Allocation Manager
Nikitha Chettiar
Science Gateway Architectures

Resource allocation manager is a new feature in Airavata which allows user to request for resources which will be manager and allocation by an admin. The allocation manager considers three main roles:

1. User: The user requests for an allocation.
2. Admin: The admin manages the allocations, assigns reviewers and gives the final decision (approve or deny) on the request.
3. Reviewer: The reviewer reviews and gives his decision.

The objective of the project was to complete a cycle which involves the following step:

1. Let a user put in the request.
2. Allow admin to assign a reviewer.
3. Reviewer can review the request and give the decision.
4. Admin gives the final decision after verifying the reviewer's decision.
5. Notify the user about the status of the request.

The implementation of the resource allocation was a collaborative task with peers. My contribution to the project is as follows:

1. Creating the notification manager module for the project.
2. Creating the reviewer and admin interface.
3. Creating API methods letting the reviewer and admin review a request and thereby update the status of the request based on their decision.

The project is present on Airavata-Sandbox repository in allocation-manager directory:
https://github.com/apache/airavata-sandbox/tree/master/allocation-manager

The allocation-manager has the consists of the following setup:

1. The airavata-allocation-manager – a multi module maven projects with the following modules:
    - airavata-allocation-manager-stubs –

This module consists of the client files for the allocation manager and the notification manager. The stubs also consist of the model files which are like bean classes representing each database table. The AllocationRegistryService.java in the stubs folder is the interface of all the API methods implemented by the micro service.

- airavata-allocation-manager-server –

This module consists of server files for the allocation manager. This module consists of the implementation which interacts with the database. The AllocationManagerServerHandler.java is the file which contains the implementation for the API methods that is implementing the AllocationRegistryService.java interface.

An example flow will be as follows:

- The allocation manager micro service uses Apache Thrift for its communication.

- The thrift client which is AllocationManagerReviewerClient.java calls a client.updateAllocationRequestStatus(projectId, status) present in the AllocationManagerServerHandler.java.
- An example function implementation is as follows :

```
public void updateAllocationRequestStatus(String projectId, String status) throws TException {
    // TODO Auto-generated method stub
    UserAllocationDetailRepository userAllocationDetail = new UserAllocationDetailRepository();
    try {
      UserAllocationDetailEntityPK userAllocationDetailPK = new UserAllocationDetailEntityPK();
      userAllocationDetailPK.setProjectId(projectId);
      userAllocationDetailPK.setUsername(new
UserAllocationDetailRepository().getPrimaryOwner(projectId));
      userAllocationDetail.get(userAllocationDetailPK).setStatus(status);

      //once status updated notify user
      (new NotificationManager()).notificationSender(projectId);
    } catch (Exception ex) {
      logger.error(ex.getMessage(), ex);
      throw new AllocationManagerException().setMessage(ex.getMessage() + " Stack trace:" +
ExceptionUtils.getStackTrace(ex));
    }
}
```

- The function uses the UserAllocationDetailEntityPK object as a composite key to get a record which is fetched by the UserAllocationDetailRepository object.
- The Repository files represent the database records.

The basic structure for the allocation manager is set and this will be built on for all the future implementations.

Notification Manager

The notification manager is a maven multi module project. The modules are as follows:

- Notification-Receiver - This module accepts the request from a microservice. The request sent is the Project ID. The project id is queued using the RabbitMq message broker. The project ID is accepted and forwarded to then Notification-Authenticator Module.

- Notification-Authenticator - This module has the NotificationDetails.java consists of methods which calls the functions in the AllocationManagerServerHandler which connect to database functions and fetch the necessary details.

- Notification-Sender - This module contains method which is called by the Notification-Receiver. The method sends email to the specified sender. The method reads messages to be sent to the user from a messages.properties resource file. The email server details should be stored in a config.properties file which should be stored directly under the Notification-Receiver project.

Admin and Reviewer Interface

- Created the thrift server (in the server module) and client(in the stubs module) for the reviewer and admin.
- The clients calls the updateAllocationRequestStatus() functions to record decisions of admin or client and getAllocationRequest() to find the request details. The function implementations are present in the AllocationManagerServerHandler();

Technologies used:

- RabbitMq.
- Apache Thrift.
- Maven

List of Jira worked on:

https://issues.apache.org/jira/browse/AIRAVATA-2532
https://issues.apache.org/jira/browse/AIRAVATA-2600

List of pull requests:

https://github.com/apache/airavata-sandbox/pull/22
https://github.com/apache/airavata-sandbox/pull/21
https://github.com/apache/airavata-sandbox/pull/18
https://github.com/apache/airavata-sandbox/pull/17
https://github.com/apache/airavata-sandbox/pull/16
https://github.com/apache/airavata-sandbox/pull/11
https://github.com/apache/airavata-sandbox/pull/9