Policy Composition

June 9-12, 2020
iRODS User Group Meeting 2020
Virtual Event
Motivation

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

Patterns
Capabilities
Policy
Core Competencies
What is Data Management

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 Original Approach

In /etc/irods/core.re ...

```c
acPostProcForPut() {
    if ($rescName == "demoResc") {
        # extract and apply metadata
    }
    else if ($rescName == "cacheResc") {
        # async replication to archive
    }
    else if ($objPath like "/tempZone/home/alice/*" &&
        $rescName == "indexResc") {
        # launch an indexing job
    }
    else if (xyz) {
        # compute checksums ...
    }
    # and so on ...
}
```
Our second approach

Expanding policy implementation across rule bases

For example: pep_data_obj_put_post(...)

- Metadata extraction and application
- Asynchronous Replication
- Initiate Indexing
- Apply access time metadata
- Asynchronous checksum computation

Rather than one monolithic implementation, separate the implementations into individual rule bases, or plugins, and allow the rule(s) to fall through
Expanding policy across rule bases

Separate the implementation into several rule bases:

/etc/irods/metadata.re

```
  # metadata extraction and application code
  RULE_ENGINE_CONTINUE
}
```

/etc/irods/checksum.re

```
  # checksum code
  RULE_ENGINE_CONTINUE
}
```

/etc/irods/access_time.re

```
  # access time application code
  RULE_ENGINE_CONTINUE
}
```
Within the Rule Engine Plugin Framework, order matters

```
"rule_engines": [
  {
    "instance_name": "irods_rule_engine_plugin-irods_rule_language-instance",
    "plugin_name": "irods_rule_engine_plugin-irods_rule_language",
    "plugin_specific_configuration": {
      "re_rulebase_set": [
        "metadata",
        "checksum",
        "access_time",
        "core"
      ],
      ...
    },
    "shared_memory_instance": "irods_rule_language_rule_engine"
  },
  {
    "instance_name": "irods_rule_engine_plugin-cpp_default_policy-instance",
    "plugin_name": "irods_rule_engine_plugin-cpp_default_policy",
    "plugin_specific_configuration": {
    }
  }
]
```
Consider Policy as building blocks towards Capabilities

Follow proven software engineering principles:

**Favor composition over monolithic implementations**

Provide a common interface across policy implementations to allow transparent configuration
Initial work with Policy Composition

Consider Storage Tiering as a collection of policies:

- Data Access Time
- Identifying Violating Objects
- Data Replication
- Data Verification
- Data Retention
Policies invoked by monolithic framework plugins and delegated by convention:

- irods_policy_access_time
- irods_policy_data_movement
- irods_policy_data_replication
- irods_policy_data_verification
- irods_policy_data_retention

Each policy may be implemented by any rule engine, or rule base to customize for future use cases or technologies.
The New Approach

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.
The When
<table>
<thead>
<tr>
<th>RPC API</th>
<th>EVENT HANDLERS</th>
<th>POLICY INVOCATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_pep_auth_agent_auth_request_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_auth_agent_auth_request_pre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_auth_agent_auth_response_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_auth_agent_auth_response_pre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_auth_agent_start_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_auth_agent_start_pre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_auth_request_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_auth_request_pre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_auth_response_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_auth_response_pre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_data_obj_get_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_data_obj_get_pre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_data_obj_put_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_data_obj_put_pre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_database_check_auth_pre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_database_check_auth_pre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_database_close_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_database_close_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_database_get_rcs_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_database_get_rcs_pre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_database_get_scm_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_database_mod_data_obj_meta_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_database_mod_data_obj_meta_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_database_open_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_database_open_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_database_reg_data_obj_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_database_reg_data_obj_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_exec_microservice_pre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_exec_rule_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_exec_rule_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_network_agent_start_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_network_agent_start_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_network_agent_stop_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_network_agent_stop_pre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_network_agent_stop_pre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_network_read_body_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_network_read_body_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_network_write_body_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_network_write_body_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_network_write_header_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_network_write_header_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_obj_list_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_obj_list_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_close_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_close_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_create_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_create_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_modified_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_modified_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_registered_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_registered_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_resolve_hierarchy_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_resolve_hierarchy_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_stat_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_stat_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_write_post</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_pep_resource_write_post</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Create
Write
Read
Replication
Unlink
Rename
Register

iRODS_Policy_Example
A Rule Engine Plugin for a specific Class of events

- Data Object
- Collection
- Metadata
- User
- Resource

The Events are specific to the class of the handler

The handler then invokes policy based on its configuration
A Rule Engine Plugin for data creation and modification events

- Create
- Read
- Replication
- Unlink
- Rename
- ...

