# Policy Composition: Configuration, Not Code

Terrell Russell, Ph.D.
@terrellrussell
Chief Technologist, iRODS Consortium

January 25-29, 2021
CS3 2021
Virtual

# iRODS CONSORTIUM

renci

RESEARCH \ ENGAGEMENT \ INNOVATION

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

# Our Membership

iRODS



















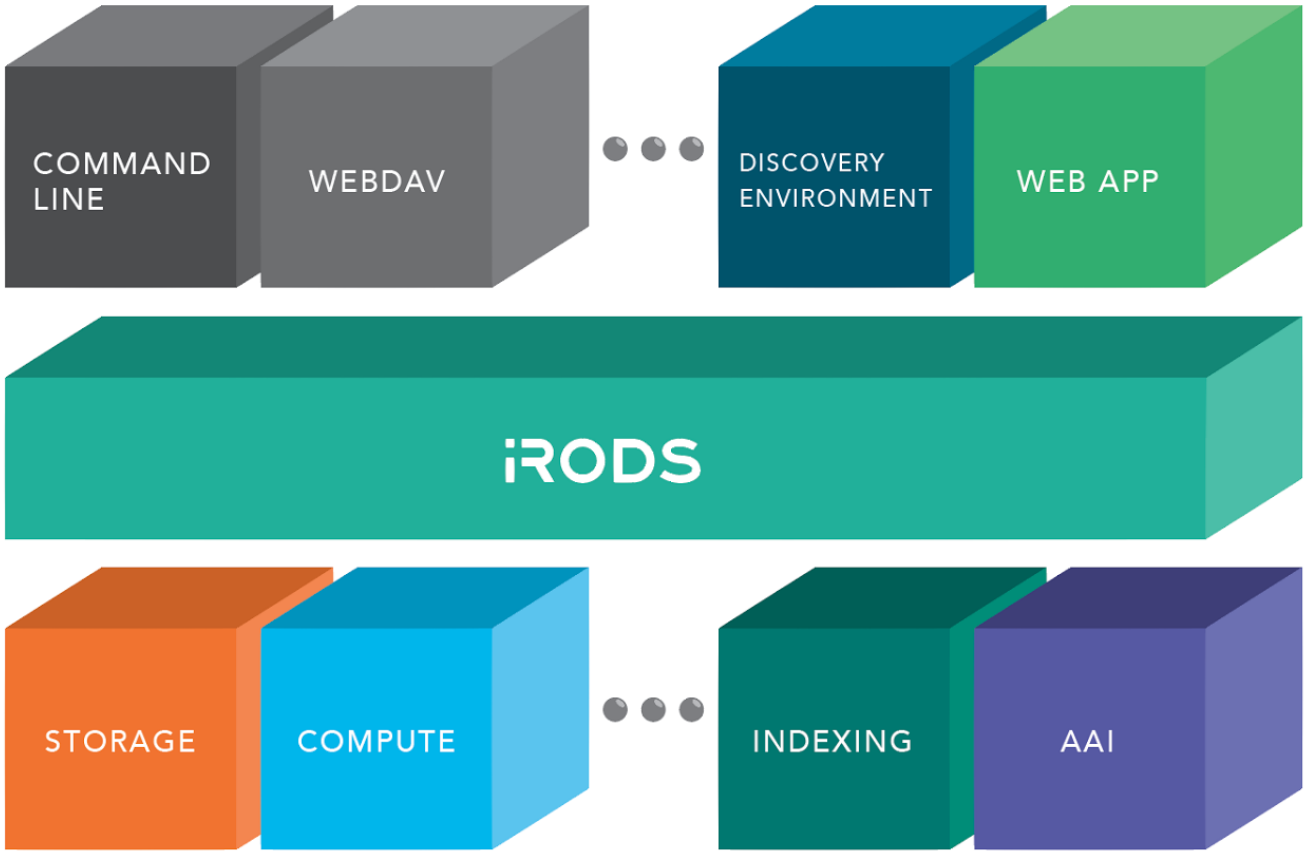








# iRODS as the Integration Layer




iRODS Clients

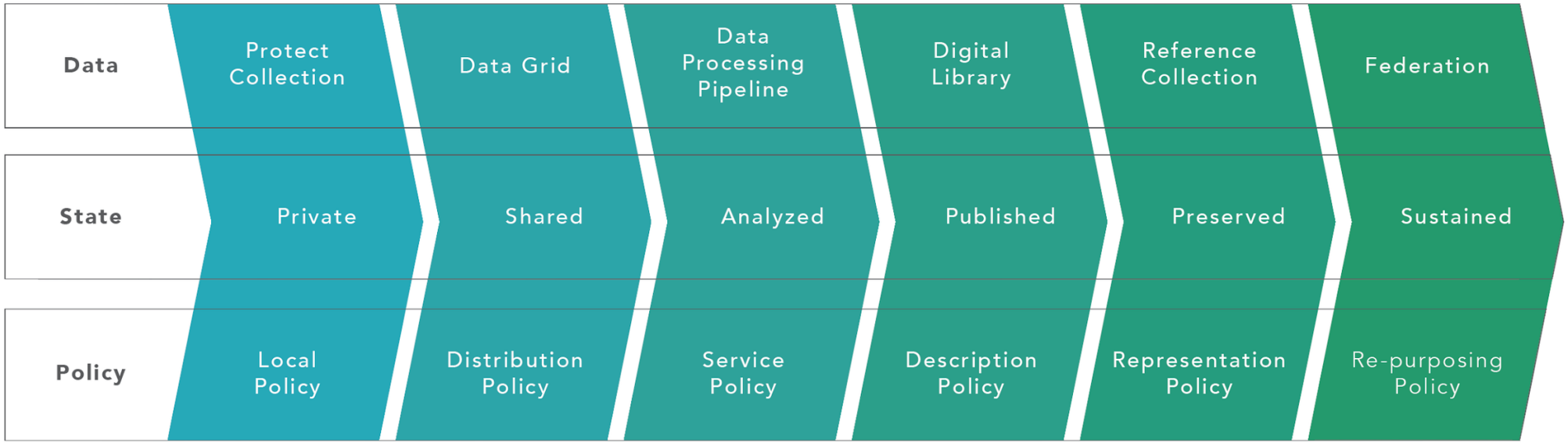
iRODS provides a layer of abstraction which integrates with your pre-existing infrastructure.

This flexibility allows your infrastructure to continue to change over time.

Existing Infrastructure

iRODS

# DATA LIFECYCLE




iRODS virtualizes the stages of the data lifecycle through policy evolution

As data matures and reaches a broader community, data management policy must also evolve to meet these additional requirements.

# Policy Composition: Configuration, Not Code

With a twenty-five year history, iRODS open source technology has been used to automate data management across many scientific and business domains. The scale and value of data across these domains drives the necessity for automation. This variety also demands a flexibility in data management policies over time. Organizations have satisfied their own needs by investing in the development of their own policy, but this has led to monolithic, specific policy sets tailored to a particular organization.

