1) Get images for display on grid

@Path("/images")

@GET

@Produces(MediaType.*APPLICATION\_JSON*)

**public** SearchImagesResponse getImagesByDeploymentType(

@QueryParam("deploymentType") String deployment\_type)

Returns list of images in TD depending on the image deployment type

supplied. Each image has an image deployment type : BareMetal or VM.

This method gets the list of images based on the deployment type provided

as a query param. Providing the deployment type is optional. If provided

the value should be VM or BareMetal . edited\_date field is updated in case of

mounting and delete. image\_upload\_status can have values Incomplete and Complete.

Since the upload happens in multiple steps, the status is Incomplete till the

sent field is equal to image\_size. image\_location is for internal server use and

it can't accessed it directly.

Sample call:

[https://<IP/HOST\_NAME>/v1/images](https://server.com:8443/v1/images)

or

https://<IP/HOST\_NAME>/v1/images?deploymentType=VM

Input: deploymentType=VM

Output: {

"images": [

{

"created\_by\_user\_id": "admin",

"created\_date": "2015-12-21",

"edited\_by\_user\_id": "admin",

"edited\_date": "2015-12-21",

"id": "465A8B27-7CC8-4A3C-BBBC-26161E3853CD",

"image\_name": "CIR1.img",

"image\_format": "qcow2",

"image\_deployments": "VM",

"image\_size": 13312,

"sent": 13312,

"deleted": false,

"trust\_policy\_id": "0e41169f-d2f3-4566-96c7-183d699417fb",

"uploads\_count": 0,

"policy\_name": "CIR1.img",

"image\_upload\_status": "Complete",

"image\_Location": "/mnt/images/"

},

{

"created\_date": "2015-12-17",

"edited\_by\_user\_id": "admin",

"edited\_date": "2015-12-17",

"id": "D7952C76-8F37-474A-B054-168ACC2C0802",

"image\_name": "10.35.35.182",

"image\_deployments": "BareMetal",

"deleted": false,

"trust\_policy\_id": "f421e8cf-8d29-40b9-b05f-6b52d549dc81",

"uploads\_count": 0,

"policy\_name": "S1",

"image\_upload\_status": "Complete"

}

}]}

Mount image with incorrect image id

{"deleted":false,"error":"No image found with id: 465A8B27-7CC8-4A3C-BBBC- 26161Es3853CD"}

2) Mount image

@Path("/v1/rpc/mount-image")

@Consumes(MediaType.*APPLICATION\_JSON*)

@Produces(MediaType.*APPLICATION\_JSON*)

@POST

**public** MountImageResponse mountImage(MountImageRequest mountImage,

@Context HttpServletRequest httpServletRequest,

@Context HttpServletResponse httpServletResponse)

Method to mount the image. This call is invoked in all types of

deployment types as we treat SSH hosts as images too. When a host is

added and entry is made in the MW\_IMAGE table, but with image\_format as

null. For images, the image format is qcow2

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

would be just one mountpoint per image. If someone tries to mount same

image again, it will throw exception. Once image is mounted,

mw\_image->mounted\_by\_user\_id database field will be updated.

This method returns a response in all scenarios, even in case of errors

while mounting the image. In the case when the user attempts to mount an

image which is already in use by another user, and error message: Unable

to mount image. Image is already in use by user: <user\_name> is thrown

and sent back in the details attribute of the response object and status

as ERROR.

In case the case of the user who has mounted the image, because of

inactivity, the session timed out; and the user logs back in. The image

will not be re-mounted.

Sample call : https://<IP/HOST\_NAME>/v1/rpc/mount-image

Input: {“id” : "465A8B27-7CC8-4A3C-BBBC-26161E3853CD"}

Output:

{

"created\_by\_user\_id": "admin",

"created\_date": 1450636200000,

"edited\_by\_user\_id": "admin",

"edited\_date": 1450685484685,

"id": "465A8B27-7CC8-4A3C-BBBC-26161E3853CD",

"image\_name": "CIR1.img",

"image\_format": "qcow2",

"image\_deployments": "VM",

"image\_size": 13312,

"sent": 13312,

"mounted\_by\_user\_id": "admin",

"deleted": false,

"image\_upload\_status": "Complete",

"image\_Location": "/mnt/images/"

}

If the user tries to mount an image which, for some reason, has been removed from the uploaded location, the response will look like :

{"error": "No image found with id: BAA5747D-B2ED-4E7D-A4D5-0256DEE7FBB1"}

If user tries to mount deleted image, the response will look like:

{"error":"Cannot launch deleted image"}

Mount image with incorrect image id:

{"id":"465A8B27-7CC8-4A3C-BBBC-26161Es3853CD","deleted":false,"error":"Error fetching image:465A8B27-7CC8-4A3C-BBBC-26161Es3853CD"}

3) Unmount image

@Path("/rpc/unmount-image")

@Consumes(MediaType.*APPLICATION\_JSON*)

@Produces(MediaType.*APPLICATION\_JSON*)

@POST

**public** UnmountImageResponse unMountImage(MountImageRequest unmountimage,

@Context HttpServletRequest httpServletRequest,

@Context HttpServletResponse httpServletResponse)

Method to unmount the mounted image.

API will first check whether image is mounted by same user or not using

mw\_image -> mounted\_by\_user\_id field. If not then, it will throw

exception. Otherwise API should figure out mount point based on image Id

and unmount the image. The default mount path is /mnt/director/UUID

As part of the unmount process, the MW\_IMAGE.mounted\_by\_user\_id field is

set to NULL again. the unmount process in the service.

Sample call: https://<IP/HOST\_NAME>/v1/rpc/unmount-image

Input: {id : "ACD7747D-79BE-43E3-BAA5-07DBEC13D272"}

Output: {

"created\_by\_user\_id": "admin",

"created\_date": 1450636200000,

"edited\_by\_user\_id": "admin",

"edited\_date": 1450685543811,

"id": "465A8B27-7CC8-4A3C-BBBC-26161E3853CD",

"image\_name": "CIR1.img",

"image\_format": "qcow2",

"image\_deployments": "VM",

"image\_size": 13312,

"sent": 13312,

"deleted": false,

"image\_upload\_status": "success",

"image\_Location": "/mnt/images/"

}

In case of error:

Output: {"status":"Error", details:"<Cause of error>"}

4) Edit policy draft from the UI via the file tree

@Path("trust-policy-drafts/{trustPolicyDraftId: [0-9a-zA-Z\_-]+}")

@POST

@Consumes(MediaType.*APPLICATION\_JSON*)

@Produces(MediaType.*APPLICATION\_JSON*)

**public** TrustPolicyDraft editPolicyDraft(

@PathParam("trustPolicyDraftId") String trustPolicyDraftId,

TrustPolicyDraftEditRequest trustPolicyDraftEditRequest)

Update the policy draft by applying the patch. Whenever an user selects

directories on the UI, a patch of the changes made to the selections is

sent to the server every 10 mins. this method does the job of merging the

patch/delta into the trust policy draft that is stored in the database.

A sample patch looks like this: <patch> <add sel="<node selector>" ><File

path="<file path>"></add> <remove sel="<node selector>" ></remove>

</patch>

We use https://github.com/dnault/xml-patch library to apply patches.

the method returns the patched trust policy as a response to successful

patch application In case of error, a DirectorException is thrown. It is

caught in the failure section of the ajax call and shown as a pop up

message.

Sample call:

https://<IP/HOST\_NAME>/v1/trust-policy-drafts/97c4b9d2-d0e6-42b5-a4e2-1642b01db21f

Method:-POST

Input: UUID of the image in path

{“patch”:“<patch>...</patch>”}

Patch examples:-

i)Selecting initrd.img file:-

{"patch":"<patch><add pos=\"prepend\" sel='//\*[local-name()=\"Whitelist\"]'><File Path=\"/initrd.img\"/></add></patch>"}

ii)Applying regex on boot folder :-

Include filter:- \*.gz

{"patch":"<patch><add sel='//\*[local-name()=\"Whitelist\"]'><Dir Path=\"/boot\" Include=\"\*.gz\" Exclude=\"\" Recursive=\"false\"/></add><add pos=\"after\" sel='//\*[local-name()=\"Whitelist\"]/\*[local-name()=\"Dir\"][@Path=\"/boot\"]'><File Path=\"/boot/tboot.gz\"/></add></patch>"}}

Output:

{

"created\_by\_user\_id": "admin",

"created\_date": "2015-12-21",

"edited\_by\_user\_id": "admin",

"edited\_date": 1450686737378,

"id": "97c4b9d2-d0e6-42b5-a4e2-1642b01db21f",

"trust\_policy\_draft": "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<TrustPolicy xmlns=\"mtwilson:trustdirector:policy:1.1\" xmlns:ns2=\"http://www.w3.org/2000/09/xmldsig#\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\">\n <Director>\n <CustomerId>testId</CustomerId>\n </Director>\n <Image>\n <ImageId>5A56C717-EF21-4A6E-8701-08EE0FF4C620</ImageId>\n <ImageHash>6a749cfc57df53bd36b6ce1cb67c54632be4bc904d084ddd8dbeb79a30f44ccc</ImageHash>\n </Image>\n <LaunchControlPolicy>MeasureOnly</LaunchControlPolicy>\n <Encryption>\n <Key URL=\"uri\">http://10.35.35.53/v1/keys/013890ed-9af3-4d87-bfe7-26a7f6605ee3/transfer</Key>\n <Checksum DigestAlg=\"md5\">cd5b437a55ae166aecd8c2fe2cf4ce29</Checksum>\n </Encryption>\n <Whitelist DigestAlg=\"sha256\">\n <File Path=\"/boot/grub/menu.lst\" />\n <File Path=\"/boot/vmlinuz-3.2.0-37-virtual\" />\n <File Path=\"/boot/config-3.2.0-37-virtual\" />\n <File Path=\"/boot/initrd.img-3.2.0-37-virtual\" />\n <File Path=\"/boot/grub/stage1\" />\n <File Path=\"/boot/grub/e2fs\_stage1\_5\" />\n <File Path=\"/boot/grub/stage2\" />\n </Whitelist>\n</TrustPolicy>\n",

