iRODS®

iRODS Policy Composition

Jason M. Coposky
@jason_coposky
Executive Director, iRODS Consortium

June 8-11, 2021
iRODS User Group Meeting 2021
Virtual Event

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

What is a Policy

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?

iRODS Policies

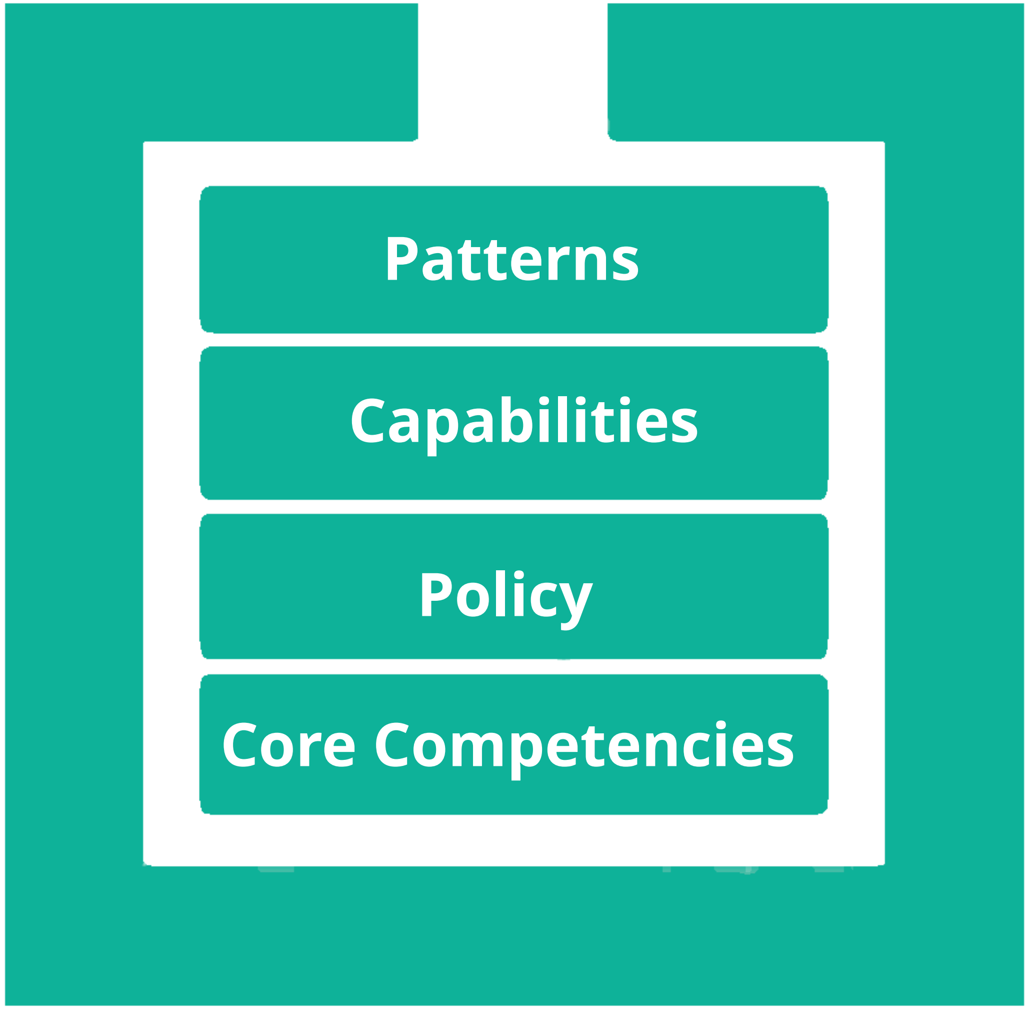
The reflection of real world data management decisions in computer actionable code.

(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



Patterns

Capabilities

Policy

Core Competencies

Some Questions

- 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:

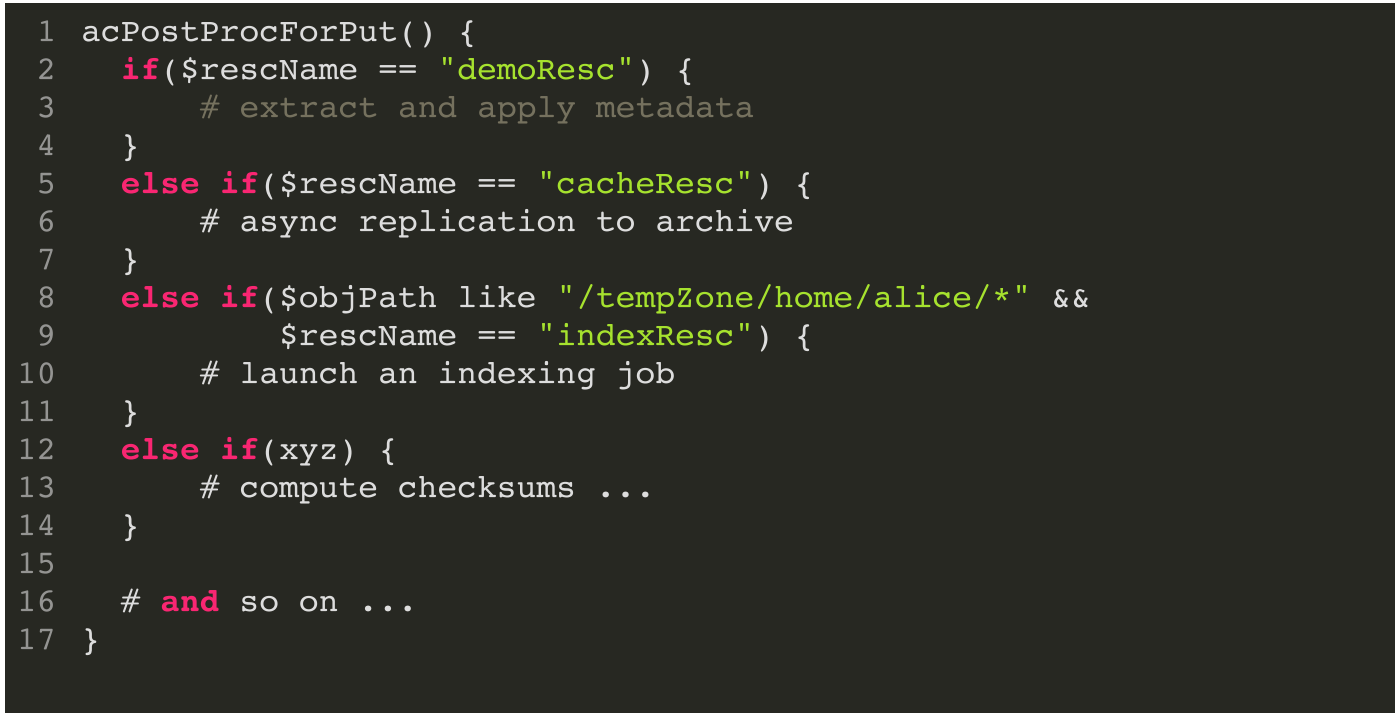
Favor composition over monolithic implementations

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

The Original Approach

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



```
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 ...
}
```

Assuming there was even a provided policy enforcement point for the desired event...

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

```
pep_api_data_obj_put_post(*INSTANCE_NAME, *COMM, *DATAOBJINP, *BUFFER, *PORTAL_OPR_OUT) {
  # metadata extraction and application code

  RULE_ENGINE_CONTINUE
}
```

/etc/irods/checksum.re

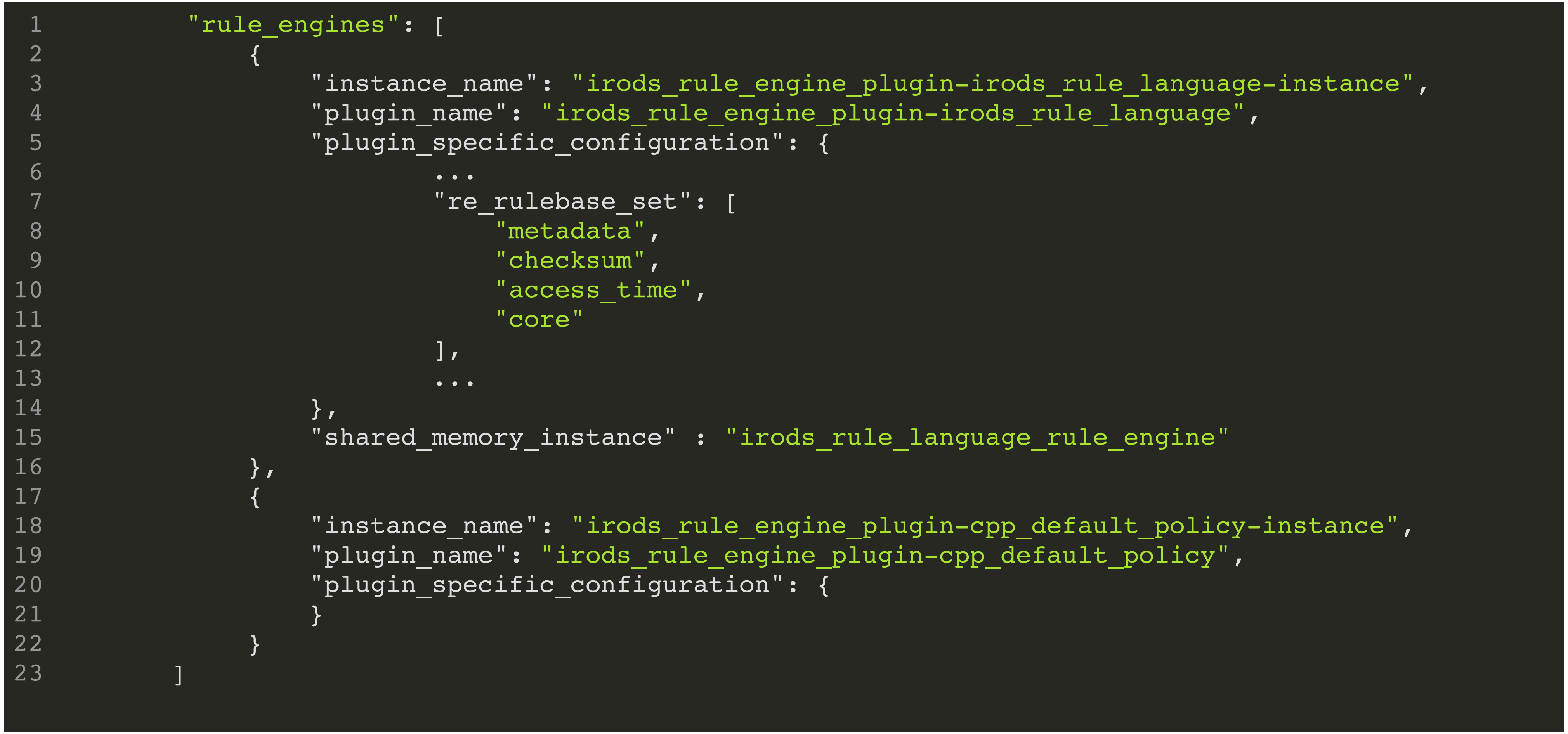
```
pep_api_data_obj_put_post(*INSTANCE_NAME, *COMM, *DATAOBJINP, *BUFFER, *PORTAL_OPR_OUT) {
  # checksum code

  RULE_ENGINE_CONTINUE
}
```