As a community, we have observed common themes and duplication of effort across these organizations and now worked to provide generalized implementations that can be deployed across multiple domains. This new approach allows multiple policies to be configured together, or composed, without the need for custom development. For example, our Storage Tiering capability is a composition of several basic policies: Replication, Verification, Retention, and the Violating Object Discovery.

iRODS

A Definition of Data Management

"The development, execution and supervision of plans, **policies**, programs, and **practices** that control, protect, deliver, and enhance the value of data and information assets."

Organizations need a **future-proof** solution to managing data and its surrounding infrastructure

A Definition of Policy

A set of ideas or a **plan** of what to do in **particular situations** that has been agreed to officially by a group of people…

So how does iRODS do this?

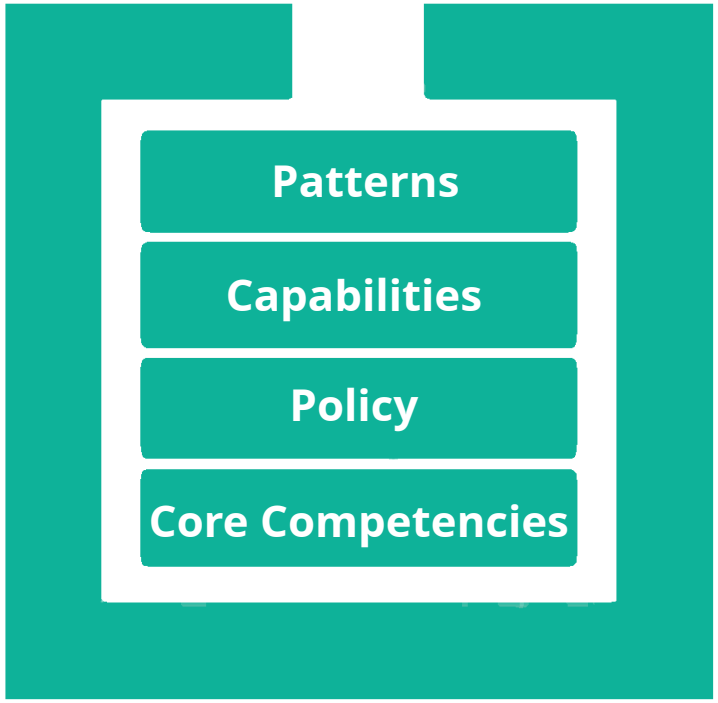
The reflection of real world data management decisions in computer actionable code.