"display\_name": "CIRROS\_ETE\_1.img"

}

In case of error:

Output: {"status":"Error", details:"<Cause of error>"}

5) Depending on user operations, get files to be displayed in the tree

@Path("images/{imageId: [0-9a-zA-Z\_-]+}/search")

@GET

@Produces(MediaType.*APPLICATION\_JSON*)

**public** SearchFilesInImageResponse searchFilesInImage(

@PathParam("imageId") String imageId, @Context UriInfo uriInfo)

Method called by the tree on Wizard 2/2 screen to find the files in the

mounted image

This method is invoked as an ajax call from the tree component of the

policy wizard. All the directory selections, viewing of child nodes,

application of regex is handled by this method.

the SearchFilesInImageRequest object contains all the flags to support

above functionalities. For example, in case of regex application, the

dir, include, exclude, include\_recursive attributes are utilized.

In case of regex reset, dir, reset\_regex is utilized.

This method expects a policy draft to be existing when the user comes to

this screen. We create a default empty policy draft when a user starts

creating one. We just keep on modifying the Whitelist tag of the xml.

The UI library for the tree also sends an "init" parameter when its

loaded the first time. This indicates that the tree needs to pick up he

current selections that might be made by the users and pre-check the tree

items. The init method reads the existing policy draft and creates a list

of files already selected earlier.

Depending on the user actions and the corresponding attributed in the

SearchFilesInImageRequest object, the DirectoryAndFileUtil class methods

are invoked to find the files and dirs inside the directory of interest.

In cases of regex and "select all" operation where user clicks on the

checkbox next to a directory in order to select all the contents, this

method created a patch as a list of strings and sends it in the

"patch\_xml" attribute. Once the UI receives it, it adds it to the current

selections on the UI and then sends back the consolidated patch to the

server

Sample call:

https://<IP/HOST\_NAME>/v1/images/08EB37D7-2678-495D-B485-59233EB51996/search

Input: QueryPAram : dir=/boot/&recursive=false&files\_for\_policy=false&init=false&include\_recursive=false&reset\_regex=false

Example Input:-

https://10.35.35.133/v1/images/E4770A39-024D-4A2A-989E-7EE1123E8204/search?dir=/&recursive=false&files\_for\_policy=false&init=true&include\_recursive=false&reset\_regex=false

output: {"tree\_content":"<Html containing the nested ul and li tags>", "patch\_xml":"<patch><list of add remove tags as per the operation></patch>"}

The output tag has the patch\_xml set only in case in the following cases of the query parameters:

1) recursive=true and files\_for\_policy=true

2) recursive=true and files\_for\_policy=false

3) reset\_regex = true

4) include="<regex expression>" & exclude="<regex expression>" with optional include\_regex=true

6) Get configured Image Deployments

@Path("/v1/image-deployments")

@GET

@Consumes(MediaType.*APPLICATION\_JSON*)

@Produces(MediaType.*APPLICATION\_JSON*)

**public** ListImageDeploymentsResponse getImageDeployments()

Retrieves list of deployment types - VM and BareMetal are the types

returned as JSON

Sample call:

https://<IP/HOST\_NAME>/v1/image-deployments

Input: None

Output: {

"image\_deployments": [

{

"name": "VM",

"display\_name": "Virtualized Server"

},

{

"name": "BareMetal",

"display\_name": "Non-Virtualized Server"

}

]

}

7) Get configured image formats

@Path("/v1/image-formats")

@GET

@Produces(MediaType.*APPLICATION\_JSON*)

**public** ListImageFormatsResponse getImageFormats()

Lookup method to fetch the image formats. Currently we return qcow2, vhd, vmdk, raw, vdi as JSON

Sample call:

https://<IP/HOST\_NAME>/v1/image-formats

Input: None

Output: {"image\_formats": [{"name": "qcow2","display\_name": "qcow2"},{"name": "vhd","display\_name": "vhd"}

\* ,{"name": "vmdk","display\_name": "vmdk"},{"name": "raw","display\_name": "raw"},{"name": "vdi","display\_name": "vdi"}]}

8) Get configured launch policies

@Path("/v1/image-launch-policies")

@GET

@Consumes(MediaType.*APPLICATION\_JSON*)

@Produces(MediaType.*APPLICATION\_JSON*)

**public** ListImageLaunchPolicyResponse getImageLaunchPoliciesList ( @QueryParam("deploymentType") String deploymentType)

lookup method to fetch the launch policies. The current launch policies

that are returned are MeasureOnly and MeasureAndEnforce If not

deploymentType is provided, all image launch policies are returned. I the

user wants to fetch launch specific to a deployment type, the possible

values for deployment type are: 1) VM 2) BareMetal

Sample call:

https://<IP/HOST\_NAME>/v1/image-launch-policies

Input: QueryParam String deploymentType=VM

deploymentType can be VM , BareMetal or Docker

Output:

{

"image\_launch\_policies": [

{

"name": "MeasureOnly",

"display\_name": "Hash Only",

"image\_deployments": [

"VM",

"BareMetal"

]

},

{

"name": "MeasureAndEnforce",

"display\_name": "Hash and enforce",

"image\_deployments": [

"VM"

]

},

{

"name": "encrypted",

"display\_name": "Encryption",

"image\_deployments": [

"VM"

]

}

]

}

9) Get current policy metadata for a policy

@Path("trust-policy-drafts/{trustPolicyDraftId:[0-9a-zA-Z\_-]+ }")

@GET

@Produces(MediaType.*APPLICATION\_JSON*)

**public** Response getPolicyDraft(

@PathParam("trustPolicyDraftId") String trustPolicyDraftId)

Retrieves trust policy draft based on uuid provided

Sample call

https://<IP/HOST\_NAME>/v1/trust-policy-drafts/ACD7747D-79BE-43E3-BAA5-7DBEC13D272

Input: PathParam String: trustPolicyDraftId=ACD7747D-79BE-43E3-BAA5-7DBEC13D272

Output: {

"launch\_control\_policy":"MeasureAndEnforce",

"encrypted":true,

"image\_name":"cirrus\_1811.img",

"display\_name":"111"

"trust\_policy": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><TrustPolicy xmlns=\"mtwilson:trustdirector:policy:1.1\" xmlns:ns2=\"http://www.w3.org/2000/09/xmldsig#\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"><Director><CustomerId>testId</CustomerId></Director><Image><ImageId>05EECBC5-C8BA-4523-A891-7AF455FAFAAB</ImageId></Image><LaunchControlPolicy>MeasureAndEnforce</LaunchControlPolicy><Whitelist DigestAlg=\"sha256\"><File Path=\"/lib/modules/3.2.0-37-virtual/modules.isapnpmap\" /> </Whitelist></TrustPolicy>"

}

If no draft exist for corresponding uuid HTTP 404 NOT Found is returned

10) Create policy after selecting files and dirs

@Path("rpc/finalize-trust-policy-draft")

@Consumes(MediaType.*APPLICATION\_JSON*)

@Produces(MediaType.*APPLICATION\_JSON*)

@POST

**public** CreateTrustPolicyResponse createTrustPolicy(

CreateTrustPolicyMetaDataRequest createPolicyRequest)

When the user has finished selecting files and dirs and clicks on the

Next button for creating a policy, we call this method to :

1) Sign with MTW

2) Generate Hashes

Sample call:

https://<IP/HOST\_NAME>/v1/rpc/finalize-trust-policy-draft

Input

{"trust\_policy\_draft\_id":"<UUID of trust policy draft>"}

In case of a success, the response would be :

{"id":"14767a34-b5a4-4f84-be7a-7604670fe8b5"}

id returned in case of success response is id of trust policy created by this call by signing draft and generating hashes.

In case of error where signing with MTW fails:

{"error":"Unable to sign the policy with MTW"}.