/etc/irods/access_time.re

```
pep_api_data_obj_put_post(*INSTANCE_NAME, *COMM, *DATAOBJINP, *BUFFER, *PORTAL_OPR_OUT) {
  # access time application code

  RULE_ENGINE_CONTINUE
}
```

Expanding policy across rule bases

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": {
        }
    }
]
```

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

The First Implementation

Policies composed by a monolithic framework plugins

Policy delegated by naming 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 overridden by another rule engine, or rule base to customize to 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

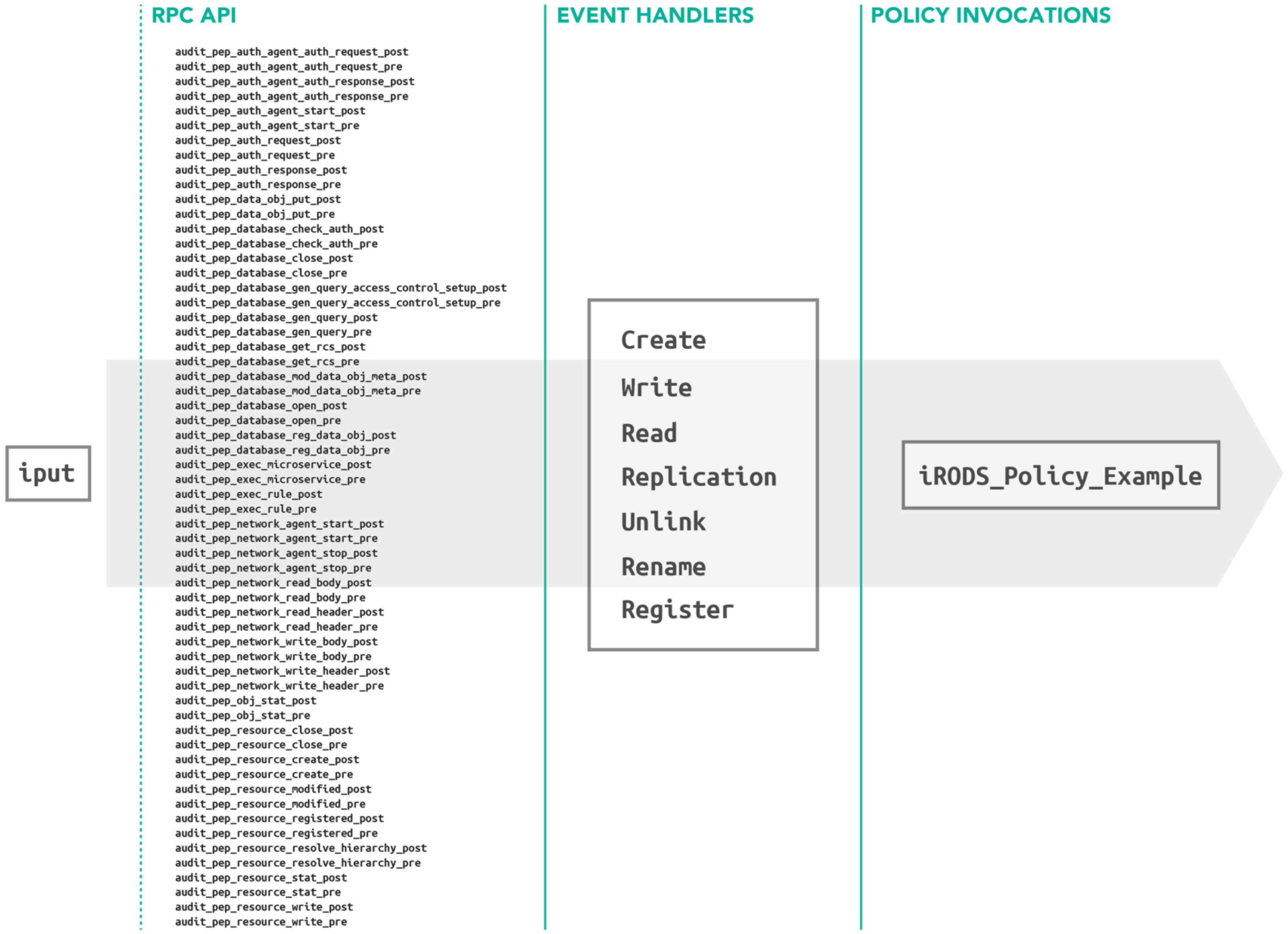
Write simple policy implementations

- 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

When - Event Handlers



iput

RPC API

```
audit_pep_auth_agent_auth_request_post
audit_pep_auth_agent_auth_request_pre
audit_pep_auth_agent_auth_response_post
audit_pep_auth_agent_auth_response_pre
audit_pep_auth_agent_start_post
audit_pep_auth_agent_start_pre
audit_pep_auth_request_post
audit_pep_auth_request_pre
audit_pep_auth_response_post
audit_pep_auth_response_pre
audit_pep_data_obj_put_post
audit_pep_data_obj_put_pre
audit_pep_database_check_auth_post
audit_pep_database_check_auth_pre
audit_pep_database_close_post
audit_pep_database_close_pre
audit_pep_database_gen_query_access_control_setup_post
audit_pep_database_gen_query_access_control_setup_pre
audit_pep_database_gen_query_post
audit_pep_database_gen_query_pre
audit_pep_database_get_rcs_post
audit_pep_database_get_rcs_pre
audit_pep_database_mod_data_obj_meta_post
audit_pep_database_mod_data_obj_meta_pre
audit_pep_database_open_post
audit_pep_database_open_pre
audit_pep_database_reg_data_obj_post
audit_pep_database_reg_data_obj_pre
audit_pep_exec_microservice_post
audit_pep_exec_microservice_pre
audit_pep_exec_rule_post
audit_pep_exec_rule_pre
audit_pep_network_agent_start_post
audit_pep_network_agent_start_pre
audit_pep_network_agent_stop_post
audit_pep_network_agent_stop_pre
audit_pep_network_read_body_post
audit_pep_network_read_body_pre
audit_pep_network_read_header_post
audit_pep_network_read_header_pre
audit_pep_network_write_body_post
audit_pep_network_write_body_pre
audit_pep_network_write_header_post
audit_pep_network_write_header_pre
audit_pep_obj_stat_post
audit_pep_obj_stat_pre
audit_pep_resource_close_post
audit_pep_resource_close_pre
audit_pep_resource_create_post
audit_pep_resource_create_pre
audit_pep_resource_modified_post
audit_pep_resource_modified_pre
audit_pep_resource_registered_post
audit_pep_resource_registered_pre
audit_pep_resource_resolve_hierarchy_post
audit_pep_resource_resolve_hierarchy_pre
audit_pep_resource_stat_post
audit_pep_resource_stat_pre
audit_pep_resource_write_post
audit_pep_resource_write_pre
```

EVENT HANDLERS

```
Create
Write
Read
Replication
Unlink
Rename
Register
```

POLICY INVOCATIONS

```
iRODS_Policy_Example
```

When - The Event Handler

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

When - event_handler-data_object_modified

A Rule Engine Plugin for data creation and modification events

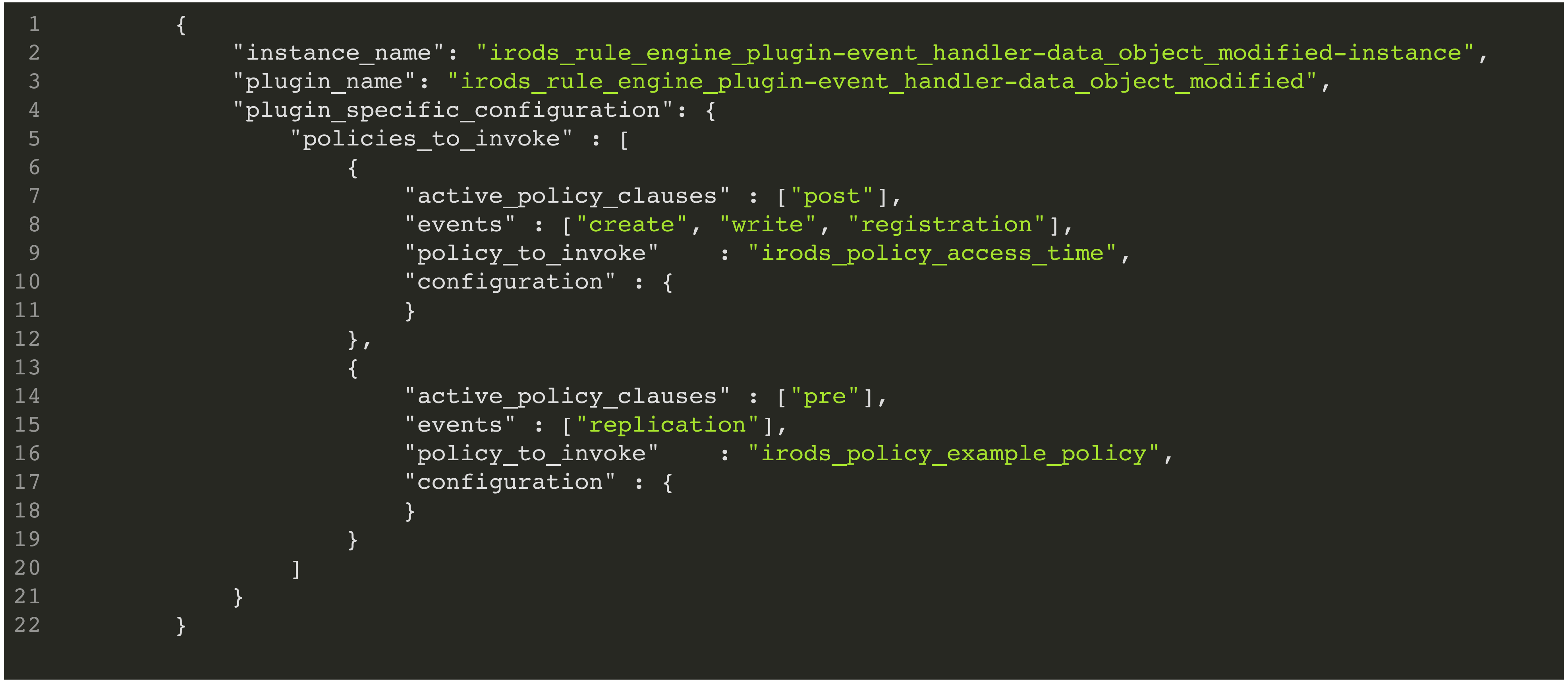
- Create
- Read
- Replication
- Unlink
- Rename
- Register

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

When - event_handler-data_object_modified

Example : Synchronous Invocation