Policy invocation is configured as an array of json objects for any given combination of events

Unifies the POSIX and Object behaviors into a single place to configure policy
Example: Synchronous Invocation

```json
1 {  
  "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_access_time",
      "configuration" : {
      }
    },

    { 
      "active_policy_clauses" : ["pre"],
      "events" : ["replication"],
      "policy" : "irods_policy_example_policy",
      "configuration" : {
      }
    }
  ]
}
```

Note that order still matters if more than one policy is configured for a given event.
The What
Basic policies that are leveraged across many deployments and capabilities:

- irods_policy_access_time
- irods_policy_query_processor
- irods_policy_data_movement
- irods_policy_data_replication
- irods_policy_data_verification
- irods_policy_data_retention

The library will continue to grow, with a cookbook of usages
What - Simple policy implementations

Standardized serialized JSON string interface: parameters, and configuration

iRODS Rule Language

```python
irods_policy_example_policy_implementation(*parameters, *configurat
  writeLine("stdout", "Hello UGM2020!")
}
```

Python Rule Language

```python
def irods_policy_example_policy_implementation(rule_args, callback,
  # Parameters    rule_args[1]
  # Configuration rule_args[2]
```

Policy can also be implemented as fast and light C++ rule engine plugins termed Policy Engines
Policy may be invoked using one of three different conventions:

- Direct Invocation: a JSON object
- Query Processor: array of query results in a JSON object
- Event Handler: a JSON object

Each invocation convention defines its interface by contract.
What - Direct Invocation

Parameters passed as serialized JSON strings

```javascript
my_rule() {
    irods_policy_access_time( "{"object_path" : "/tempZone/home/rods/file0.txt"}")
}
```

Directly invoked policy via irule

```javascript
{  
   "policy" : "irods_policy_execute_rule",
   "payload" : {
      "policy_to_invoke" : "irods_policy_storage_tiering",
      "parameters" : {
         "object_path" : "/tempZone/home/rods/file0.txt"
      },
      "configuration" : {
      }
   }
}
```

INPUT null

OUTPUT ruleExecOut
What - Query Processor Invocation

Serializes results to JSON array and passed to the policy via the parameter object as "query_results"

```
1 { 
2   "policy": "irods_policy_enqueue_rule",
3   "delay_conditions": "<PLUSET>1s</PLUSET>",
4   "payload": {
5     "policy": "irods_policy_execute_rule",
6     "payload": {
7       "policy_to_invoke": "irods_policy_query_processor",
8       "parameters": {
9         "query_string": "SELECT USER_NAME, COLL_NAME, DATA_NAME, RESC_NAME WHERE COLL_NAME like '/tempZone/home",
10        "query_limit": 10,
11        "query_type": "general",
12        "number_of_threads": 4,
13        "policy_to_invoke": "irods_policy_engine_example"
14     } 
15   } 
16 } 
17 }
```

For example the invoked policy would receive a row like:
"query_results" : ['rods', '/tempZone/home/rods', 'file0.txt', 'demoResc']


```json
1 {
2   "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
3   "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
4   'plugin_specific_configuration': {
5     "policies_to_invoke": [
6       {
7         "active_policy_clauses": ["post"],
8         "events": ["put"],
9         "policy": "irods_policy_data_replication",
10        "configuration": {
11           "source_to_destination_map": {
12             "demoResc": ["AnotherResc"]
13           }
14        }
15      }
16      ...
17    ]
18   }
19 }
20 }
```
What - Event Handler Invocation

Serializes dataObjInp_t and rsComm_t to the Parameter object

```json
{
  "comm": {
    "auth_scheme": "native", "client_addr": "152.54.8.141", "proxy_auth_info_auth_flag": "5", "proxy_auth_info_auth_scheme": "", "proxy_auth_info_auth_str": "", "proxy_auth_info_flag": "0", "proxy_auth_info_host": "", "proxy_auth_info_ppid": "0", "proxy_rods_zone": "tempZone", "proxy_sys_uid": "0", "proxy_user_name": "rods", "proxy_user_other_info_user_comments": "", "proxy_user_other_info_user_create": "", "proxy_user_other_info_user_info": "", "proxy_user_other_info_user_modify": "", "proxy_user_type": "", "user_auth_info_auth_flag": "5", "user_auth_info_auth_scheme": "", "user_auth_info_auth_str": "", "user_auth_info_flag": "0", "user_auth_info_host": "", "user_auth_info_ppid": "0", "user_rods_zone": "tempZone", "user_sys_uid": "0", "user_user_name": "rods", "user_user_other_info_user_comments": "", "user_user_other_info_user_create": "", "user_user_other_info_user_info": "", "user_user_other_info_user_modify": "", "user_user_type": ""
  },
  "cond_input": {
    "dataIncluded": "", "dataType": "generic", "destRescName": "ufs0", "noOpenFlag": "", "openType": "1", "recursiveOpr": "1", "resc_hier": "ufs0", "selObjType": "dataObj", "translatedPath": ""
  },
  "create_mode": "33204",
  "data_size": "1",
  "event": "CREATE",
  "num_threads": "0",
  "obj_path": "/tempZone/home/rods/test_put_gt_max_sql_rows/junk0083",
  "offset": "0",
  "open_flags": "2",
  "opr_type": "1",
  "policy_enforcement_point": "pep_api_data_obj_put_post"
}
```
Any additional static context passed into the policy

```json
{
  "policy" : "irods_policy_access_time",
  "configuration" : {
    "attribute" : "irods::access_time"
  }
}
```