11)

@Path("trust-policy-drafts")

@Consumes(MediaType.*APPLICATION\_JSON*)

@Produces(MediaType.*APPLICATION\_JSON*)

@POST

**public** CreateTrustPolicyMetaDataResponse createTrustPolicyDraft(

CreateTrustPolicyMetaDataRequest createTrustPolicyMetaDataRequest)

Creates an initial trust policy draft for image.

image\_id, display\_name and launch\_control\_policy are mandatory fields

Sample call:

https://<IP/HOST\_NAME>/v1/trust-policy-drafts

Input: {"image\_id":"08EB37D7-2678-495D-B485-59233EB51996","image\_name":"cirrus\_1811.img","display\_name":"cirrus\_1811.img","launch\_control\_policy":"MeasureOnly","encrypted":false}

Output: {"id":"50022e9c-577a-4bbd-9445-197a3e1a349f","trust\_policy":"<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>\n<TrustPolicy xmlns:ns2=\"http://www.w3.org/2000/09/xmldsig#\" xmlns=\"mtwilson:trustdirector:policy:1.1\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\">\n <Director>\n <CustomerId>testId</CustomerId>\n </Director>\n <Image>\n <ImageId>08EB37D7-2678-495D-B485-59233EB51996</ImageId>\n <ImageHash>6413fccb72e36d2cd4b20efb5b5fe1be916ab60f0fe1d7e2aab1a2170be1ff40</ImageHash>\n </Image>\n <LaunchControlPolicy>MeasureOnly</LaunchControlPolicy>\n <Whitelist DigestAlg=\"sha256\">\n <File Path=\"/boot/grub/stage1\"></File>\n <File Path=\"/boot/grub/menu.lst\"></File>\n <File Path=\"/initrd.img\"></File>\n <File Path=\"/boot/vmlinuz-3.2.0-37-virtual\"></File>\n <File Path=\"/boot/config-3.2.0-37-virtual\"></File>\n <File Path=\"/boot/initrd.img-3.2.0-37-virtual\"></File>\n <File Path=\"/boot/grub/e2fs\_stage1\_5\"></File>\n <File Path=\"/boot/grub/stage2\"></File>\n </Whitelist>\n</TrustPolicy>\n"}

12) Import policy template

@Path("rpc/apply-trust-policy-template/")

@Consumes(MediaType.*APPLICATION\_JSON*)

@Produces(MediaType.*APPLICATION\_JSON*)

@POST

**public** ImportPolicyTemplateResponse importPolicyTemplate(GenericRequest req) {

This call is made during policy create flow for Live host. We have

templates defined in the database. Depending on the type of the live host

(with vrtm installed or not), a certain template is picked and applied

during creating a new blank policy draft.

Sample call:

https://<IP/HOST\_NAME>/v1/rpc/apply-trust-policy-template

Input: {"image\_id":"08EB37D7-2678-495D-B485-59233EB51996"}

Output: In case of success :: {"trust\_policy":"<policy xml>"}

In case of error::

{"error":"No image found during import of policy"}

details attribute in output describes the error. If the image id provided does not exist, an error

"No image found during import of policy" would be returned in the details.

13) Recreate draft in case user after creating a policy, decides to edit it again as part of wizard flow.

@Path("rpc/create-draft-from-policy")

@POST

@Produces(MediaType.*APPLICATION\_JSON*)

**public** TrustPolicyDraft createPolicyDraftFromPolicy(GenericRequest req)

**throws** DirectorException

After the user has finalized the list of files and dirs and created a

policy, if he chooses to revisit the files/dirs selection we need to

recreate the policy draft. This method creates draft from existing policy.

Unlike POST trust-policy-drafts method it does not accepts launch

control policy , encrytion details.

Sample call:

https://<IP/HOST\_NAME>/v1/rpc/create-draft-from-policy

Input: {"image\_id":"08EB37D7-2678-495D-B485-59233EB51996"}

Output: {"id":"<UUID of Policy draft>", "trust\_policy\_draft":"<XML representation of policy>", "display\_name":"<name provided by user for the policy>", "image\_attributes

":"{"id":"<UUID of image>", "image\_format":"qcow2", ..... }"}

14) Download trust policy

@Path("images/{imageId: [0-9a-zA-Z\_-]+}/downloads/policy")

@GET

@Produces(MediaType.*APPLICATION\_XML*)

**public** Response downloadPolicyForImageId(

@PathParam("imageId") String imageId)

Method lets the user download the policy from the grids page. The user

can visit the grid any time and download the policy. This method looks

into the MW\_TRUST\_POLICY table and gets the policy string and sends it as

an xml content to the user.

In case the policy is not found for the image id, HTTP 404 is returned

Sample call:

https://<IP/HOST\_NAME>/v1/images/08EB37D7-2678-495D-B485-59233EB51996/downloads/policy

Input: Image id as path param

Output: Content sent as stream

15) Download image

@Path("/v1/images/{imageId: [0-9a-zA-Z\_-]+}/downloadImage")

@GET

@Produces(MediaType.*APPLICATION\_XML*)

**public** Response downloadImage(@PathParam("imageId") String imageId,

@QueryParam("modified") **boolean** isModified)

Method that downloads the BM image which has been modified to push the

trust policy in the /boot/trust folder. The user, on the third step of

the wizard, gets a link which downloads the modified image

NOT USED IN CURRENT VERSION

16) Download the trust policy and manifest as a tarball for live hosts

@Path("images/{imageId: [0-9a-zA-Z\_-]+}/downloads/policyAndManifest")

@GET

@Produces(MediaType.*APPLICATION\_OCTET\_STREAM*)

**public** Response downloadPolicyAndManifestForImageId(

@PathParam("imageId") **final** String imageId)

Method lets the user download the policy and manifest as a tarball from the grids page. The user can visit the grid any time and download the policy and manifest as it was created in the wizrd. This method looks into the MW\_TRUST\_POLICY table and gets the policy string, creates a manifest and sends it as an tarball content to the user.

In case the policy is not found for the image id, HTTP 404 is returned

Sample call:

https://<IP/HOST\_NAME>/v1/images/08EB37D7-2678-495D-B485-59233EB51996/downloads/policyAndManifest

Input: Image UUID

Output: Content of tarball as stream

17) Delete policy draft

@Path("trust-policy-drafts/{trustPolicyDraftId: [0-9a-zA-Z\_-]+}")

@DELETE

@Produces(MediaType.*APPLICATION\_JSON*)

**public** GenericResponse deletePolicyDraft(

@PathParam("trustPolicyDraftId") String trustPolicyDraftId) {

Delete the trust policy draft by the provided ID

Sample call:

https://{IP/HOST\_NAME}/v1/trust-policy-drafts/04EB37D7-2678-495D-B485-59233EB51933

Input: UUID of the policy draft to be deleted

Output: In case of successful deletion:

{"deleted":true}

In case of error:

{"deleted":false; "error":"Error in deleting policy draft"}

18)

@Path("trust-policy/{trustPolicyId: [0-9a-zA-Z\_-]+}")

@DELETE

@Produces(MediaType.*APPLICATION\_JSON*)

**public** GenericResponse deletePolicy(

@PathParam("trustPolicyId") String trustPolicyId)

Deletes the signed trust policy by the provided id

Sample call:

https://<IP/HOST\_NAME>/v1/trust-policy/08EB37D7-2678-495D-B485-59233EB51996

Input: UUID of the policy to be deleted

Output: In case of successful deletion:

{"deleted":true}

In case of error:

{"deleted":false; "error":"Error in deleting trust policy: <UUID>"}

19)

@Path("images")

@POST

@Consumes(MediaType.*APPLICATION\_JSON*)

@Produces(MediaType.*APPLICATION\_JSON*)

**public** TrustDirectorImageUploadResponse createUploadImageMetadata(

TrustDirectorImageUploadRequest uploadRequest)

**throws** DirectorException

API for uploading image metadata like image format, deployment type(VM,BareMetal, Docker), image file name, image size, etc. Creates image upload metadata with specified parameters and returns metadata along with image id.

Sample call:

https://{IP/HOST\_NAME}/v1/images

Input: {"image\_name":"test.img","image\_deployments":"VM","image\_format": "qcow2", "image\_size":202354}

Output: {"created\_by\_user\_id":"admin","created\_date":1446801301639,"edited\_by\_user\_id":"admin",

"edited\_date":1446801301639,"id":"B79EDFE9-4690-42B7-B4F0-71C53E36368C"

,”image\_name":"test.img","image\_format":"qcow2","image\_deployments":"VM","status":"In Progress","image\_size":407552,"sent":0,"deleted":false,"location":"/mnt/images/"}

In Case of error such as image name already exists on the server :

{

"status" : "Error",

"details" : "Image with Same Name already exists, choose a different name"

}

20) Upload image content

@Path("/v1/rpc/images/content/{image\_id: [0-9a-zA-Z\_-]+}")

@POST

@Consumes(MediaType.*APPLICATION\_OCTET\_STREAM*)

@Produces(MediaType.*APPLICATION\_JSON*)

**public** TrustDirectorImageUploadResponse uploadImageToTrustDirector(

@PathParam("image\_id") String image\_id,

InputStream filInputStream)

API for uploading image data for the given image id. Before Uploading image it is divided in chunks and sent to server one by one. Once the chunk is received location to save image is retrieved from DB using given image id and chunk is saved to that location.