```
{
    "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_to_invoke"    : "irods_policy_access_time",
                "configuration" : {
                }
            },
            {
                "active_policy_clauses" : ["pre"],
                "events" : ["replication"],
                "policy_to_invoke"    : "irods_policy_example_policy",
                "configuration" : {
                }
            }
        ]
    }
}
```

Note that order still matters if more than one policy needs to be invoked for a given event

The What

What - Simple policy implementations

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 JSON interface : parameters, and configuration

iRODS Rule Language

```
irods_policy_example_policy_implementation(*parameters, *configuration) {

}
```

Python Rule Language

```
def irods_policy_example_policy_implementation(rule_args, callback, rei):
# Parameters    rule_args[1]
# Configuration rule_args[2]
```

Policy can also be implemented as fast and light C++ rule engine plugins

What - Simple policy implementations

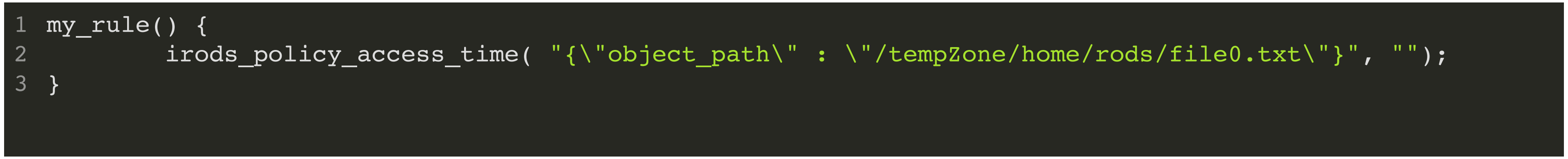
Policy may be invoked using one of three different conventions:

- Direct Invocation : a JSON object
- Query Processor : a JSON array of parameters
- Event Handler : a JSON object

Each invocation convention defines its interface by contract.

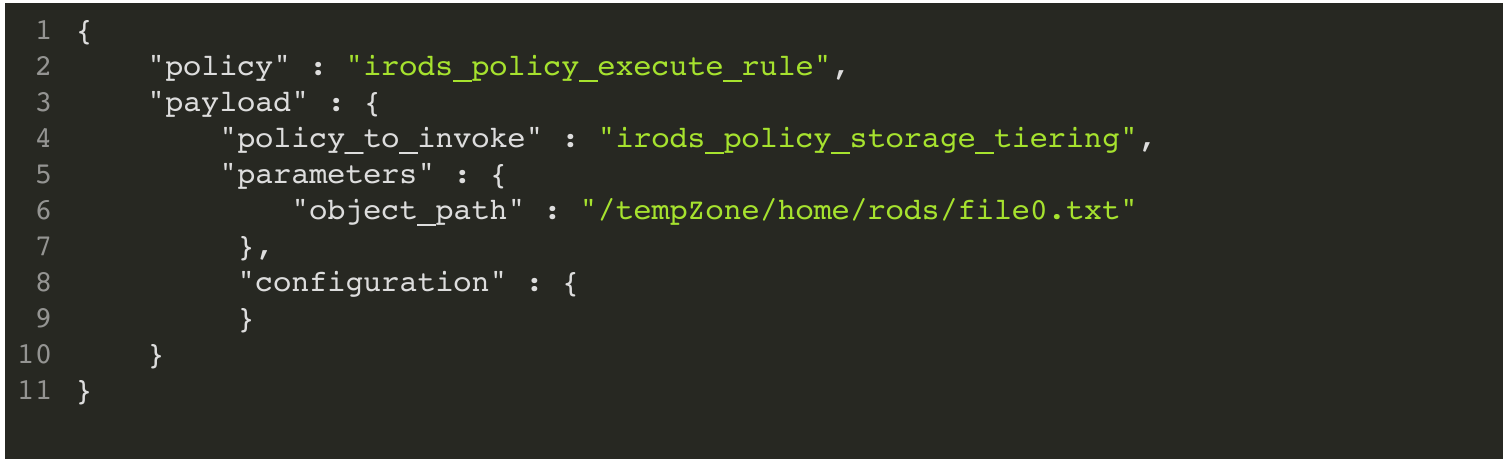
What - Simple policy implementations

Direct Invocation : Parameters passed as a JSON object



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

Parameters may also be configured statically

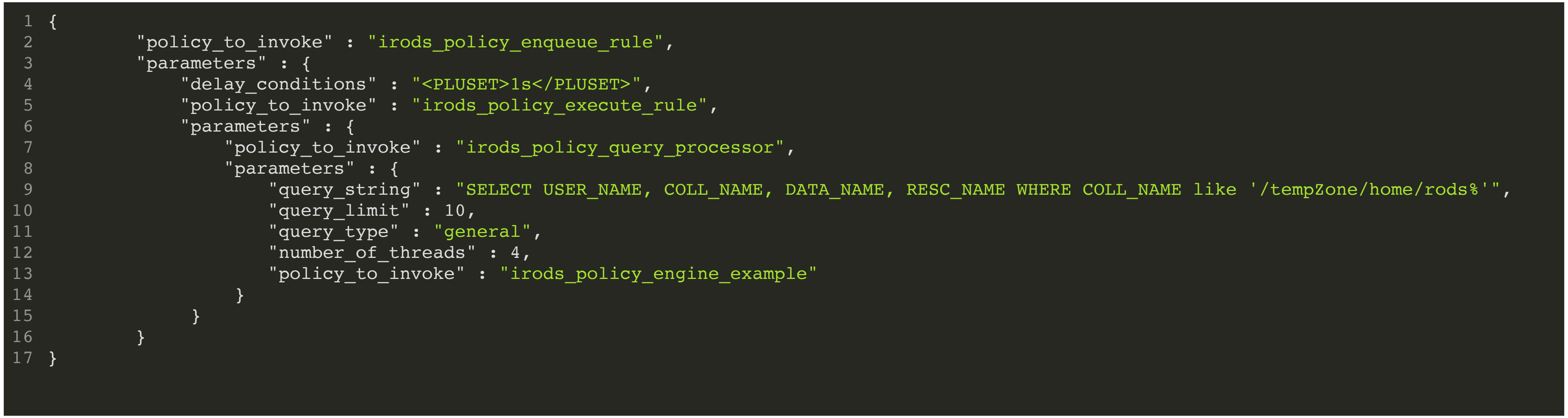


```
{
    "policy" : "irods_policy_execute_rule",
    "payload" : {
        "policy_to_invoke" : "irods_policy_storage_tiering",
        "parameters" : {
            "object_path" : "/tempZone/home/rods/file0.txt"
         },
         "configuration" : {
         }
    }
}
```

What - Simple policy implementations

Query Processor Invocation

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



```
{
        "policy_to_invoke" : "irods_policy_enqueue_rule",
        "parameters" : {
            "delay_conditions" : "<PLUSET>1s</PLUSET>",
            "policy_to_invoke" : "irods_policy_execute_rule",
            "parameters" : {
                "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%'",
                    "query_limit" : 10,
                    "query_type" : "general",
                    "number_of_threads" : 4,
                    "policy_to_invoke" : "irods_policy_engine_example"
                 }
             }
        }
}
```

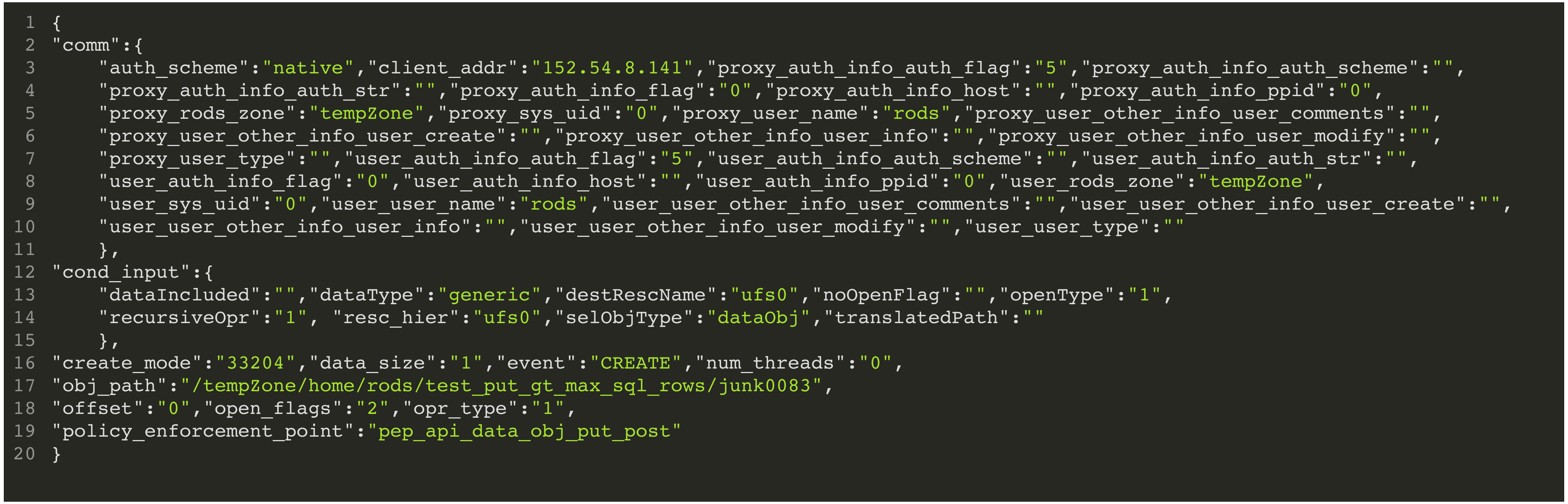
For example the invoked policy would receive a row:

['rods', '/tempZone/home/rods/', 'file0.txt', 'demoResc']

What - Simple policy implementations

Event Handler Invocation

Serializes dataObjInp_t and rsComm_t to a JSON object



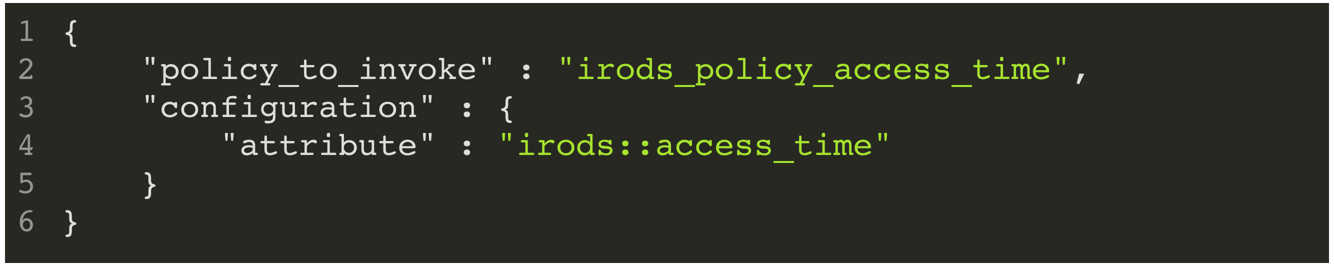
```
{
"comm":{
    "auth_scheme":"native","client_addr":"152.54.8.141","proxy_auth_info_auth_flag":"5","proxy_auth_info_auth_scheme":"",
    "proxy_auth_info_auth_str":"","proxy_auth_info_flag":"0","proxy_auth_info_host":"","proxy_auth_info_ppid":"0",
    "proxy_rods_zone":"tempZone","proxy_sys_uid":"0","proxy_user_name":"rods","proxy_user_other_info_user_comments":"",
    "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"
}
```

Which is also passed in as the parameter object

What - Simple policy implementations

Configuration

Any additional statically set context passed into the policy



