# Swift Integration:

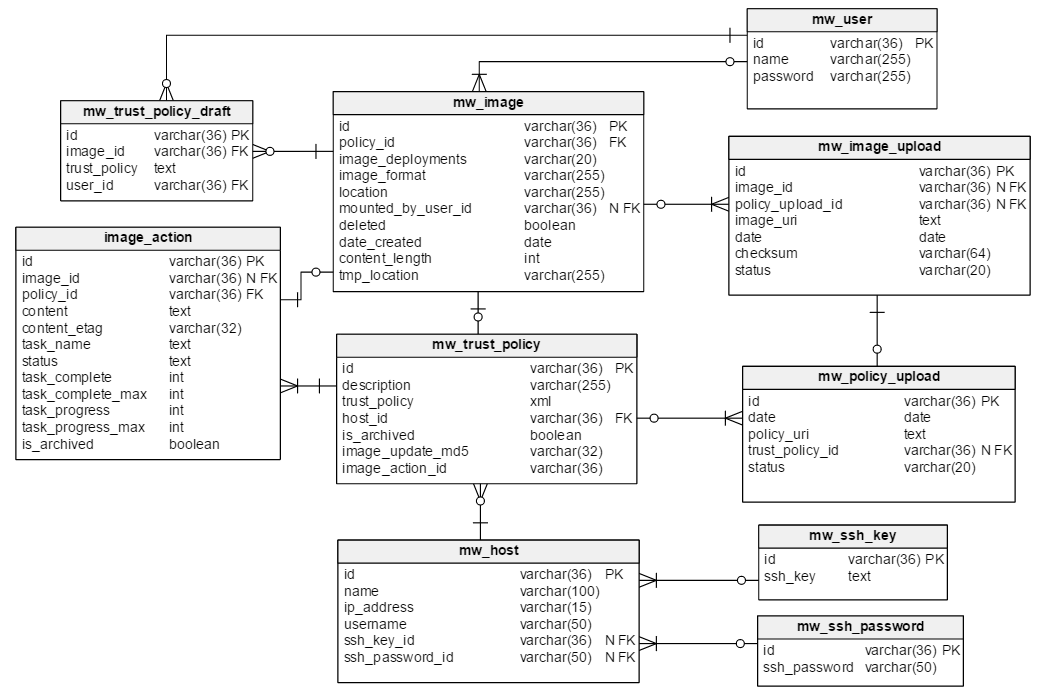
We support three usecases based on the image and policy upload options

1. Upload image only in glance
2. Upload image and policy as tar in glance
3. Upload image in glance and policy in swift

This document will discuss overall flow and database changes from trust policy creation to trust policy upload. We have to make changes in database to support certain features.

Let’s first discuss about DB changes.

Here is updated ERD for the reference



**DB changes in detail:**

mw\_image:

* tmp\_location – if policy is encryption this field contains encrypted image location. Once image is uploaded in image store, encrypted image will be deleted from the temp location and this field will be cleared.

mw\_trust\_policy:

* is\_archive: it’s a Boolean value to show whether trust policy is active or not. The default value should be false which means policy is the latest policy. When it is true, it means policy is some old version but kept for future reference.
* image\_update\_md5 – contains hash of all variables those enforce image to update in image store once policy is updated. At this moment dekUrl is the only variable that enforces image to update. If user changes encryption policy or changes dekUrl in policy, director needs to upload image again with new policy. Director should find dekUrl from associated trust policy and update hash of dek in this field.

mw\_image\_upload:

* policy\_upload\_id – maps uploaded image with policy
* tmp\_location – moved to mw\_image\_upload
* content\_length – moved to image\_action
* content\_sent – moved to image\_action

mw\_image\_action:

* id – unique UUID
* image\_id – image id associated with actions
* policy\_id – policy id for which image actions are created
* content – json array of actions to be performed on an image. Sample json is:

{ actions: [

{ task\_name: "encrypt\_image" },

{ task\_name: "create\_tar", location: "/path/to/tar" },

{ task\_name: "upload\_image", repository\_type: "glance", repository\_id: "uuid here" }

]}

* contnt\_etag – md5 of content, used to make sure multiple processes are not updating the json content at the same time and clobbering each others changes.