(a plan of what to do in particular situations)

# Possible Policies

- Data Movement
- Data Verification
- Data Retention
- Data Replication
- Data Placement
- Checksum Validation
- Metadata Extraction
- Metadata Application
- Metadata Conformance
- Replica Verification
- Vault to Catalog Verification
- Catalog to Vault Verification
- …

# The iRODS Data Management Model




- How can we help new users get started?
- How can we make policy reusable?
- How can we simplify policy development?
- How can we provide a cook book of deployments?
- How do we get from Policy to Capabilities?

# Policy Composition

Consider Policy as building blocks towards Capabilities

Follow proven software engineering principles:

- Favor composition over monolithic implementations

Rules and Dynamic Policy Enforcement Points
can be overloaded and fall through

Implement or configure several rule bases or rule
engine plugins to achieve complex use cases

Consider Storage Tiering as a collection of policies:

- Data Access Time
- Identifying Violating Objects
- Data Replication
- Data Verification
- Data Retention

Continue to separate the concerns:

- When : Which policy enforcement points
- What : The policy to be invoked
- Why : What are the conditions necessary for invocation
- How : Synchronous or Asynchronous

Write simple policy implementations

- Not tied to a Policy Enforcement Point
- Do one thing well
- How it is invoked is of no concern

Each policy may now be reused in a generic fashion, favoring configuration over code.

# Examples

# Synchronous Access Time

```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {
                "active_policy_clauses" :  ["post"],
                "events" : ["put", "get", "create", "read", "write", "rename",
                            "register", "unregister", "replication", "checksum",
                            "copy", "seek", "truncate"],
                "policy"    : "irods_policy_access_time",
                "configuration" : {
                }
            }
        ]
    }
}
```

# Synchronous Replication

```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {   "active_policy_clauses" : ["post"],
                "events" : ["create", "write", "registration"],
                "policy"    : "irods_policy_data_replication",
                "configuration" : {
                    "source_to_destination_map" : {
                        "source_resource_0" : ["destination_resource_0a", "destination_resource_0b"],
                        "source_resource_1" : ["destination_resource_1a"],
                    }
                }
            },
            {   "active_policy_clauses" : ["post"],
                "events" : ["create", "write", "registration"],
                "policy"    : "irods_policy_data_replication",
                "configuration" : {
                    "destination_resource" : "destination_resource_3"
                    }
                }
            },
        ]
    }
}
```

# Asynchronous Replication

```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {
                "active_policy_clauses" : ["post"],
                "events" : ["create", "write", "registration"],
                "policy" : "irods_policy_enqueue_rule",
                "delay_conditions" : "<ET>PLUSET 1</ET>",
                "payload" : {
                    "policy" : "irods_policy_execute_rule",
                    "payload" : {
                        "policy_to_invoke" : "irods_policy_data_replication",
                        "configuration" : {
                            "source_to_destination_map" : {
                                "source_resource_0" : ["destination_resource_0a", "destination_resource_0b"],
                                "source_resource_1" : ["destination_resource_1a"],
                            }
                        }
                    }
                }
            }
        ]
    }
}
```

# Synchronous Retention

```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {
                "active_policy_clauses" : ["post"],
                "events" : ["replication"],
                "policy" : "irods_policy_data_retention",
                "configuration" : {
                    "mode" : "trim_single_replica",
                    "source_resource_list" : ["source_resource_1", "source_resource_2"]
                }
            }
        ]
    }
}
```

# Query Driven Asynchronous Retention

```
{
    "policy" : "irods_policy_enqueue_rule",
    "delay_conditions" : "<EF>REPEAT FOR EVER</EF>",
    "payload" : {
        "policy" : "irods_policy_execute_rule",
        "payload" : {
            "policy_to_invoke" : "irods_policy_query_processor",
            "parameters" : {
                "query_string" : "SELECT USER_NAME, COLL_NAME, DATA_NAME, RESC_NAME WHERE
                                  COLL_NAME like '/tempZone/home/rods%' AND
                                  RESC_NAME IN ('source_resource_1', 'source_resource_2')",
                "query_limit" : 10,
                "query_type" : "general",
                "number_of_threads" : 4,
                "policy_to_invoke" : "irods_policy_data_retention",
                "configuration" : {
                    "mode" : "trim_single_replica",
                    "source_resource_list" : ["source_resource_1", "source_resource_2"]
                }
            }
        }
    }
}
```