```
{
    "policy_to_invoke" : "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

The Why

Why - Policy Conditionals

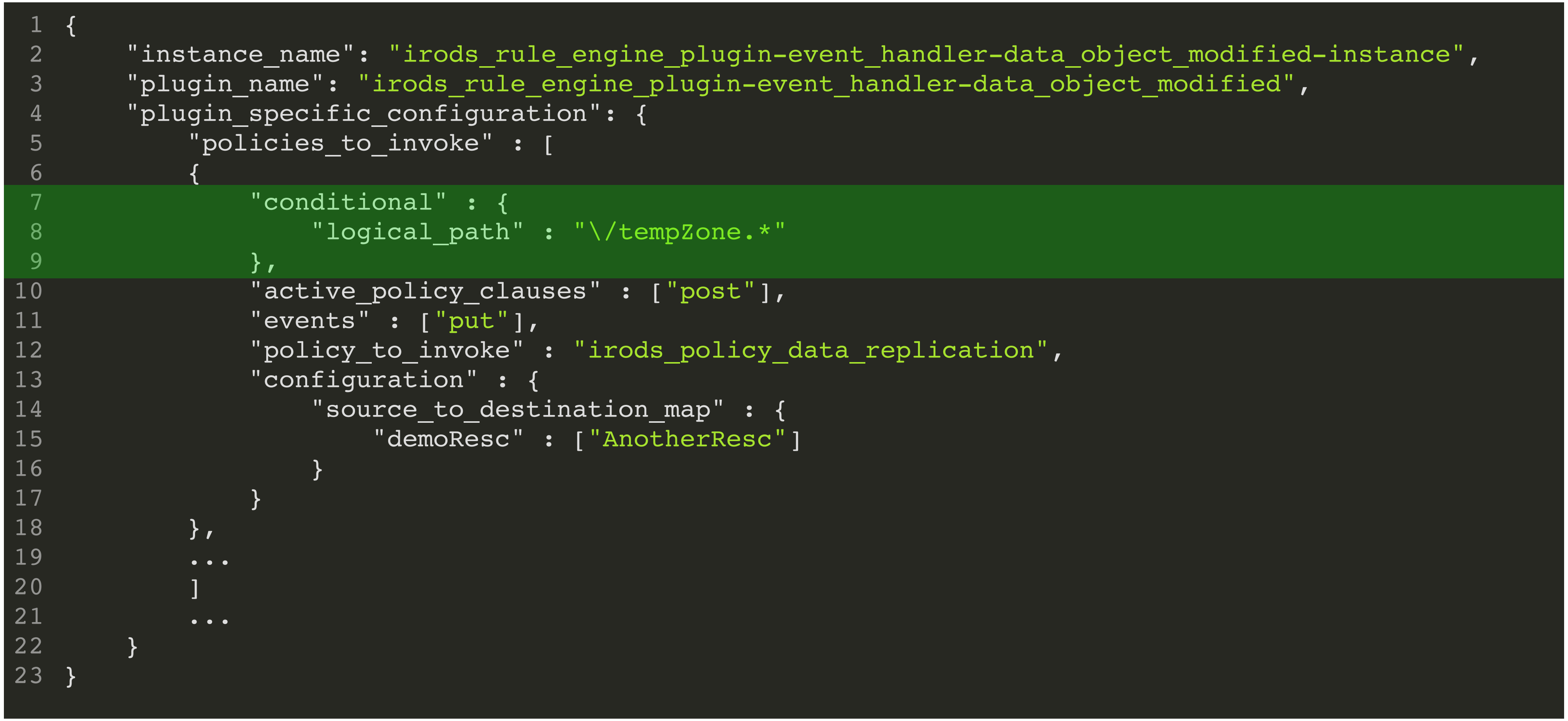
Each invoked policy may set a conditional around each noun within the system which gates the invocation

- Data Object
- Collection
- Metadata
- User
- Resource

Leverages boost::regex to match any combination of logical_path, metadata, resource name, etc.

Why - Policy Conditionals

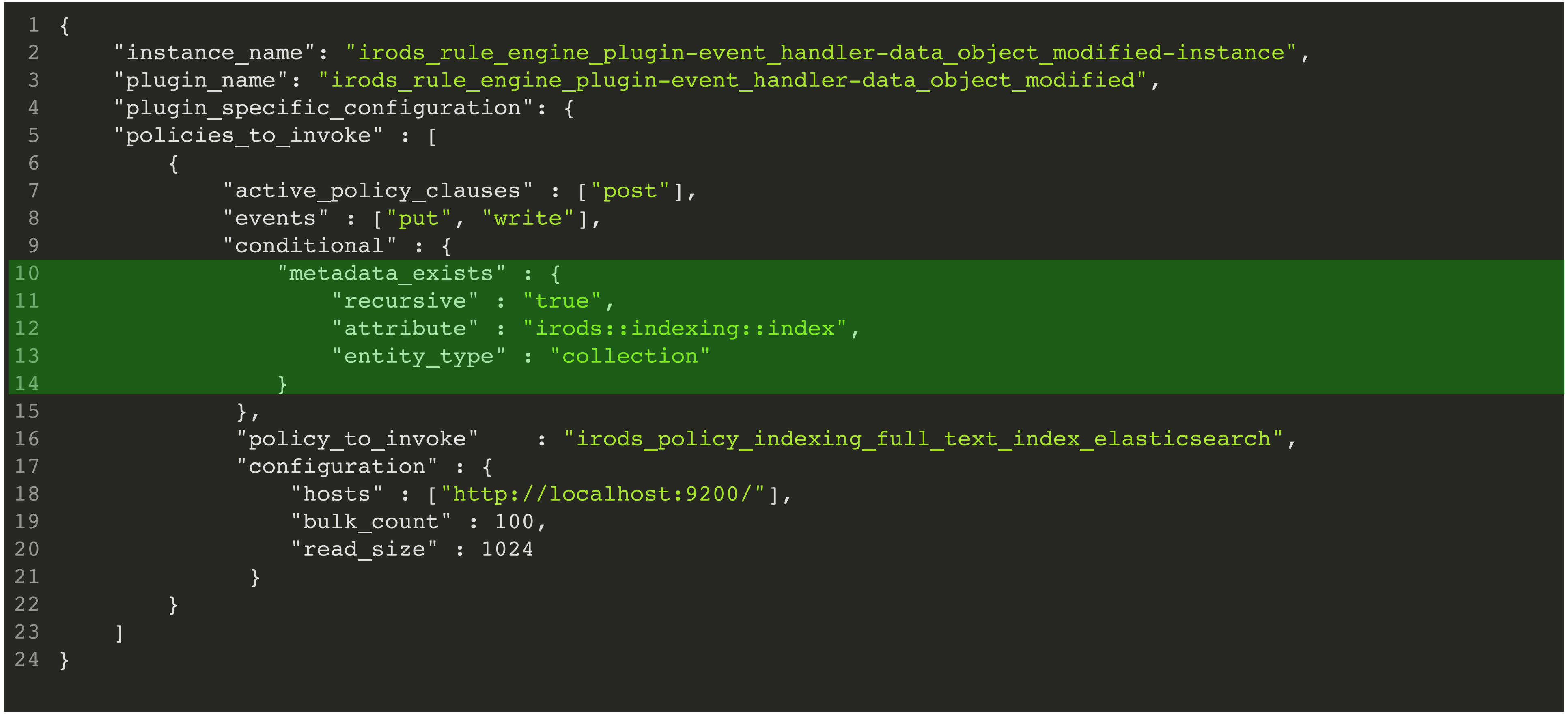
Matching a logical path for replication policy invocation



```
{
    "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" : ["put"],
            "policy_to_invoke" : "irods_policy_data_replication",
            "configuration" : {
                "source_to_destination_map" : {
                    "demoResc" : ["AnotherResc"]
                }
            }
        },
        ...
        ]
        ...
    }
}
```

Why - Policy Conditionals

Matching metadata for indexing policy invocation



```
{
    "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"],
            "conditional" : {
                "metadata_exists" : {
                    "recursive" : "true",
                    "attribute" : "irods::indexing::index",
                    "entity_type" : "collection"
                }
             },
             "policy_to_invoke"    : "irods_policy_indexing_full_text_index_elasticsearch",
             "configuration" : {
                 "hosts" : ["http://localhost:9200/"],
                 "bulk_count" : 100,
                 "read_size" : 1024
              }
        }
    ]
}
```

The How

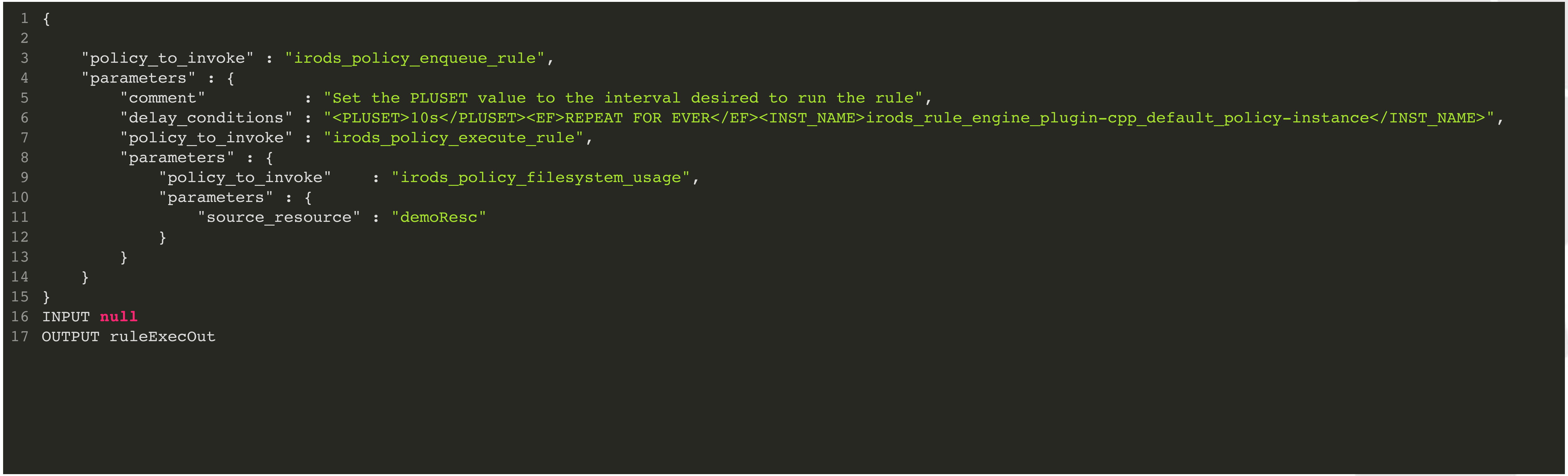
How - Asynchronous Execution

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.

