Delegation Service Design

A Resource Rights Management Service

Executive Summary

The delegation service provides a unified API for managing delegated authority over storage and compute resources, enabling autonomous agents to operate with bounded, auditable, and revocable permissions. While this document uses ALCF as the reference deployment, the service design is facility-agnostic: the same architecture applies to any site with Globus-managed storage and a job submission system.

What it does. The service tracks a tree of delegations: a PI delegates rights to an agent, which may further delegate to sub-agents. Each delegation specifies a resource (Eagle storage, Aurora compute), a scope (paths, queues, node limits), a quota (bytes, node-hours), and an expiration. The service enforces three invariants: authority cannot be amplified through delegation, consumption cannot exceed quota, and revocation cascades to all derived rights.

How it works. When a delegation is created, the service provisions enforcement at the resource level: Globus ACLs for storage, submission proxy authorization for compute. Agents then interact with resources directly—Globus for transfers, the proxy for jobs—while the service tracks consumption via webhooks. When quota thresholds are crossed, the service suspends access (read-only for storage, blocked submissions for compute) and alerts the PI.

Key design choices.

- *Implicit access*: Agents use their Globus identity; no separate capability tokens required
- *Unified model*: Storage and compute share the same delegation tree, API, and revocation logic
- *Soft quotas with suspension*: Alerts at thresholds, automatic suspension at exhaustion, PI-controlled restoration
- *Phased deployment*: Local pilot first, ALCF-hosted production second, Globus-hosted service as the long-term goal

Integration points. The service requires: (1) Globus application credentials with ACL management permissions on target collections, (2) integration with a submission proxy for compute authorization, and (3) webhook endpoints to receive transfer and job completion events.

Scope of this document. This document specifies the data model, API, enforcement adapters, and deployment approach. It uses ALCF (Eagle storage, Aurora/Polaris compute) as the reference deployment, but the design applies to any facility. The submission proxy and PI-facing portal are covered in separate documents.

Contents

1 Introduction

This document specifies a delegation service for managing resource rights across scientific cyberinfrastructure. The service provides a unified API for storage and compute resources, with a pluggable adapter architecture that supports deployment at any facility with Globus-managed storage and a job submission system. ALCF serves as the reference deployment throughout this document, but the design is facility-agnostic.

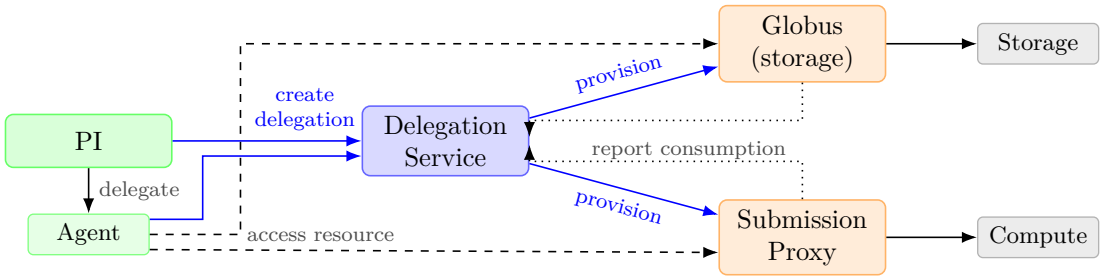

Figure 1: Architecture overview. **Solid arrows**: delegation management—PIs and agents create/revoke delegations via the service, which provisions enforcement (Globus ACLs, proxy authorization). **Dashed arrows**: runtime operations—agents access resources directly using their provisioned permissions. **Dotted arrows**: consumption tracking via webhooks.

The delegation service sits between agents and resource-specific enforcement points. Agents request delegations through this service, which:

- Validates authority (can this principal delegate this right?)
- Provisions enforcement (Globus ACLs for storage, proxy authorization for compute)
- Tracks provenance and consumption
- Enforces quotas and lifetimes

Agents then interact with the underlying resources directly—Globus for transfers, the submission proxy for jobs—while the delegation service tracks consumption via webhooks and callbacks.

