Hortonworks Data Platform

Adding a New Component to Apache Ranger

(Dec 2, 2014)

docs.hortonworks.com

Hortonworks Data Platform: Adding a New Component to Apache Ranger

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, Zookeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, training and partner-enablement services. All of our technology is, and will remain free and open source.

Please visit the Hortonworks Data Platform page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the Support or Training page. Feel free to Contact Us directly to discuss your specific needs.

Table of Contents

1. Adding a New Component to Apache Ranger 1
2. Developing a Custom Authorization Module 4

1. Adding a New Component to Apache Ranger

This document provides a general description of how to add a new component to Apache Ranger.

Apache Ranger has three main components:

- **Admin Tool** – Provides web interface & REST API for managing security policies
- **Custom Authorization Module for components** – Provides custom authorization within the (Hadoop) component to enforce the policies defined in Admin Tool
- **UserGroup synchronizer** – Enables the user/group information in Apache Ranger to synchronize with the Enterprise user/group information stored in LDAP or Active directory.

For supporting new component authorization using Apache Ranger, the component details needs to be added to Apache Ranger as follows:

- Add component details to the Admin Tool
- Develop a custom authorization module for the new component

Adding Component Details to the Admin Tool

The Apache Ranger Admin tool supports policy management via both a web interface (UI) and support for (public) REST API. In order to support a new component in both the UI and the Server, the Admin Tool needs to be modified.

Required UI changes to support the new component:

1. Add a new component template to the Policy Manager page (console home page):

   Show new component on the policy manager page i.e home page[#!/policymanager]. Apache Ranger needs to add table template to policy manager page and make changes in corresponding JS files. Ranger also needs to create a new repository type enum to distinguish the component for which the repository/policy is created/updated.

   For example: Add a table template to PolicyManagerLayout_tmpl.html file to view the new component on the Policy Manager page and make changes in the PolicyManagerLayout.js file related to the new componen, such as passing knox repository collection data to the PolicyManagerLayout_tmpl template. Also create a new repository type enum (for example, ASSET_KNOX) in the XAEnums.js file.

2. Add new configuration information to the Repository Form:

   Add new configuration fields to Repository Form [AssetForm.js] as per new component configuration information. This will cause the display of new configuration fields in the corresponding repository Create/Update page. Please note that the AssetForm.js is a common file for every component to create/update the repository.

For example: Add new field(configuration) information to AssetForm.js and AssetForm_tmpl.js.

3. Add a new Policy Listing page:

   Add a new policy listing page for the new component in the View Policy list. For example: Create a new KnoxTableLayout.js file and add JS-related changes as per the old component[HiveTableLayout.js] to the View Policy listing. Also create a template page, KnoxTableLayout_tmpl.html.

4. Add a new Policy Create/Update page:

   Add a Policy Create/Update page for the new component. Also add a policy form JS file and its template to handle all policy form-related actions for the new component. For example: Create a new KnoxPolicyCreate.js file for Create/Update Knox Policy. Create a KnoxPolicyForm.js file to add knox policy fields information. Also create a corresponding KnoxPolicyForm_tmpl.html template.

5. Other file changes, as needed:

   Make changes in existing common files as per our new component like Router.js, Controller.js, XAUtils.js, FormInputList.js, UserPermissionList.js, XAEnums.js, etc.

Required server changes for the new component:

Let's assume that Apache Ranger has three components supported in their portal and we want to introduce one new component, Knox:

1. Create New Repository Type

   If Apache Ranger is introducing new component i.e Knox, then they will add one new repository type for knox. i.e repositoryType = “Knox”. On the basis of repository type, while creating/updating repository/policy, Apache Ranger will distinguish for which component this repository/policy is created/updated.

2. Add new required parameters in existig objects and populate objects

   For Policy Creation/Update of any component (i.e HDFS, Hive, Hbase), Apache Ranger uses only one common object, `VXPolicy.` The same goes for the Repository Creation/Update of any component: Apache Ranger uses only one common object `VXRepository.` As Apache Ranger has three components, it will have all the required parameters of all of those three components in `VXPolicy/VXRepository.` But for Knox, Apache Ranger requires some different parameters which are not there in previous components. Thus, it will add only required parameters into `VXPolicy/VXRepository` object. When a user sends a request to the Knox create/update policy, they will only send the parameters that are required for Knox to create/update the VXPolicy object.

   After adding new parameters into VXPolixy/VXRepository, Apache Ranger populates the newly-added parameters in corresponding services, so that it can map those objects with Entity Object.

3. Add newly-added fields (into database table) related parameters into entity object and populate them

As Apache Ranger is using JPA-EclipseLink for database mapping into java, it is necessary to update the Entity object. For example, if for Knox policy Apache Ranger has added two new fields (`topology` and `service`) into db table `x_resource`, it will also have to update the entity object of table (i.e `XXResource`), since it is altering table structure.

After updating the entity object Apache Ranger will populate newly-added parameters in corresponding services (i.e XResourceService), so that it can communicate with the client using the updated entity object.

4. Change middleware code business logic

   After adding and populating newly required parameters for new component, Apache Ranger will have to write business logic into file `AssetMgr`, where it may also need to do some minor changes. For example, if it wants to create a default policy while creating the Repository, then on the basis of repositoryType, Apache Ranger will create one default policy for the given repository. Everything else will work fine, as it is common for all components.

Required database changes for the new component:

For repository and policy management, Apache Ranger includes the following tables:

- x_asset (for repository)
- x_resource (for repository)

As written above, if Apache Ranger is introducing new component then it is not required to create individual table in database for each component. Apache Ranger has common tables for all components.

If Apache Ranger has three components and wants to introduce a fourth one, then it will add required fields into these two tables and will map accordingly with java object. For example, for Knox, Apache Ranger will add two fields (`topology`, `service`) into `x_resource`. After this, it will be able to perform CRUD operation of policy and repository for our new component, and also for previous components.

2. Developing a Custom Authorization Module

In the Hadoop ecosystem, each component (i.e., Hive, HBase) has its own authorization implementation and ability to plug in a custom authorization module. To implement the centralized authorization and audit feature for a component, the component should support a customizable (or pluggable) authorization module.

The custom component Authorization Plugin should do the following:

- Provide authorization based on Policies defined in Policy Admin Tool
- Provide audit information based on the authorization decisions

Implementing Custom Component Authorization

To implement the custom component authorization plugin, the Ranger common agent framework provides the following functionalities:

- Ability to read all policies from Policy Manager for a given repository-id
- Ability to log audit information

When the custom authorization module is initialized, the module should do the following:

1. Initiate a REST API call to the "Policy Admin Tool" to retrieve all policies associated with the specific component.
2. Once the policies are available, it should:
    - be built into a custom data structure for enabling the authorization module.
    - kick off the policy updater thread to refresh policies from "Policy Admin Tool" at a regular interval.

When the custom authorization module is called to perform authorization of a component action (such as READ action) on a specific component resource (such as /app folder), the authorization module will:

- **Identify authorization decision** - For each policy:policyList:
    - If (resource in policy <match> auth-requested-resource)
    - If (action-in-policy <match>action-requested
    - If (current-user or current-user-groups or public-group <allowed> for the policy), Return access-allowed
- **Identify auditing needs** - For each policy:policyList
    - If (resource in policy <match> auth-requested-resource), return policy.isAuditEnabled()