Support for $username variable

- Introduction
- Rationale
- Use Cases
  - HDFS
    - Resource Names
    - Access Control Regime
  - Hive
    - Resource Names
    - Access Control Regime
- Recommended practices and Customizability
- For Experienced Ranger Users and developers

Introduction

This article provides rationale, use-cases, recommended practices/customizability, and more information for experienced Ranger users and developers.

Rationale

It is customary for large data centers to identify user-accessible data resources (HDFS files/directories, Hive databases/tables, etc.) encoded with the user-name or some other user-specific attribute value in their name. In such cases, Ranger administrator needs to author multiple policies addressing these distinct resource names. RANGER-698 proposes to provide a generic way to author ranger policies exploiting such relationships between resource-name, and access permissions for its users to achieve equivalent access control regime with a single policy or a small number of policies. In addition to adding clarity to the mapping of enterprise-wide high-level access-control regime to ranger policy specifications, fewer policies, in general, lead to significant improvement in capacity and performance in ranger administration as well as ranger-enabled components.

Use Cases

HDFS

Resource Names

There are multiple HDFS users, each with a 'home' directory under '/home' which is named by the user's name.

Access Control Regime

A user can access all files only under their own 'home' directory.

Ranger Policies:

resource: path=/home/{USER}
user: {USER}
permissions: all, delegateAdmin=true

This will allow all access for

- User 'user1' to path /home/user1
- User 'user2' to path /home/user2
- And so on...

In this way, this one policy can replace many ranger policies each with a different resource specification and different user in its policy-item specification.

Hive

Resource Names

There are multiple databases within Data Center/Data Lake. Database names contain the user's name, which has all access permissions to it. There may be thousands of hive users.

Access Control Regime
A user can access only the database that is named with their name, and cannot access any other database unless it is a special user named 'hive'.

Ranger Policies:

Policy-1: Grants all access to all databases/table/columns for user 'hive'

resource: database=.*, table=.*, column=*  
user: hive  
permissions: all, delegateAdmin=true

Policy-2: Grants all access to the database which is named by the user making the access request.

resource: database=db_{USER}; table=.*, column=*  
user: {USER}  
permissions: all, delegateAdmin=true

This will allow all access for

- User 'user1' on database 'db_user1'
- User 'user2' on database 'db_user2'
- And so on...
- User 'hive' on database 'db_user1' and 'db_user2'

Recommended practices and Customizability

Ranger requires that string '{USER}' is used to represent accessing user as the user in the policy-item in a Ranger policy. However, Ranger provides flexible way of customizing the string that is used as shorthand to represent the accessing user’s name in the policy resource specification. By default, Ranger policy resource specification expects characters '{' and '}' as delimiters for string 'USER', however, ranger supports customizable way of specifying delimiter characters, escaping those delimiters, and the string 'USER' itself by prefixing it with another, user-specified string on a per resource-level basis in the service definition of each component supported by Ranger.

For example, if for a certain HDFS installation, if the path names may contain '{' or '}' as valid characters, but not '%' character, then the service-definition for HDFS can be specified as

```json
"resources": [  
  {  
    "itemId": 1,  
    "name": "path",  
    "type": "path",  
    "level": 10,  
    "parent": "," ,  
    "mandatory": true,  
    "lookupSupported": true,  
    "recursiveSupported": true,  
    "excludesSupported": false,  
    "matcher": "org.apache.ranger.plugin.resourcematcher.RangerPathResourceMatcher",  
    "matcherOptions": {"wildcard": true, "ignoreCase": false, "replaceTokens":true, "tokenDelimiterStart": ";", "tokenDelimiterEnd": ";", "tokenDelimiterPrefix":"rangerToken:"}  
    "validationRegEx": "",  
    "validationMessage": "",  
    "uiHint": "",  
    "label": "Resource Path",  
    "description": "HDFS file or directory path"  
  }  
]
```

Corresponding ranger policy for the use case for HDFS will be written as follows.

resource: path=/home/%rangerToken:USER%

user: (USER)

permissions: all, delegateAdmin=true

The following customizable matcherOptions are available for this feature.
For Experienced Ranger Users and developers

This feature, at its core, supports general-purpose identification of special patterns in the resource specification and their replacement at run-time with other strings to derive the name of the resource, before matching it with the resource being accessed by user. Therefore, it is not limited to replacement of string 'USER' with current-user's name; it is just something that is offered out of the box.

In order that Ranger user can use the underlying core functionality, they need to be familiar with interfaces provided by Ranger for customizing Ranger, such as RangerContextEnricher, RangerAccessRequest and RangerConditionEvaluator.

The following methods are provided to populate and reference context in RangerAccessRequest object that represents access request in the policy evaluation engine.

- `RangerAccessRequestUtil.setTokenInContext(Map<String, Object> context, String tokenName, Object tokenValue);`
- `Object RangerAccessRequestUtil.getTokenFromContext(Map<String, Object> context, String tokenName);`

An advanced Ranger user will need to provide the following to use the core functionality offered by this feature.

1. Provide an implementation of RangerContextEnricher and include it in the service-definition. The implementation of RangerContextEnricher.enrich() method needs to get the context of the RangerAccessRequest provided to it and use RangerAccessRequestUtil.setTokenInContext() API to populate it with specific 'tokenName' (such as 'USER') and its value, derived based on some run-time information and enricher's configuration parameters.
2. Customize component's service definition with appropriate "matcherOptions" for each resource definition supported by it as described above in customizability section.
3. Provide an implementation of RangerConditionEvaluator and include it in the service definition. The implementation of RangerConditionEvaluator.isMatched() API needs to retrieve value of the 'tokenName' from the request's context using RangerAccessRequestUtil.getTokenFromContext() API, and return appropriate result.
4. Author a ranger-policy with the resource-specification containing the 'tokenName' to implement appropriate access-control for the resource-name built at run-time by Ranger.