The service supports multiple resource types through a common abstraction. Each delegation specifies:

- **Resource type**: What kind of resource (storage, compute, . . .)
- **Resource identifier**: Which specific resource (Eagle collection, Aurora system)
- **Scope**: What operations are permitted (paths and read/write for storage; queues and node limits for compute)
- **Quota**: Quantitative limit (bytes for storage; node-hours for compute)

The delegation tree, provenance tracking, expiration, and revocation work identically across resource types. Only enforcement differs: storage delegations create Globus ACLs; compute delegations register authorization with the submission proxy.

2 Design Alternatives

Before describing the implementation, we review three possible models for how agents interact with resource rights.

2.1 Implicit (Identity-Based)

The delegation service creates Globus ACLs tied to the agent's identity. Once a delegation exists, the agent calls Globus directly with its own OAuth credentials. Globus checks the ACL and permits or denies the transfer. The agent does not present a separate capability token; its authenticated identity is sufficient.

Advantages:

- Simple agent implementation—agents use Globus as normal
- No token management or caching required
- Leverages existing Globus infrastructure

Disadvantages:

- Agent must query the delegation service to discover what it can do
- Revocation requires ACL deletion (eventual consistency window)

2.2 Explicit (Token-Based)

The delegation service issues a signed JWT representing the resource right. The agent presents this token when initiating transfers, either to a proxy that validates it before calling Globus, or to an extended Globus endpoint that accepts delegation tokens.

Advantages:

- Agent carries its authority—no discovery needed
- Token encodes scope, quota, and constraints explicitly
- Supports offline validation

Disadvantages:

- Adds a second token type (Globus OAuth + delegation JWT)
- Requires a validating proxy or Globus extension
- Token refresh/revocation adds complexity

2.3 Hybrid

Combine both: use Globus ACLs for enforcement (implicit), but also issue delegation tokens that agents can cache and introspect locally. The token serves as a portable proof of authority and quota status, while the ACL remains the enforcement point.

Advantages:

- Agents can check authority locally before attempting transfers
- Enforcement remains at Globus (simple, reliable)

Disadvantages:

- Two representations of the same right must stay synchronized
- Added implementation complexity

2.4 Chosen Approach

This document describes the **implicit (identity-based)** model. It is the simplest to implement, requires no changes to how agents interact with Globus, and leverages Globus's existing ACL infrastructure. Agents that need to know their rights can query the delegation service; agents that do not care can simply attempt transfers and handle failures.

If future requirements demand portable tokens (e.g., for cross-facility federation or offline operation), the service can be extended to issue delegation JWTs without changing the underlying ACL-based enforcement.

3 Data Model

3.1 Delegations Table

The primary table stores all delegation records with a generalized schema that supports multiple resource types:

```
CREATE TABLE delegations (
    delegation_id       UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    parent_id           UUID REFERENCES delegations(delegation_id),
    delegator           TEXT NOT NULL,
    grantee             TEXT NOT NULL,

    -- Resource specification (generalized)
    resource_type       TEXT NOT NULL,      -- 'storage' | 'compute'
```

```
    resource_id          TEXT NOT NULL,      -- collection UUID or
        system name
    scope                JSONB NOT NULL,     -- type-specific scope

    -- Quota (generalized)
    quota                JSONB,              -- {bytes: N} or {
        node_hours: N}
    consumed             JSONB NOT NULL DEFAULT '{}',
    suspended            BOOLEAN NOT NULL DEFAULT false,

    -- Enforcement handle (type-specific)
    enforcement_ref      TEXT,               -- ACL ID, proxy auth ID,
        etc.

    -- Lifetime
    expires_at           TIMESTAMPTZ,

    -- Status
    revoked              BOOLEAN NOT NULL DEFAULT false,
    revoked_at           TIMESTAMPTZ,

    created_at           TIMESTAMPTZ NOT NULL DEFAULT now()
);

CREATE INDEX idx_delegations_grantee ON delegations(grantee);
CREATE INDEX idx_delegations_parent ON delegations(parent_id);
CREATE INDEX idx_delegations_type ON delegations(resource_type);
CREATE INDEX idx_delegations_expires ON delegations(expires_at)
    WHERE NOT revoked;
```

Field descriptions:

