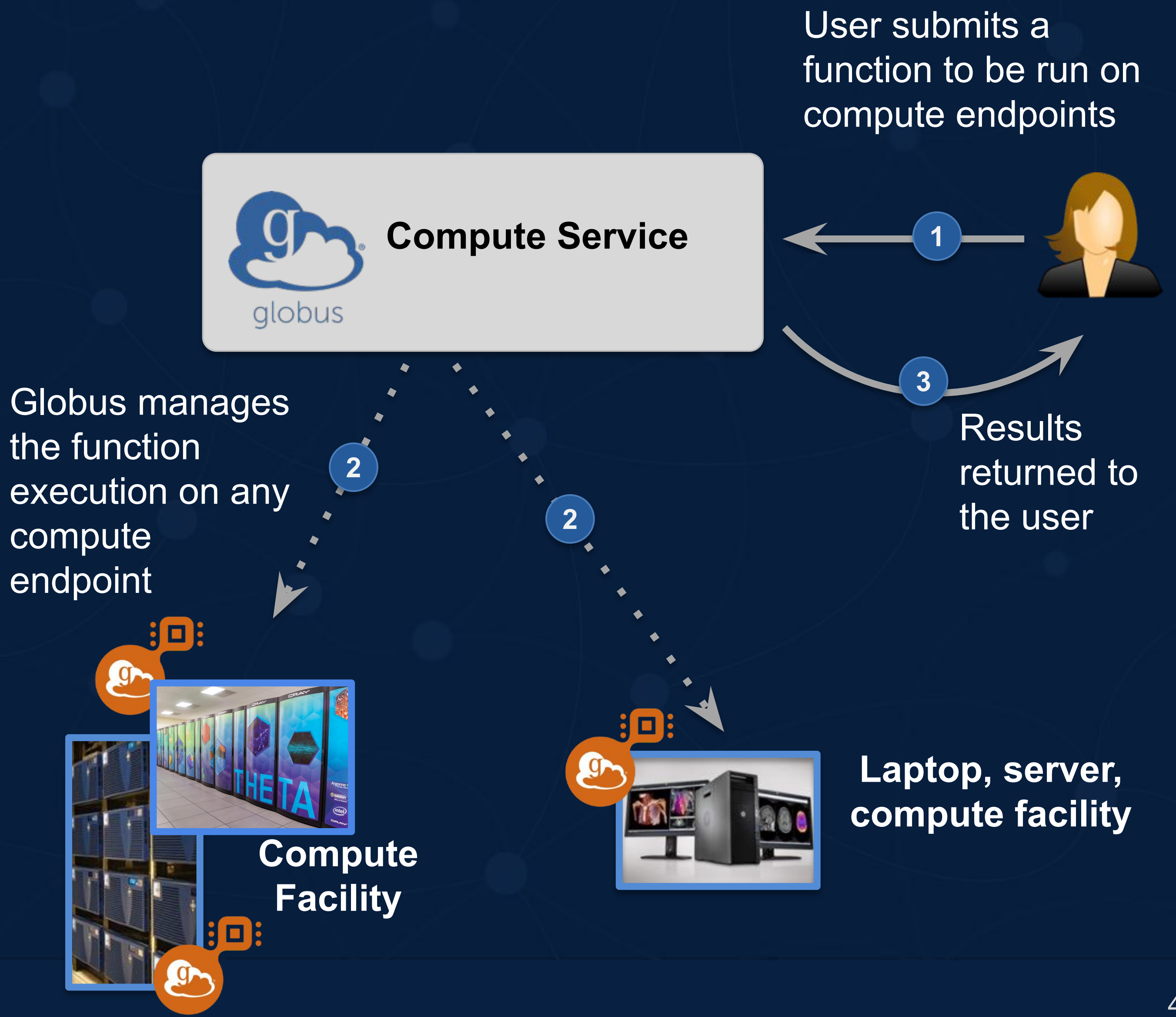

**Managed remote execution**

Unified data access

Reliable automation

# Managed compute …on any system

- Support use of Python for functions
- Fire and forget function execution
- Federated authentication, and local access control
- Uniform interface to various compute resources



# How are researchers using Globus Compute?

# Interactive computing via familiar tools

- No terminal login required, or scheduler specific scripts
- Define function like any other code, and code is run on compute resource
- Use with JupyterNB, Python applications

```python
# Define the function for remote execution
def hello_name(name: str):
    return f'Hello, {name}!'

from globus_compute_sdk import Client, Executor

MY_COMPUTE_ENDPOINT = "ad4f48be-9c03-49bc-9dc4-e240bc599bef"
compute_executor = Executor(endpoint_id=MY_COMPUTE_ENDPOINT)

# Run the function on the remote compute resource
my_result = compute_executor.submit(hello_name, 'you')

print(my_result)
```

# Reliable repeated task management

```python
# Function that estimates pi
def pi(num_points):
    from random import random
    inside = 0

    for i in range(num_points):
        x, y = random(), random()
        if x**2 + y**2 < 1:
            inside += 1
    return (inside*4 / num_points)

# Register function with  Globus Compute
from globus_compute_sdk import Client

gc_client = Client()
my_function = gc_client.register_function(pi)

# Execute the function N times
N = 1000
estimates = [
    gce.submit(pi, 10**5)
    for _ in range(N)
]

# Get the results and calculate the total
results = [est.result() for est in estimates]
```

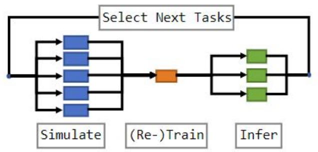
- **Manage bag of tasks**
- **No need to manage process on login node**
  - Reliable outsourcing of task management
  - Automated retry on certain failures
- **No special tools required to access compute resource**

# Example: AI-enabled bag of tasks

## Molecular design ML-in-the-loop workflow

This notebook demonstrates an increasingly commmon ML-in-the-loop molecular design application. We use ML to guide the choice of simulations to perform. The objective of this application is to identify which molecules have the largest ionization energies (IE, the amount of energy required to remove an electron).

IE can be computed using various simulation packages (here we use xTB ); however, execution of these simulations is expensive, and thus, given a finite compute budget, we must carefully select which molecules to explore. We use machine learning to predict high IE molecules based on previous computations (a process often called active learning). We iteratively retrain the machine learning model to improve the accuracy of predictions. The resulting ML-in-the-loop workflow proceeds as follows.




In this notebook, we use Globus Compute to execute functions (simulation, model training, and inference) in parallel on remote computers. We show how Globus Compute's use of (i.e., `concurrent.futures` ) allows applications to be easily written that dynamically respond to the completion of asynchronous tasks.

In [ ]:
```
# Set this ID to your Globus Compute endpoint
compute_endpoint = ''
```

In [ ]:
```
%matplotlib inline
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
from concurrent.futures import as_completed
from matplotlib import pyplot as plt
from tqdm.notebook import tqdm
from time import monotonic
```

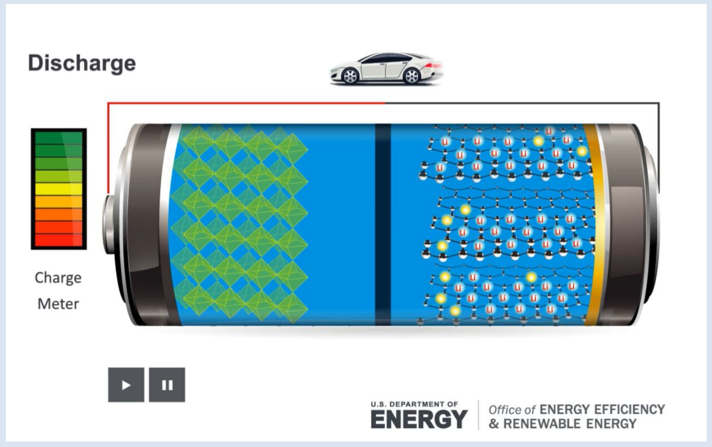
## Run a quantum chemistry simulation to calcuate IE

We need data to train our ML models. We'll do that by selecting a set of molecules at random from our search space, performing some simulations on those molecules, and training on the results.

**Aim:** Identify high value molecules (high ionization energy) among a search space of billions of candidates

**Problem:** Simulation is expensive

**Solution:** Create an active learning loop, coupling simulation with ML to simulate only high value candidates



# Simplify execution on different systems

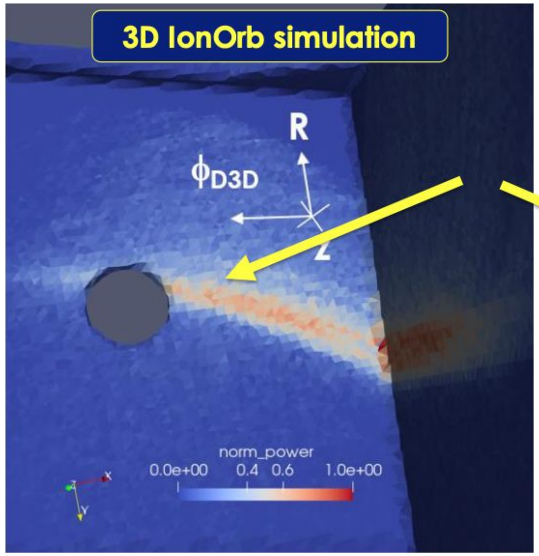
- **Simplified workload migration**
  - Register function once with Globus Compute
  - One-time setup of runtime environment on different systems
- **Same interface for function invocation**

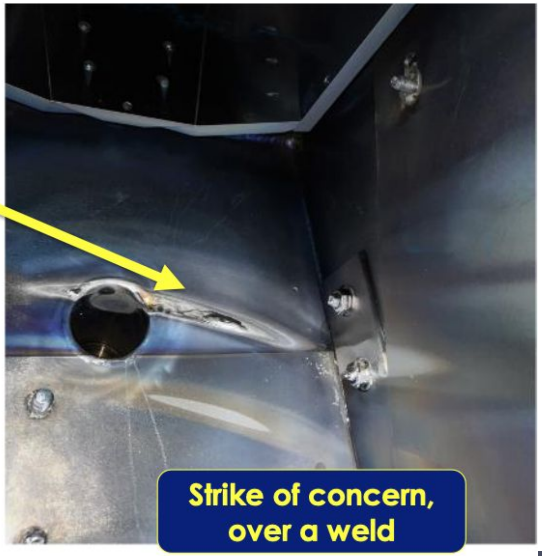
```python
def process_image(input_file):

    import os
    from PIL import Image

    image = Image.open(file)
    image.thumbnail((200, 200))
    image.save(f"{os.path.basename(file)}")

# Register the function with the Globus Compute service
from globus_compute_sdk import Client

gc_client = Client()
my_function = gc_client.register_function(process_images)

campus_cluster = "ad4f48be-9c03-49bc-9dc4-e240bc599bef"
purdue_anvil_access = "e93b4289-35c1-4de1-838a-0c0512cdf61e"

# Run code on Campus Cluster
my_campus_task = gc_client.run('image.png', my_function, campus_cluster)

# Run code on Anvil supercomputer at Purdue
my_anvil_task = gc_client.run('image.png', my_function, purdue_anvil_access)
```