Sample call:

https://{IP/HOST\_NAME}/v1/rpc/images/content/B79EDFE9-4690-42B7-B4F0-71C53E36368C

Input: chunk for image upload

Output: {"created\_by\_user\_id":"admin","created\_date":1446801301639,"edited\_by\_user\_id":"admin",

"edited\_date":1446801301639,"id":"B79EDFE9-4690-42B7-B4F0-71C53E36368C","name":"test.img",

"image\_format":"qcow2","image\_deployments":"VM","status":"Complete","image\_size":407552,"sent":407552,"deleted":false,"location":"/mnt/images/"}

While the image upload is in progress:

{"created\_by\_user\_id":"admin","created\_date":1446801301639,"edited\_by\_user\_id":"admin",

"edited\_date":1446801301639,"id":"B79EDFE9-4690-42B7-B4F0-71C53E36368C","name":"test.img",

"image\_format":"qcow2","image\_deployments":"VM","status":"In Progress","image\_size":407552,

"sent":407552,"deleted":false,"location":"/mnt/images/"}

21) Delete image

@Path("/v1/images/{imageId: [0-9a-zA-Z\_-]+}")

@DELETE

@Produces(MediaType.*TEXT\_PLAIN*)

**public** String deleteImage(

@PathParam("imageId") String imageId)

Mark image as deleted. We turn the deleted flag=true in the MW\_IMAGE table

Sample call

https://{IP/HOST\_NAME}/v1/images/08EB37D7-2678-495D-B485-59233EB51996

Method:DELETE

Input: pass the UUID of the image as path param

Output: {"deleted": true}

In case of error:

{"deleted": false , "error":"Error in deleteImage"}

22)

@Path("trust-policies/{trustPolicyId: [0-9a-zA-Z\_-]+}")

@Consumes(MediaType.*APPLICATION\_JSON*)

@Produces(MediaType.*APPLICATION\_JSON*)

@POST

**public** MonitorStatus updateTrustPolicy(@PathParam("trust\_policy\_id") String trust\_policy\_id, UpdateTrustPolicyRequest updateTrustPolicyRequest)

This method is just used to update the trust policy display name.

On the step 3/3 of the wizard for VM, when the user clicks on the “Upload now” button, we accept the last moment changes in the name of the policy and update it.

Sample call:

https://{IP/HOST\_NAME}/v1/trust-policies/d80ce469-39fd-4940-bb67-c0573551ce4c2

Input: UUID of trust policy in path {"display\_name":"policy\_renamed"}

Output:

In case of success : {"status":"success","deleted":false}

In case of error such as policy name already exists:

{ "error" : "Policy Name Already Exists","deleted":false }

23) Get list of configured image stores

@Path("/v1/imagestores")

@GET

@Produces(MediaType.*APPLICATION\_JSON*)

**public** GetImageStoresResponse getImageStores()

in case of multiple configured image stores, we would show the list on the 3/3 step of the wizard.

https://{IP/HOST\_NAME}/v1/image-stores

Input: NA

Output: {“image\_stores”:[“Glance\_1”, “Glance\_2”]}

NOT USED IN CURRENT VERSION

24) Fetch the image-action data for given action\_id

@Path("/v1/image-actions/{action\_id: [0-9a-zA-Z\_-]+}")

@Consumes(MediaType.*APPLICATION\_JSON*)

@Produces(MediaType.*APPLICATION\_JSON*)

@GET

**public** ImageActionObject fetchImageAction(@PathParam("action\_id") String action\_id)

This method will fetch an image-action which has actionId on which

actions are performed, list of actions to be performed, etc. if

actionId do not exist it give 404 Not Found.

action\_count and action\_completed are provided for convenience. These two

attributes are guaranteed to have the right data corresponding to the

actions collection. action size - is the data uploaded, action\_size\_max

- total size of the image uploaded

The tasks in "action" attribute are processed sequentially. The current

task holds the name of the task in the "action" array that is currently

being processed.

https://{IP/HOST\_NAME}/v1/image-actions

Input: action\_id = CF0A8FA3-F73E-41E9-8421-112FB22BB057

Output: {

"id": "CF0A8FA3-F73E-41E9-8421-112FB22BB057",

"image\_id": "08EB37D7-2678-495D-B485-59233EB51996",

"action\_count": 2,

"action\_completed": 2,

"action\_size": 66570,

"action\_size\_max": 66570,

"action": [ { "status": "Complete","task\_name": "Create Tar"},

"status": "Complete", "storename": "Glance", "task\_name": "Upload Tar" }],

"current\_task\_status": "Complete",

"current\_task\_name": "Upload Tar" }

{

"id": "CF0A8FA3-F73E-41E9-8421-112FB22BB057",

"image\_id": "08EB37D7-2678-495D-B485-59233EB51996",

"action\_count": 2,

"action\_completed": 1,

"action\_size": 66570,

"action\_size\_max": 66570,

"action": [ { "status": "Complete","task\_name": "Create Tar"},

"status": "In Progress", "storename": "Glance", "task\_name": "Upload Tar" }],

"current\_task\_status": "Complete",

"current\_task\_name": "Create Tar"

}

In case of error creating tar :

{

"id": "CF0A8FA3-F73E-41E9-8421-112FB22BB057",

"image\_id": "08EB37D7-2678-495D-B485-59233EB51996",

"action\_count": 2,

"action\_completed": 0,

"action\_size": 66570,

"action\_size\_max": 66570,

"action": [ { "status": "Error","task\_name": "Create Tar","error":"Error Creating tar"},

"status": "Incomplete", "storename": "Glance", "task\_name": "Upload Tar" }],

"current\_task\_status": "Error : Error Creating tar ",

"current\_task\_name": "Create Tar" }

In case of error uploading tar :

{

"id": "CF0A8FA3-F73E-41E9-8421-112FB22BB057",

"image\_id": "08EB37D7-2678-495D-B485-59233EB51996",

"action\_count": 2,

"action\_completed": 1,

"action\_size": 66570,

"action\_size\_max": 66570,

"action": [ { "status": "Complete","task\_name": "Create Tar"},

"status": "Error", "storename": "Glance", "task\_name": "Upload Tar","error":"Error Uploading tar"}],

"current\_task\_status": "Error : Error Uploading tar ",

"current\_task\_name": "Upload Tar" }

25) Create the image-action data for given data

@Path("/v1/image-actions")

@Consumes(MediaType.*APPLICATION\_JSON*)

@Produces(MediaType.*APPLICATION\_JSON*)

@POST

**public** Response createImageAction(ImageActionRequest imageActionRequest)

This method will create an image-action. Data required by this method is image\_id and artifacts and store id user want to upload. Output contains actionId(id),image\_id, actions etc.

Sample call:

https://<IP/HOST\_NAME>/v1/image-actions

Input: {"artifact\_store\_list":[{"artifact\_name":"Tarball","image\_store\_id":"78D1FF99-7412-4AA6-8351-8FD6902412CB"}],"image\_id":"64E8AFCC-182F-42C9-8A7B-42AD3C93EDCF"}

Output:

{"id":"30869EF3-9809-48F6-AC36-21994318313F",

"image\_id":"64E8AFCC-182F-42C9-8A7B-42AD3C93EDCF",

"action\_count":3,

"action\_completed":0,

"action\_size":0,

"Action\_size\_max":0,

"actions":[{"status":"Incomplete","task\_name":"Recreate Policy"}, {"status":"Incomplete", "task\_name":"Create Tar"}, {"status":"Incomplete", "task\_name":"Upload Tar", "store\_id":"78D1FF99-7412-4AA6-8351-8FD6902412CB"}],

"current\_task\_status":"Incomplete",

"current\_task\_name":"Recreate Policy",

"created\_date\_time":1458630019513,

"deleted":false

}

26) Delete the image-action data for given data

@Path("image-actions/{actionId: [0-9a-zA-Z\_-]+}")

@Produces(MediaType.*APPLICATION\_JSON*)

@DELETE

**public** ImageActionResponse deleteImageAction(

@PathParam("actionId") String actionId)

This method will delete existing image-action. Data required by this method is action\_id. Output will contain status of delete task initated.

Sample call:

https://<IP/HOST\_NAME>/v1/image-actions

Input: PathParam = actionId : CF0A8FA3-F73E-41E9-8421-112FB22BB057

Output: {"deleted":true,"action\_count":0,"action\_completed":0,"action\_size":0,"action\_size\_max":0}

27) Get all trust policy drafts

@Path("trust-policy-drafts")

@GET

@Produces(MediaType.*APPLICATION\_JSON*)

**public** ListTrustPolicyDrafts getPolicyDraftList() **throws** DirectorException

Sample call:

https://<IP/HOST\_NAME>/v1/trust-policy-drafts

Input: NA

Output:

{

"trust\_policy\_drafts": [

{

"created\_by\_user\_id": "admin",

"created\_date": "2015-12-29",

"edited\_by\_user\_id": "admin",

"edited\_date": "2015-12-29",

"id": "ac3044b0-e842-4fc1-a4b6-8b41a8be9b66",

"trust\_policy\_draft": "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?><TrustPolicy xmlns:ns2=\"http://www.w3.org/2000/09/xmldsig#\" xmlns=\"mtwilson:trustdirector:policy:1.1\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"><Director><CustomerId>testId</CustomerId></Director><Image><ImageId>4439C209-6CD8-40DA-801B-E91DA0D3E639</ImageId><ImageHash>427b8344669ccd10697225768b2bbc9d9cad0517440f358112f8b6e8118c7ba4</ImageHash></Image><LaunchControlPolicy>MeasureAndEnforce</LaunchControlPolicy><Whitelist DigestAlg=\"sha256\"><File Path=\"/boot/grub/menu.lst\"></File></Whitelist></TrustPolicy>",

"display\_name": "cirros-pme",

"image\_attributes": {

"created\_by\_user\_id": "admin",

"created\_date": "2015-12-17",

"edited\_by\_user\_id": "admin",

"edited\_date": "2015-12-29",

"id": "4439C209-6CD8-40DA-801B-E91DA0D3E639",

"image\_name": "cirros-gauri",

"image\_format": "qcow2",

"image\_deployments": "VM",

"image\_size": 12976,

"sent": 12976,

"deleted": false,

"image\_upload\_status": "Complete",

"image\_Location": "/mnt/images/"