Field	Description
`delegation_id`	Unique identifier (UUID)
`parent_id`	Parent delegation (null for root delegations from PIs)
`delegator`	Globus identity URN of the principal creating this delegation
`grantee`	Globus identity URN of the principal receiving the right
`resource_type`	Type of resource: `storage` or `compute`
`resource_id`	Identifier for the resource (collection UUID, system name)
`scope`	Type-specific scope (see below)
`quota`	Type-specific quota (see below)
`consumed`	Type-specific consumption tracking
`suspended`	True if access is suspended due to quota exhaustion
`enforcement_ref`	Reference to enforcement mechanism (ACL ID, etc.)
`expires_at`	Expiration timestamp (null = no expiration)
`revoked`	True if explicitly revoked
`revoked_at`	Timestamp of revocation
`created_at`	Timestamp of creation

3.2 Type-Specific Fields

The `scope`, `quota`, and `consumed` fields are JSON objects whose structure depends on the resource type.

Storage:

```
{
  "scope": {
    "path": "/projects/materials-discovery",
    "operations": ["read", "write"]
  },
  "quota": {
    "bytes": 10995116277760
  },
  "consumed": {
    "bytes": 4500000000000
  }
}
```

Compute:

```
{
  "scope": {
    "queues": ["prod", "debug"],
    "max_nodes_per_job": 512,
```

```
    "max_walltime_hours": 24
  },
  "quota": {
    "node_hours": 100000
  },
  "consumed": {
    "node_hours": 45000
  }
}
```

Delegation validation enforces that child scope is a subset of parent scope (narrower path, fewer queues, lower node limits) and child quota does not exceed parent's available capacity.

3.3 Consumption Tracking Tables

For storage, track Globus transfer tasks:

```
CREATE TABLE storage_tasks (
    task_id             TEXT PRIMARY KEY,
    delegation_id       UUID NOT NULL
                        REFERENCES delegations(delegation_id),
    bytes_transferred   BIGINT,
    status              TEXT NOT NULL DEFAULT 'pending',
    created_at          TIMESTAMPTZ NOT NULL DEFAULT now(),
    completed_at        TIMESTAMPTZ
);
```

For compute, track jobs:

```
CREATE TABLE compute_jobs (
    job_id              TEXT PRIMARY KEY,
    delegation_id       UUID NOT NULL
                        REFERENCES delegations(delegation_id),
    nodes               INTEGER NOT NULL,
    walltime_seconds    INTEGER,
    node_hours_charged  REAL,
    status              TEXT NOT NULL DEFAULT 'pending',
    created_at          TIMESTAMPTZ NOT NULL DEFAULT now(),
    completed_at        TIMESTAMPTZ
);
```

The `status` field takes values: `pending`, `running`, `completed`, `failed`, or `cancelled`.

4 API Specification

Base URL: `https://delegations.<facility>.gov/v1` (e.g., `delegations.alcf.anl.gov`)

All endpoints require a valid Globus Auth token. The service extracts the caller's identity from the token.

4.1 Create Delegation

```
POST /delegations
```

Storage example:

```
{
  "grantee": "urn:globus:auth:identity:12345-...",
  "resource_type": "storage",
  "resource_id": "alcf-eagle-uuid",
  "scope": {
    "path": "/projects/battery-data/agent-workspace",
    "operations": ["read", "write"]
  },
  "quota": {"bytes": 1099511627776},
  "expires_at": "2026-09-01T00:00:00Z"
}
```

Compute example:

```
{
  "grantee": "urn:globus:auth:identity:12345-...",
  "resource_type": "compute",
  "resource_id": "aurora",
  "scope": {
    "queues": ["prod", "debug"],
    "max_nodes_per_job": 512
  },
  "quota": {"node_hours": 50000},
  "expires_at": "2026-09-01T00:00:00Z"
}
```

Response (201 Created):

```
{
```

```
  "delegation_id": "uuid-...",
  "parent_id": null,
  "delegator": "urn:globus:auth:identity:pi-...",
  "grantee": "urn:globus:auth:identity:12345-...",
  "resource_type": "storage",
  "resource_id": "alcf-eagle-uuid",
  "scope": {
    "path": "/projects/battery-data/agent-workspace",
    "operations": ["read", "write"]
  },
  "quota": {"bytes": 1099511627776},
  "consumed": {"bytes": 0},
  "suspended": false,
  "expires_at": "2026-09-01T00:00:00Z",
  "created_at": "2026-07-01T12:00:00Z"
}
```

Validation rules:

- Caller must have a valid (non-revoked, non-expired) delegation covering the requested scope, OR be authorized via project membership (for root delegations)
- Child scope must be a subset of parent scope (narrower path, fewer queues, lower limits)
- `expires_at` must be $\leq$ parent's `expires_at`
- Quota must be $\leq$ parent's available capacity

On success:

1. Provision enforcement (storage: create Globus ACL; compute: register with submission proxy)
2. Store delegation record with `enforcement_ref`

4.2 List Delegations

```
GET /delegations
```

Query parameters:

Parameter	Description
`grantee`	Filter by grantee identity
`delegator`	Filter by delegator identity
`resource_id`	Filter by resource
`include_revoked`	Include revoked delegations (default: false)

Returns an array of delegation objects.

4.3 Get Delegation

```
GET /delegations/{delegation_id}
```

Returns the delegation object, or 404 if not found.

4.4 Update Delegation

```
PATCH /delegations/{delegation_id}
```

Request body (all fields optional):

```
{
  "quota": {"bytes": 2199023255552},
  "expires_at": "2026-12-01T00:00:00Z"
}
```

Constraints:

- Only the delegator (or an ancestor in the chain) can update
- Can only increase quota up to parent's available capacity
- Can only extend expiration up to parent's expiration
- If quota is increased and `suspended` is true, restore permissions (write access for storage, job submission for compute)

4.5 Revoke Delegation

```
DELETE /delegations/{delegation_id}
```

On success:

1. Mark delegation as `revoked = true`, set `revoked_at`
2. Delete Globus ACL rule
3. Recursively revoke all child delegations

Returns 204 No Content.

4.6 Register Transfer

Agents should register transfers before submitting to Globus so the service can track them:

```
POST /delegations/{delegation_id}/transfers
```

Request body:

```
{
  "task_id": "globus-task-uuid",
  "destination_path": "/projects/battery-data/agent-workspace/results"
}
```

This creates a `storage_tasks` record. The webhook handler uses this to associate completions with delegations.

4.7 Transfer Completion Webhook

```
POST /webhooks/globus
```

Globus calls this endpoint when a transfer task completes. The service:

1. Looks up the task in `storage_tasks`
2. Updates `bytes_transferred` and `status`
3. If successful, updates delegation's `consumed` field
4. Checks quota thresholds:
   - At 80%: queue alert to PI
   - At 100%: suspend writes (update Globus ACL to read-only, set `suspended = true`)

5 Background Jobs

5.1 Expiration Worker

Runs periodically (hourly) to revoke expired delegations:

```
SELECT delegation_id FROM delegations
WHERE expires_at < now()
  AND NOT revoked;
```

For each result, invoke the revoke operation (which cascades to children and deletes ACLs).

5.2 Quota Alert Worker

Runs periodically (every 15 minutes) to send threshold alerts:

```
SELECT delegation_id, delegator, consumed, quota
FROM delegations
WHERE quota IS NOT NULL
  AND NOT revoked
  AND NOT suspended;
-- Application code checks: consumed >= quota * 0.8
```

For delegations crossing thresholds, send alerts to the delegator via the portal notification system.

6 Authentication and Authorization

The service uses Globus Auth for authentication:

1. Agent presents a Globus access token in the `Authorization` header
2. Service introspects the token to obtain the caller's identity
3. For delegation operations, service validates authority by traversing the delegation chain

For root delegations (no parent), the service verifies project membership via ALCF's allocation database or a configured policy endpoint.

Authorization rules:

- *Create*: Caller must hold a valid delegation covering the requested scope, or have project-level authority

- *Update*: Caller must be the delegator or an ancestor
- *Revoke*: Caller must be the delegator or an ancestor
- *Read*: Caller must be the delegator, grantee, or an ancestor

7 Agent Interaction Model

This section describes how agents interact with resource rights in practice. As described in Section 2, we use the implicit (identity-based) model: the delegation service creates enforcement entries (Globus ACLs for storage, proxy authorizations for compute) tied to the agent's identity. Agents access resources directly without presenting separate capability tokens.