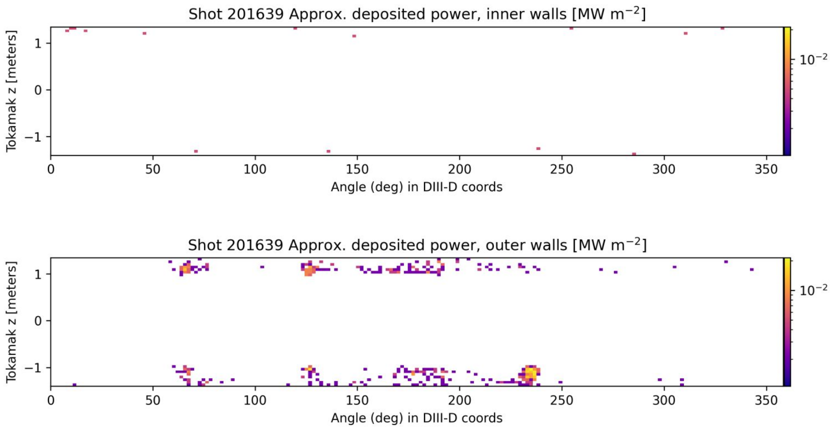
# Rapid and Reliable Particle Tracking for Heat Deposition for DIII-D

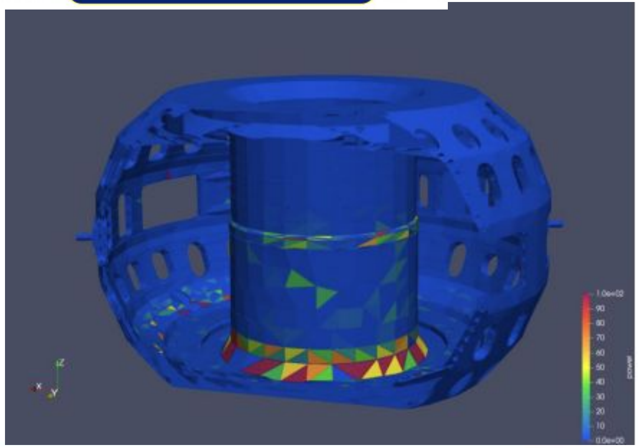
Simulate energy deposition for Tokamak shots and **alert to potential instrument damage within 7 minutes** (previously 4.5 hours).

- **Failover to available resources:** Seamlessly leverage resources at different DOE national labs for online feedback



DIII-D NATIONAL FUSION FACILITY

Argonne NATIONAL LABORATORY

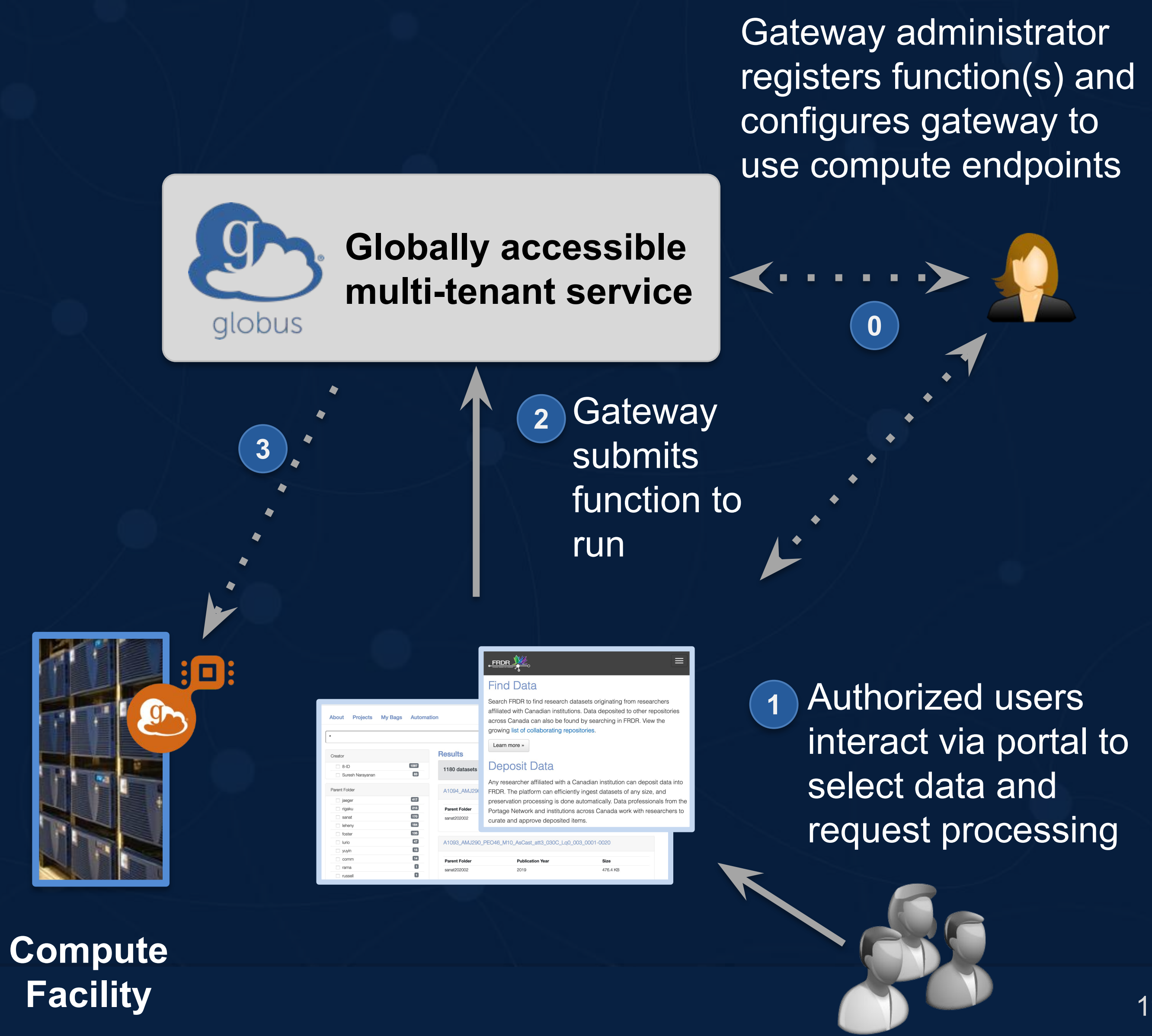
BERKELEY LAB

U.S. DEPARTMENT OF ENERGY | Office of Science

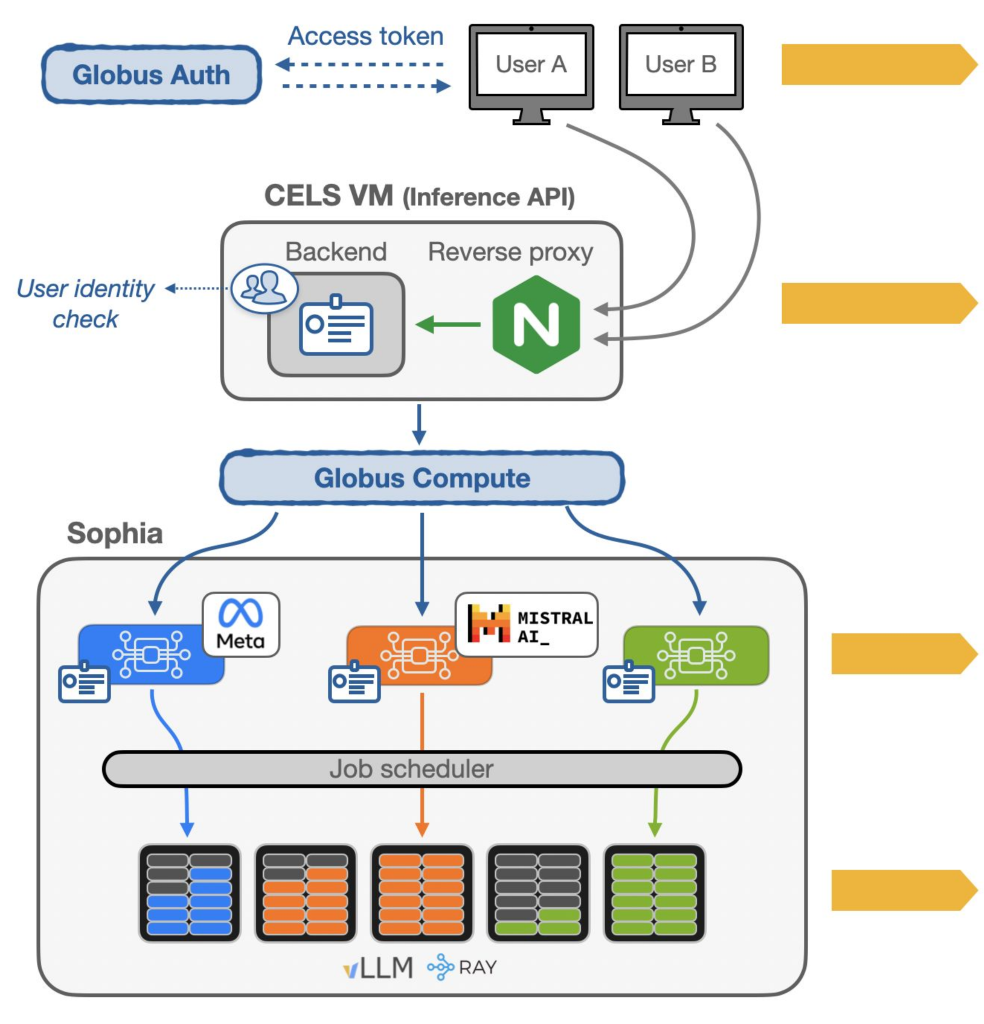
DIII-D: Juan Colmenares, Sterling Smith, Torrin Bechtel Amara, Akshay DeshPande
Argonne: Christine Simpson, Tom Uram
NERSC: Nick Tyler, Peng Fei

# Compute access from Science Gateways/Portals

- **Secure access to execution environment from science gateways/portals**
- **Support for community account or service account model**
- **Authentication & authorization policies**
  - Allowed users
  - Permitted functions