}

},

{

"created\_by\_user\_id": "admin",

"created\_date": "2015-12-17",

"edited\_by\_user\_id": "admin",

"edited\_date": "2015-12-29",

"id": "5b287b2d-c8c2-4550-9d8e-6154b4e41b1f",

"trust\_policy\_draft": "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?><TrustPolicy xmlns:ns2=\"http://www.w3.org/2000/09/xmldsig#\" xmlns=\"mtwilson:trustdirector:policy:1.1\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"><Director><CustomerId>testId</CustomerId></Director><Image><ImageId>BD14A8F7-4423-413E-B12E-13E9EBC37925</ImageId><ImageHash>6a749cfc57df53bd36b6ce1cb67c54632be4bc904d084ddd8dbeb79a30f44ccc</ImageHash> </Image> <LaunchControlPolicy>MeasureOnly</LaunchControlPolicy><Whitelist DigestAlg=\"sha256\"><File Path=\"/boot/grub/stage2\"></File></Whitelist></TrustPolicy>",

"display\_name": "cirrus.img",

"image\_attributes": {

"created\_by\_user\_id": "admin",

"created\_date": "2015-12-16",

"edited\_by\_user\_id": "admin",

"edited\_date": "2015-12-16",

"id": "BD14A8F7-4423-413E-B12E-13E9EBC37925",

"image\_name": "cirrus.img",

"image\_format": "qcow2",

"image\_deployments": "VM",

"image\_size": 13312,

"sent": 13312,

"deleted": true,

"image\_upload\_status": "Complete",

"image\_Location": "/mnt/images/"

}

}

]

}

27) Add host for non-virtualized hosts

@POST

@Path("images/host")

@Produces(MediaType.*APPLICATION\_JSON*)

@Consumes(MediaType.*APPLICATION\_JSON*)

**public** SshSettingResponse addHost(SshSettingRequest sshSettingRequest)

**throws** DirectorException

This method adds the host related details provided by the user. The

connection details are used to verify whether connection can be

established with the remote host.

Sample call:

https://{IP/HOST\_NAME}/v1/images/host

Input: {"policy\_name":"Host\_1","ip\_address":"10.35.35.182","username":"admin","password":"password","image\_id":"","name":"10.35.35.182"}

Output: {"deleted":false,"ip\_address":"10.35.35.182","username":"root","image\_name":"10.35.35.182","image\_id":"FAA5AA92-5872-44CD-BBF4-AD3EFB61D7C9"}

In case of error:

Input: {"policy\_name":"Host\_1","ip\_address":"","username":"admin","password":"password","image\_id":"","name":"10.35.35.182"}

Lets say the user does not provide the IP:

{

"error": "No Ip address provided",

"deleted": false

}

In case of any back end error, the error would contain the error occurred at the backed.

28) Update remote host details

@PUT

@Path("images/host")

@Produces(MediaType.*APPLICATION\_JSON*)

@Consumes(MediaType.*APPLICATION\_JSON*)

**public** SshSettingResponse updateHost(SshSettingRequest sshSettingRequest)

This method updates the host related details provided by the user. The

connection details are used to verify whether connection can be

established with the remote host. This call acts similar to the POST

call, only difference being it expects an image id for update. If that is

not provided the method returns an error.

https://{IP/HOST\_NAME}/v1/images/host

Input: {"policy\_name":"Host\_1","ip\_address":"10.35.35.182","username":"admin","password":"password","image\_id":"FAA5AA92-5872-44CD-BBF4-AD3EFB61D7C9","name":"10.35.35.182"}

Output:

{

"deleted": false,

"ip\_address": "10.35.35.182",

"username": "root",

"image\_name": "10.35.35.182",

"image\_id": "FAA5AA92-5872-44CD-BBF4-AD3EFB61D7C9"

}

In case of error:

Input: {"policy\_name":"Host\_1","ip\_address":"","username":"admin","password":"password","image\_id":"","name":"10.35.35.182"}

Lets say the user does not provide the correct details to connect to the remote host :

{

"error": "Unable to connect to remote host",

"deleted": false

}

In case of any back end error, the error would contain the error occurred at the backed.

29) Setup Docker Image

@Path("/rpc/docker-setup/{image\_id: [0-9a-zA-Z\_-]+}")

@POST

@Produces(MediaType.*APPLICATION\_JSON*)

@Consumes(MediaType.*APPLICATION\_JSON*)

**public** Response dockerSetup(@PathParam("image\_id") String image\_id)

This method sets up docker image uploaded manually for given image\_id for further operation provided that deployment\_type of image must be 'Docker' and repo tag should be given before. This is mandatory step before performing any further operation on docker image.Prior to calling this service user needs to create image metadata using https://{IP/HOST\_NAME}/v1/images POST an upload image using https://{IP/HOST\_NAME}/v1/rpc/images/content/3DED763F-99BA-4F99-B53B-5A6F6736E1E9 POST method.

This method iternally performs following:

docker load: load image to docker container

docker tag:- tag with \_source to mark it as original (In database no change in image name, which end user sees)

docker rmi:- remove the first one without \_source tag from container

https://{IP/HOST\_NAME}/v1/rpc/docker-setup/3DED763F-99BA-4F99-B53B-5A6F6736E1E9

Input: Pathparam: 3DED763F-99BA-4F99-B53B-5A6F6736E1E9

Output:

{"details":"Docker Image successfully uploaded","status":"Success"}

In case of error:

Input: Pathparam: FAA5AA92-5872-44CD-BBF4-AD3EFB61D7C9

Lets say the user provide the image id which does not have image\_deployment as 'Docker'

{"error": "Cannot Perform Docker Setup Operation in this Image" }

30) Pull Docker Image

@Path("/rpc/docker-pull/{image\_id: [0-9a-zA-Z\_-]+}")

@POST

@Produces(MediaType.*APPLICATION\_JSON*)

@Consumes(MediaType.*APPLICATION\_JSON*)

**public** Response dockerPull(@PathParam("image\_id") String image\_id)

This method initiates docker pull task which downloads image from docker hub to trust director for given image\_id provided that deployment\_type of image must be 'Docker' and repo tag should be given before.

It spawns a new thread which asynchronously executes image download from hub and saving it in director. Internally it executes following docker commands:

docker pull: download image to docker container

docker save:- save as tar file

docker tag:- tag with \_source to mark it as original (In database no change in image name, which end user sees)

docker rmi:- remove the one we first downloaded without \_source tag from container

https://{IP/HOST\_NAME}/v1/rpc/docker-pull/3DED763F-99BA-4F99-B53B-5A6F6736E1E9

Input: Pathparam: 3DED763F-99BA-4F99-B53B-5A6F6736E1E9

Output:

{"details":"Docker Image successfully queued for download","status":"Success"}

In case of error:

Input: Pathparam: FAA5AA92-5872-44CD-BBF4-AD3EFB61D7C9

Lets say the user provide the image id which does not have image\_deployment as 'Docker'

{"error": "Cannot Perform Docker Setup Operation in this Image" }

31) Remove Orphan Policies

@Path("/rpc/remove-orphan-policies")

@POST

@Produces(MediaType.*APPLICATION\_JSON*)

@Consumes(MediaType.*APPLICATION\_JSON*)

**public** Response removeOrphanPolicies()

This method removes policies from configured external storages whose associated image is deleted from one or more configured external storages.

https://{IP/HOST\_NAME}/v1/remove-orphan-policies

Input: {}

Output:

In case of error:

{"error": "No image stores configured"}

32) Fetch Image Uploads History

@Path("/image-actions/history/{imageId: [0-9a-zA-Z\_-]+}")

@GET

@Produces(MediaType.*APPLICATION\_JSON*)

**public** Response getImageActionHistory(@PathParam("imageId") String imageId)

This method will fetch image upload history of given imageId.

https://{IP/HOST\_NAME}/v1/image-actions/history/FAA5AA92-5872-44CD-BBF4-AD3EFB61D7C9

Input: PathParam : imageId = 64E8AFCC-182F-42C9-8A7B-42AD3C93EDCF

Output:

{"image\_action\_history\_list":[

{"store\_names":"Glance-36","execution\_status":"Complete","id":"30869EF3-9809-48F6-AC36-21994318313F","artifact\_name":"Image With Policy As Tarball","datetime":"2016 Mar 22 12:30:46"}, {"store\_names":"Glance-36","execution\_status":"Complete","id":"689AF185-2232-4E61-A1ED-2435FF7DF337","artifact\_name":"Image With Policy As Tarball","datetime":"2016 Mar 18 15:34:36"}]

}

33) Create Image Store

@Path("/v1/image-stores")

@POST

@Produces(MediaType.*APPLICATION\_JSON*)

@Consumes(MediaType.*APPLICATION\_JSON*)

**public** Response createImageStore(ImageStoreTransferObject imageStoreTransferObject) throws DirectorException

Creates the external store. the user can either pass the whole object, if the details are knows or just the name, artifacts and connector to reserve a external store. The user can then call the PUT call to update the values of the connection properties.

https://{IP/HOST\_NAME}/v1/image-stores

Input: {"name":"ExtStore\_1", "connector":"Glance", "artifact\_types":["Image", "Tarball"]}

Output:{

"id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"name": "ExtStore\_1",

"artifact\_types": [

"Image",

"Tarball"