7.1 Discovery

Although agents do not need to know about their rights to use them, they may want to:

- Avoid failed transfer attempts by checking permissions in advance
- Monitor quota consumption
- Detect impending expiration
- Choose among multiple delegated paths

Agents query their rights via the List Delegations endpoint:

```
GET /delegations?grantee={my_identity}
```

Response:

```
[{
  "delegation_id": "uuid-...",
  "resource_type": "storage",
  "resource_id": "eagle-uuid",
  "scope": {
    "path": "/projects/battery-data",
    "operations": ["read", "write"]
  },
  "quota": {"bytes": 1099511627776},
  "consumed": {"bytes": 450000000000},
  "suspended": false,
  "expires_at": "2026-09-01T00:00:00Z"
}]
```

This tells the agent: where it can read/write, how much quota remains, and when access expires.

7.2 Caching

Agents may cache discovered rights to reduce API calls. Recommended cache behavior:

- **TTL**: 5–15 minutes (rights can be revoked or suspended at any time)
- **Invalidation**: On transfer failure (403 or quota error), re-query immediately
- **Refresh**: Before operations on cached rights nearing expiration

For batch workflows that run infrequently, caching is unnecessary—query once at startup. For interactive agents making many small transfers, caching avoids redundant discovery calls.

7.3 Delegation by Agents

Agents holding delegations can further delegate to sub-agents via the Create Delegation endpoint. This enables hierarchical workflows where a coordinating agent parcels out storage access to specialized workers. The sub-agent receives its own Globus ACL and can operate independently within its delegated scope.

8 Example: Hierarchical Delegation

This section illustrates a complete scenario: a PI creates an agent with storage rights, and that agent delegates to sub-agents.

8.1 Scenario

Dr. Smith runs a materials discovery project. She deploys a coordinator agent that manages the workflow, spawning specialized sub-agents for different tasks:

- **Coordinator agent**: Orchestrates the workflow, needs full read/write access
- **Simulation sub-agent**: Runs DFT calculations, writes results (5 TB)
- **ML sub-agent**: Trains models on simulation data, writes checkpoints (5 TB)
- **Analysis sub-agent**: Reads all data for post-processing, no write access

All agents have Globus identities (registered as Globus applications).

8.2 Step 1: PI Creates Root Delegation

Dr. Smith uses the delegation portal (or calls the API directly) to grant the coordinator agent 10 TB of write access on Eagle:

```
POST /delegations
Authorization: Bearer <dr-smith-token>
```

```
{
  "grantee": "urn:globus:auth:identity:coord-agent-uuid",
  "resource_type": "storage",
  "resource_id": "alcf-eagle-uuid",
  "scope": {
    "path": "/projects/materials-discovery",
    "operations": ["read", "write"]
  },
  "quota": {"bytes": 10995116277760},
  "expires_at": "2027-01-01T00:00:00Z"
}
```

Response:

```
{
  "delegation_id": "d1-root-uuid",
  "parent_id": null,
  "delegator": "urn:globus:auth:identity:dr-smith-uuid",
  "grantee": "urn:globus:auth:identity:coord-agent-uuid",
  "resource_type": "storage",
  "resource_id": "alcf-eagle-uuid",
  "scope": {
    "path": "/projects/materials-discovery",
    "operations": ["read", "write"]
  },
  "quota": {"bytes": 10995116277760},
  "consumed": {"bytes": 0},
  "suspended": false,
  "expires_at": "2027-01-01T00:00:00Z",
  "created_at": "2026-07-01T10:00:00Z"
}
```

The delegation service creates a Globus ACL granting the coordinator agent read/write access to `/projects/materials-discovery` on Eagle.

8.3 Step 2: Coordinator Delegates to Simulation Sub-Agent

The coordinator agent creates a delegation for the simulation sub-agent with 5 TB write quota, scoped to a subdirectory:

```
POST /delegations
Authorization: Bearer <coord-agent-token>
```

```
{
  "grantee": "urn:globus:auth:identity:sim-agent-uuid",
  "resource_type": "storage",
  "resource_id": "alcf-eagle-uuid",
  "scope": {
    "path": "/projects/materials-discovery/simulations",
    "operations": ["read", "write"]
  },
  "quota": {"bytes": 5497558138880},
  "expires_at": "2026-12-01T00:00:00Z"
}
```

