# Trust Director API

## Add Image

Add image to director works in two steps

1. POST /images

Type: resources

This API uploads image metadata

**Input**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| Image\_name | String | Name of the image |
| image\_deployments | Object | Array of deployments supported |
| format | String | Image format |

Response:

{

“name”:”cirros-x86.img”,

”id”: 8569c040-1a0f-11e5-9fee-0002a5cca51b,

“format”:”qcow2”,

“image\_deployments”:[”VM”]

}

1. POST /uploads/{image\_id}/content

Type: rpc

Uploads actual image using image ID received in previous step

|  |  |  |
| --- | --- | --- |
| Image\_id | String | Image id received in previous step |
| image | Bytes | Image to be uploaded on server |

Response:

{

“name”:”cirros-x86.img”,

”id”: 8569c040-1a0f-11e5-9fee-0002a5cca51b,

“format”:”qcow2”,

“image\_deployments”:[”VM”]

“date\_created”: “2015-07-05”

“content\_length”: “10223616”

}

## Search Images

GET /images

Type: resources

**Input**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| image\_deployment | String | Image deployment type |

Response: returns list of image name, id and format

{“images”:[  
    {“name”:”cirros-x86.img”, ”id”: 8569c040-1a0f-11e5-9fee-0002a5cca51b, “format”:”qcow2”, “image\_deployments”:[”VM”]},  
    {“name”:”cirros-x86.vhd”, ”id”: 9569c550-9fee-11e5-9fee-0002a5d5cbac, “format”:”vhd”, “image\_deployments”:[”VM”,“Bare\_Metal”]},

{“name”:”cirros-x86.ami”, ”id”: 7569c040-1a0f-11e5-11e5-0002a5d5c44b, “format”:”ami”, “image\_deployments”:[”VM”]}}     
]}

Milestone 1 supports just qcow2

## Get image deployments

GET /image-deployments

Type: resources

Response: returns image deployments supported

{

“image\_deployments”: [

{“name”: “VM”},

{“name”: “Bare\_Metal”}

]

}

In the first version, supported application deployment type are retrieved from configuration file.

Milestone 1 supports just VM

## Get image formats

GET /image-formats

Type: resources

Response: returns list of image types supported

{

“image\_formats”: [

{“name”: “qcow2”},

{“name”: “raw”},

{“name”: “vhd”}

]

}

In the first version supported image formats are retrieved from configuration file.

Milestone 1 supports just qcow2

## Get image launch policies

GET /image-launch-policies

Type: resources

**Input(optional)**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| image\_deployments | String | Image deployment type |

Response: returns list of policies supported

{

“image-launch-policies”: [

{“name”: “measureOnly”, “image\_deployments”: [“VM”, “BM”]},

{“name”: “measureAndEnforce”, “image\_deployments”: [“VM”]},

{“name”: “encryption”, “image\_deployments”: [“VM”]}

]

}

Milestone 1 supports just measureOnly and measureAndEnforce

## MountImage

POST /images/{image-id}/mount

Type: rpc

**Input**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| id | String | Database UUID of the image that needs be mounted |

Response:

{

“id”: “1eebe380-1a36-11e5-9472-0002a5d5c51b”,

“name”: “cirros-x86.img”

“image\_deployments”: “VM,Bare\_Metal”

“image\_format”: “qcow2”

“mounted”: “true”

}

Mount path for VM and BM image is /mnt/director/images/<db\_image\_uuid> .

There would be just one mountpoint per image. If someone tries to mount same image again, it shoud throw exception.

Once image is mounted, mw\_image->mounted\_by\_user\_id database field should be updated.

Milestone 1 supports just qcow2

**Source Code:**

Take reference of MountVMImage.java (do not use it).

* Use com.intel.mtwilson.util.exec.ExecUtil from mtwilson-util to execute commands. Throw exception if exitcode is not zero
* Image must be mounted read only. (already there in Mount\_vm\_image.sh)

## Unmount Image

POST /images/{image-id}/unmount

Type: rpc

**Input**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| id | String | Database UUID of the image that needs be unmounted |

Response:

{

“id”: “1eebe380-1a36-11e5-9472-0002a5d5c51b”,

“name”: “cirros-x86.img”

“image\_deployments”: “VM,Bare\_Metal”

“image\_format”: “qcow2”

“mounted”: “false”

}

API should first check whether image is mounted by same user or not using mw\_image -> mounted\_by\_user\_id field. If not then, it should throw exception otherwise API should figure out mount point based on image Id and unmount the image.

There should be a common function to get mountpath which can be used by mount and unmount APIs.

Milestone 1 supports just qcow2

**Source Code:**

Take reference of MountVMImage.java (do not use it).