],

"connector": "Glance",

"deleted": false,

"image\_store\_details": [

{

"id": "331BD472-5F66-4EFB-811C-461A0CEF6517",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.api.endpoint"

},

{

"id": "AA24C0E1-9F33-442C-B872-49364D933F35",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.keystone.public.endpoint"

},

{

"id": "FFF71D82-29E2-46FA-9789-ED67880B2266",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.image.store.username"

},

{

"id": "02073D14-648F-41C8-97CF-E3BD831D4F03",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.image.store.password"

},

{

"id": "F76EC7A8-703C-4645-8FE0-2666C377D033",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.tenant.name"

}

]

}

34) Edit Image Store

@Path("/v1/image-stores")

@PUT

@Produces(MediaType.*APPLICATION\_JSON*)

@Consumes(MediaType.*APPLICATION\_JSON*)

**public** ImageStoreTransferObject updateImageStores(ImageStoreTransferObject imageStoreTransferObject) throws DirectorException

This methods update image store and its details

https://{IP/HOST\_NAME}/v1/image-stores

Input:

{

"id":"78D1FF99-7412-4AA6-8351-8FD6902412CB",

"name":"Glance-36",

"artifact\_types":[

"Image",

"Tarball"

],

"connector":"Glance",

"deleted":false,

"image\_store\_details":[

{

"id":"9FED428E-EE94-4EA6-A0B9-B0C21A6DF0D8",

"image\_store\_id":"78D1FF99-7412-4AA6-8351-8FD6902412CB",

"key":"glance.api.endpoint",

"value":"http://10.35.35.36:9292",

"key\_display\_value":"API Endpoint"

},

{

"id":"A11F3A00-7E01-4694-A897-C9FBD4E4497B",

"image\_store\_id":"78D1FF99-7412-4AA6-8351-8FD6902412CB",

"key":"glance.image.store.password",

"value":"intelmh",

"key\_display\_value":"Password"

},

{

"id":"64337211-D011-4B63-A8EE-F6FF0AC62D33",

"image\_store\_id":"78D1FF99-7412-4AA6-8351-8FD6902412CB",

"key":"glance.image.store.username",

"value":"admin",

"key\_display\_value":"UserName"

},

{

"id":"FACBD7AD-3409-44E2-8552-6EA8D85913DF",

"image\_store\_id":"78D1FF99-7412-4AA6-8351-8FD6902412CB",

"key":"glance.keystone.public.endpoint",

"value":"http://10.35.35.36:5000",

"key\_display\_value":"Authorization Endpoint"

},

{

"id":"C855F736-0319-4812-9D5C-342CAD913F45",

"image\_store\_id":"78D1FF99-7412-4AA6-8351-8FD6902412CB",

"key":"glance.tenant.name",

"value":"admin",

"key\_display\_value":"Tenant Name"

}

]

}

Output: {

"id":"78D1FF99-7412-4AA6-8351-8FD6902412CB",

"name":"Glance-36",

"artifact\_types":[

"Image",

"Tarball"

],

"connector":"Glance",

"deleted":false,

"image\_store\_details":[

{

"id":"9FED428E-EE94-4EA6-A0B9-B0C21A6DF0D8",

"image\_store\_id":"78D1FF99-7412-4AA6-8351-8FD6902412CB",

"key":"glance.api.endpoint",

"value":"http://10.35.35.36:9292",

"key\_display\_value":"API Endpoint"

},

{

"id":"A11F3A00-7E01-4694-A897-C9FBD4E4497B",

"image\_store\_id":"78D1FF99-7412-4AA6-8351-8FD6902412CB",

"key":"glance.image.store.password",

"value":"XNxEwGPUzMWFoUmzaZBV6gP+tblx3h7hF/RwIvPoKFo62DIOm34Z3W0ef/dZWrJq",

"key\_display\_value":"Password"

},

{

"id":"64337211-D011-4B63-A8EE-F6FF0AC62D33",

"image\_store\_id":"78D1FF99-7412-4AA6-8351-8FD6902412CB",

"key":"glance.image.store.username",

"value":"admin",

"key\_display\_value":"UserName"

},

{

"id":"FACBD7AD-3409-44E2-8552-6EA8D85913DF",

"image\_store\_id":"78D1FF99-7412-4AA6-8351-8FD6902412CB",

"key":"glance.keystone.public.endpoint",

"value":"http://10.35.35.36:5000",

"key\_display\_value":"Authorization Endpoint"

},

{

"id":"C855F736-0319-4812-9D5C-342CAD913F45",

"image\_store\_id":"78D1FF99-7412-4AA6-8351-8FD6902412CB",

"key":"glance.tenant.name",

"value":"admin",

"key\_display\_value":"Tenant Name"

}

]

}

35) Delete Image Store

@Path("/image-stores/{imageStoreId: [0-9a-zA-Z\_-]+}")

@DELETE

@Produces(MediaType.*APPLICATION\_JSON*)

**public** Response deleteImageStores(@PathParam("imageStoreId") String imageStoreId)

Turns deleted flag of image store to false for the provided imageStoreId if not exists respond with HTTP 404 Not Found.

Input: https://{IP/HOST\_NAME}/v1/image-stores/9EC519A5-F57C-4A07-ABEA-D7C5C58B5CD8

PathParam: 9EC519A5-F57C-4A07-ABEA-D7C5C58B5CD8

Output:

In case of successful deletion:

{"deleted" : true}

In case ImageStore doesn't exist, gives HTTP 404 Not Found

36) Validate Image Store

@Path("/rpc/image-stores/{imageStoreId: [0-9a-zA-Z\_-]+}/validate")

@POST

@Produces(MediaType.*APPLICATION\_JSON*)

@Consumes(MediaType.*APPLICATION\_JSON*)

**public** Response validateImageStore(@PathParam("imageStoreId") String imageStoreId)

This method validates Image Store provided with image Store id

Input: https://{IP/HOST\_NAME}/v1/rpc/image-stores/3DA92563-A2A6-4D52-9337-3201D11105E1/validate

Output:

{

"id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"name": "ExtStore\_1",

"artifact\_types": [

"Image",

"Tarball"

],

"connector": "Glance",

"deleted": false,

"is\_valid": true,

"image\_store\_details": [

{

"id": "FFF71D82-29E2-46FA-9789-ED67880B2266",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.image.store.username"

},

{

"id": "331BD472-5F66-4EFB-811C-461A0CEF6517",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.api.endpoint"

},

{

"id": "F76EC7A8-703C-4645-8FE0-2666C377D033",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.tenant.name"

},

{

"id": "AA24C0E1-9F33-442C-B872-49364D933F35",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.keystone.public.endpoint"

},

{

"id": "02073D14-648F-41C8-97CF-E3BD831D4F03",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.image.store.password"

}

]}

37) Fetch Image Store

@Path("image-stores/{imageStoreId: [0-9a-zA-Z\_-]+}")

@GET

@Produces(MediaType.*APPLICATION\_JSON*)

**public** Response getImageStore(@PathParam("imageStoreId") String imageStoreId)

Get image store for the provided imageStoreId if not exists respond with HTTP 404 Not Found.

Input:

https://<TD\_HOST>/v1/image-stores/3DA92563-A2A6-4D52-9337-3201D11105E1

Output:

{

"id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"name": "ExtStore\_1",

"artifact\_types": [

"Image",

"Tarball"

],

"connector": "Glance",

"deleted": false,

"image\_store\_details": [

{

"id": "FFF71D82-29E2-46FA-9789-ED67880B2266",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.image.store.username"

},

{

"id": "331BD472-5F66-4EFB-811C-461A0CEF6517",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.api.endpoint"

},

{

"id": "F76EC7A8-703C-4645-8FE0-2666C377D033",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.tenant.name"

},

{

"id": "AA24C0E1-9F33-442C-B872-49364D933F35",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.keystone.public.endpoint"

},

{

"id": "02073D14-648F-41C8-97CF-E3BD831D4F03",

"image\_store\_id": "3DA92563-A2A6-4D52-9337-3201D11105E1",

"key": "glance.image.store.password"

}]

}

If an invalid ID for image store is given, a HTTP 404 is returned

38) Fetch Image Hash Algo Type

@Path("image-hash-type")

@GET

@Produces(MediaType.*APPLICATION\_JSON*)

**public** Response getImageHashType(@QueryParam("deploymentType") String deploymentType)

This method returns hash algorithm used for hash calculation according to valid deployment type. If deployment type is not provided returns hash algorithm used by all supported deployment type.

Input:

https://<TD\_HOST>/v1/image-hash-type

deploymentType : VM

Output:

{

"deployment\_type": "VM",

"hash\_type": "sha256"

}

39) Fetch Stalled Images

@Path("images-stalled")

@GET

@Produces(MediaType.*APPLICATION\_JSON*)

**public** Response getStalledImages()

This method returns list of stalled images.

Input:

https://<TD\_HOST>/v1/images-stalled

Output:

[

{

"created\_by\_user\_id": "admin",

"created\_date": "2016-04-14 18:49:11",

"edited\_by\_user\_id": "admin",

"edited\_date": "2016-04-14 18:49:11",

"id": "AC4750E4-4018-449D-B471-9517122FE29B",

"image\_name": "123.img",

"image\_format": "qcow2",

"image\_deployments": "VM",

"image\_size": 13631488,

"sent": 0,

"deleted": false,

"image\_uploads\_count": 0,

"policy\_uploads\_count": 0,

"image\_upload\_status": "In Progress",

"image\_Location": "/mnt/images/"

}

]

40) Fetch Image deployment Type which supports encryption

@Path("encryption-supported")

@GET

@Produces(MediaType.*APPLICATION\_JSON*)

**public** Response listEncryptionSupported()

This method returns list of deployment types which allow image encryption.