Response:

```
{
  "delegation_id": "d2-sim-uuid",
  "parent_id": "d1-root-uuid",
  "delegator": "urn:globus:auth:identity:coord-agent-uuid",
  "grantee": "urn:globus:auth:identity:sim-agent-uuid",
  "resource_type": "storage",
  "resource_id": "alcf-eagle-uuid",
  "scope": {
    "path": "/projects/materials-discovery/simulations",
    "operations": ["read", "write"]
  },
  "quota": {"bytes": 5497558138880},
  "consumed": {"bytes": 0},
  ...
}
```

The service validates:

- Coordinator holds a valid delegation covering this path and operations
- Requested path is a subdirectory of coordinator's path

- Requested quota (5 TB) $\leq$ coordinator's available capacity (10 TB)
- Requested expiration $\leq$ coordinator's expiration

On success, the delegation service creates a new Globus ACL granting the simulation sub-agent read/write access to `/projects/materials-discovery/simulations`. The coordinator does not interact with Globus directly—it calls the delegation service, which provisions enforcement. This keeps the delegation tree and ACLs synchronized.

8.4 Step 3: Coordinator Delegates to ML Sub-Agent

Similarly, for the ML sub-agent with 5 TB:

```
POST /delegations
Authorization: Bearer <coord-agent-token>
```

```
{
  "grantee": "urn:globus:auth:identity:ml-agent-uuid",
  "resource_type": "storage",
  "resource_id": "alcf-eagle-uuid",
  "scope": {
    "path": "/projects/materials-discovery/ml-training",
    "operations": ["read", "write"]
  },
  "quota": {"bytes": 5497558138880},
  "expires_at": "2026-12-01T00:00:00Z"
}
```

After this delegation, the coordinator's available capacity is exhausted: 10 TB allocated, 5 TB to simulation, 5 TB to ML. Any further write delegations would be rejected until quota is freed or increased.

8.5 Step 4: Coordinator Delegates Read-Only to Analysis Sub-Agent

The analysis sub-agent needs to read all data but should not write. Since read-only access does not consume write quota, this delegation succeeds:

```
POST /delegations
Authorization: Bearer <coord-agent-token>
```

```
{
```

```
  "grantee": "urn:globus:auth:identity:analysis-agent-uuid",
  "resource_type": "storage",
  "resource_id": "alcf-eagle-uuid",
  "scope": {
    "path": "/projects/materials-discovery",
    "operations": ["read"]
  },
  "expires_at": "2026-12-01T00:00:00Z"
}
```

Response:

```
{
  "delegation_id": "d4-analysis-uuid",
  "parent_id": "d1-root-uuid",
  "delegator": "urn:globus:auth:identity:coord-agent-uuid",
  "grantee": "urn:globus:auth:identity:analysis-agent-uuid",
  "resource_type": "storage",
  "resource_id": "alcf-eagle-uuid",
  "scope": {
    "path": "/projects/materials-discovery",
    "operations": ["read"]
  },
  "quota": null,
  ...
}
```

No `quota` is specified because the analysis agent has no write access and therefore no transfer quota.

8.6 Resulting Delegation Tree

```
Dr. Smith (PI)
    |
    +-- d1: Coordinator Agent
        |   path: /projects/materials-discovery
        |   operations: [read, write]
        |   quota: 10 TB
        |
        +-- d2: Simulation Sub-Agent
        |       path: /projects/materials-discovery/simulations
        |       operations: [read, write]
```

```
|       quota: 5 TB
|
+-- d3: ML Sub-Agent
|       path: /projects/materials-discovery/ml-training
|       operations: [read, write]
|       quota: 5 TB
|
+-- d4: Analysis Sub-Agent
        path: /projects/materials-discovery
        operations: [read]
        quota: (none)
```

8.7 Sub-Agent Operation

Each sub-agent can now use Globus directly:

Simulation sub-agent transfers DFT results to Eagle:

```
POST https://transfer.api.globus.org/v0.10/transfer
Authorization: Bearer <sim-agent-token>

{
  "source_endpoint": "compute-scratch-uuid",
  "destination_endpoint": "alcf-eagle-uuid",
  "DATA": [{
    "source_path": "/scratch/dft-results/",
    "destination_path": "/projects/materials-discovery/simulations/run
       -042/",
    "recursive": true
  }]
}
```

Globus checks the ACL (created when delegation d2 was issued) and permits the transfer. The delegation service receives the completion webhook and updates the `consumed` field for delegation d2.