# Synchronous Verification

```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {   "active_policy_clauses" : ["post"],
                "events" : ["create", "write", "registration"],
                "policy" : "irods_policy_data_verification",
                "configuration" : {
                    "attribute" : "irods::verification::type"
                }
            }
        ]
    }
}
```

The type of verification to perform is stored as metadata on the resource

- catalog
- filesystem
- checksum

# Asynchronous Verification

```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {   "active_policy_clauses" : ["post"],
                "events" : ["create", "write", "registration"],
                "policy" : "irods_policy_enqueue_rule",
                "delay_conditions" : "<ET>PLUSET 1</ET>",
                "payload" : {
                    "policy" : "irods_policy_execute_rule",
                    "payload" : {
                        "policy" : "irods_policy_data_verification",
                        "configuration" : {
                            "attribute" : "irods::verification::type"
                        }
                    }
                }
            }
        ]
    }
}
```

The type of verification to perform is stored as metadata on the resource

- catalog
- filesystem
- checksum

# Policy Composed Capabilities

# Storage Tiering Overview

iRODS

*Periodically, the storage tiering policy discovers data objects in violation via a **default query** and schedules their migration to the next tier group.*

UNIFIED NAMESPACE

*After 1800 seconds, any data objects in violation are automatically replicated to tier 1, and then once at rest, they are trimmed from tier 0.*

*After 9000 seconds, any data objects in violation are automatically replicated to tier 2, and then once at rest, they are trimmed from tier 1.*

DATA SOURCES

Coordinating Resource

A: irods::storage_tier_time
V: 1800
U:

A: irods::storage_tier_group
V: example_group
U: 0

A: irods::storage_tier_time
V: 9000
U:

A: irods::storage_tier_group
V: example_group
U: 1

A: irods::storage_tier_group
V: example_group
U: 2

Tier 0
(*FAST*)

Tier 1
(*INTERMEDIATE*)

Tier 2
(*SLOW*)

*The **default query** that determines which data objects are in violation can be overridden by adding a new metadata attribute **irods::storage_tier_query** with a value that defines the custom query.*

YOUR ORGANIZATION

FEDERATE SECURELY

OTHER ORGANIZATION

- Data Virtualization ( Unified Namespace )
- Data Discovery ( Metadata )
- Workflow Automation ( Rule Engine )
- Secure Collaboration ( Federation )

# Policy Composed Storage Tiering

- Asynchronous Discovery
- Asynchronous Replication
- Synchronous Retention
- Resource associated metadata
- Identified by 'tiering groups'

# Asynchronous Discovery and Replication

```
{
    "policy" : "irods_policy_execute_rule",
    "payload" : {
        "policy_to_invoke" : "irods_policy_query_processor",
        "configuration" : {
            "query_string" : "SELECT META_RESC_ATTR_VALUE WHERE META_RESC_ATTR_NAME = 'irods::storage_tiering::group'",
            "query_limit" : 0,
            "query_type" : "general",
            "number_of_threads" : 8,
            "policy_to_invoke" : "irods_policy_event_generator_resource_metadata",
            "configuration" : {
                "conditional" : {
                    "metadata" : {
                        "attribute" : "irods::storage_tiering::group",
                        "value" : "{0}"
                    }
                },
                "policies_to_invoke" : [
                    {
                        "policy" : "irods_policy_query_processor",
                        "configuration" : {
                            "query_string" : "SELECT META_RESC_ATTR_VALUE WHERE META_RESC_ATTR_NAME = 'irods::storage_tiering::query' AND RESC_NAME = 'IRODS_TOKEN_S
                            "default_results_when_no_rows_found" : ["SELECT USER_NAME, COLL_NAME, DATA_NAME, RESC_NAME WHERE META_DATA_ATTR_NAME = 'irods::access_ti
                            "query_limit" : 0,
                            "query_type" : "general",
                            "number_of_threads" : 8,
                            "policy_to_invoke" : "irods_policy_query_processor",
                            "configuration" : {
                                "lifetime" : "IRODS_TOKEN_QUERY_SUBSTITUTION_END_TOKEN(SELECT META_RESC_ATTR_VALUE WHERE META_RESC_ATTR_NAME = 'irods::storage_tieri
                                "query_string" : "{0}",
                                "query_limit" : 0,
                                "query_type" : "general",
                                "number_of_threads" : 8,
                                "policy_to_invoke" : "irods_policy_data_replication",
                                "configuration" : {
                                    "comment" : "source_resource, and destination_resource supplied by the resource metadata event generator"
                                }
                            }
                        }
                    }
                ]
            }
        }
    }
}
INPUT null
OUTPUT ruleExecOut
```