* Use com.intel.mtwilson.util.exec.ExecUtil from mtwilson-util to execute commands. Throw exception if exitcode is not zero

## Create trust policy

POST /trustpolicies

Type: resources

**Input**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| id | String | Database UUID of the image |
| trustPolicy | String (xml) | Trust policy xml |

This API should take trustpolicy schema as reference to generate trustpolicy. Schema is available in mtwilson-vmquote-xml project. Trust policy xml as input parameter should contain imageId, LaunchControlPolicy, Dir and Files.

API should retrieve Director id from user profile, create keys using KMS, calculate image hash and calculate whitelist values. Once policy is created it should be saved in the mw\_trust\_policy table.

AR: need to figure out a way to generate XML from schema in java script. For java we already have jaxb utility in place.

Milestone 1 is supports just measureOnly and measureAndEnforce policy not encryption policy

Source Code:

* Use GenerateTrustPolicy. createTrustPolicy(..).
* Replace executeShellCommand() with com.intel.mtwilson.util.exec.ExecUtil. executeShellCommand() deletes new line at the end of result which is not done by util class. So please make sure you make necessary changes.

## Search files in image

GET /images/{image-id}/search

Type: rpc

**Input**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| directory | String | Directory path |
| include | String | Include regular expression |
| exclude | String | Exclude regular expression |
| recursive | boolean | Specifies whether to go into subdirectories or not. non-recursive ( recursive: false ) would be default to show just contents of the specified directory (w/ includes or excludes) and not go into subdirectories |

Response: returns list of files that matches include and exclude filter for a specific directory. The file path is relative to directory path, it should not include leading “/”

{

“directory”:”/root”

“include”:”\*.sh”

“exclude”:””

“recursive”:true

“files”: [

{“name”: “director.sh”},

{“name”: “mtwilson.sh”},

{“name”: “scripts/uninstall.sh”}

]

}

Source Code:

* Use GenerateTrustPolicy. createWhitelist(..).
* Generate find command based on include and exclude and recursive. Recursive option was not there previously you will need to update find command accordingly.
* Replace executeShellCommand() with mtwilson-util com.intel.mtwilson.util.exec.ExecUtil.

## Sign policy using Mt Wilson

POST /trustpolicies/{trustpolicy\_id}/sign

Type: rpc

**Input**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| id | String | Database UUID of trust policy |

Client should use this API in two steps: 1) call create trust policy API which creates and saves policy in database 2) call sign policy API to sign an existing policy and update database with signed policy.

Response:

{

“id”: “9569c550-9fee-11e5-9fee-0002a5d5cbac”

“image\_id”: “5454c550-9fee-11e5-9fee-0002a5d5caaa”

“trust\_policy”: “<TrustPolicy>….</TrustPolicy>”

“host\_id”: null

}

**Source Code:**

* Use SignWithMtWilson
* This class is going to be changed. We will no longer need to add certificate in keystore.

## Upload image to image store

POST /uploads

Type: resource

Input

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| image\_id | String | Database UUID of the image |
| trust\_policy\_id | String | Database UUID of trust policy |
| image\_store\_id | String | Id of the image store |
| format | String | Options: tar, null. Default: null |

Response:

{

“id”: “9569c550-9fee-11e5-9fee-0002a5d5cbac”

“image\_id”: “5454c550-9fee-11e5-9fee-0002a5d5caaa”

“image\_uri”: “glance:http://glance.example.com/v1/images/71c675ab-d94f-49cd-a114-e12490b328d9”

“date”: “2015-06-28 01:00:00”

“tmp\_location”: null

“checksum”: “bd84446696ff953efe91735aae4d2882”

“status”: “active”

“content\_length”: “10223616”

“content\_sent”: “785564”

}

Uploading an image to a remote server is an operation that will probably take some time. The TD should not hold the client connection open while this is happening. So the response from this RPC should not wait for the operation to complete, it should immediately return a reference to the client and then perform the operation in the background. The client should then be able to query the status of the upload and eventually see that it is completed or failed. The client would create a new “upload task” object with the image id & image store information, and let the server update this “upload task” with the progress of the upload itself. The client can then query it (imagine progress bar in the UI) and when done the #bytes for image\_size == #bytes sent.

Source code:

* Use GlanceImageStoreImpl.uploadImage() as a reference

## ~~Create Tarball~~

~~POST /create-tar~~

~~Type: rpc~~

**~~Input~~**

|  |  |  |
| --- | --- | --- |
| **~~Name~~** | **~~Type~~** | **~~Description~~** |
| ~~trustPolicy~~ | ~~String~~ | ~~Whole trust policy~~ |
| ~~imageName~~ | ~~String~~ | ~~Name of the image~~ |

~~Response: location where tar ball is saved~~

~~{~~

~~“location”:”/opt/images/cirros-x86.tar”~~

~~}~~