# **HAWQ Ranger Support**

[HAWQ Ranger Support](#_2y562yky45h9)

[Revision History](#_djxn1iicx036)

[1. Introduction](#_8uqsvil5rjwa)

[2. Requirement](#_6p76yq37wqxc)

[3. Architecture](#_k1jot0chks9d)

[4. HAWQ Policy](#_h43iynost487)

[5. Implementations](#_jn5c5e5vquo)

[6. Interface with RESTful Ranger Authorizer Service](#_qfratuinbc6f)

[7. Future works](#_80c30pf9097q)

[8. References](#_2662n9rl95zg)

# **Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Date | Reason For Changes | Version | Status |
| Hubert Zhang,  Wen Lin,  Lili Ma,  Lei Chang | 2016.07.28 | Initial version | v0.1 | Draft |
| Hubert Zhang,  Lili Ma | 2016.08.29 | Forbid Grant/Revoke in HAWQ if Ranger configured;  Change the checkPrivilege API | v0.2 | Draft |
| Hubert Zhang,  Wen Lin,  Hong Wu,  Chunling Wang,  Lili Ma | 2016.11.18 | Add resource hierarchy  Remove ranger fallback to HAWQ | v0.3 | Draft |
|  |  |  |  |  |

# 

# 

# 1. Introduction

Apache Ranger is a centralized security framework to manage fine grained access control over Hadoop and related components. To sync with other components such as Hive/HBase, it’s useful for HAWQ to support Ranger.

Ranger Usage:

1. Manage policy for accessing certain sources to certain user/groups, and enforce the policy
2. Audit tracking
3. Policy analysis for deeper control of the system
4. Delegate administration of certain data to other group owners

Ranger Structure:

1. Ranger Portal. A centralized web application providing policy administration, audit and report functions. Can be used by authorized users through web tools or REST APIs.
2. Ranger Plugin. Embedded in processes of cluster. In hadoop eco-systems, are enforced by lightweight Java plugins. Plugin extracts the policy from Ranger Portal server at a regular intervals.
3. User Group Sync. Sync user information from Unix, LDAP, or active Directory.

# 2. Requirement

1. Central control user list for HAWQ and other systems on Hadoop, user Information can be established in LDAP/AD/Unix.
2. Use Ranger to do user access policy definition. If Ranger is configured, can define policy in Ranger console, also can use grant privilege sql in HAWQ and pass that to Ranger; else, use HAWQ own’s grant privilege.
3. Use Ranger to do user access rights authorization. If Ranger is configured, need access Ranger to do user privilege checking. Else, use HAWQ own catalog privilege checking. Note that this is HAWQ level privilege checking, not HDFS level.
4. Audit Function. Can use Ranger to track the user login behaviour. Also can use Ranger to look up the user account modification history.

# 3. Architecture



Figure 1. HAWQ-Ranger Integration Architecture

There are several components to break down.

1. Ranger Server. For policy storage and user privilege check.
2. Ranger REST Service. Provide a REST Service for HAWQ. Embed in HAWQ Ranger Plugin.
3. HAWQ.
4. LDAP/Unix. In charge of user management.

Functions Provided.

1. Create user. Can create user in LDAP/Unix(Corresponding to 1.1,1.2,1.3 in Figure 1.)

For Ranger, the UserSync process will sync user information from LDAP/Uni

periodically. For HAWQ, we can choose one from below two options:

1. Use HAWQ legacy user management. Use scripts or manually to execute “create role” statement in HAWQ for each user in LDAP/Unix.
2. Don’t create any user in HAWQ. Just keep one superuser(usually gpadmin) in HAWQ to communicate with Ranger. Don’t store any other user information in HAWQ catalog. (Need spike feasibility and effort in code change in HAWQ if deprecating user creation part)
3. Create user privilege policy. Can do this through Ranger Web UI (Corresponding to 2 in diagram). Note that we will forbid Grant/Revoke/Reassign SQL in HAWQ side if Ranger integration is configured.
4. User privilege check policy. Every time a connection request or a query is submitted in HAWQ side, need connect Ranger server to verify whether the user has the permission to do that. (Corresponding to 4.1-4.4 in Figure 1).
5. There is another function in Ranger Admin server to lookup resources such as databases/tables in HAWQ(corresponding to 5 in Figure 1). Can be implemented through JDBC.