How - Asynchronous Execution



```
{

    "policy_to_invoke" : "irods_policy_enqueue_rule",
    "parameters" : {
        "comment"          : "Set the PLUSET value to the interval desired to run the rule",
        "delay_conditions" : "<PLUSET>10s</PLUSET><EF>REPEAT FOR EVER</EF><INST_NAME>irods_rule_engine_plugin-cpp_default_policy-instance</INST_NAME>",
        "policy_to_invoke" : "irods_policy_execute_rule",
        "parameters" : {
            "policy_to_invoke"    : "irods_policy_filesystem_usage",
            "parameters" : {
                "source_resource" : "demoResc"
            }
        }
    }
}
INPUT null
OUTPUT ruleExecOut
```

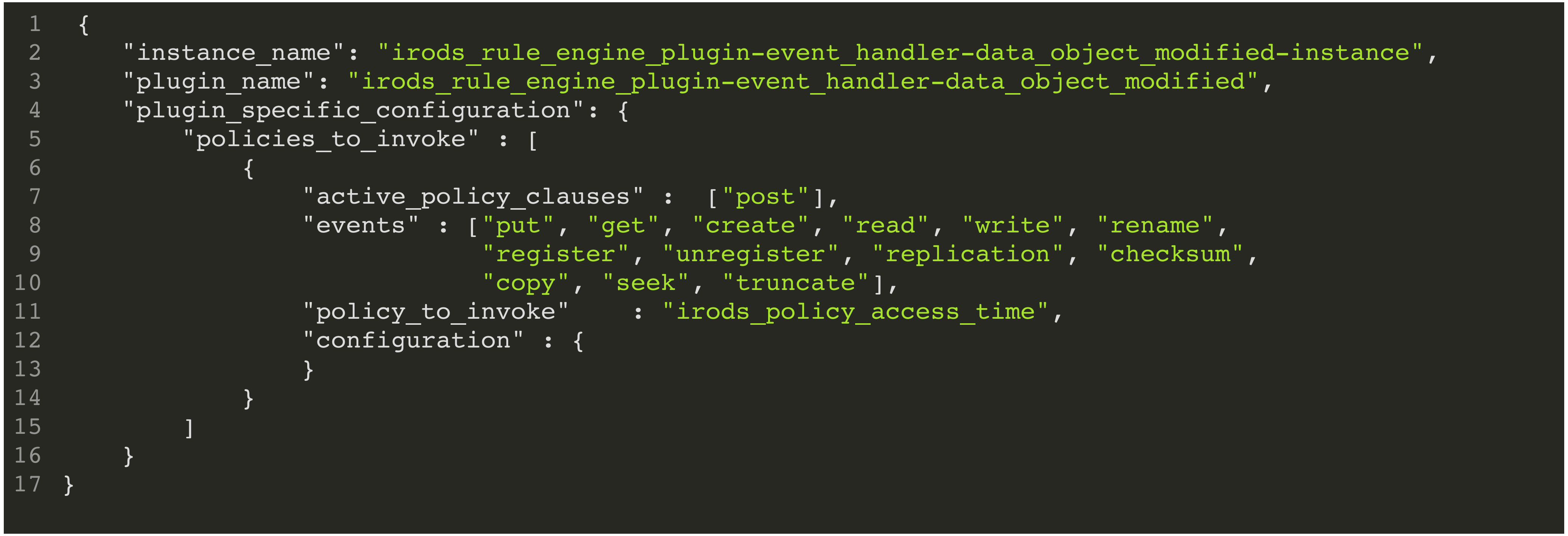
The execute rule policy invokes a policy engine either from the delayed execution queue or as a direct invocation