May be "plugin_specific_configuration" from a rule engine plugin or "configuration" from within the event framework

May hold additional policy which to be subsequently invoked, e.g. the Query Processor
The Why
Each invoked policy may set a conditional around each noun within the system which gates the invocation

- Data Object
- Collection
- Metadata
- User
- Resource

Leverages boost::regex to match any combination of logical_path, metadata, resource name, or user name
Matching a logical path for replication policy invocation

```json
1 {  
  "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-inst",  
  "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",  
  'plugin_specific_configuration': {  
    "policies_to_invoke": [  
      {  
        "conditional": {  
          "logical_path": "\tempZone.*"
        },  
        "active_policy_clauses": ["post"],  
        "events": ["put"],  
        "policy": "irods_policy_data_replication",  
        "configuration": {  
          "source_to_destination_map": {  
            "demoResc": ["AnotherResc"]
          }
        }
      }
    ]
  }
}
```
import shutil

"instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
"plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
policy_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
          }
        }
      }
    ]
  }
}
The How
The cpp_default rule engine plugin in 4.2.8 will now support two new policies:

- irods_policy_enqueue_rule
- irods_policy_execute_rule

The enqueue rule policy will push a job onto the delayed execution queue. The "payload" object holds the rule which is to be executed.
The execute rule policy will invoke a policy engine either from the delayed execute queue or as a direct invocation.

```json
1 {  
2   "policy" : "irods_policy_execute_rule",
3   "payload" : {
4     "policy_to_invoke" : "irods_policy_example",
5     "parameters" : {
6       },
7     "configuration" : {
8       }
9   }
10 }
11 }
12 INPUT null
13 OUTPUT ruleExecOut
```
Sample Delayed Rule for Asynchronous Execution by the cpp default rule engine

```json
1 {  
  "policy" : "irods_policy_enqueue_rule",
  "delay_conditions" : "<EF>REPEAT FOR EVER</EF>",
  "payload" : {  
    "policy" : "irods_policy_execute_rule",
    "payload" : {  
      "policy_to_invoke" : "irods_policy_example",
      "parameters" : {  
      },  
      "configuration" : {  
      }  
    }  
  }  
}
```

We no longer need to pay the penalty of instantiating an interpreted language
Policy Composed Capabilities
Storage Tiering Overview

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

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.

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.

Data Virtualization (Unified Namespace)
Data Discovery (Metadata)
Workflow Automation (Rule Engine)
Secure Collaboration (Federation)
• Asynchronous Discovery
• Asynchronous Replication
• Synchronous Retention
• Resource associated metadata
• Identified by 'tiering groups'
Asynchronous Replication

```json
1 {  
2   "policy": "irods_policy_execute_rule",  
3   "payload": {  
4     "policy_to_invoke": "irods_policy_query_processor",  
5     "configuration": {  
6       "query_string": "SELECT META_RESC_ATTR_VALUE WHERE META_RESC_ATTR_NAME = 'irods::storage_tiering::group'",  
7       "query_limit": 0,  
8       "query_type": "general",  
9       "number_of_threads": 8,  
10      "policy_to_invoke": "irods_policy_event_generator_resource_metadata",  
11      "configuration": {  
12        "metadata": {  
13          "attribute": "irods::storage_tiering::group",  
14          "value": "{0}"  
15        }  
16      },  
17      "policies_to_invoke": [],  
18      "policy": "irods_policy_query_processor",  
19      "configuration": {  
20        "query_string": "SELECT META_RESC_ATTR_VALUE WHERE META_RESC_ATTR_NAME = 'irods::storage_tiering::query' AND RESC_NAME = 'IRODS_TOKEN_SO
21        "default_results_when_no_rows_found": [{"SELECT USER_NAME, COLL_NAME, DATA_NAME, RESC_NAME WHERE META_DATA_ATTR_NAME = 'irods::access_ti
22        "query_limit": 0,  
23        "query_type": "general",  
24        "number_of_threads": 8,  
25        "policy_to_invoke": "irods_policy_query_processor",  
26        "configuration": {  
27          "lifetime": "IRODS_TOKEN_QUERY_SUBSTITUTION_END_TOKEN(SELECT META_RESC_ATTR_VALUE WHERE META_RESC_ATTR_NAME = 'irods::storage_tiering::query
28          "query_string": "{0}",  
29          "query_limit": 0,  
30          "query_type": "general",  
31          "number_of_threads": 8,  
32          "policy_to_invoke": "irods_policy_data_replication",  
33          "configuration": {  
34            "comment": "source_resource, and destination_resource supplied by the resource metadata event generator"  
35          }  
36        }  
37      }  
38     }  
39   }  
40 }  
41 }  
42 }  
43 }  
44 }  
45 }  
46 INPUT null
```
Policy Composed Storage Tiering

