# Ranger Stacks - How to add a custom plugin?

## Introduction

Apache Ranger provides centralized security for Enterprise Hadoop ecosystem, including fine-grained access control and centralized auditing. Apache Ranger 0.4 provides authorization and auditing for the following services: HDFS, HBase, Hive, Knox and Storm. Adding support for more services would require changes to various modules of Apache Ranger implementation, like – UI, APIs, database schema, etc.

Apache Ranger 0.5 supports stack-model to enable easier onboarding of new components, without requiring code changes in Apache Ranger. This document provides details of the stack model and the steps to enable a new service to integrate with Apache Ranger.

## Steps to create a Ranger Authorization Plugin

This section provides a high level overview of the steps involved in creating a  anger authorization plugin. More details on each step are provided in later sections.

### Define Service-type

1. Create a JSON file with the following details about the service:
    - Resources. Example: database, table, column, etc.
    - Access types. Example: select, update, create, drop, etc.
    - Configuration to connect to the service. Example: JDBC URL, JDBC driver, credentials, etc.
- Load the JSON into Ranger.

### Develop Ranger Authorization Plugin

#### During initialization of the service:

1. Create a static/global instance of RangerBasePlugIn class (or a class derived from this). Keep a reference to this instance for later – to authorize resource access.
2. Call init() on this instance. This will initialize the policy-engine with authorization policies from Ranger Admin and trigger a background thread to periodically update policies from the Ranger Admin.
3. Register an audit handler, like RangerDefaultAuditHandler, with the plugin instance. Plugin will use this audit handler to generate audit logs of resource accesses.

### To authorize access to a resource:

1. Create an instance of RangerAccessRequest implementation, like RangerAccessRequestImpl, with details of the access that needs to be authorized – resource, access-type, user, etc.
2. Call isAccessAllowed() on the plugin instance created earlier.
3. Depending upon the returned result, either allow or deny the access.

### To support resource lookup:

1. Extend class RangerBaseService and provide implementation of lookupResource() and validateConfig() methods.
2. Provide the name of this class in service-type definition.
3. Copy the library (jar file) that includes the class implementation, and other libraries referenced by this class, under ranger-plugins/*<service-type>* directory in CLASSPATH of Ranger Admin.

## Install the plug-in in the service:

The ranger-plugin for the service must be installed and configured to run in the service where the access authorization is to be performed. Please consult the documentation of the service for details of registering an authorizer.

### Service Type

## Service Type Definition

Resources of a service, along with other details like type of resource accesses (read/write/create/delete/submit/…), configuration needed to connect to the service (url, username, password, …) , custom conditions to evaluate in policies (IP range, …), etc., are defined using JSON – as shown in the following example. For more up-to-date service-type JSON, please refer to one of the service-types in Apache Ranger source (here).

## Example: YARN Service Type definition

```
{
 "name": "yarn",
 "implClass": "org.apache.ranger.services.yarn.RangerServiceYarn",
 "label": "YARN",
 "description": "YARN",
 "guid": "5b710438-edcf-4e20-834c-a9a267b5b963",
 "resources":
 [
  {
   "name": "queue",
   "type": "string",
   "level": 10,
   "mandatory": true,
   "lookupSupported": true,
   "recursiveSupported": true,
   "matcher": "org.apache.ranger.plugin.resourcematcher.RangerPathResourceMatcher",
   "matcherOptions": {"wildCard":true, "ignoreCase":true, "pathSeparatorChar":"."},
   "label": "Queue",
   "description": "Queue"
  }
 ],
 "accessTypes":
 [
  {
   "name": "submit-app",
   "label": "submit-app"
  },
  {
   "name": "admin-queue",
   "label": "admin-queue"
  }
 ],

 "configs":
 [
  {
```

```
    "name": "username",
    "type": "string",
    "mandatory": true,
    "label": "Username"
   },
   {
    "name": "password",
    "type": "password",
    "mandatory": true,
    "label": "Password"
   },
   {
    "name": "yarn.url",
    "type": "string",
    "mandatory": true,
    "label": "YARN REST URL"
   },

   {
    "name": "commonNameForCertificate",
    "type": "string",
    "mandatory": false,
    "label": "Common Name for Certificate"
   }
  ],

  "policyConditions":
  [
   {
    "name": "ip-range",
    "evaluator": "org.apache.ranger.plugin.conditionevaluator.RangerIpMatcher",
    "label": "IP Address Range",
    "description": "IP Address Range"
   }
 ]
}
```

## Register Service Type definition with Ranger

Service type definition should be registered with Ranger using REST API provided by Ranger Admin. Once registered, Ranger admin will provide UI to create service instances (called as repositories in previous releases) and policies for the service-type. Ranger plugin uses the service type definition and the policies to determine if an access request should be granted or not. The REST API can be invoked using curl command as shown in the example below:

curl -u admin:admin -X POST -H "Accept: application/json" -H "Content-Type: application/json" –d @ranger-servicedef-yarn.json http://ranger-admin-host:port/service/plugins/definitions

## Ranger Plugin Development

## Ranger Authorizer

Ranger authorization for a service is generally built as a library that implements service specific hooks to intercept resource access requests and call Ranger APIs to authorize and audit the accesses. These hooks would be registered with the service while installing the plugin in a service. In this section, we will go through the details of Ranger plugin implementation for YARN. Service type definition for YARN is provided in the previous section.