# Argonne Inference Service




**Authentication**
Allows authorized users to interact with inference service.

**Django Portal**
Provides API access to running vLLM endpoints.

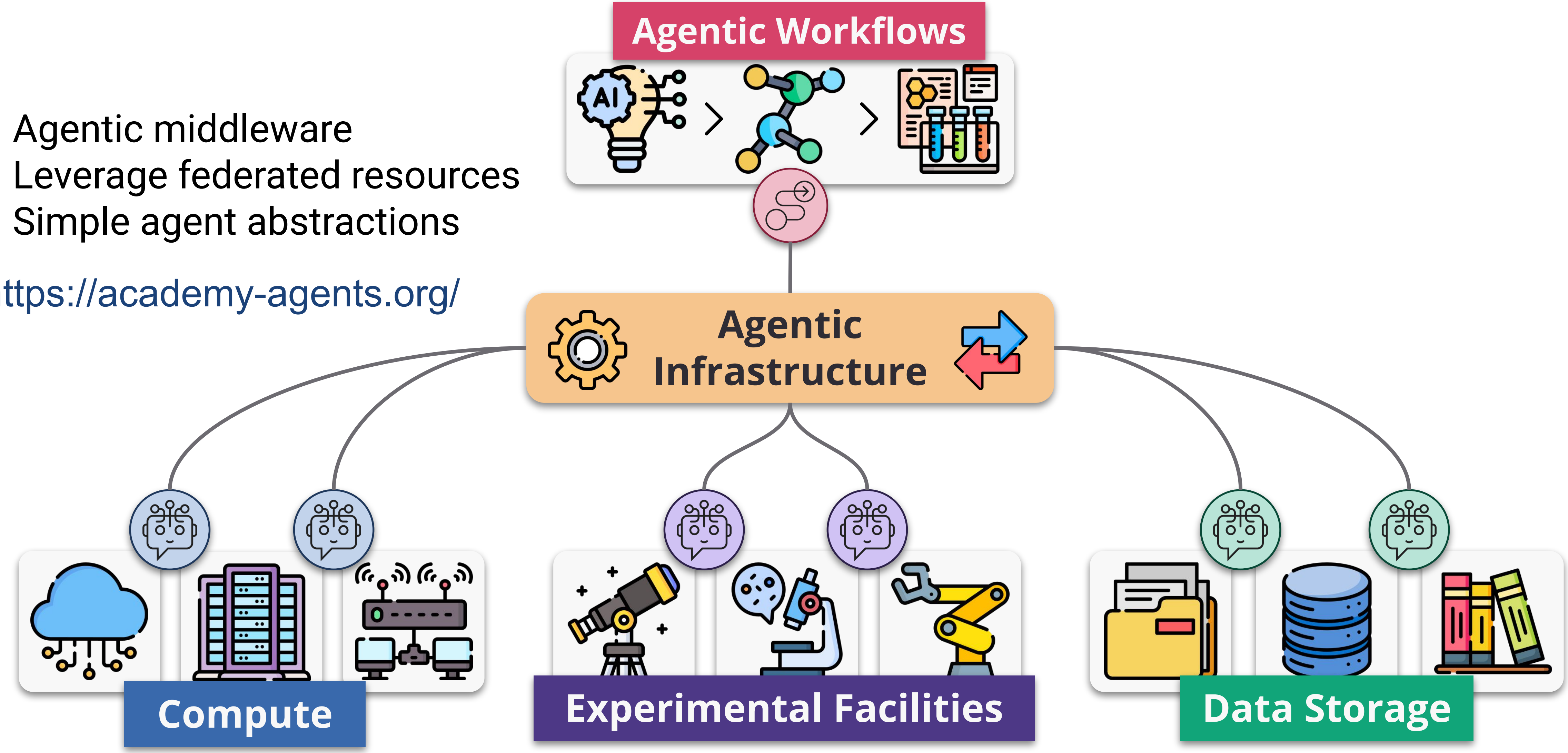
**Endpoints**
Multiple pre-registered endpoints owned by a service account.

**GPU Nodes**
Compute resources running vLLM + Ray with various models. Elastic based on requested model.

# Enabling deployment and management of agent systems

- Agentic middleware
- Leverage federated resources
- Simple agent abstractions

https://academy-agents.org/



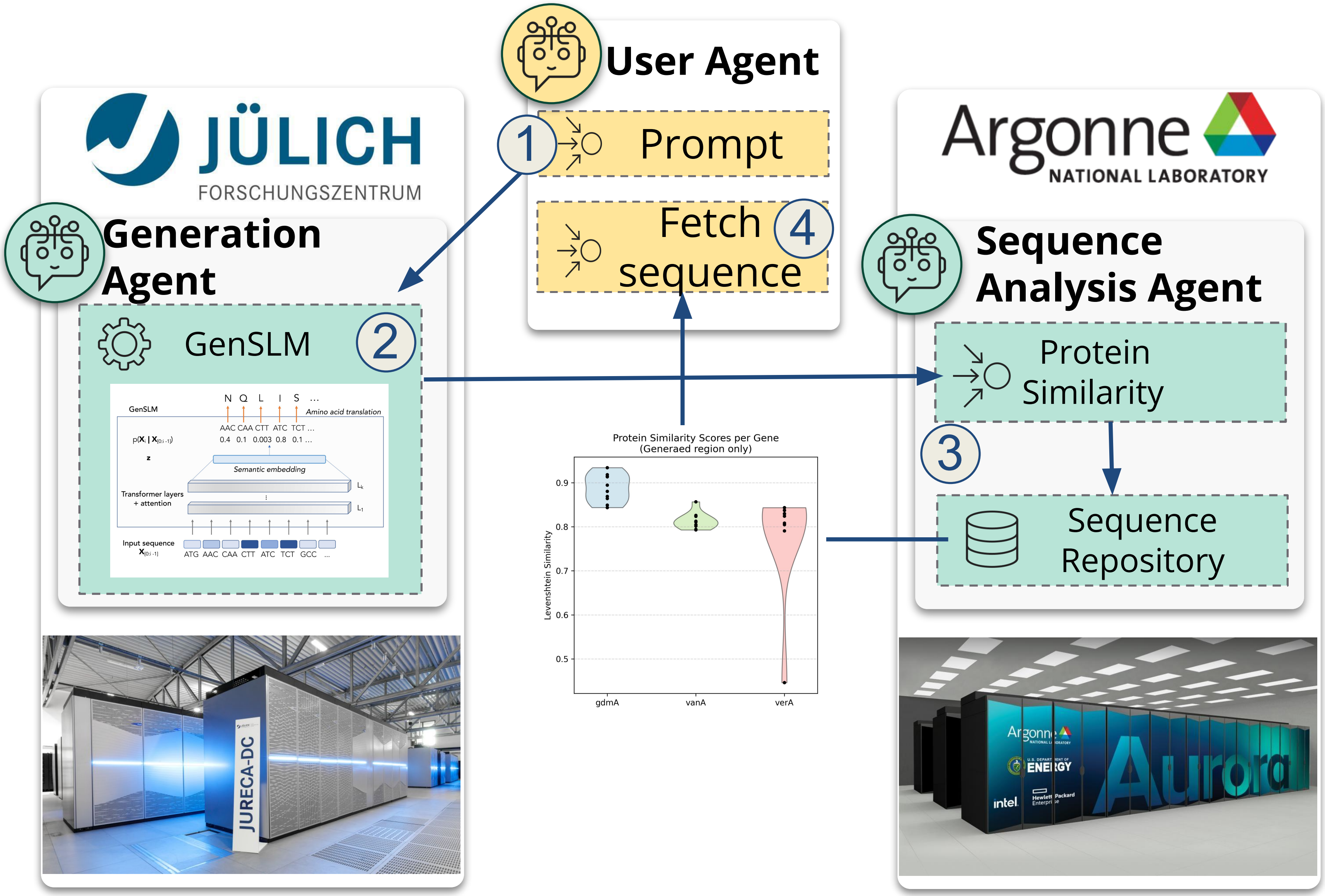
# Agentic Protein Design with Academy

GenSLM is a genome-scale language model that can generate bacterial and viral protein sequences.

We implement three agents:

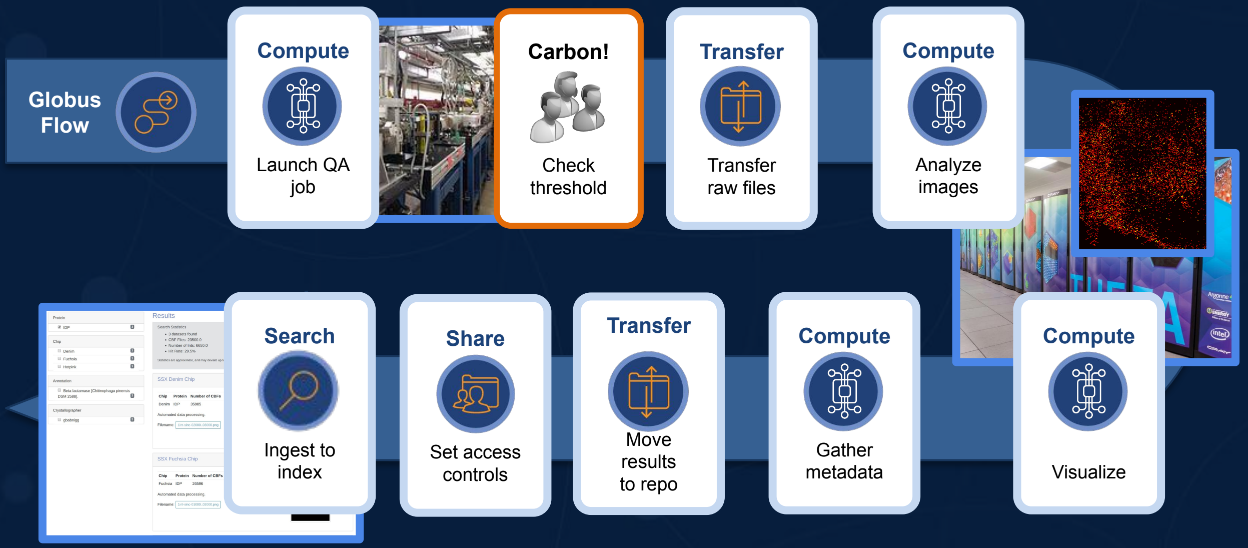
1. **User agent** triggers new analyses
2. **Generation agent**, which runs GenSLM models, trained on proprietary data, to generate sequences
3. **Sequence analysis agent,** which hosts methods for evaluating protein similarity, and stores promising sequences
4. **User agent** (again) monitors sequence repository for promising candidate sequences

https://academy-agents.org/



# Programmatic access to compute for automation

- **Incorporate compute tasks into Globus Flows**
- **Perform actions that don't have an action provider**
- **Automate execution of different workloads on suitable compute resources**