The New Approach

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

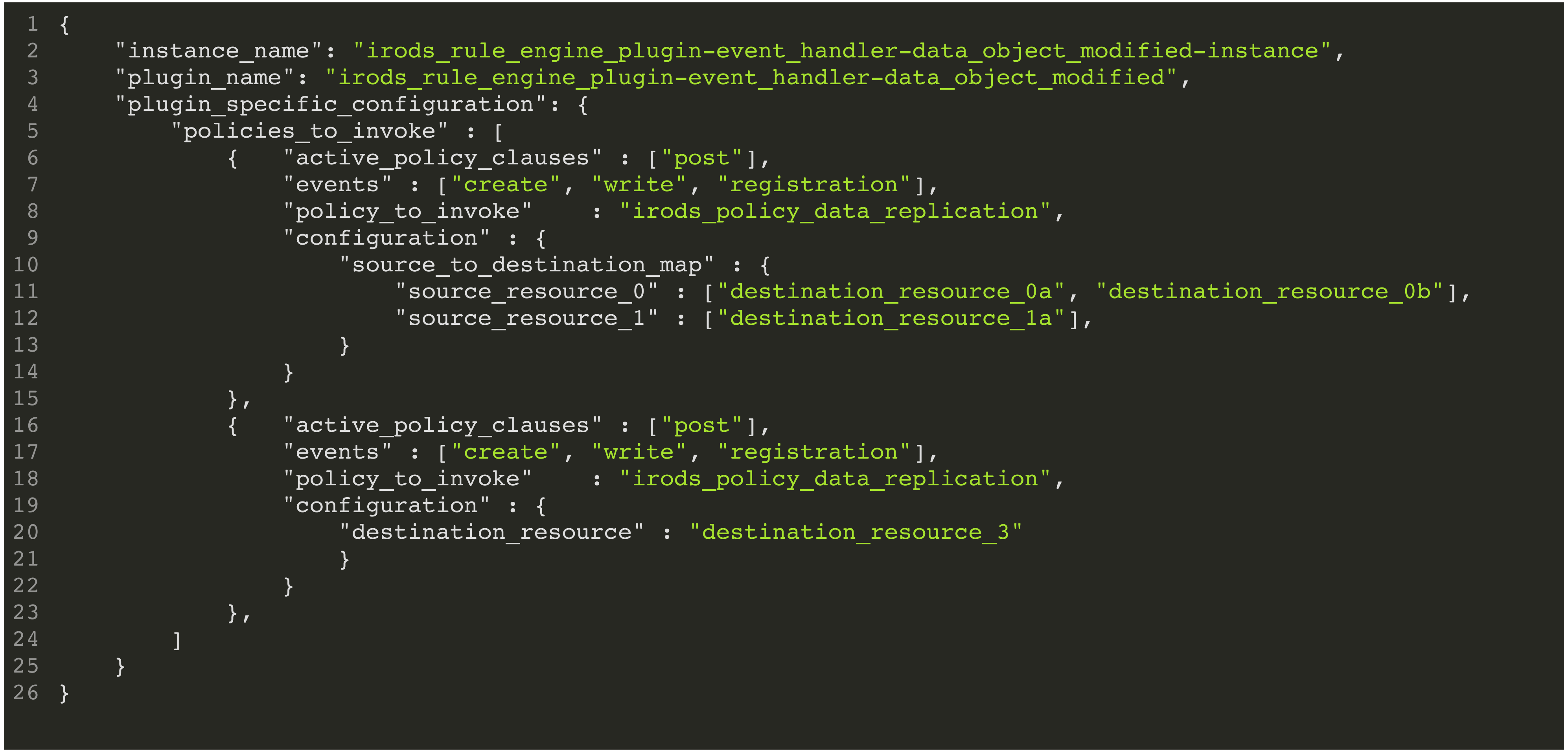
Examples

Synchronous Access Time



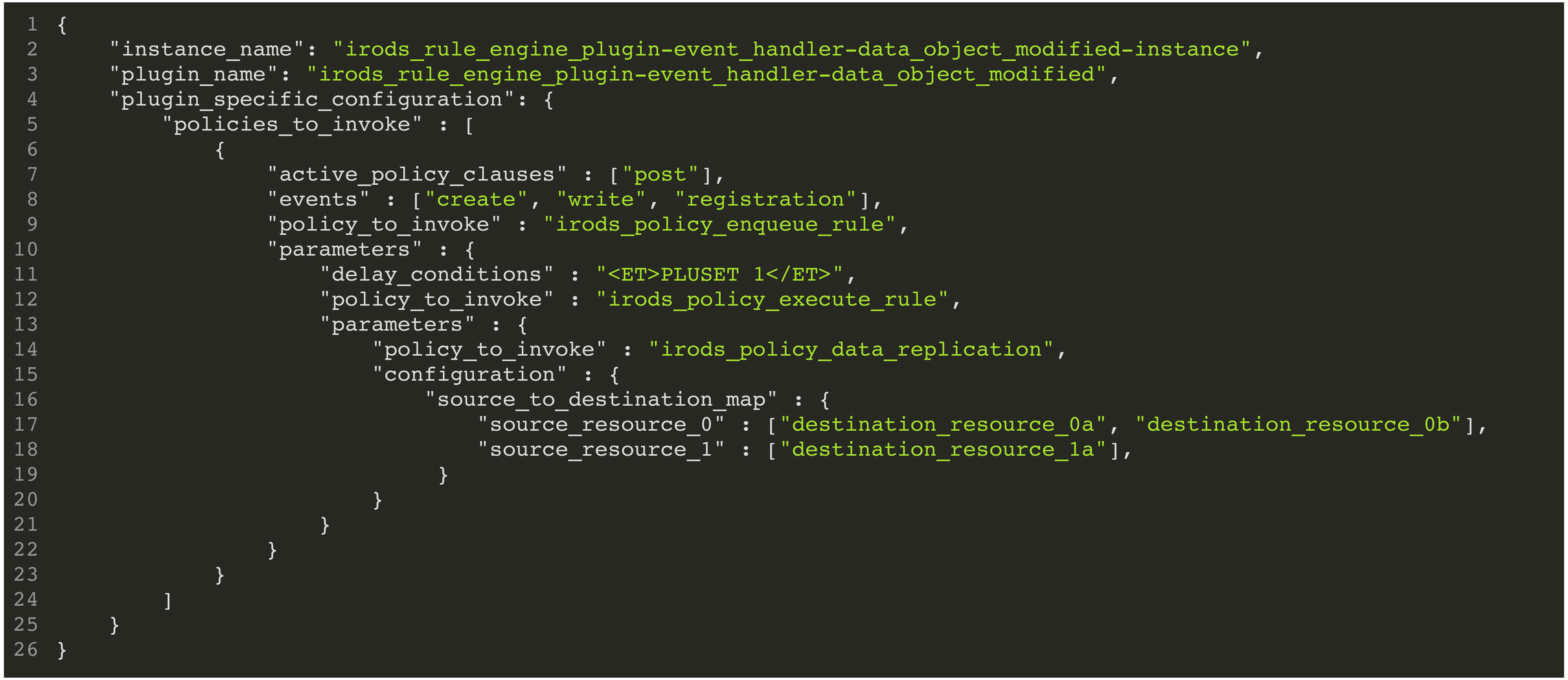
```
 {
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {
                "active_policy_clauses"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

Synchronous Replication



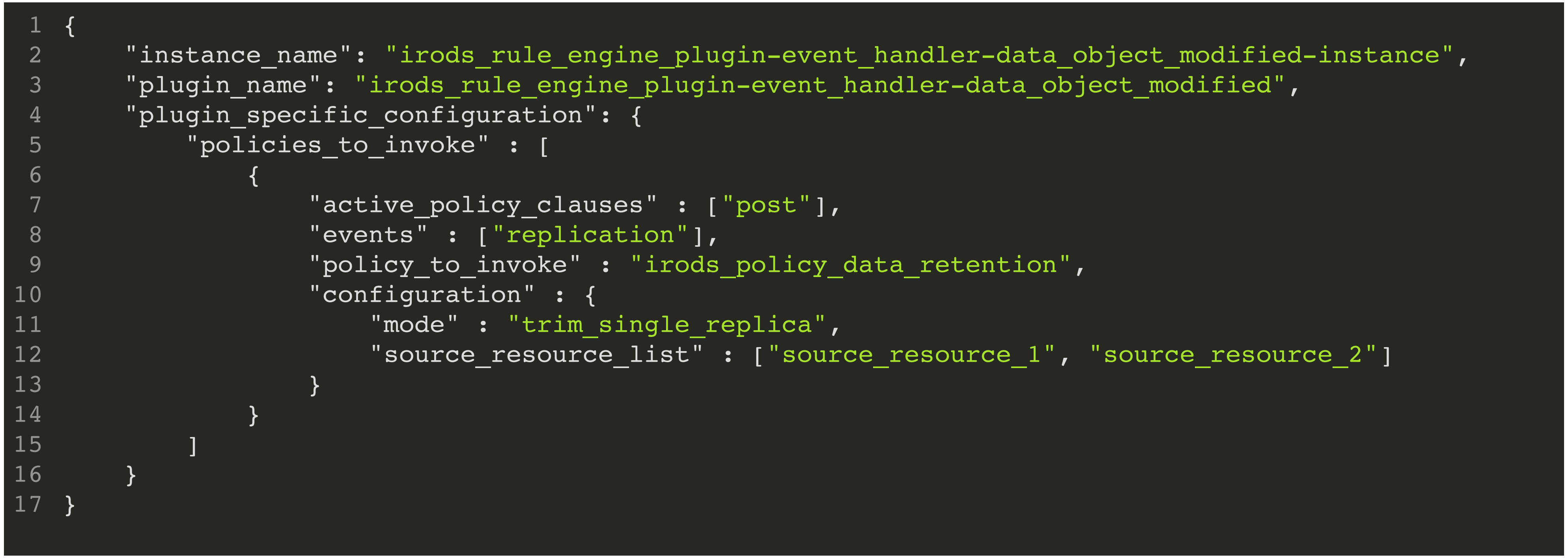
```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {   "active_policy_clauses" : ["post"],
                "events" : ["create", "write", "registration"],
                "policy_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"],
                    }
                }
            },
            {   "active_policy_clauses" : ["post"],
                "events" : ["create", "write", "registration"],
                "policy_to_invoke"    : "irods_policy_data_replication",
                "configuration" : {
                    "destination_resource" : "destination_resource_3"
                    }
                }
            },
        ]
    }
}
```

Asynchronous Replication



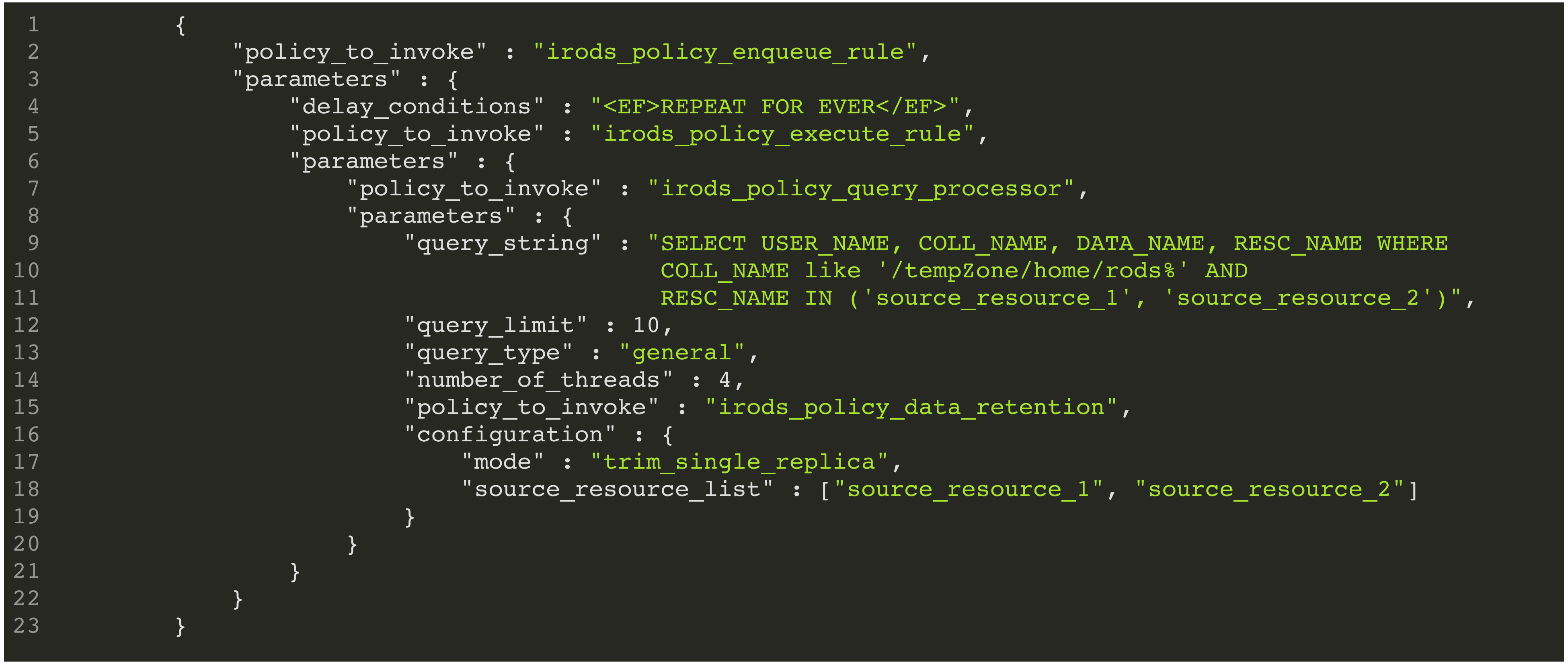
```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {
                "active_policy_clauses" : ["post"],
                "events" : ["create", "write", "registration"],
                "policy_to_invoke" : "irods_policy_enqueue_rule",
                "parameters" : {
                    "delay_conditions" : "<ET>PLUSET 1</ET>",
                    "policy_to_invoke" : "irods_policy_execute_rule",
                    "parameters" : {
                        "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



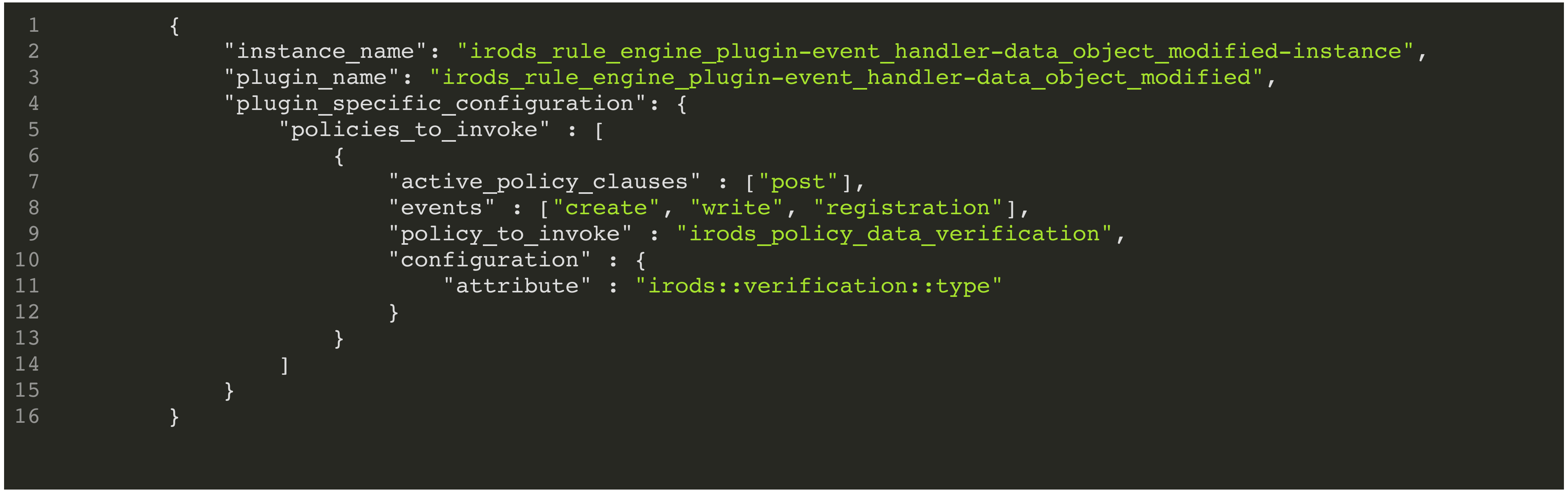
```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {
                "active_policy_clauses"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

Query Driven Asynchronous Retention



```
{
    "policy_to_invoke" : "irods_policy_enqueue_rule",
    "parameters" : {
        "delay_conditions" : "<EF>REPEAT FOR EVER</EF>",
        "policy_to_invoke" : "irods_policy_execute_rule",
        "parameters" : {
            "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