A static/global instance of RangerBasePlugin class should be created and initialized during the initialization of the service. The reference to the instantiated plugin object should be preserved for later reference during authorization of access requests.

During initialization, the plugin will load policies from local cache, if exists, and setup a policy refresher to get the updated policies from Ranger admin.

YARN service requires authorization providers to implement YarnAuthorizationProvider interface. Ranger plugin for YARN implements the init() and c heckPermission() methods, as shown below, to provide authorization and auditing of access to YARN queues.

```
public class RangerYarnAuthorizer extends YarnAuthorizationProvider {
 private static RangerBasePlugin plugin = null;
 @Override
 public void init(Configuration conf) {
  plugin = new RangerBasePlugin("yarn", "yarn");
  plugin.init(); // this will initialize policy engine and policy refresher
  plugin.setDefaultAuditHandler(new RangerDefaultAuditHandler());
 }
 @Override
 public boolean checkPermission(AccessType accessType, PrivilegedEntity entity, UserGroupInformation ugi) {
  RangerAccessRequestImpl request  = new RangerAccessRequestImpl();
  RangerResourceImpl      resource = new RangerResourceImpl();

  resource.setValue("queue", entity.getName());
   request.setResource(resource);
  request.setAccessType(getRangerAccessType(accessType));
   request.setUser(ugi.getShortUserName());
   request.setUserGroups(Sets.newHashSet(ugi.getGroupNames()));
   request.setAccessTime(new Date());
   request.setClientIPAddress(getRemoteIp());
  RangerAccessResult result = plugin.isAccessAllowed(request);
```

```
  return result == null ? false : result.getIsAllowed();
 }
}
```

## Resource Lookup

When authoring policies in Ranger admin, the users enter the name of the resources whose access need to be protected. To make it easier for users to enter the resource names, Ranger admin provides autocomplete feature, which looks up the available resources in the service that match the input entered so far.

The implementation of this lookup is specific to the service in which the resources are accessed. It involves using the APIs provided by the service to connect and retrieve the available resources. To facilitate the autocomplete feature, Ranger Admin requires the plugin to provide the implementation of RangerBaseService interface. The implementation class should be registered with Ranger in service type definition and be made available in the CLASSPATH of Ranger Admin.

```
public class RangerServiceYarn extends RangerBaseService {
 public HashMap<String, Object> validateConfig() throws Exception {
  // TODO: connect to YARN resource manager; throw Exception on failure
}
 public List<String> lookupResource(ResourceLookupContext context) throws Exception {
  // TODO: retrieve the resource list from YARN resource manager using REST API
}
}
```

# Install and configure plugin in the service

The ranger-plugin for the service must be installed and configured to run in the service where the access authorization is to be performed. Please consult the documentation of the service for details of registering an authorization.

Jar file(s) containing the implementation of the plugin and the following Ranger libraries should be available at runtime for the service. Please ensure that these files are in the CLASSPATH of the service:

- ranger-plugins-audit-<version>.jar
- ranger-plugins-common-<version>.jar
- ranger-plugins-cred-<version>.jar

Ranger plugin libraries read the following configuration files during initialization of the plugin. Please ensure that these files are present in the CLASSPATH of the service:

- ranger-<serviceType>-audit.xml
- ranger-<serviceType>-security.xml
- ranger-policymgr-ssl.xml

Ranger plugin requires the following configuration to be available at runtime. These properties would typically be in ranger-<serviceType>-security.xml.

| Configuration | Default Value | Comments |
|---|---|---|
| ranger.plugin.<serviceType>.service.name | No default value. This configuration must be provided. | Name of the service containing policies for the plugin |
| ranger.plugin.<serviceType>.policy.source.impl | org.apache.ranger.admin.client.RangerAdminRESTClient | Name of the class used to retrieve policies. |

| | | |
|---|---|---|
| ranger.plugin.<serviceType>.policy.rest.url | No default value. | URL to Ranger Admin |
| ranger.plugin.<serviceType>.policy.rest.ssl.config.file | No default value. This configuration must be provided if SSL is enabled between plugin and Ranger Admin. | Path to the file containing SSL details to contact Ranger Admin |
| ranger.plugin.<serviceType>.policy.cache.dir | No default value. If no valid value is specified, local caching of policies will not be done. | Directory where Ranger policies are cached after successful retrieval from the source |
| ranger.plugin.<serviceType>.policy.pollIntervalMs | 30000 | How often to poll for changes in policies? |