Universiteit Utrecht

# iRODS user empowerment: A matter of Sudo microservices

Chris Smeele - ITS/ResearchIT

## The need for automated privileged operations

**Privileged operations can be needed to ensure**

- process quality
- system security

**Implementation:**

- manually: by designated staff
- automated: policy controlled process (**user empowerment**)

Example of "manual" implementation: iadmin requires actor to be rodsadmin type user

***Real-time response requirements drive automation of privileged operations***

## Model of automated privileged operations

| | |
|---|---|
| control | • start supervised process |
| check | • ensure that preconditions are met |
| execute | • execute operation with elevated privileges |
| return | • end supervised process |

## So how can we empower our iRODS users?

**Why not create a set of microservices that**
**1) wrap selected existing iRODS functions**
**2) execute them with 'rodsadmin' privileges on behalf of the user**

***analog to the Linux "Sudo" command***

Universiteit Utrecht

## Design principles for our Sudo microservices

**Security by design**

- not enabled unless a precondition policy is defined

**Each privileged operation must be singular and targeted**

- to minimize impact of any potential flaws in design, implementation or configuration

**A privileged operation should be able to take part in a workflow**

- this allows for support of more complex use cases

## Extended model of privileged operations

| | |
|---|---|
| control | • start supervised process |
| check | • ensure that preconditions are met |
| execute | • execute operation with elevated privileges |
| trigger | • **optionally trigger next workflow step** |
| return | • end supervised process |

## Implementation in Sudo microservices

control
- user rule calls Sudo microservice XXX

check
- **acPreSudoXXX** policy used to check preconditions and to authorize user (process stops here if this policy fails)

execute
- execute XXX operation with elevated privileges

trigger
- **acPostSudoXXX** policy can initiate next workflow step

return
- Sudo microservice returns control to user rule

## Index of the set of SUDO microservices

- **iRODS User management**
  - msiSudoUserAdd
  - msiSudoUserRemove
- **iRODS Group management**
  - msiSudoGroupAdd
  - msiSudoGroupRemove
  - msiSudoGroupMemberAdd
  - msiSudoGroupMemberRemove
- **Access control**
  - msiSudoObjAclSet
- **Metadata management**
  - msiSudoObjMetaAdd
  - msiSudoObjMetaSet
  - msiSudoObjMetaRemove

## Example use case: delegated group management

iRODS users 'ton' and 'chris' are regular rodsusers.
Via a policy they are allowed to manage the iRODS group 'humanities'

In a rule they can call a Sudo microservice to add user 'john' to this group:

```
....
....
msiSudoGroupMemberAdd ('humanities', 'john', *policyKv);
....
....
```

NB: *policyKv is a variable that can be used to optionally pass additional information

## Preparation for this use case by rodsadmin

Use metadata so that users "ton" and "chris" act as admins for group "humanities"

```
imeta add –u humanities Admin ton
imeta add –u humanities Admin chris
```

and add the following policy to the rulebase to allow admins to add group members:

```
acPreSudoGroupMemberAdd( *groupName, *username, *policyKv) {
   foreach (*admin in SELECT META_DATA_ATTR_VALUE
               WHERE   USER_NAME = 'humanities'
               AND META_DATA_ATTR_NAME = 'Admin'
               AND META_DATA_ATTR_VALUE = '$userNameClient' ) {
          succeed;
   }
   fail;  # disallow all other users
}
```

## Sudo feature to overcome chicken and egg issue

Extended use case:

- we also want to use Sudo services to allow 'ton' and 'chris' to add groups
- (only) the actor should also act as the initial 'admin' of the newly created group

Preparation:

- establish policy to allow ton and chris to use msiSudoGroupAdd()

rule body that ton can use to add group 'science':

```
....
msiSudoGroupAdd('science', 'Admin', 'ton', '', *policyKv);
....
....
```

will be added by Sudo service as initial metadata to the 'science' group object, immediately after the group is created

## Demo

## Status of the Sudo microservices set

- Used by our university in our Yoda production systems (iRODS 4.1.8 based)

- Microservices plugin set
  - source and an RPM binary installable with iRODS 4.1.8 located at
  - https://github.com/UtrechtUniversity/irods-sudo-microservices/

- Current plan: package and make available as open source by July
  - with binaries for latest releases iRODS 4.1 and 4.2

Universiteit Utrecht

ITS – Research IT

# iRODS User Empowerment

## A matter of Sudo microservices

For more information: yoda@uu.nl
Chris Smeele can be reached at c.j.smeele@uu.nl

## SUDO microservices in detail:  user management

- msiSudoUserAdd (
```
    *userName,
    *initialAttr,
    *initialValue,
    *initialUnit,
    *policyKv
    )
```
- msiSudoUserRemove (
```
    *username,
    *policyKv
    )
```

## SUDO microservices in detail: group management

- msiSudoGroupAdd (
  ```
  *groupName,
  *initialAttr,
  *initialValue,
  *initialUnit,
  *policyKv
  )
  ```
- msiSudoGroupRemove (
  ```
  *groupName,
  *policyKv
  )
  ```
- msiSudoGroupMemberAdd (
  ```
  *groupName,
  *username,
  *policyKv
  )
  ```
- msiSudoGroupMemberRemove (
  ```
  *groupName,
  *userName,
  *policyKv
  )
  ```

## SUDO microservices in detail: ACL management

- msiSudoObjAclSet (
  *recursive,
  *accessLevel,
  *otherName,
  *objPath,
  *policyKv
  )

*recursive flag is of type integer
0 = no recursion, 1 = apply recursion

*accesslevel can be any of "null", "read", "write", "own", "inherit", "noinherit"
*NB: do not specify an* 'admin:' *prefix, this will be applied automatically*

*otherName is the name of an existing user or group

## SUDO microservices in detail: metadata management

- msiSudoObjMetaAdd (
  - *objName,
  - *objType,
  - *attribute,
  - *value,
  - *unit,
  - *policyKv
  - )

- msiSudoObjMetaSet:

(parameters similar to microservice msiSudoObjMetaAdd)

- msiSudoObjMetaRemove (
  - *objName,
  - *objType,
  - *wildcards,
  - *attribute,
  - *value,
  - *unit,
  - *policyKv
  - )

*wildcard flag is of type integer
0 = no wildcard, 1 = apply wildcard
wildcards follow imeta command syntax