# Pipeline with diverse computing tasks
## (serial crystallography)

**Globus Flow**

**Compute**
Run code on **adjacent server**
Launch QA job

**Carbon!**
Check threshold

**Transfer**
Transfer raw files

**Compute**
Run code on **supercomputer**
Analyze images

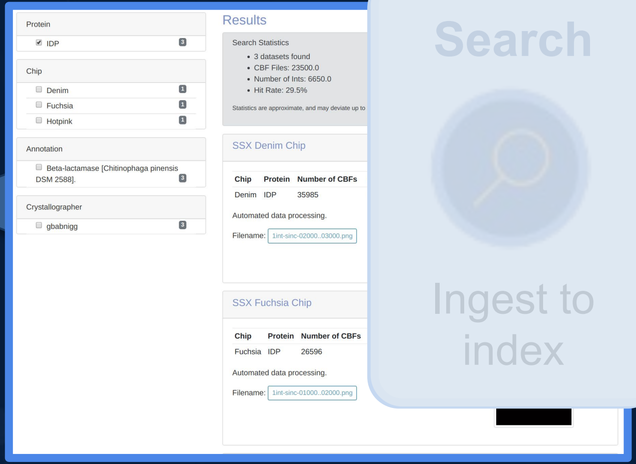

**Compute**
Run code on **GPU cluster**
Visualize

**Compute**
Access custom **metadata extraction service**
Gather metadata

**Transfer**
Move results to repo

**Share**
Set access controls

**Search**
Ingest to index

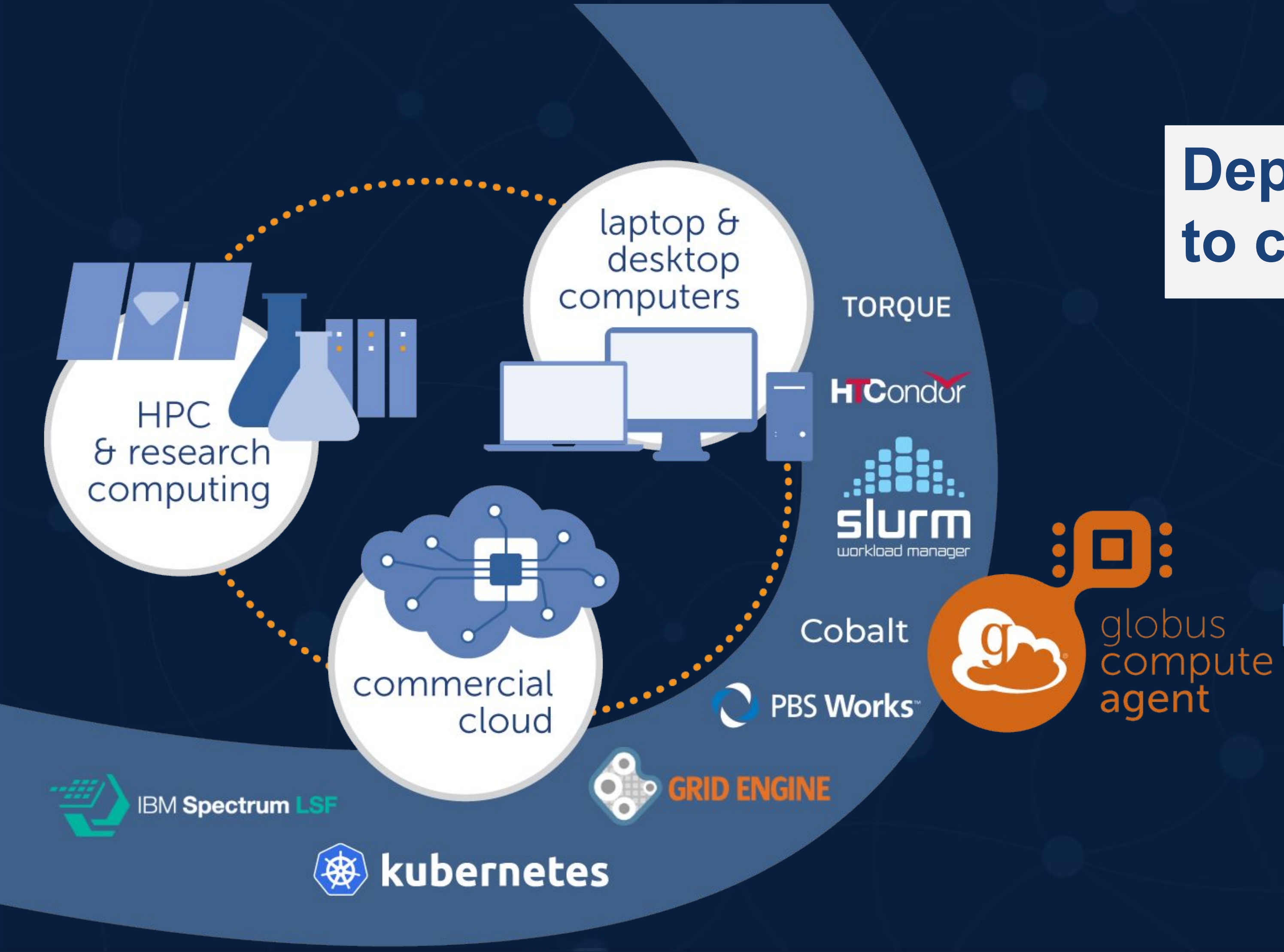


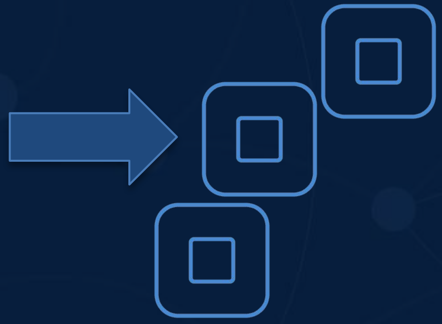
# SDK

https://jupyter.demo.globus.org/

# Enabling compute at your institution

**Deploy Globus Compute Agent to connect to ecosystem**

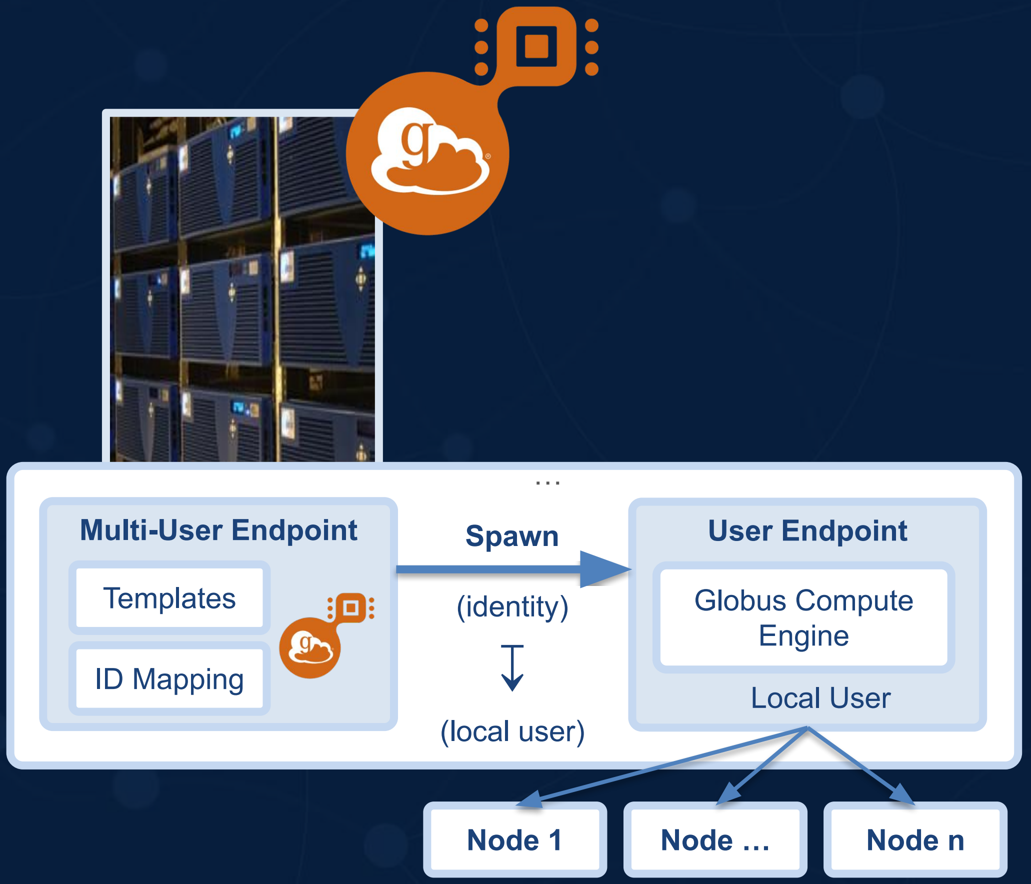


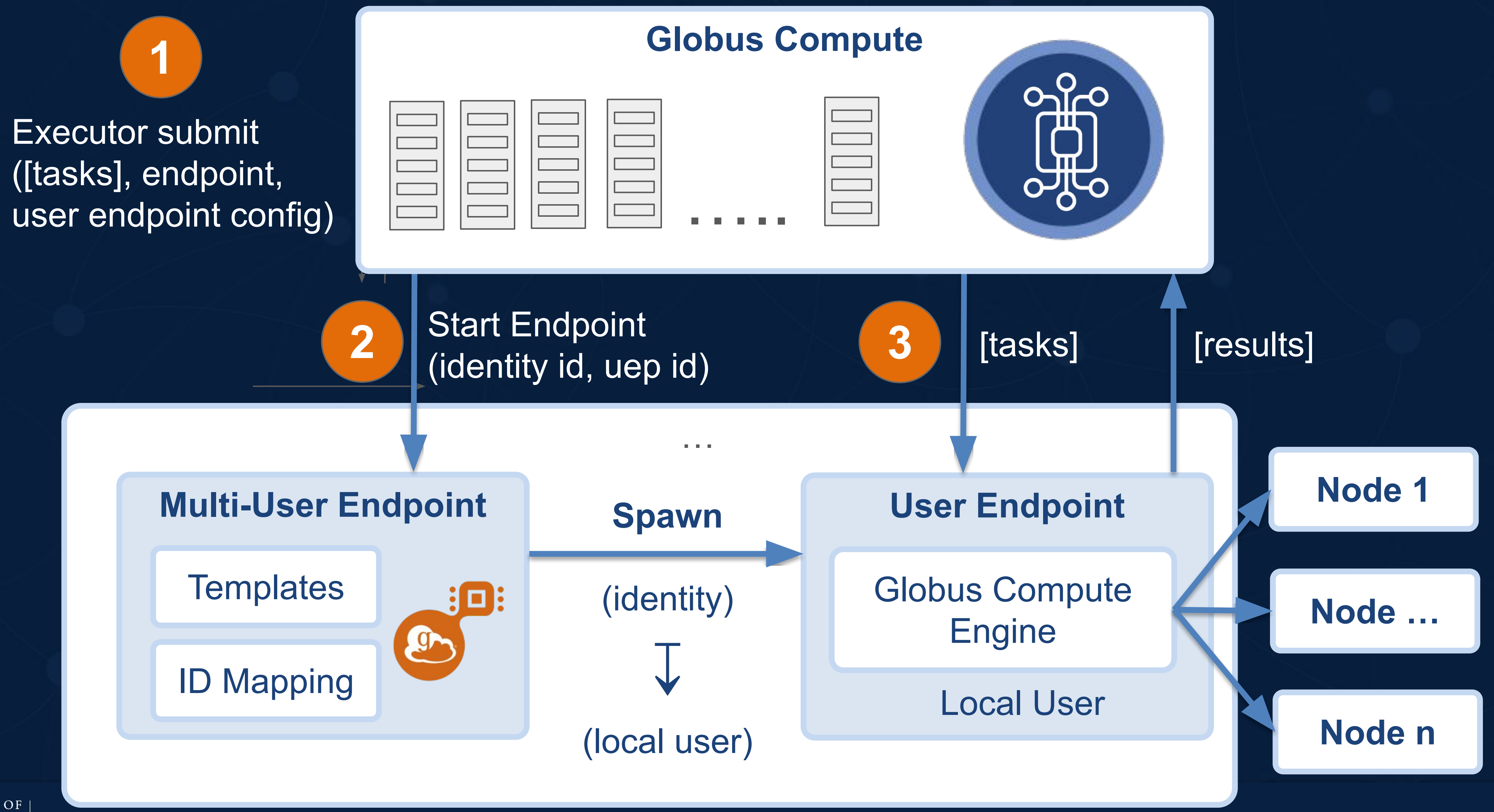
**Compute endpoint**

# Globus Compute Endpoint

- **Install compute agent using Linux package manager**
  - Campus cluster, cloud/other shared resource □ Multi-user endpoint
  - Facility/instrument-adjacent cluster □ Single-user endpoint
- **Define authentication and authorization policies**
  - User identity mapping for authorization
- **Configure executor template (Slurm, PBS, LSF, Cobalt, K8s, *et al*)**
  - Fixed values, defaults for common job parameters
  - Parameters that may be overridden by user



# Multi-user endpoint execution flow



# Administrator Benefits of Multi-user Compute

- **Templatable endpoint configurations**
  - Pre-select executor, enforce limits
- **Standard Globus identity mapping for authorization**
  - Support for services accounts
- **Simplify user requirement to use HPC**
  - No user run processes, or SSH required
  - No training on scheduler/execution system specifics required
- **Administrator console**
  - Manage and monitor use

# Endpoints

# Installing single user compute endpoint

```
$ pip install globus-compute-endpoint

$ globus-compute-endpoint configure eresearch-2025-endpoint
Created profile for endpoint named <eresearch-2025-endpoint>

        Configuration file:
/home/ec2-user/.globus_compute/eresearch-2025-endpoint/config.yaml

Use the `start` subcommand to run it:

        $ globus-compute-endpoint start eresearch-2025-endpoint
```

```
$ globus-compute-endpoint start eresearch-2025-endpoint
Starting endpoint; registered ID: 54460200-b652-4f43-a918-02882fa6114a
```

# Configuring a single user compute endpoint

```yaml
# ~/.globus_compute/eresearch-2025-endpoint/config.yaml
amqp_port: 443
display_name: My Endpoint
engine:
 type: GlobusComputeEngine
 provider:
   type: LocalProvider
```

https://globus-compute.readthedocs.io/en/latest/endpoints.html#example-configurations


The following snippet shows an example configuration for executing remotely on Delta, a supercomputer at the National Center for Supercomputing Applications. The configuration assume user is running on a login node, uses the `SlurmProvider` to interface with the scheduler, and uses `SrunLauncher` to launch workers.

```yaml
amqp_port: 443
display_name: NCSA Delta 2 CPU
engine:
    type: GlobusComputeEngine
    max_workers_per_node: 2

    address:
        type: address_by_interface
        ifname: eth6.560

    provider:
        type: SlurmProvider
        partition: cpu
        account: {{ ACCOUNT NAME }}

        launcher:
            type: SrunLauncher

        # Command to be run before starting a worker
        # e.g., "module load anaconda3; source activate gce_env"
        worker_init: {{ COMMAND }}
```

# Configuring endpoints - Scaling

```yaml
# ~/.globus_compute/eresearch-2025-endpoint/config.yaml
amqp_port: 443
display_name: My Endpoint
engine:
  type: GlobusComputeEngine
  max_workers_per_node: 8

  provider:
    type: LocalProvider
```

# Managing the Execution Environment

```yaml
# ~/.globus_compute/eresearch-2025-endpoint/config.yaml
display_name: My Endpoint
engine:
  type: GlobusComputeEngine
  container_type: docker
  container_uri: python:3.12.10-bookworm
  container_cmd_options: -v /tmp:/tmp

provider:
   type: LocalProvider
   worker_init: conda activate pearc-conda
```

# Configuring endpoints - Batch Schedulers

```yaml
# ~/.globus_compute/eresearch-2025-endpoint/config.yaml
amqp_port: 443
display_name: My Endpoint
engine:
 provider:
   type: SlurmProvider
   partition: compute
   account: {{ ACCOUNT }}
   launcher:
     type: SrunLauncher
   scheduler_options: {{ OPTIONS}}
   worker_init: {{ COMMAND }}
   walltime: 01:00:00
   nodes_per_block: 1
 type: GlobusComputeEngine
 max_workers_per_node: 8
```


he following snippet shows an example configuration for executing remotely on Expanse, a upercomputer at the San Diego Supercomputer Center. The configuration assumes the user is running n a login node, uses the `SlurmProvider` to interface with the scheduler, and uses the `SrunLauncher` to aunch workers.

```yaml
display_name: Expanse@SDSC

engine:
    type: GlobusComputeEngine
    max_workers_per_node: 2
    worker_debug: False

    address:
        type: address_by_interface
        ifname: ib0

    provider:
        type: SlurmProvider
        partition: compute
        account: {{ ACCOUNT }}

        launcher:
            type: SrunLauncher

        # string to prepend to #SBATCH blocks in the submit
        # script to the scheduler
        # e.g., "#SBATCH --constraint=knl,quad,cache"
        scheduler_options: {{ OPTIONS }}

        # Command to be run before starting a worker
        # e.g., "module load anaconda3; source activate gce_env"
        worker_init: {{ COMMAND }}
```

# Configuring endpoints - Scaling Batch Schedulers

```yaml
# ~/.globus_compute/eresearch-2025-endpoint/config.yaml
amqp_port: 443
display_name: My Endpoint
engine:
 type: GlobusComputeEngine
 nodes_per_block: 8
 init_blocks: 1
 min_blocks: 0
 max_blocks: 4

 max_workers_per_node: 8

 provider:
   type: SlurmProvider
   partition: compute
  ...
```

# Debugging and Diagnostics

```
/home/name/.globus_compute/default/
├── config.yaml
├── endpoint.json
├── endpoint.log
├── GlobusComputeEngine-HighThroughputExecutor
│   ├── block-0
│   │   └── 22980c57e30a
│   │       ├── manager.log
│   │       └── worker_0.log
│   └── interchange.log
├── submit_scripts
│   ├── parsl.GlobusComputeEngine-HighThroughputExecutor.block-0.1731697961.0310187.sh
│   ├── parsl.GlobusComputeEngine-HighThroughputExecutor.block-0.1731697961.0310187.sh.ec
│   ├── parsl.GlobusComputeEngine-HighThroughputExecutor.block-0.1731697961.0310187.sh.err
│   └── parsl.GlobusComputeEngine-HighThroughputExecutor.block-0.1731697961.0310187.sh.out
```

# Globus Automation Overview

## eResearch Australia 2025

THE UNIVERSITY OF CHICAGO

# Research Automation

**Executing research tasks reliably, at scale, with minimal (or no) human intervention when required.**

# Research Automation

**Executing research tasks reliably, at scale, with minimal (or no) human intervention when required.**

**Drive efficiency, handle scale (tasks/data/users), implement best practices…**

# Globus Automation Capabilities

## Command Line Interface

Ad hoc scripting and integration

## Timer Service

Scheduled and recurring transfers
(*a.k.a. Globus cron*)

## Globus Flows service

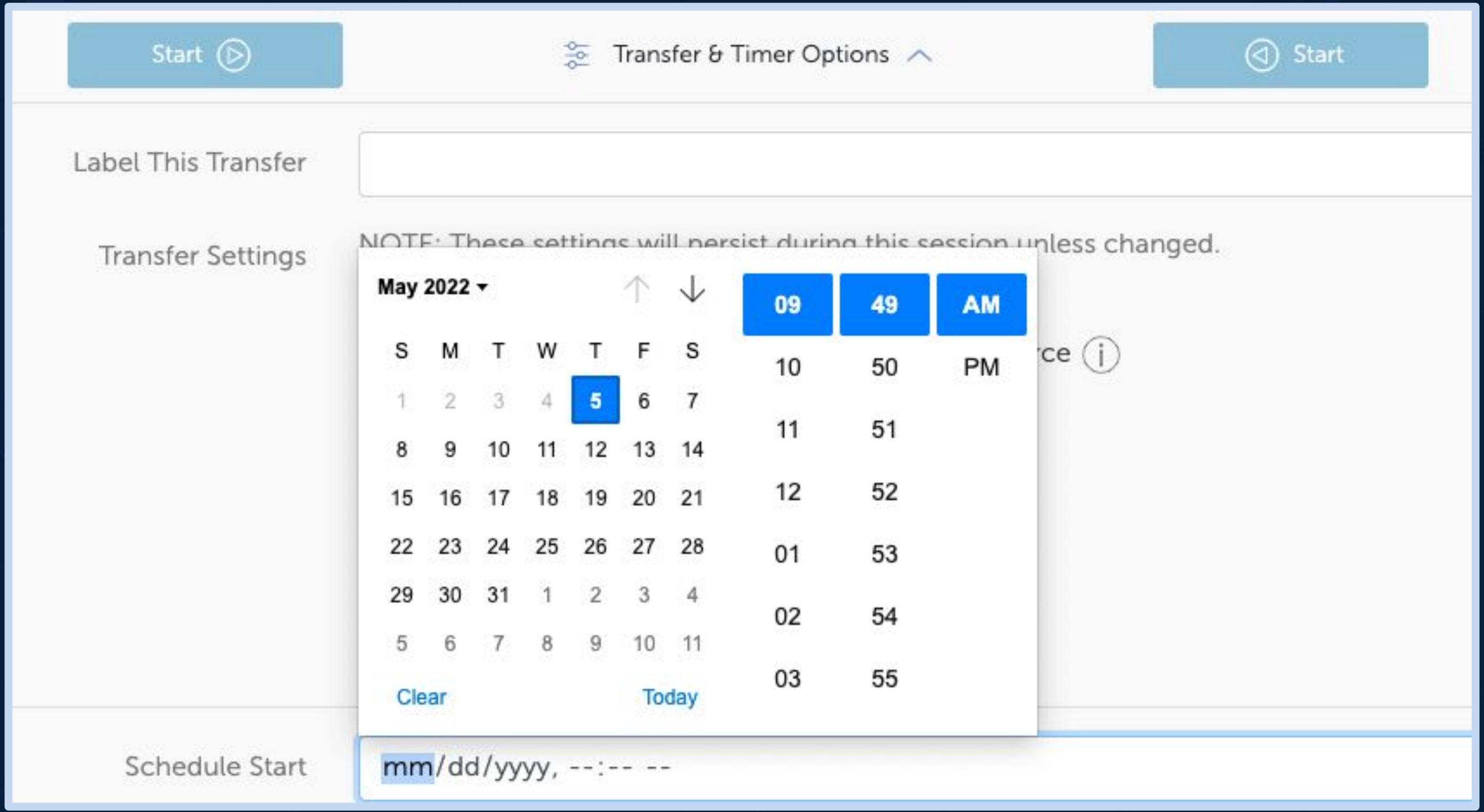
Comprehensive task (data and compute) orchestration with human in the loop interactions