# Synchronous Configuration for Storage Tiering

```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {
                "active_policy_clauses" : ["post"],
                "events" : ["put", "get", "create", "read", "write", "rename", "register", "unregister", "replication", "checksum", "copy", "seek", "truncate"],
                 "policy" : "irods_policy_access_time",
                 "configuration" : {
                    "log_errors" : "true"
                 }
            },
            {
                "active_policy_clauses" : ["post"],
                "events" : ["read", "write", "get"],
                "policy"    : "irods_policy_data_restage",
                "configuration" : {
                }
            },
            {
                "active_policy_clauses" : ["post"],
                "events" : ["replication"],
                    "policy"    : "irods_policy_tier_group_metadata",
                    "configuration" : {
                    }

            },
            {
                "active_policy_clauses" : ["post"],
                "events" : ["replication"],
                    "policy"    : "irods_policy_data_verification",
                    "configuration" : {
                    }

            },
            {
                "active_policy_clauses" : ["post"],
                "events" : ["replication"],
                    "policy"    : "irods_policy_data_retention",
                    "configuration" : {
                        "mode" : "trim_single_replica",
                        "log_errors" : "true"
                    }

            }
        ]
    }
}
```

# Possible Metadata Driven Restage for Storage Tiering

```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-metadata_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-metadata_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {
                "active_policy_clauses" : ["post"],
                "events" : ["set", "add"],
                "attribute" : "irods::storage_tiering::restage",
                "value"     : "*.*",
                "policy"    : "irods_policy_data_restage",
                "configuration" : {
                }
            }
        ]
    }
}
```