Analysis sub-agent reads data:

```
POST https://transfer.api.globus.org/v0.10/transfer
Authorization: Bearer <analysis-agent-token>

{
```

```
  "source_endpoint": "alcf-eagle-uuid",
  "destination_endpoint": "analysis-workspace-uuid",
  "DATA": [{
    "source_path": "/projects/materials-discovery/",
    "destination_path": "/workspace/incoming/",
    "recursive": true
  }]
}
```

Globus permits the read. If the analysis agent attempted to write to Eagle, Globus would reject it (ACL grants read-only).

8.8 Revocation

If Dr. Smith revokes the coordinator's delegation (d1):

```
DELETE /delegations/d1-root-uuid
Authorization: Bearer <dr-smith-token>
```

The delegation service:

1. Marks d1 as revoked
2. Deletes the Globus ACL for the coordinator
3. Recursively revokes d2, d3, and d4
4. Deletes their Globus ACLs

All four agents immediately lose access to Eagle storage.

9 Resource Type Adapters

The delegation service uses pluggable adapters to provision and manage enforcement for different resource types. Each adapter implements the same interface: provision, update, and revoke.

9.1 Storage Adapter: Globus Integration

The storage adapter uses the Globus SDK for ACL management.

9.1.1 Creating an ACL Rule

When a delegation is created:

```
from globus_sdk import TransferClient

tc = TransferClient(authorizer=...)

rule_data = {
    "DATA_TYPE": "access",
    "principal_type": "identity",
    "principal": grantee_identity,
    "path": path,
    "permissions": "rw" if "write" in operations else "r",
}
if expires_at:
    rule_data["expiration"] = expires_at.isoformat()

result = tc.add_endpoint_acl_rule(collection_id, rule_data)
acl_id = result["access_id"]
```

9.1.2 Updating an ACL Rule

For write suspension when quota is exceeded:

```
tc.update_endpoint_acl_rule(
    collection_id,
    acl_id,
    {"permissions": "r"}
)
```

For restoring write access after quota increase:

```
tc.update_endpoint_acl_rule(
    collection_id,
    acl_id,
    {"permissions": "rw"}
)
```

9.1.3 Deleting an ACL Rule

On revocation:

```
tc.delete_endpoint_acl_rule(collection_id, acl_id)
```

9.1.4 Transfer Completion Events

The service must receive notifications when transfers complete. Options:

- **Globus Flows**: Subscribe to task completion events via a flow that calls the webhook endpoint
- **Polling**: Periodically query the Tasks API for pending tasks in `storage_tasks`

Flows are preferred for lower latency; polling provides a fallback.

9.2 Compute Adapter: Submission Proxy Integration

The compute adapter integrates with a facility's submission proxy. The examples here use the ALCF proxy (described in the ALCF implementation document), but the same pattern applies to any proxy that validates resource rights before submitting jobs to a batch scheduler.

9.2.1 Registering Authorization

When a compute delegation is created, the adapter registers it with the submission proxy:

```
import requests

auth_data = {
    "delegation_id": str(delegation_id),
    "agent_identity": grantee_identity,
    "project": project_id,
    "scope": {
        "queues": scope["queues"],
        "max_nodes_per_job": scope.get("max_nodes_per_job"),
        "max_walltime_hours": scope.get("max_walltime_hours")
    },
    "quota": {"node_hours": quota["node_hours"]},
    "expires_at": expires_at.isoformat() if expires_at else None
}
```

```
response = requests.post(
    f"{PROXY_URL}/authorizations",
    json=auth_data,
    headers={"Authorization": f"Bearer␣{service_token}"}
)
auth_ref = response.json()["authorization_id"]
```

The `authorization_id` is stored as `enforcement_ref` in the delegation record.

9.2.2 Job Submission Flow

When an agent submits a job through the proxy:

1. Agent calls proxy with job specification and its Globus token
2. Proxy looks up authorizations for the agent's identity
3. Proxy validates: requested queue in allowed queues? nodes $\leq$ max? walltime $\leq$ max?
4. Proxy checks quota: estimated node-hours $\leq$ remaining?
5. If valid, proxy submits to PBS and records job in `compute_jobs`
6. On job completion, proxy reports actual node-hours to delegation service

9.2.3 Consumption Tracking

The proxy reports job completions to the delegation service:

```
POST /webhooks/compute
```

```
{
  "job_id": "12345.aurora-pbs",
  "delegation_id": "uuid-...",
  "status": "completed",
  "nodes": 256,
  "walltime_seconds": 3600,
  "node_hours_charged": 256.0
}
```

The service updates `consumed.node_hours` and checks quota thresholds.