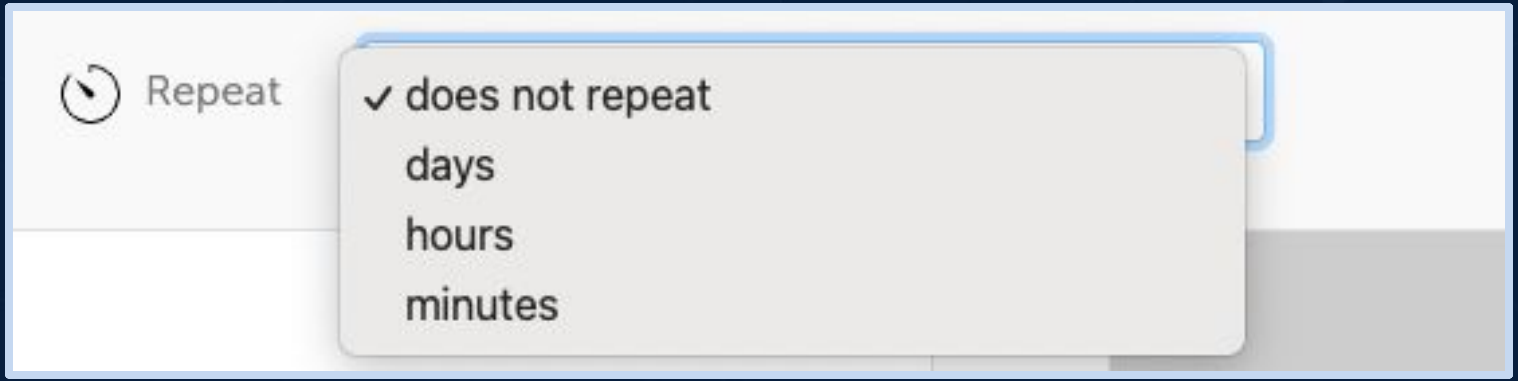
# Globus Timers

# Timers

Scheduled and recurring file transfers

Supports all transfer and sync options




hpc.nih.gov/docs/globus/globus_cron.php#cron

# Globus Flows

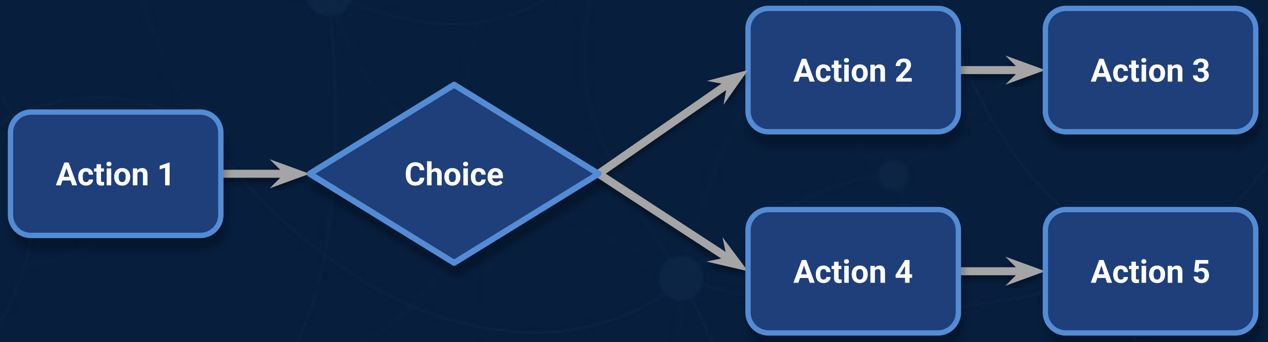
# Globus Flows for reliable automation

- Managed reliable task orchestration
- Declarative language for flow definition
- Event driven execution model
- Extensible to integrate external services

# What is a Flow?

## A sequence of steps…

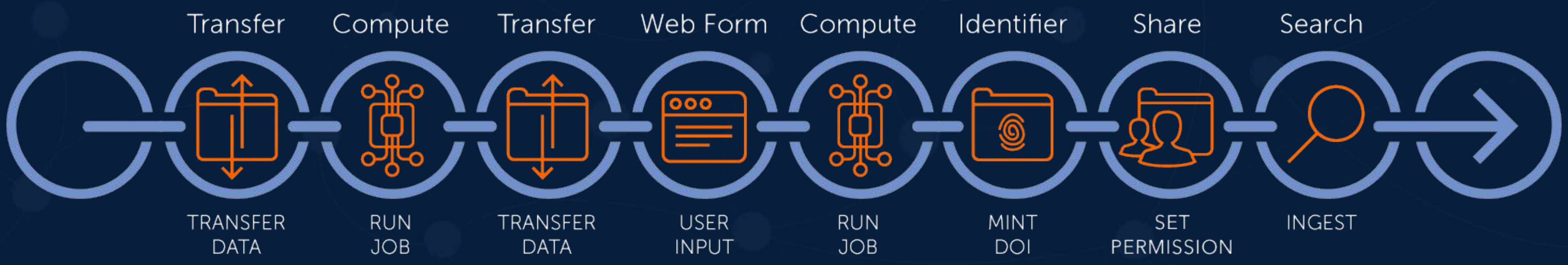
- Hosted
- Reusable
- Flexible
- Shareable



# What's an Action?

## Each step in the flow is an ***action***

- **Action calls other services to perform some operation**
  - E.g. move data, get a persistent identifier, send email to curator
- **Flows service manages interactions**
  - Authenticate, authorize, validate, store, auto-retry



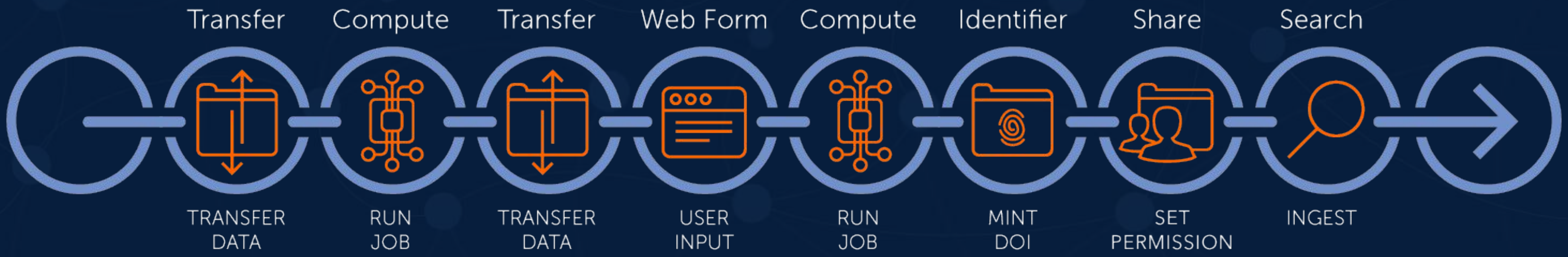
# Globus-Provided Actions

# Extensible to any operation

# How Do You Use a Flow?

## Each time a flow is started, it's called a ***run***

- Start from Web App, CLI, Python SDK, API
- Provide input
- Performs a series of ***actions*** with that input
- Manage the run (Evaluate; Share)



# Anatomy of a Flow

## Definition

Operations

Transformations

Conditions

## Input Schema

Data requirements

Three key benefits:

1. Prevent errors
2. Guide input
3. Validate flow

# Flow definition

```
"StartAt": "TransferFiles",
"States": {
    "TransferFiles": {
        "Comment": "Transfer to a guest collection",
        "Type": "Action",
        "ActionUrl": "https://actions.automate.globus.org/transfer/transfer",
        "Parameters": {
            "source_endpoint_id.$": "$.input.source.id",
            "destination_endpoint_id.$": "$.input.destination.id",
            "transfer_items": [
                {
                    "source_path.$": "$.input.source.path",
                    "destination_path.$": "$.input.destination.path",
                    "recursive.$": "$.input.recursive_tx"
                }
            ]
        },
        "ResultPath": "$.TransferFiles",
        "WaitTime": 60,
        "Next": "SetPermission"
    },
    "SetPermission": {
        .....
        "End": True
    }
}
```

Action

Action Provider URL

Action inputs

Timeout (seconds)

Next state

# Flow input schema

```
{
    ....
    "properties": {
        "input": {
            "type": "object",
            "required": [
                "source",
                "destination",
                "recursive_tx"
            ],
            "properties": {
                "source": {
                    "type": "object",
                    "title": "Select source collection and path",
                    "description": ”Source collection/path (MUST end with '/')",
                    "format": "globus-collection",
                    "required": [
                        "id",
                        "path"
                    ],
            "transfer_label": {
                "type": "string",
                "title": "Label for Transfer Task",
                "pattern": "^[a-zA-Z0-9-_, ]+$",
                "maxLength": 128,
            }
 ....
```

Required inputs

Input type

Custom schema

Input type

# Anatomy of a Flow

## Input Schema

*Expresses data requirements*

Three key benefits

1. Prevent errors
2. Guide input
3. Validate Flow

```
{
  "required":
["entry"],
  "properties": {
    "source": {
      "title": "Entry",

      "minimum": 1,
      "maximum": 50,
      "default": 1
    }
  }
}
```