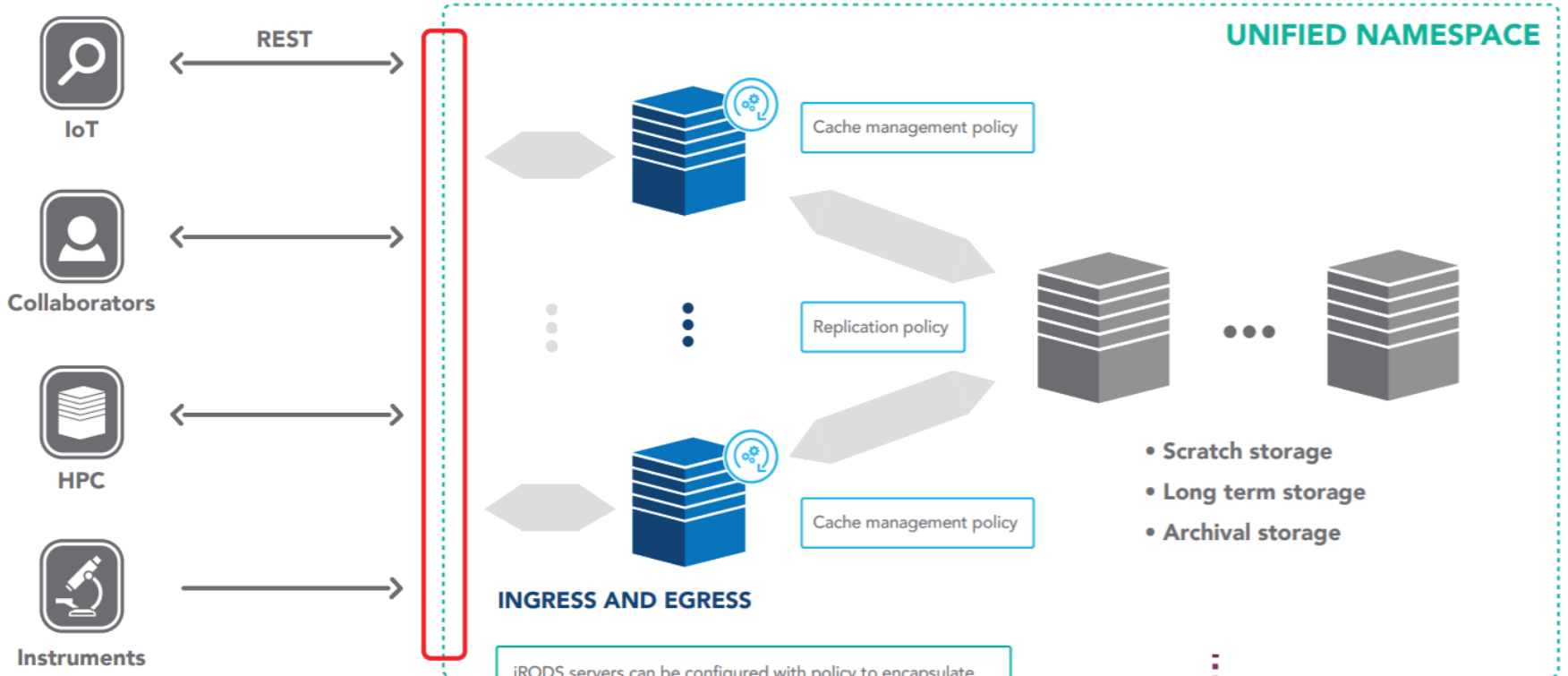
# Data Transfer Nodes Pattern

iRODS

Moving large datasets across organizational boundaries remains a challenge due to the requirement of exposing high performance hardware to the public network. Data Transfer Nodes (DTNs) provide a secure location for ingress and egress of data while avoiding the performance impact of an organizational firewall.

In the following deployment pattern, iRODS satisfies the requirements of a Science DMZ while also providing automated data management.

> The Science DMZ is a portion of the network, built at or near the campus or laboratory's local network perimeter that is designed such that the equipment, configuration, and security policies are optimized for high-performance scientific applications rather than for general-purpose business systems or 'enterprise' computing.
>
> —ESnet




iRODS servers can be configured with policy to encapsulate read and write operations which trigger replication within the Zone. This allows the staging of data for ingress and egress to be separated from scratch storage, long term storage, or archival storage.

Because the iRODS servers act as Data Transfer Nodes (DTNs) the staged data should not be persistent. These servers implement active cache management via iRODS policy which limits the amount of required storage at the edge.

FEDERATE SECURELY

OTHER ORGANIZATION

- Data Virtualization ( Unified Namespace )
- Data Discovery ( Metadata )
- Workflow Automation ( Rule Engine )
- Secure Collaboration ( Federation )

irods.org

# Policy Composed Data Transfer Node

- Asynchronous Discovery
- Asynchronous Retention
- Synchronous Replication
- Resource associated metadata
- Identified by 'replication groups'

# Asynchronous Retention on Edge Resources

```
{
    "policy" : "irods_policy_enqueue_rule",
    "delay_conditions" : "<EF>REPEAT FOR EVER</EF>",
    "payload" : {
        "policy" : "irods_policy_execute_rule",
        "payload" : {
            "policy_to_invoke" : "irods_policy_query_processor",
            "parameters" : {
                "query_string" : "SELECT USER_NAME, COLL_NAME, DATA_NAME, RESC_NAME WHERE COLL_NAM
                "query_limit" : 10,
                "query_type" : "general",
                "number_of_threads" : 4,
                "policy_to_invoke" : "irods_policy_data_retention",
                "configuration" : {
                    "mode" : "trim_single_replica",
                    "source_resource_list" : ["edge_resource_1", "edge_resource_2"]
                }
            }
        }
    }
}
```

# Policy Composed Capability

## Synchronous Replication

```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {
                "conditional" : {
                    "logical_path" : "\/tempZone.*"
                },
                "pre_or_post_invocation" : ["post"],
                "events" : ["create", "write", "registration"],
                "policy" : "irods_policy_data_replication",
                "configuration" : {
                    "source_to_destination_map" : {
                        "edge_resource_0" : ["long_term_resource_0"],
                        "edge_resource_1" : ["long_term_resource_1"],
                    }
                }
            },
            {
                "conditional" : {
                    "logical_path" : "\/tempZone.*"
                },
                "active_policy_clauses" : ["pre"],
                "events" : ["get"],
                "policy" : "irods_policy_data_replication",
                "configuration" : {
                    "source_to_destination_map" : {
                        "long_term_resource_0" : ["edge_resource_0"],
                        "long_term_resource_1" : ["edge_resource_1"]
                    }
                }
            }
        ]
    }
}
```