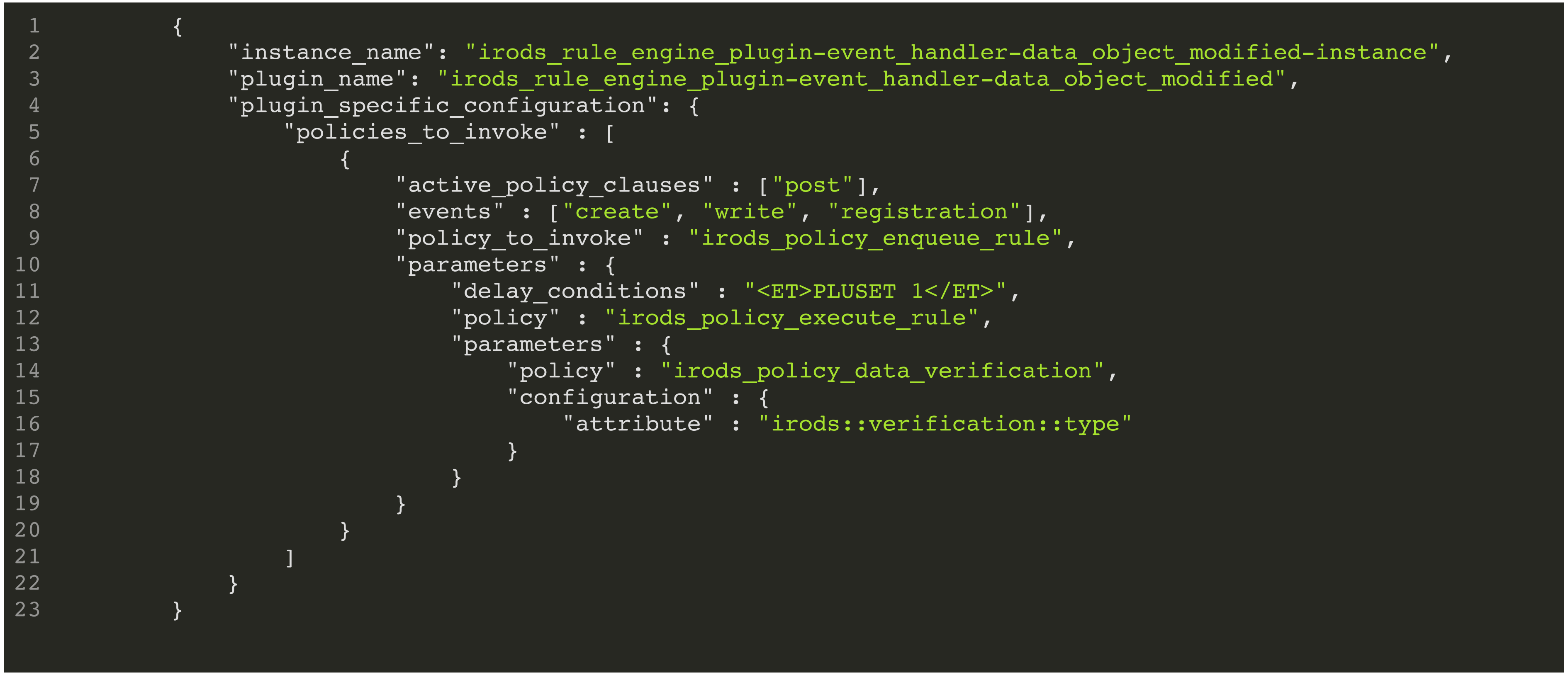


```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {
                "active_policy_clauses" : ["post"],
                "events" : ["create", "write", "registration"],
                "policy_to_invoke" : "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



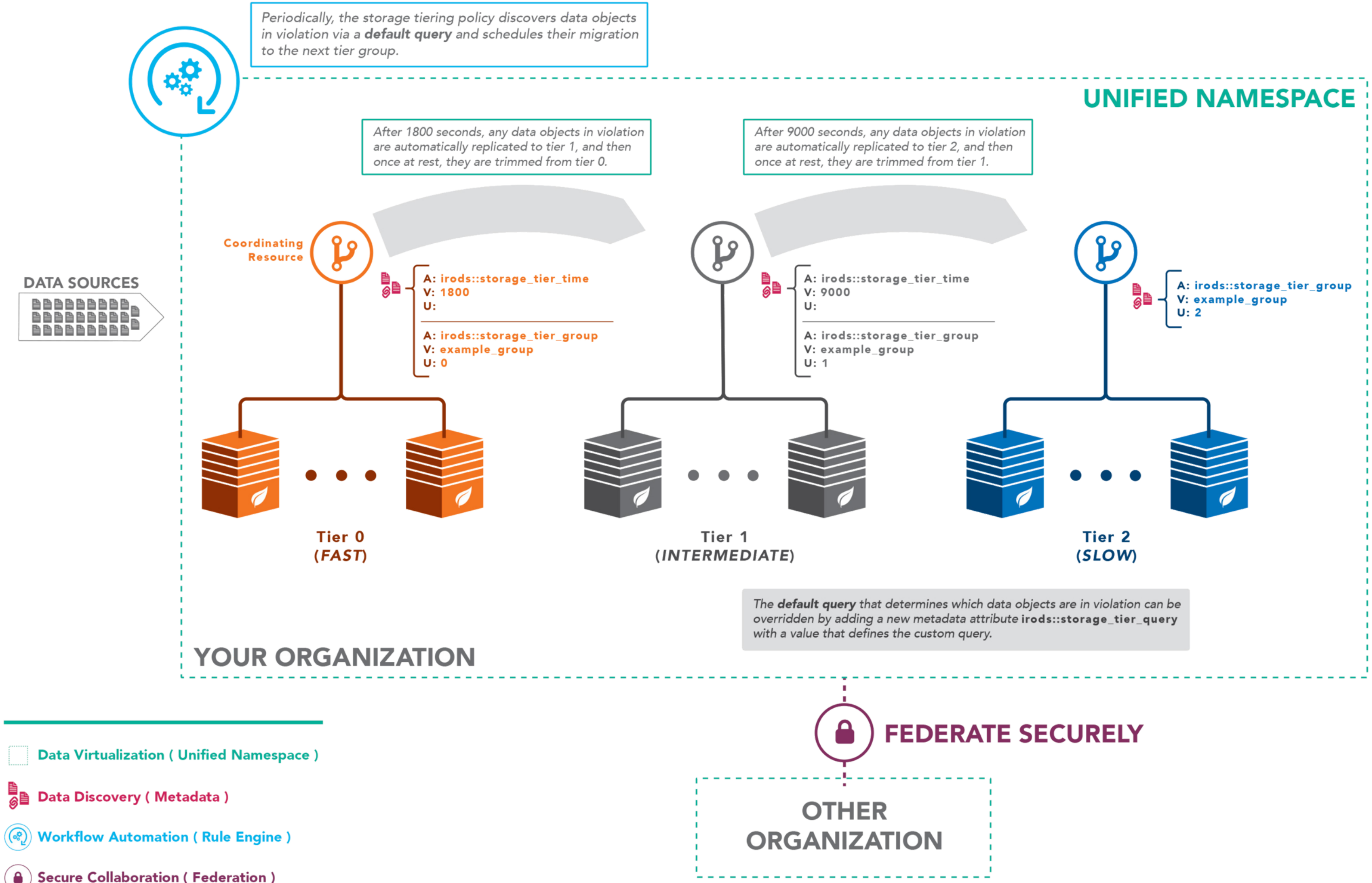
```
{
    "instance_name": "irods_rule_engine_plugin-event_handler-data_object_modified-instance",
    "plugin_name": "irods_rule_engine_plugin-event_handler-data_object_modified",
    "plugin_specific_configuration": {
        "policies_to_invoke" : [
            {
                "active_policy_clauses" : ["post"],
                "events" : ["create", "write", "registration"],
                "policy_to_invoke" : "irods_policy_enqueue_rule",
                "parameters" : {
                    "delay_conditions" : "<ET>PLUSET 1</ET>",
                    "policy" : "irods_policy_execute_rule",
                    "parameters" : {
                        "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



*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.

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

*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.*

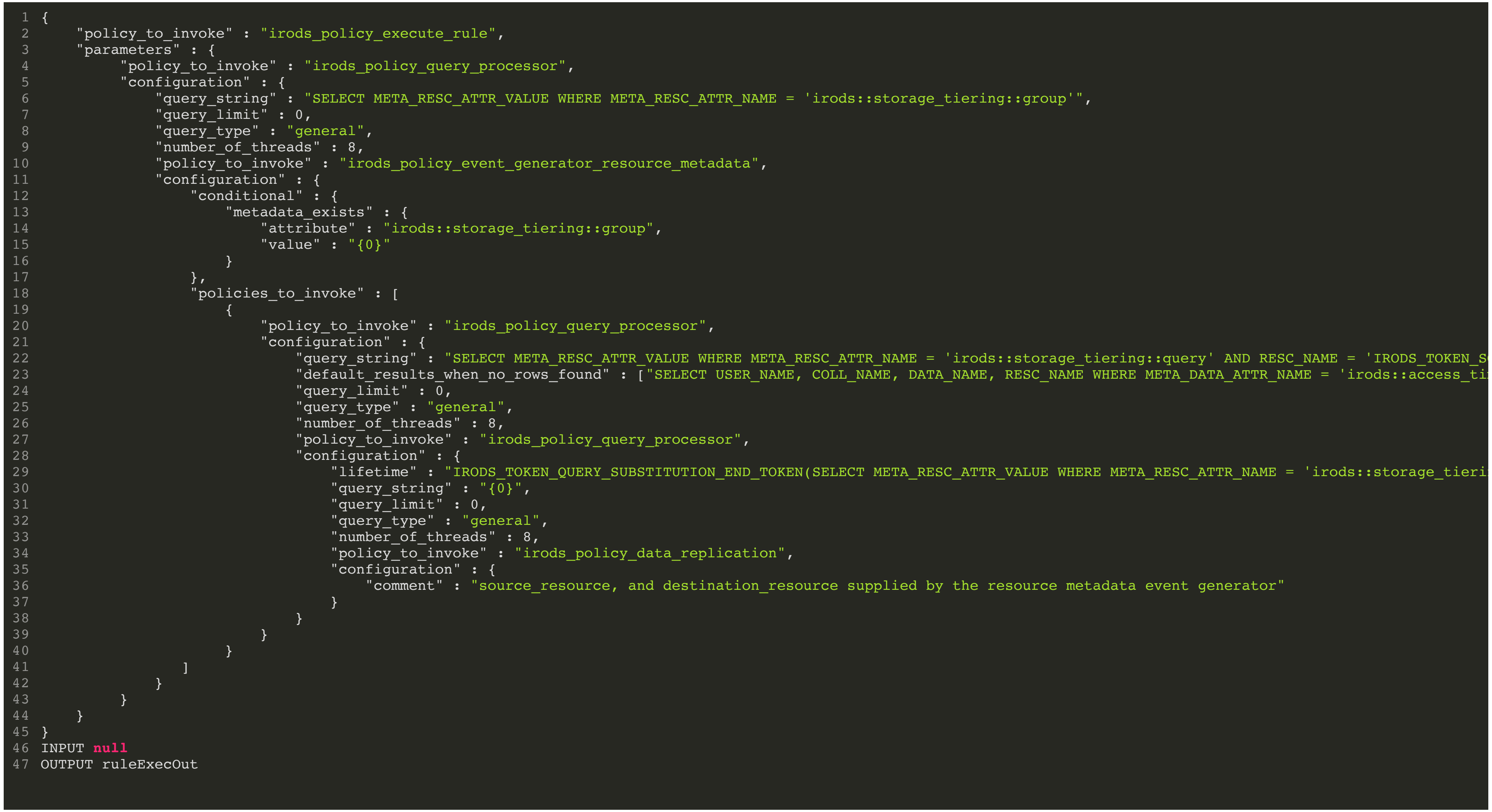
- Data Virtualization (Unified Namespace)
- Data Discovery (Metadata)
- Workflow Automation (Rule Engine)
- Secure Collaboration (Federation)