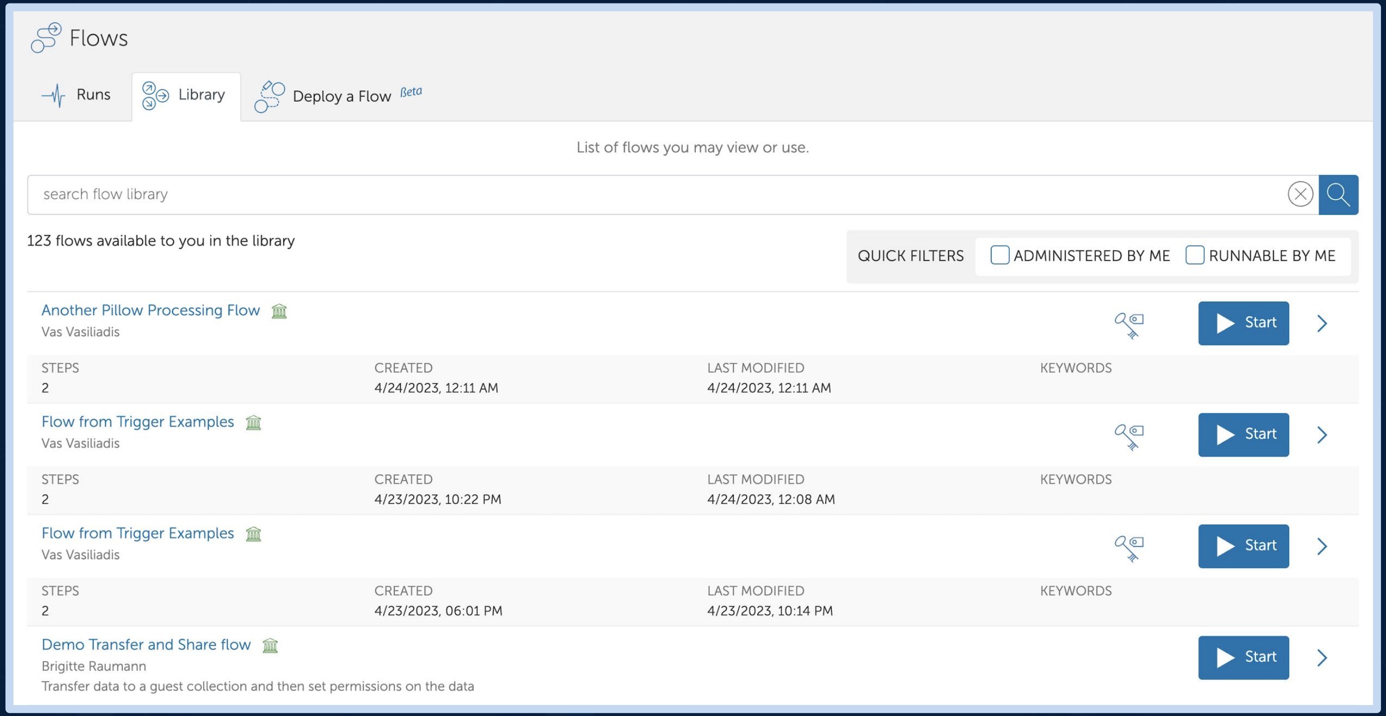
# Flow lifecycle

- **Define using JSON**

```
{
  "States": {
    "ProcessFiles": {
      "End": true,
      "Type": "Action",
      "Comment": "Process files - generate thumbnails",
      "WaitTime": 180,
      "ActionUrl": "https://compute.actions.globus.org/fxap",
      "Parameters": {
        "kwargs.$": "$.input.compute_function_kwargs",
        "endpoint.$": "$.input.compute_endpoint_id",
        "function.$": "$.input.compute_function_id"
      },
      "ResultPath": "$.ProcessFiles"
    },
    "TransferFiles": {
      "Next": "ProcessFiles",
      "Type": "Action",
      "Comment": "Transfer to a guest collection",
      "WaitTime": 60,
      "ActionUrl": "https://actions.automate.globus.org/transfer/transfer",
      "Parameters": {
        "transfer_items": [
          {
            "recursive.$": "$.input.recursive_tx",
            "source_path.$": "$.input.source.path",
            "destination_path.$": "$.input.destination.path"
          }
        ],
        "source_endpoint_id.$": "$.input.source.id",
        "destination_endpoint_id.$": "$.input.destination.id"
      },
      "ResultPath": "$.TransferFiles"
    }
  },
  "Comment": "Transfer and process files by invoking a funcX function",
  "StartAt": "TransferFiles"
}
```

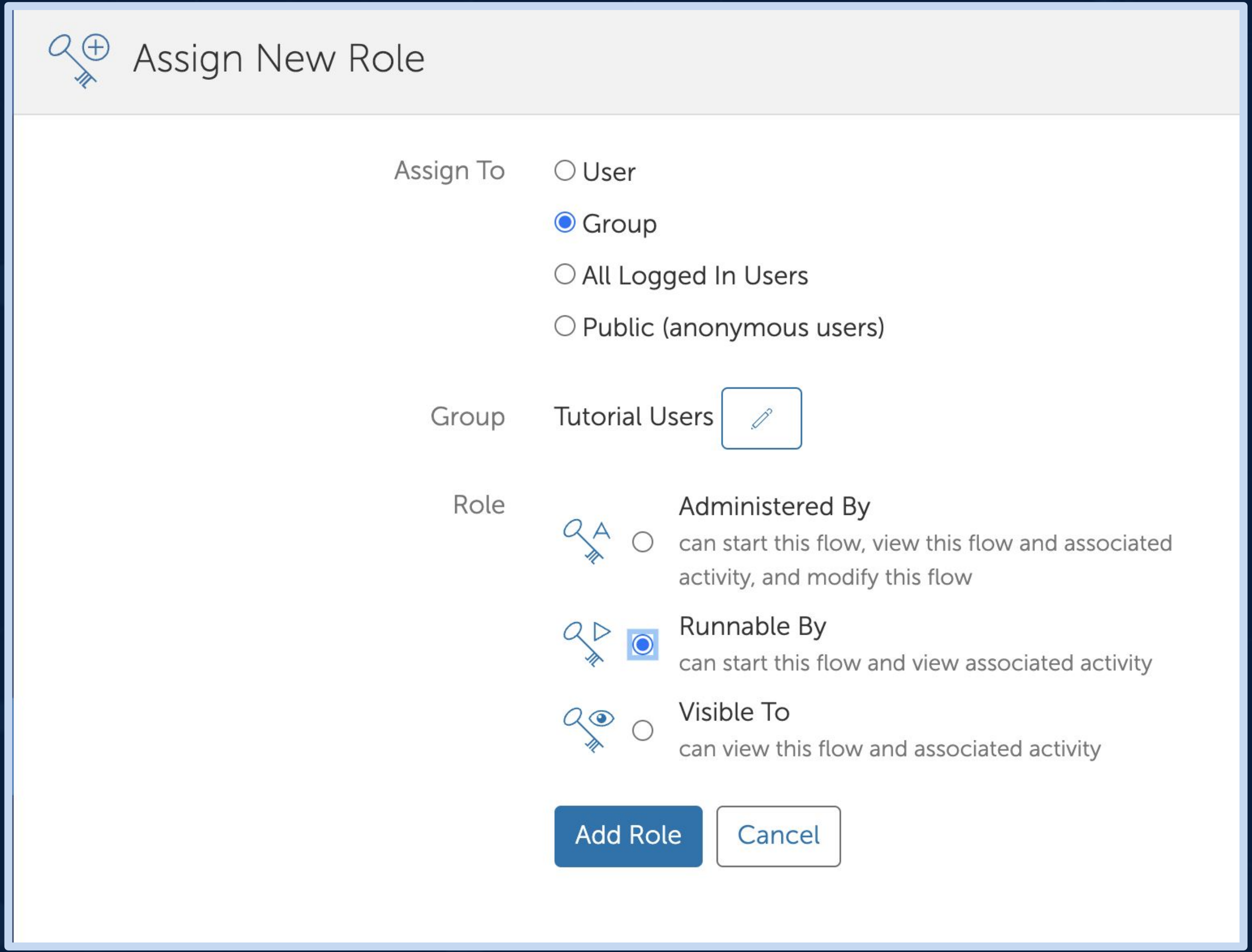
# Flow lifecycle

- Define using JSON/YAML
- Deploy to Flows service



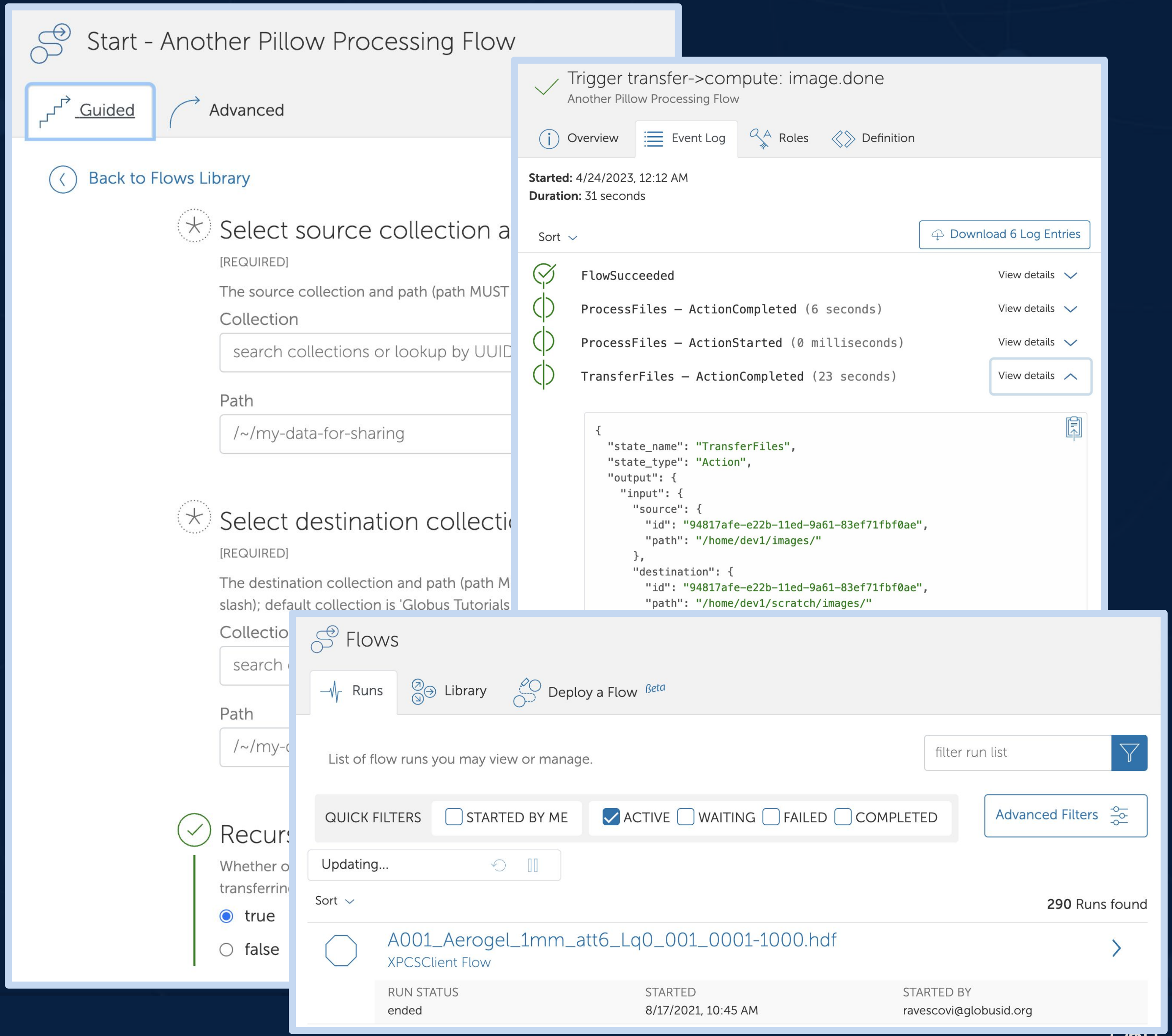
# Flow lifecycle

- **Define using JSON/YAML**
- **Deploy to Flows service**
- **Set access policy for visibility and execution**