# Indexing Capability

iRODS

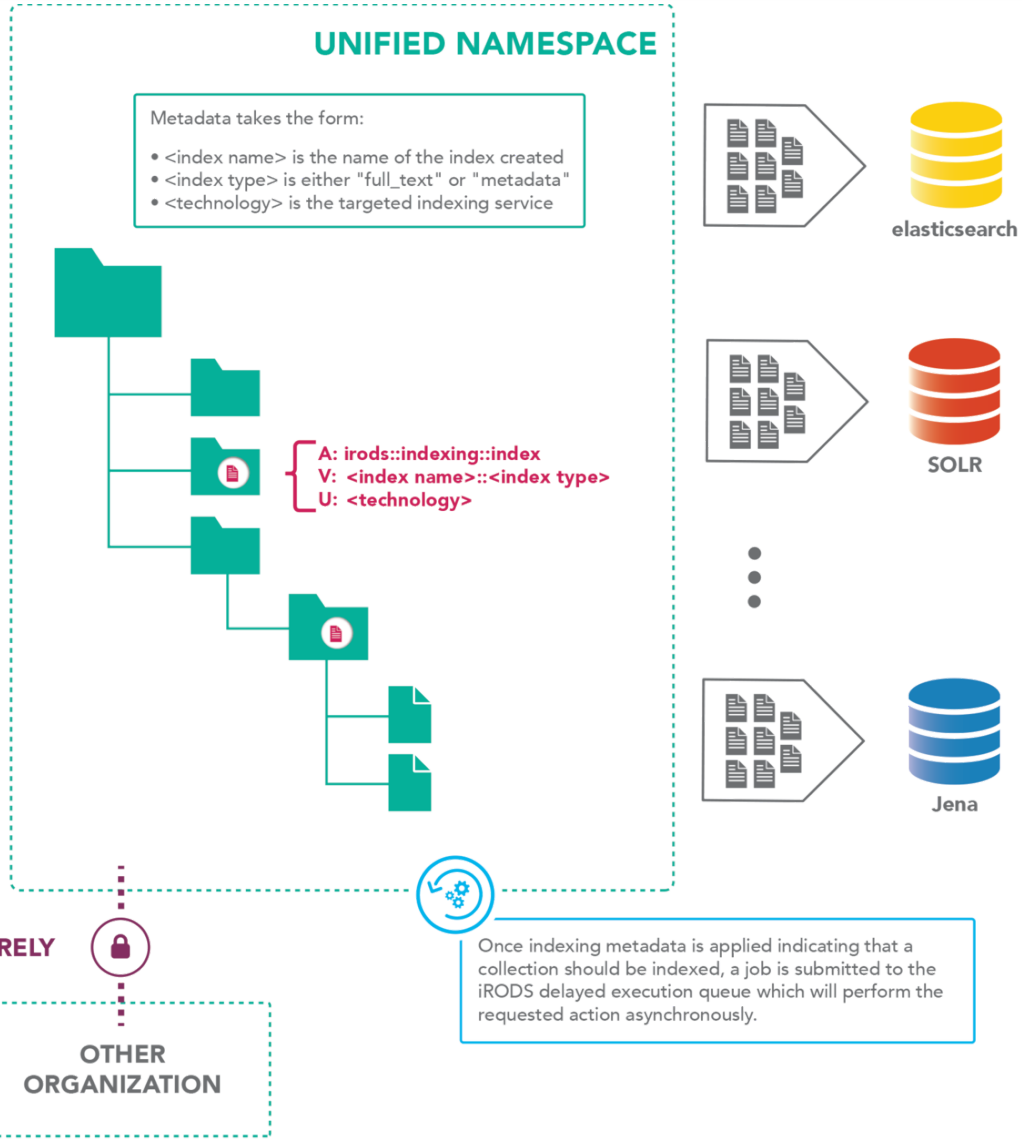
The iRODS Indexing Capability provides a policy framework around both full text and metadata indexing for the purposes of enhanced data discovery.

Logical collections are annotated with metadata which indicates that any data objects or nested collections of data objects should be indexed given a particular indexing technology, index type, and index name.

From the configured metadata, the framework composes a rule name and then delegates to the policy implementation through the rule engine.

A new indexing technology can be supported via a rule base or policy engine which provides policy implementations of the form:

- irods_policy_indexing_object_index_<technology>
- irods_policy_indexing_object_purge_<technology>
- irods_policy_indexing_metadata_index_<technology>
- irods_policy_indexing_metadata_purge_<technology>




Data Virtualization ( Unified Namespace )

Data Discovery ( Metadata )

Workflow Automation ( Rule Engine )

Secure Collaboration ( Federation )

irods.org

Policy implemented as separate policy engine plugins

- irods_policy_indexing_full_text_index_elasticsearch
- irods_policy_indexing_full_text_purge_elasticsearch
- irods_policy_indexing_metadata_index_elasticsearch
- irods_policy_indexing_metadata_purge_elasticsearch

iRODS

# Synchronously configured full text indexing

```
"instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
"plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
'plugin_specific_configuration': {
    "policies_to_invoke" : [
        {
            "active_policy_clauses" : ["post"],
            "events" : ["put", "write"],
            "policy"    : "irods_policy_event_delegate_collection_metadata",
            "configuration" : {
                "policies_to_invoke" : [
                    {
                        "conditional" : {
                            "metadata" : {
                                "attribute" : "irods::indexing::index",
                                "entity_type" : "data_object"
                            },
                        },
                        "policy"    : "irods_policy_indexing_full_text_index_elasticsearch",
                        "configuration" : {
                            "hosts" : ["http://localhost:9200/"],
                            "bulk_count" : 100,
                            "read_size" : 1024
                        }
                    }
                ]
            }
        }
        ...
```