Note that. If Ranger is not configured, we directly use HAWQ own’s user access control mechanism, say, catalog permission. If configured, we will always use Ranger’s authorization, say, if Ranger is down, HAWQ is not available.

# 4. HAWQ Policy

1. HAWQ Policy Resources
   1. Database
   2. Table
   3. Sequence
   4. Function
   5. Language
   6. Schema
   7. Tablespace
   8. Protocol

Resources has a hierarchy to distinguish the resources with the same name, for example two table with the same name under different schema is allowed and will be treated as different resource.

|--database

| |--schema

| | |--table

| | |--sequence

| | |--function

| |

| |--language

|

|--tablespace

|--protocol

1. Permission Type
   1. Select
   2. Insert
   3. Update
   4. Delete
   5. References
   6. Usage
   7. Create
   8. Connection
   9. Execute
   10. Temporary(Temp): create a temp table

Below is the privileges list for objects should be outlined in Ranger.

* 1. Database: Create/Connect/Temp
  2. Table: Select/Insert/Update/Delete/Reference
  3. Sequence: Usage/Select/Update
  4. Function: Execute
  5. Language: Usage
  6. Schema: Create/Usage
  7. Tablespace: Create
  8. Protocol: Select/Insert

Note that the permission types for different resources are different. For example, database has connection privilege, while table has insert/select/.. Privilege without connection. Below lists the details of mapping between resource and privilege.

# 5. Implementations

For grant privilege in HAWQ, need modify the function ExecuteGrantStmt to forbid grant/revoke command when HAWQ is in ranger mode.

For check privilege in HAWQ, need modify the functions in aclchk.c, auth.c.

# 6. Interface with RESTful Ranger Authorizer Service

HAWQ call REST API to communicate with Ranger Authorizer Service for checking privilege.

The requirement will be converted to json request and sent to Ranger Authorizer Service(RAS for short). RAS pass the json request and call ranger-hawq-plugin to check privilege.

Input JSON Format:

For check privilege:

{

"requestor": u1,

"access":[

{

"resource":

{

"database": "db1"

},

"privileges": ["create"]

"how": ANY

},

{

"resource":

{

"database": "db1",

"schema": "schema1",

"table": "table1"

},

"privileges": ["select"]

"how": ALL

}

]

}

*Explanations: u1 wants to manipulate the table t1 in database db1(one of select and insert privilege is OK), and manipulate the table t2 in database db1(both of select and insert privilege is needed). Calling of checkPrivileges() need check the privilege for this operation.*

Return Value JSON Format:

{

“Result” : [ 1, 0 ]

}

An integer array should be returned from REST service. Since we may combine several requests in one REST service call, the array indicates the privilege state of every request. 0 for Success, 1 for Failed

Ranger-hawq-plugin(JAVA) related member functions(suggested):

*public int[] checkPrivileges(HawqObject[] object, HawqAction[][] action, privilegeType[] how User grantor)*

# 7. Future works

1. HAWQ Local Cache of policy for performance issues.
2. A tool for migrating user privilege policy from Ranger to HAWQ and from HAWQ to Ranger, in case that user may want to use Ranger and then doesn’t want to use it anymore, or user may not use Ranger and then want use it. Provide a tool for policy migration.

# 8. References

1. Apache Ranger Links:

<http://ranger.apache.org/faq.html>

<http://hortonworks.com/apache/ranger/#section_2>

2. HAWQ Side Links:

<http://hdb.docs.pivotal.io/20/reference/sql/GRANT.html>

3. LDAP

<http://www.openldap.org/>