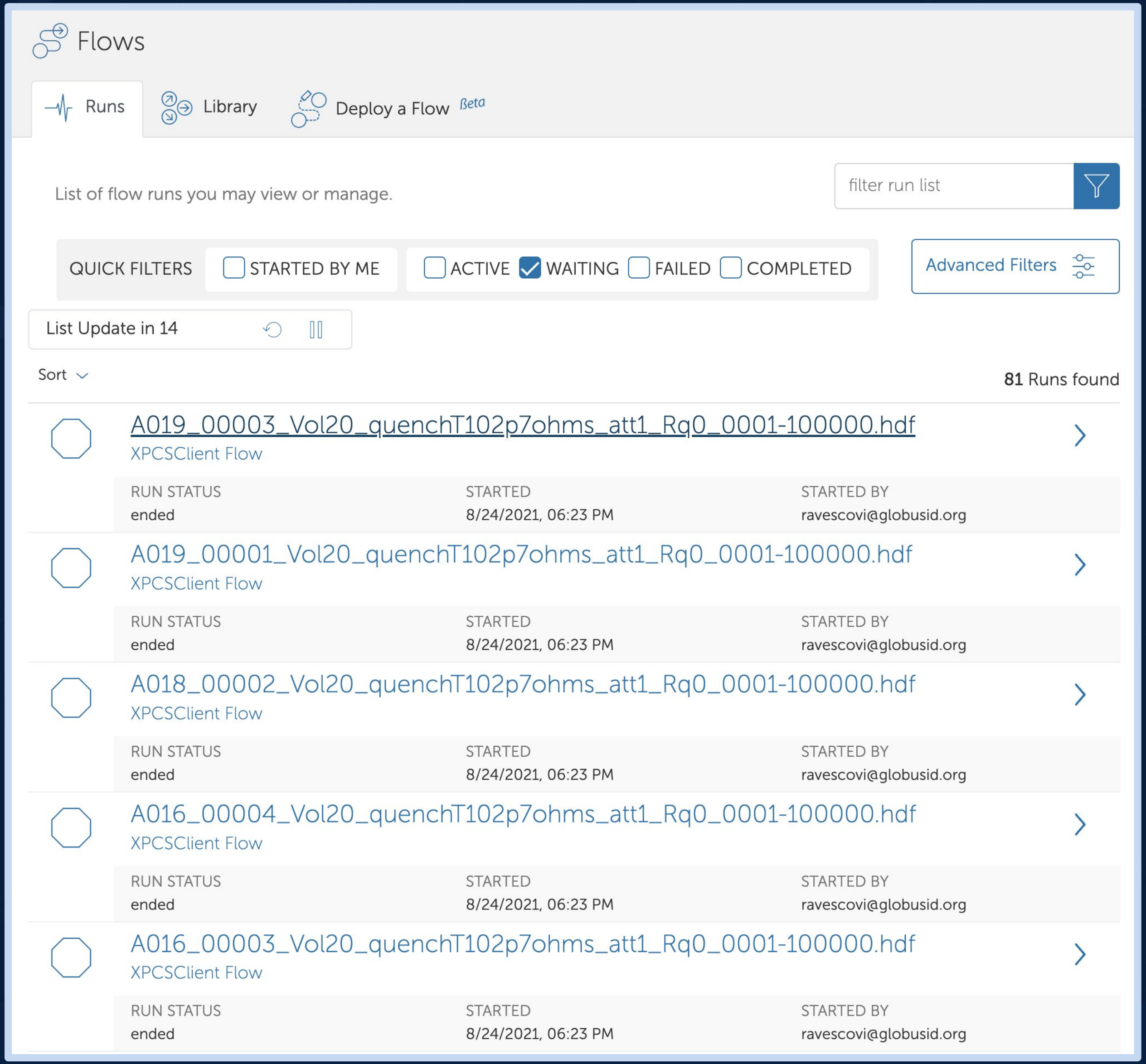
# Flow lifecycle

- Define using JSON/YAML
- Deploy to Flows service
- Set access policy for visibility and execution
- **Run (debug) and monitor**



# Flow lifecycle: Write once, run many

- **Define using JSON/YAML**
- **Deploy to Flows service**
- **Set access policy for visibility and execution**
- **Run (debug) and monitor**
- **…and run again!**



# Examples of Flows

## Two Stage Globus Transfer

- Temporarily transfer to an intermediate location before transferring to a destination

## Move (Copy and Delete)

- After transferring to a destination, cleans up the files on the source

## Transfer and Set Permissions

- After transferring to a destination, share with users and groups

# Run a flow

app.globus.org/flows

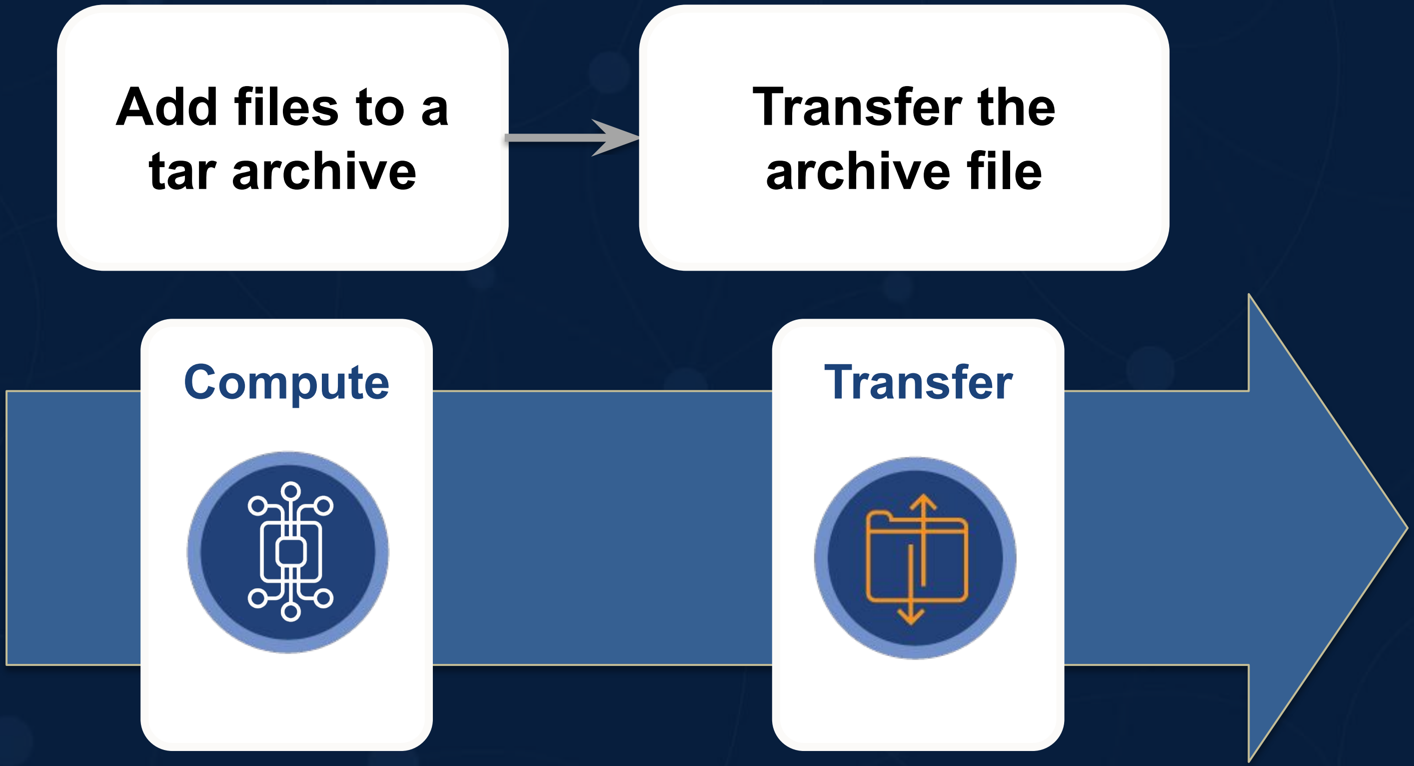
# Exercise: Pre-requisites

- We will use your existing Guest collection
- This grants you access to manage permissions on a guest collection

# Exercise: Run Globus Flow using the web app

- **Find “Transfer and Set Permissions” in flows library**
  - Go to Flows □ Library and check “Globus-Provided Flows”
- **Click “Start”**
- **Select “AARNet Globus Endpoint NSW (ARTM) POSIX Gateway Public RO Guest Collection” as the source**
- **Select <your previous guest collection> as the destination**
- **Choose your target path (must be a directory): /**
- **Select your sharee(s)**
- **Enter a label for the flow run and click “Start Run”**
- **Monitor  flow progress on the “Event log” tab**

# Simple* automation: tar and transfer



# Exercise: Pre-requisites

- **You will need co-located Globus Connect and Globus Compute endpoints**

```python
def simple_tar(src_dir, dest_path, gcs_base_path):
    import tarfile
    outfile = os.path.join(gcs_base_path, dest_path)
    with tarfile.open(outfile, "w:gz") as tar:
        tar.add(src_dir)
    return outfile


gce.client.register_function(simple_tar, public=True)
```

# Exercise: Run Globus Flow using the web app

- Find “eResearch 2025 Tar and Transfer” in flows library
- Click “Start”
- Select your source collection and path (a directory)
- Set the base path for the Globus collection (e.g., /home/<username>)
- Select your Globus Compute endpoint
- Select “Tutorial Endpoint” as the destination and any path
- Enter a label for the flow run and click “Start Run”
- Monitor  flow progress on the “Event log” tab

# Resources

- **Getting started:**

jupyter.demo.globus.org/hub/user-redirect/lab/tree/globus-jupyter-notebooks/Compute_Introduction.ipynb

globus-compute.readthedocs.io/en/stable/quickstart.html

- **Endpoint configuration and sample templates:**

globus-compute.readthedocs.io/en/stable/endpoints/multi_user.html

- **Endpoint configuration and sample templates:**

globus-compute.readthedocs.io/en/stable/endpoints/multi_user.html

# Ask for help! Really, please.

- **Guidance on best practices**
- **Sounding board for your design/implementation**
- **Assistance with configuring endpoint, templates, etc.**
- **All at no cost to you …just reach out**