# Synchronously configured full text purge

```
        {
            "active_policy_clauses" : ["pre"],
            "events" : ["unlink", "unregister"],
            "policy"    : "irods_policy_event_delegate_collection_metadata",
            "configuration" : {
                "policies_to_invoke" : [
                    {
                        "conditional" : {
                            "metadata" : {
                                "attribute" : "irods::indexing::index",
                                "entity_type" : "data_object"
                            },
                        },
                        "policy"    : "irods_policy_indexing_full_text_purge_elasticsearch",
                        "configuration" : {
                            "hosts" : ["http://localhost:9200/"],
                            "bulk_count" : 100,
                            "read_size" : 1024
                        }
                    }
                ]
            }
        }
    ]
}
```

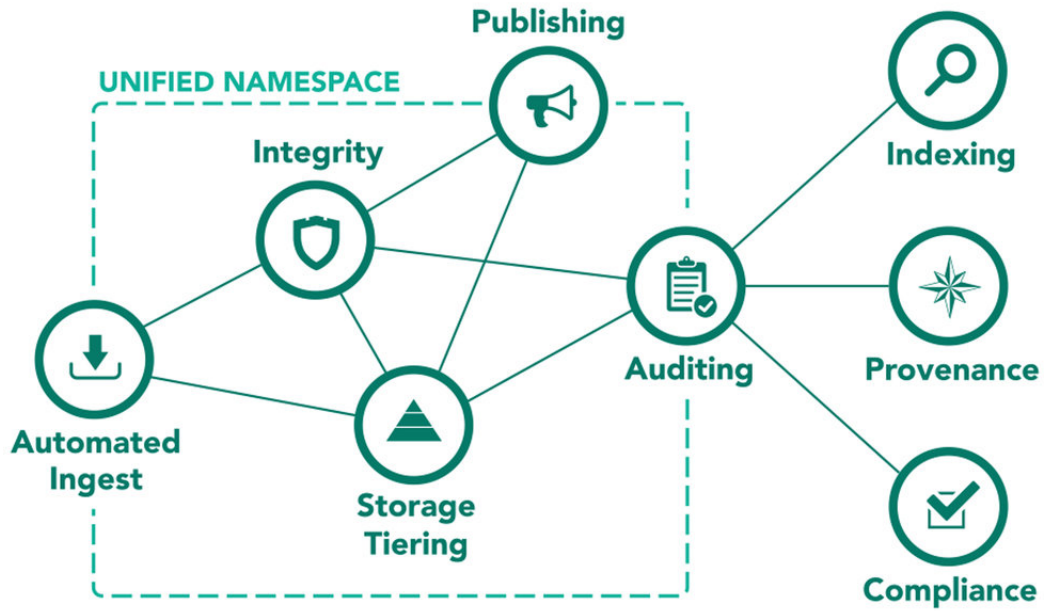
# iRODS | Data Management Model
## Ingest to Institutional Repository

iRODS provides eight packaged capabilities, each of which can be selectively deployed and configured.

These capabilities represent the most common use cases as identified by community participation and reporting.

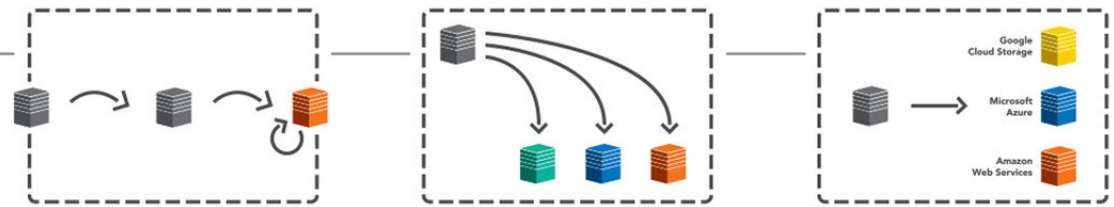
The flexibility provided by this model allows an organization to address its immediate use cases.

Additional capabilities may be deployed as any new requirements arise.




A pattern represents a combination of iRODS capabilities and data management policy consistent across multiple organizations.

Three common patterns of iRODS deployment have been observed within the community:




irods.org

# Summary - Configuration, Not Code

**Capabilities become easily configured recipes.**

**A Policy GUI is now a possibility with**

**simple manipulation of server side JSON.**

**Data management**

**should be**

**data-centric and metadata driven.**

**Future-proof automated data management**

**requires**

**open formats and open source.**