Anytime we update content we also update content\_etag. Adding content\_etag to where clause of update sql query will prevent from updating record if content\_etag has changed since we read it. If update fails to find record (because etag changed) query record again, merge desired changes into that json content , then try update again.

* task\_name – name of task in progress, ex. upload\_image, upload\_policy, encrypt\_image, create\_tar, etc.
* status – status of a task\_name pending, in progress, done, cancelled, error etc
* task\_complete – number of completed tasks
* task\_complete\_max – number of tasks to be completed
* task\_progress – progress of a current task
* task\_progress\_max – represents status when current task is 100% completed
* is\_archived - Boolean value represents if current image\_action is archived or not

**Policy creation work flow:**

Once user completes creating policy, policy will be moved to mw\_trust\_policy table and new policy\_id should updated in mw\_image table.

Now, if policy already exist in mw\_trust\_policy table director would move that policy to archive, that means director would set is\_archived to true and create new entry in mw\_trust\_policy table.

Before replacing policy, director must make sure that there is no pending action in the image\_action table associated with an image id (action\_count == action\_completed). If there is any pending action, director should show warning to the user that old policy is in use so can not replace new policy at this time. User may try after some time or we can create feature in director to do it automatically.

When director creates new entry in mw\_trust\_policy table if image\_update\_md5 is different from previous policy value, director should create new entry in image\_action table and define action, action\_total\_count and action\_completed. If there already exist image\_action related to the image\_action director should archive that action and create new one. At this point, there would be just one action at max which can be ‘encrypt’. If encryption policy is not set, action\_total\_count would be zero but it is important to create entry in the image\_action table to assure that preprocess actions are taken into account. Director should start preprocessing actions in background and update total\_count as each action is completed.

**Policy Upload work flow:**

When user uploads just image without policy, no extra image processing is required from director end. When user uploads image and policy both, director should add image upload related tasks in the image\_action table ex. ‘image\_tar’, ‘image\_upload’ and/or policy\_upload.

1. Upload as new policy and image:

When director creates/edits policy, if image\_update\_md5 is different from previous value, director creates new entry in image\_action table and starts preprocessing action in background, if any. Now upload action does not require to determine preprocess actions it only requires to add upload actions(‘tar’ and/or ‘upload’) in action table.

1. Upload same policy and image again:

When director uploads policy and image again, there wouldn’t be any entry in image\_action table associated with an image. At this point, upload task needs to determine preprocess actions and update action table and start all processes.

1. Replace existing policy in image store from the given list:

User should only be able to replace policy if associated image remains valid for new policy also. Let’s discuss how to generate the list of policies those are eligible for replacement. Let’s say a policy is uploaded 5 times in the image store so we would have 5 entries in the image\_upload table. Now for each 5 entries we should find respective image\_update\_md5 from trust\_policy table. We should now compare old and new upload\_variables\_md5, if both values match then we would allow user to update respective policy. Let’s say among five, three policies are qualified with above mentioned criteria then user would get list of just 3 policies as an update option.

Once upload is completed director should delete encrypted image and temp\_location from image table. Also action(entire row) associated with image is deleted from action\_table.

Once user selects upload new row is created in mw\_image\_upload and mw\_policy\_upload tables.

**More Details on mw\_image\_action table:**

When user creates policy, action \_count would be at max one. When user replaces policy we check action \_count is same as action \_completed. Here we do not need to enforce user to upload image before replacing policy so checking action \_count is same as action \_completed is sufficient. When user uploads image action\_count would increment. If user wants to replace policy while image and policy are being uploaded, action \_count would not be same as action \_completed and director would not allow to replace policy.

**Progress bar**

The dashboard widget will search for non-archived actions to display progress. Once a progress bar is displayed , it should be initialized with specific image action id. Then it can display status all the way until the action is complete or cancelled even if it's archived. When status is final progress bar will stop queries and display final status until user closes it.

**Pending image Uploads:**

On director main page one of the widget is to show list of pending images to be uploaded.

Pending image upload means images which are never uploaded to the image store. For each image director can check image ID in the mw\_image\_upload table, if that does not exist then the image should be added as pending image upload.

**Pending policy Uploads:**

On director main page, one of the widget is to show list of pending policies to be uploaded.

Pending policy upload means policy is never uploaded or not uploaded after it is edited last time. For each policy, which is not archived director can check policy ID in the mw\_policy\_upload table, if that does not exist then that policy should be added as pending policy upload.