9.2.4 Quota Suspension for Compute

Unlike storage (where writes can be suspended while reads continue), compute quota exhaustion blocks all job submissions. When quota is exceeded:

1. Mark delegation as `suspended = true`
2. Notify proxy to reject submissions for this delegation
3. Notify PI via portal

Running jobs are not terminated—only new submissions are blocked.

9.2.5 Revoking Authorization

On delegation revocation:

```
requests.delete(
    f"{PROXY_URL}/authorizations/{auth_ref}",
    headers={"Authorization": f"Bearer␣{service_token}"}
)
```

The proxy immediately stops accepting jobs for this delegation. Running jobs may continue to completion (or can be killed, depending on policy).

10 Deployment

The delegation service can be deployed at multiple scales, from a local pilot to a cloud-hosted multi-facility service. This section describes a phased approach using ALCF as the reference facility, but the same progression applies to any deployment.

10.1 Phase 1: Local Deployment

For initial development and pilot with a small number of agents:

- **API server**: Single FastAPI instance running on a VM or container at the facility
- **Database**: SQLite for initial prototyping, PostgreSQL for pilot (single instance)
- **Background jobs**: In-process scheduler (e.g., APScheduler) or cron—no distributed worker needed
- **Secrets**: Environment variables or local file; Globus client credentials for a test application

This configuration supports:

- Rapid iteration on the API and data model
- Testing with real Globus collections
- Pilot with 2–5 agents and a small number of PIs

No high availability, no load balancing. Acceptable for a pilot where downtime is tolerable.

10.2 Phase 2: Facility-Hosted Production

For production use at facility scale (e.g., ALCF):

- **API server**: Multiple replicas behind a load balancer (nginx or HAProxy)
- **Database**: Managed PostgreSQL with automated backups
- **Background jobs**: Celery with Redis, or a managed job scheduler
- **Secrets**: HashiCorp Vault or Kubernetes secrets
- **Monitoring**: Prometheus metrics, structured logging, alerting

This configuration supports hundreds of agents and PIs with high availability. Each facility runs its own instance with facility-specific adapter configuration.

10.3 Phase 3: Globus-Hosted Service

The long-term goal is for the delegation service to be operated by Globus as a cloud-hosted component, similar to Globus Auth and Globus Transfer. Benefits:

- No facility operational burden
- Consistent API across facilities (ALCF, OLCF, NERSC, etc.)
- Built-in federation: delegations at one facility recognized at others
- Integration with Globus portal and developer tools

The API and data model described in this document are designed to be portable: the same interface works whether the service runs locally, at a facility, or in Globus cloud infrastructure. Migration from Phase 2 to Phase 3 requires data export/import and DNS cutover, not application changes.

10.4 Globus Application Registration

Regardless of deployment phase, the service requires a Globus application with:

- Scopes for Transfer API (ACL management on target collections)
- Scopes for Auth API (token introspection)
- A client identity granted ACL management permissions on the facility's Globus collections

For Phase 1, register a "native app" or "confidential client" via the Globus developer portal. For Phase 3, the application would be managed by Globus operations.

11 Future Extensions

Hard quota enforcement. Route all transfers through the service with reservation tracking. Before submitting to Globus, reserve capacity; on completion, reconcile actual bytes against reservation.

Policy constraints. Add a `constraints` JSONB column to store arbitrary predicates (time-of-day restrictions, source/destination filtering, approval requirements). Evaluate constraints at transfer registration time.

Delegation templates. Pre-approved delegation patterns that PIs can instantiate without specifying all parameters. For example, a "standard agent workspace" template with default quota and expiration.

Audit log. A separate append-only table recording all operations (create, update, revoke, transfer) with full context for compliance and forensics.

Multi-facility federation. Accept delegation tokens from peer facilities (OLCF, NERSC) by validating signatures against a federated trust store.

This service implements the resource rights model described in "Implementing Resource Rights at ALCF."