Policy Composed Storage Tiering

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

Policy Composed Capability

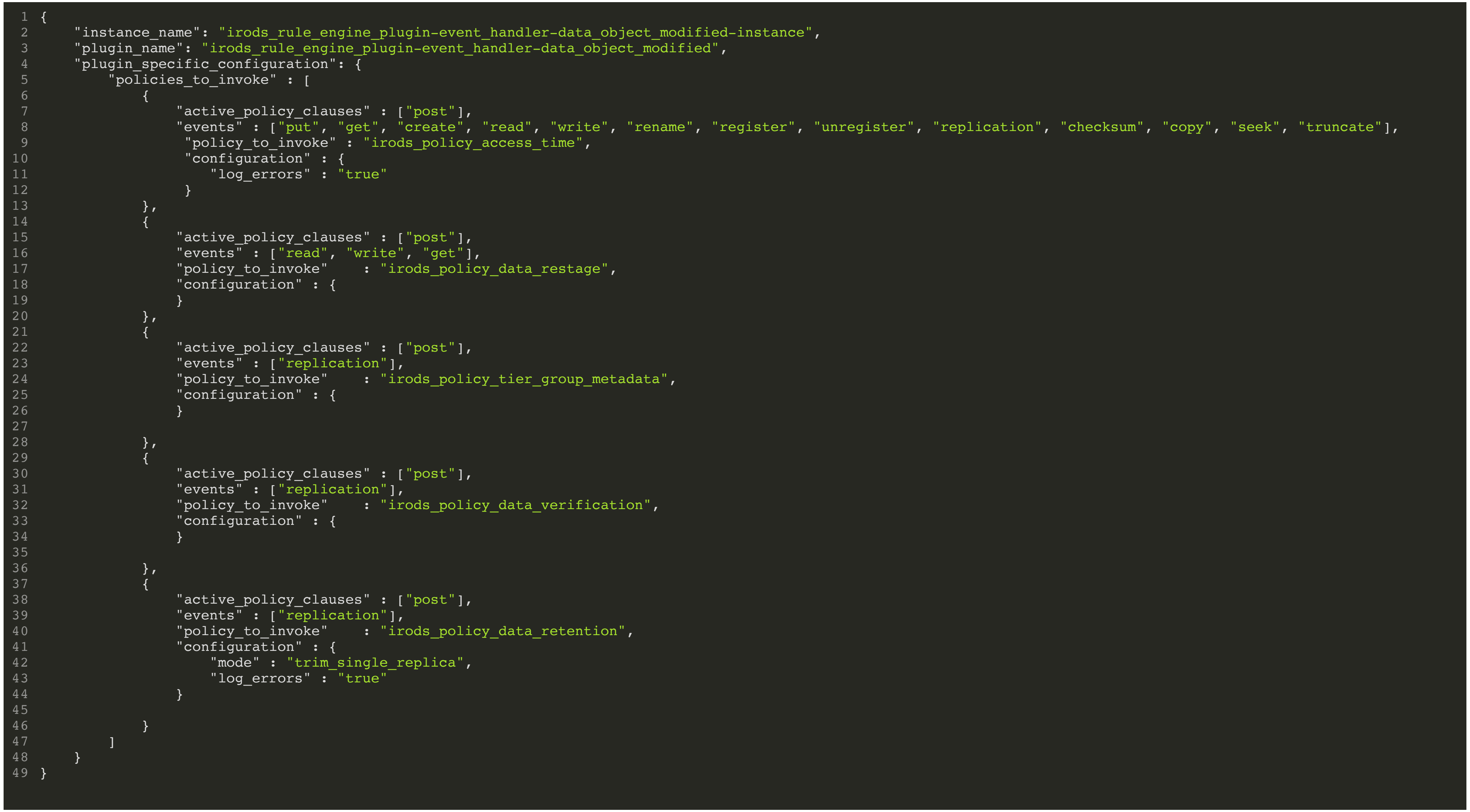
Asynchronous Discovery and Replication



```
{
    "policy_to_invoke" : "irods_policy_execute_rule",
    "parameters" : {
        "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_exists" : {
                        "attribute" : "irods::storage_tiering::group",
                        "value" : "{0}"
                    }
                },
                "policies_to_invoke" : [
                    {
                        "policy_to_invoke" : "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
```

Policy Composed Capability

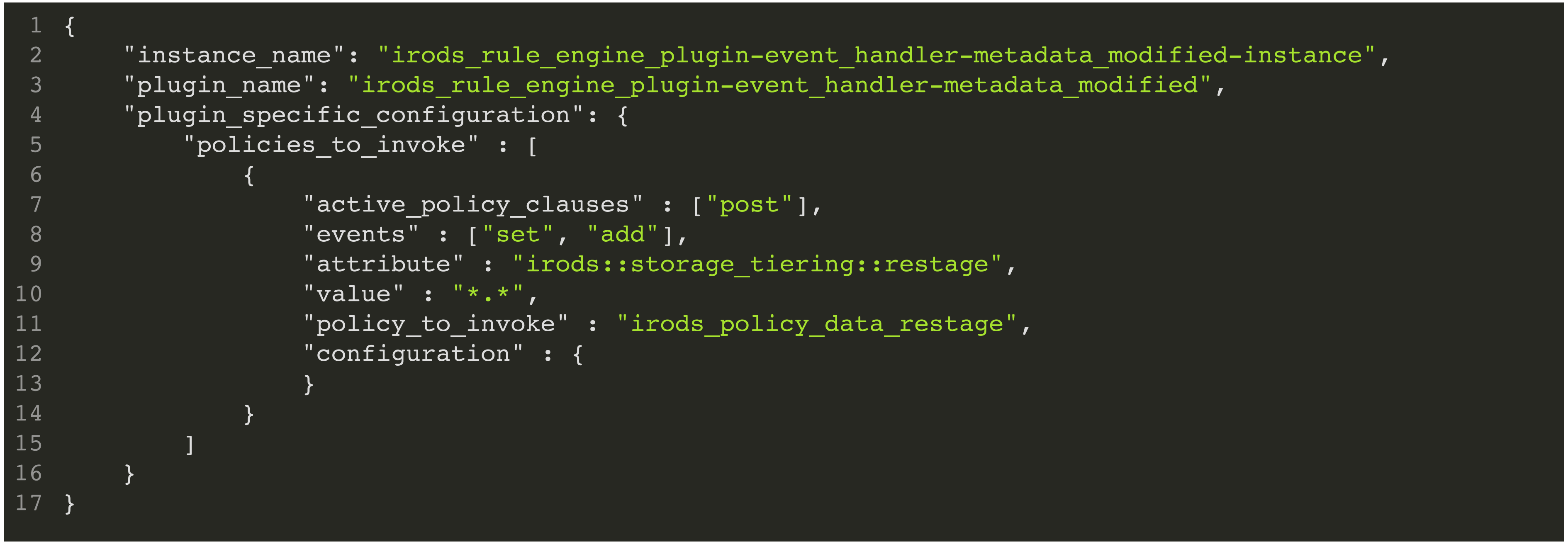
Synchronous Configuration for Storage Tiering



```
{
    "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_to_invoke" : "irods_policy_access_time",
                 "configuration" : {
                    "log_errors" : "true"
                 }
            },
            {
                "active_policy_clauses" : ["post"],
                "events" : ["read", "write", "get"],
                "policy_to_invoke"    : "irods_policy_data_restage",
                "configuration" : {
                }
            },
            {
                "active_policy_clauses" : ["post"],
                "events" : ["replication"],
                "policy_to_invoke"    : "irods_policy_tier_group_metadata",
                "configuration" : {
                }

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

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

            }
        ]
    }
}
```

Policy Composed Capability

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_to_invoke" : "irods_policy_data_restage",
                "configuration" : {
                }
            }
        ]
    }
}
```

iRODS

Data Management Model

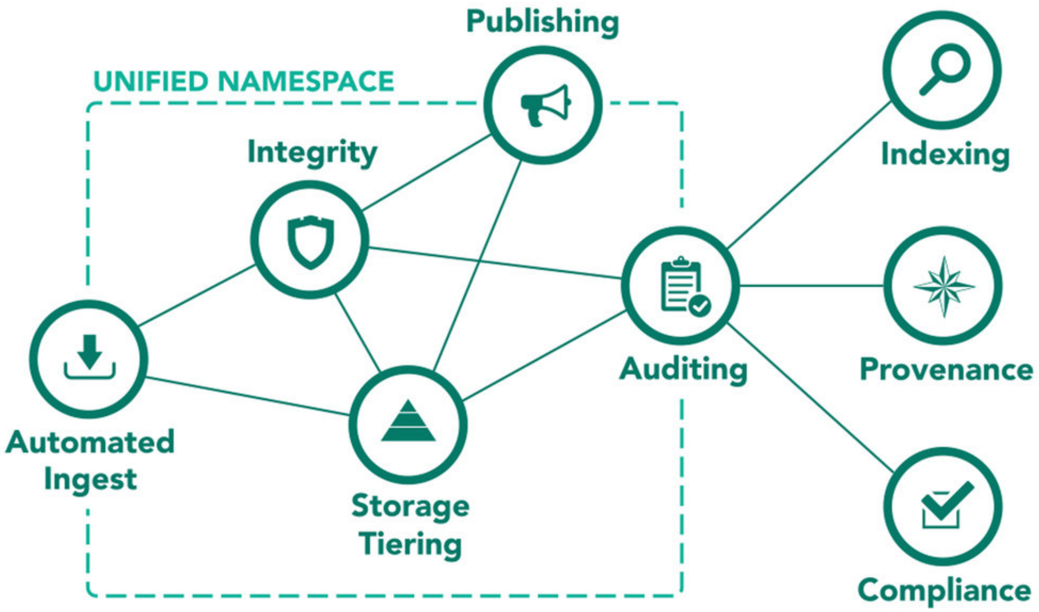
Ingest to Institutional Repository

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.

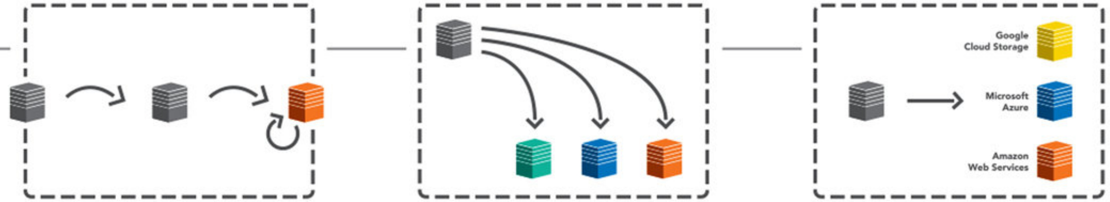
The flexibility provided by this model allows an organization to address its immediate use cases.

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:



Data to Compute

Compute to Data

Synchronization

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.