Synchronous Configuration for Storage Tiering

```json
{
    "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"],
                "policy": "irods_policy_access_time",
                "configuration": {
                    "log_errors": "true"
                }
            },
            {
                "active_policy_clauses": ["post"],
                "events": ["read", "write", "get"],
                "policy": "irods_policy_data_restage",
                "configuration": {
                    "mode": "trim_single_replica",
                    "log_errors": "true"
                }
            },
            {
                "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"
                }
            }
        ]
    }
}
```
Policy Composed Storage Tiering

Metadata Driven Restage for Storage Tiering

```json
{
    "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": [
            {
                "conditional": {
                    "attribute": "irods::storageTiering::restage",
                },
                "active_policy_clauses": ["post"],
                "events": ["set", "add"],
                "policy": "irods_policy_data_restage",
                "configuration": {}
            }
        ]
    }
}
```
Data Transfer Nodes Pattern

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.
• Asynchronous Discovery
• Asynchronous Retention
• Synchronous Replication
• Resource associated metadata
• Identified by 'replication groups'
Asynchronous Retention on Edge Resources

```json
1 {  
2   "policy" : "irods_policy_enqueue_rule",
3   "delay_conditions" : "<EF>REPEAT FOR EVER</EF>",
4   "payload" : {
5     "policy" : "irods_policy_execute_rule",
6     "payload" : {
7       "policy_to_invoke" : "irods_policy_query_processor",
8       "parameters" : {
9         "query_string" : "SELECT USER_NAME, COLL_NAME, DATA_NAME, RESC_NAME WHERE COLL_NAME",
10        "query_limit" : 10,
11        "query_type" : "general",
12        "number_of_threads" : 4,
13        "policy_to_invoke" : "irods_policy_data_retention",
14        "configuration" : {
15          "mode" : "trim_single_replica",
16          "source_resource_list" : ["edge_resource_1", "edge_resource_2"]
17        }
18     }
19   }
20 }  
21 }
```
Synchronous Replication

```json
{
    "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.**"
                },
                "active_policy_clauses": ["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"]
                }
            }
        }
    }
}
```
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>`

Metadata takes the form:

- `<index name>` is the name of the index created
- `<index type>` is either "full_text" or "metadata"
- `<technology>` is the targeted indexing service

Once indexing metadata is applied indicating that a collection should be indexed, a job is submitted to the iRODS delayed execution queue which will perform the requested action asynchronously.
Implemented as individual Policy Engines

- 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
Synchronously configured full text indexing

```json

1 "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
2 "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
3 'plugin_specific_configuration': {
4   "policies_to_invoke": [
5     {
6       "active_policy_clauses": ["post"],
7       "events": ["put", "write"],
8       "policy": "irods_policy_event_delegate_collection_metadata",
9       "configuration": {
10      "policies_to_invoke": [
11     {
12       "conditional": {
13       "metadata": {
14       "attribute": "irods::indexing::index",
15       "entity_type": "data_object"
16     },
17   },
18     "policy": "irods_policy_indexing_full_text_index_elasticsearch",
19     "configuration": {
20     "hosts": ["http://localhost:9200/"]
21     "bulk_count": 100,
22     "read_size": 1024
23   }
24 
25   }
26 
27 }]
28 ...
```
Synchronously configured full text purge

```json
{
    "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
                }
            }
        ]
    }
}
```
Capabilities become recipes which are easily configured

A Policy GUI is now a possibility with the manipulation of server side JSON

Continue to build a library of supported policy engines, driven by the community

Data Integrity Capability will now be a collection of policy engines
Questions?