Input:

https://<TD\_HOST>/v1/encryption-supported

Output:

["VM"]

41) Download Trust Policy Using trust policy id

@Path("trust-policies/{trustPolicyId: [0-9a-zA-Z\_-]+}/download")

@GET

@Produces(MediaType.*APPLICATION\_XML*)

**public** Response downloadPolicyForTrustPolicyId(

@PathParam("trustPolicyId") String trustPolicyId)

This method looks into the MW\_TRUST\_POLICY table and gets the policy string and sends it as an xml content to the user.

In case the policy is not found for the trust policy id, HTTP 404 is returned.

Input:

https://<TD\_HOST>/v1/trust-policies/08EB37D7-2678-495D-B485-59233EB51996

/download

42) Download Trust Policy Draft Using trust policy draft id

@Path("trust-policy-drafts/{trustPolicyDraftId: [0-9a-zA-Z\_-]+}/download")

@GET

@Produces(MediaType.*APPLICATION\_XML*)

**public** Response downloadPolicyForTrustPolicyDraftId(@PathParam("trustPolicyDraftId")

String trustPolicyDraftId)

This method looks into the MW\_TRUST\_POLICY\_DRAFTS table and gets the policy draft

string and sends it as an xml content to the user.

In case the policy draft is not found for the trust policy draft id, HTTP 404 is returned.

Input:

https://<TD\_HOST>/v1/rust-policy-drafts/08EB37D7-2678-495D-B485-59233EB51996/download

***Sample flows***

*Initial Login to get auth token:-*

Request URL:[https://10.35.35.133/v1/login](https://10.35.35.119/v1/login)

Request Method:POST

RequestHeader:-

Content-Type:application/json

Accept:application/json

{"username": "admin", "password": "passwd"}

Response:-

{"authorization\_token":"ztDsT6Ou4Y4ppemlYY6oOMF8O22yRU2ez8c9Y/6Ezv0="}

**Use Cases:-**

**VM**

1. Uploading a File To Trust Director involves 2 steps:

a. First metadata for image is sent.

b. File is uploaded to TD with obtained id in first step.

A. Sending Metadata For Image

Request URL: https://10.35.35.133/v1/images

Request Method:POST

Request Header:

Authorization:Token ntDkpAyRcHh2yVWW2I7ztIsCx0nJH2sQ6N9K9XKPc6c=

Content-Type:application/json

Payload:

{

"image\_deployments": "VM",

"image\_format" : "qcow2",

"image\_name": "cirros\_v2.tar",

"image\_size" : 13312

}

Response:

{

"created\_by\_user\_id": "admin",

"created\_date": 1452761459085,

"edited\_by\_user\_id": "admin",

"edited\_date": 1452761459085,

"id": "3DED763F-99BA-4F99-B53B-5A6F6736E1E9",

"image\_name": "cirros\_v2.tar",

"image\_format": "qcow2",

"image\_deployments": "VM",

"image\_size": 13312,

"sent": 0,

"deleted": false,

"image\_upload\_status": "In Progress",

"image\_Location": "/mnt/images/"

}

B. Sending Image

Request URL: https://10.35.35.133/v1/rpc/images/content/3DED763F-99BA-4F99-B53B-5A6F6736E1E9

Request Method:POST

Request Header:

Authorization:Token ntDkpAyRcHh2yVWW2I7ztIsCx0nJH2sQ6N9K9XKPc6c=

Content-Type:application/octet-stream

Payload:

File

Response:

{

"created\_by\_user\_id": "admin",

"created\_date": 1452761459085,

"edited\_by\_user\_id": "admin",

"edited\_date": 1452761459085,

"id": "3DED763F-99BA-4F99-B53B-5A6F6736E1E9",

"image\_name": "cirros\_v2.tar",

"image\_format": "qcow2",

"image\_deployments": "VM",

"image\_size": 13312,

"sent": 13312,

"deleted": false,

"image\_upload\_status": "Complete",

"image\_Location": "/mnt/images/"

}

2. For Creating Trust Policy Drafts for an Image, We send required metadata such as image\_id, display\_name for policy, launch\_control\_policy, encrypted, etc.

Request URL:https://10.35.35.133/v1/trust-policy-drafts

Request Method:POST

Request Header:

Authorization:Token ntDkpAyRcHh2yVWW2I7ztIsCx0nJH2sQ6N9K9XKPc6c=

Content-Type:application/json

Payload:

{

"display\_name" : "cirros\_v2.tar",

"encrypted" : true,

"image\_id" : "3DED763F-99BA-4F99-B53B-5A6F6736E1E9"

"image\_name" : "cirros\_v2.tar"

"launch\_control\_policy" : "MeasureAndEnforce"

}

Response:

{

"encrypted":false,

"id":"dae9c1ee-4b35-4b3b-8c38-163a4127b51d",

"trust\_policy":"<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>\n<TrustPolicy xmlns:ns2=\"http://www.w3.org/2000/09/xmldsig#\" xmlns=\"mtwilson:trustdirector:policy:1.1\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\">\n <Director>\n <CustomerId>testId</CustomerId>\n </Director>\n <Image>\n <ImageId>3DED763F-99BA-4F99-B53B-5A6F6736E1E9</ImageId>\n </Image>\n <LaunchControlPolicy>MeasureAndEnforce</LaunchControlPolicy>\n <Encryption>\n <Key URL=\"uri\">uri</Key>\n <Checksum>1</Checksum>\n </Encryption>\n <Whitelist DigestAlg=\"sha256\"/>\n</TrustPolicy>\n",

"status":"success"

}

3. For Mount Image we need to pass id for an image.

Request URL:https://10.35.35.133/v1/rpc/mount-image

Request Method:POST

Request Header:

Authorization:Token ntDkpAyRcHh2yVWW2I7ztIsCx0nJH2sQ6N9K9XKPc6c=

Content-Type:application/json

Payload:

{

"id": "3DED763F-99BA-4F99-B53B-5A6F6736E1E9"

}

Response:

{

"created\_by\_user\_id":"admin",

"created\_date":1452761459085,

"edited\_by\_user\_id":"admin",

"edited\_date":1452761782716,

"id":"3DED763F-99BA-4F99-B53B-5A6F6736E1E9",

"image\_name":"cirros\_v2.tar",

"image\_format":"qcow2",

"image\_deployments":"VM",

"image\_size":13312,

"sent":13312,

"mounted\_by\_user\_id":"admin",

"deleted":false,

"image\_upload\_status":"Complete",

"image\_Location":"/mnt/images/"

}

4. Drafts are usually updated by sending patches.

Request URL:https://10.35.35.133/v1/trust-policy-drafts/dae9c1ee-4b35-4b3b-8c38-163a4127b51d

Request Method:POST

Request Header:

Authorization:Token ntDkpAyRcHh2yVWW2I7ztIsCx0nJH2sQ6N9K9XKPc6c=

Content-Type:application/json

Payload:

{

"patch" : "<patch><add pos=\"prepend\" sel='//\*[local-name()=\"Whitelist\"]'><File Path=\"/init\"/></add></patch>"

}

Response:

{

"created\_by\_user\_id":"admin",

"created\_date":"2016-01-14",

"edited\_by\_user\_id":"admin",

"edited\_date":1452762152959,

"id":"dae9c1ee-4b35-4b3b-8c38-163a4127b51d",

"trust\_policy\_draft":"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<TrustPolicy xmlns=\"mtwilson:trustdirector:policy:1.1\" xmlns:ns2=\"http://www.w3.org/2000/09/xmldsig#\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\">\n <Director>\n <CustomerId>testId</CustomerId>\n </Director>\n <Image>\n <ImageId>3DED763F-99BA-4F99-B53B-5A6F6736E1E9</ImageId>\n </Image>\n <LaunchControlPolicy>MeasureAndEnforce</LaunchControlPolicy>\n <Encryption>\n <Key URL=\"uri\">uri</Key>\n <Checksum>1</Checksum>\n </Encryption>\n <Whitelist DigestAlg=\"sha256\"><File xmlns=\"\" Path=\"/init\" /></Whitelist>\n</TrustPolicy>\n",

"display\_name":"cirros\_v2.tar"

}

5. Drafts is finally converted to Trustpolicy.

Request URL: https://10.35.35.133/v1/rpc/finalize-trust-policy-draft

Request Method:POST

Request Header:

Authorization:Token ntDkpAyRcHh2yVWW2I7ztIsCx0nJH2sQ6N9K9XKPc6c=

Content-Type:application/json

Payload:

{

"trust\_policy\_draft\_id":"dae9c1ee-4b35-4b3b-8c38-163a4127b51d"

}

Response:

{

"id":"7753bbe2-69b4-4324-b0c3-4d2ad99238b3"

}

6. Unmouting an Image requires id of an image

Request URL: https://10.35.35.133/v1/rpc/unmount-image

Request Method:POST

Request Header:

Authorization:Token ntDkpAyRcHh2yVWW2I7ztIsCx0nJH2sQ6N9K9XKPc6c=

Content-Type:application/json

Payload:

{

"id":"3DED763F-99BA-4F99-B53B-5A6F6736E1E9"

}

Response:

{

"created\_by\_user\_id":"admin",

"created\_date":1452761459085,

"edited\_by\_user\_id":"admin",

"edited\_date":1452761782716,

"id":"3DED763F-99BA-4F99-B53B-5A6F6736E1E9",

"image\_name":"cirros\_v2.tar",

"image\_format":"qcow2",

"image\_deployments":"VM",

"image\_size":13312,

"sent":13312,

"mounted\_by\_user\_id":"admin",

"deleted":false,

"image\_upload\_status":"Complete",

"image\_Location":"/mnt/images/"

}

6. Policy Metadata such as it's name can be converted in following ways:

Request URL: https://10.35.35.133/v1/trust-policies/7753bbe2-69b4-4324-b0c3-4d2ad99238b3

Request Method: POST

Request Header:

Authorization:Token ntDkpAyRcHh2yVWW2I7ztIsCx0nJH2sQ6N9K9XKPc6c=

Content-Type:application/json

Payload:

{

"display\_name":"cirros\_v2.tar"

}

Response:

{

"status":"success",

}

7. For Creating Async Tasks such encrypting image, Creating tar, Uploading it to Glance, etc. :

Request URL: https://10.35.35.133/v1/image-actions

Request Method: POST

Request Header:

Authorization:Token ntDkpAyRcHh2yVWW2I7ztIsCx0nJH2sQ6N9K9XKPc6c=

Content-Type:application/json

Payload:

{

"image\_id":"3DED763F-99BA-4F99-B53B-5A6F6736E1E9",

"actions":[

{

"task\_name":"Create Tar",

"status":"Incomplete"

},

{

"task\_name":"Upload Tar",

"status":"Incomplete",

"storename":"Glance"

}

]

}

Response:

{

"deleted":false,

"id":"E2B32D67-E9F6-4FD5-97A2-1E2FC5EE93D3",

"image\_id":"3DED763F-99BA-4F99-B53B-5A6F6736E1E9",

"action\_count":3,

"action\_completed":0,

"action\_size":0,

"action\_size\_max":0,

"actions":[

{

"status":"Incomplete",

"task\_name":"Encrypt Image"

},

{

"status":"Incomplete",

"task\_name":"Create Tar"

},

{

"status":"Incomplete",

"storename":"Glance",

"task\_name":"Upload Tar"

}

],

"current\_task\_status":"Incomplete",

"current\_task\_name":"Encrypt Image"

}

**Bare Metal**

a. Create Host:-

Request URL:https://10.35.35.133/v1/images/host

Request Method:POST

Request headers:-

Authorization:Token ztDsT6Ou4Y4ppemlYY6oOMF8O22yRU2ez8c9Y/6Ezv0=

Content-Type:application/json

Accept:application/json

Payload:-

{"policy\_name":"A1","ip\_address":"10.35.35.182","username":"root","password":"intelmh","name":"10.35.35.182"}

Response:-

{  
 "deleted": false,  
 "ip\_address": "10.35.35.182",  
 "username": "root",  
 "image\_name": "10.35.35.182",  
 "image\_id": "2EE8E2F3-7F69-480B-B730-B932BFAE7CFF"  
}

b. Mount Image:-

Request URL:https://10.35.35.133/v1/rpc/mount-image

Request Method:POST

Request headers:-

Content-Type:application/json

Accept:application/json

Authorization:Token ztDsT6Ou4Y4ppemlYY6oOMF8O22yRU2ez8c9Y/6Ezv0=

Payload:-

{"id":"2EE8E2F3-7F69-480B-B730-B932BFAE7CFF"}

Response:-

{  
 "created\_date": 1452760218270,  
 "edited\_by\_user\_id": "admin",  
 "edited\_date": 1452760451487,  
 "id": "2EE8E2F3-7F69-480B-B730-B932BFAE7CFF",  
 "image\_name": "10.35.35.182",  
 "image\_deployments": "BareMetal",  
 "mounted\_by\_user\_id": "admin",  
 "deleted": false,  
 "image\_upload\_status": "Complete"  
}

c. Create trust policy Draft:-

Request URL:https://10.35.35.133/v1/trust-policy-drafts

Request Method:POST

Request headers:-

Authorization:Token ztDsT6Ou4Y4ppemlYY6oOMF8O22yRU2ez8c9Y/6Ezv0=

Content-Type:application/json

Accept:application/json

Payload:-

{"image\_id":"2EE8E2F3-7F69-480B-B730-B932BFAE7CFF","display\_name":"A1\_renamed","launch\_control\_policy":"MeasureOnly"}

Response:-

{  
 "encrypted": false,  
 "id": "cf55afca-6b97-4bf7-a0b0-9dd2e4b038a7",  
 "trust\_policy": "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>\n<TrustPolicy xmlns:ns2=\"http://www.w3.org/2000/09/xmldsig#\" xmlns=\"mtwilson:trustdirector:policy:1.1\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\">\n <Director>\n <CustomerId>testId</CustomerId>\n </Director>\n <Image>\n <ImageId>2EE8E2F3-7F69-480B-B730-B932BFAE7CFF</ImageId>\n </Image>\n <LaunchControlPolicy>MeasureOnly</LaunchControlPolicy>\n <Whitelist DigestAlg=\"sha1\"/>\n</TrustPolicy>\n",  
 "status": "success"  
}

d. Apply Trust Policy template:-

Request URL:https://10.35.35.133/v1/rpc/apply-trust-policy-template/

Request Method:POST

Request headers:-

Authorization:Token ztDsT6Ou4Y4ppemlYY6oOMF8O22yRU2ez8c9Y/6Ezv0=

Content-Type:application/json

Accept:application/json

Payload:-

{"image\_id":"2EE8E2F3-7F69-480B-B730-B932BFAE7CFF"}

Response:-

{  
"trust\_policy": "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?><TrustPolicy.......... .......</TrustPolicy>"  
}

e. Search:-

Request URL:https://10.35.35.133/v1/images/2EE8E2F3-7F69-480B-B730-B932BFAE7CFF/search?dir=/&recursive=false&files\_for\_policy=false&init=true&include\_recursive=false&reset\_regex=false

Request Method:GET

Request headers:-

Authorization:Token ztDsT6Ou4Y4ppemlYY6oOMF8O22yRU2ez8c9Y/6Ezv0=

Content-Type:application/json

Response:-

{  
 "patch\_xml": [],  
 "tree\_content": "<ul class=\"jqueryFileTree\" style=\"display: none;\"><li class=\"directory collapsed \">...”

}

f.

1)Selecting initrd.img file:-

Request URL:https://10.35.35.133/v1/trust-policy-drafts/ cf55afca-6b97-4bf7-a0b0-9dd2e4b038a7

Request Method:POST

Request headers:-

Authorization:Token ztDsT6Ou4Y4ppemlYY6oOMF8O22yRU2ez8c9Y/6Ezv0=

Content-Type:application/json

Payload:-

{"patch":"<patch><add pos=\"prepend\" sel='//\*[local-name()=\"Whitelist\"]'><File Path=\"/initrd.img\"/></add></patch>"}

Response:-

{"created\_by\_user\_id":"admin","created\_date":"2016-01-14","edited\_by\_user\_id":"admin","edited\_date":1452761917177,"id":"cf55afca-6b97-4bf7-a0b0-9dd2e4b038a7","trust\_policy\_draft":"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<TrustPolicy......</TrustPolicy>\n","display\_name":"A1\_renamed"}

2)Applying regex on boot folder :-

Include filter:- \*.gz

Request URL:https://10.35.35.133/v1/trust-policy-drafts/ cf55afca-6b97-4bf7-a0b0-9dd2e4b038a7

Request Method:POST

Request headers:-

Authorization:Token ztDsT6Ou4Y4ppemlYY6oOMF8O22yRU2ez8c9Y/6Ezv0=

Content-Type:application/json

Payload:-

{"patch":"<patch><add sel='//\*[local-name()=\"Whitelist\"]'><Dir Path=\"/boot\" Include=\"\*.gz\" Exclude=\"\" Recursive=\"false\"/></add><add pos=\"after\" sel='//\*[local-name()=\"Whitelist\"]/\*[local-name()=\"Dir\"][@Path=\"/boot\"]'><File Path=\"/boot/tboot.gz\"/></add></patch>"}

Response:-

{"created\_by\_user\_id":"admin","created\_date":"2016-01-14","edited\_by\_user\_id":"admin","edited\_date":1452761917178,"id":"cf55afca-6b97-4bf7-a0b0-9dd2e4b038a7","trust\_policy\_draft":"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<TrustPolicy......</TrustPolicy>\n","display\_name":"A1\_renamed"}

g. Create Trust policy:-

Request URL:https://10.35.35.133/v1/rpc/finalize-trust-policy-draft

Request Method:POST

Request headers:-

Authorization:Token ztDsT6Ou4Y4ppemlYY6oOMF8O22yRU2ez8c9Y/6Ezv0=

Content-Type:application/json

Accept:application/json

Payload:-

{"trust\_policy\_draft\_id":"cf55afca-6b97-4bf7-a0b0-9dd2e4b038a7"}

Response:-

{"deleted":false,"id":"8c9ce6e4-120e-437c-84f0-b8ad4e65af8b"}

h.Unmount Image:-

Request URL:https://10.35.35.133/v1/rpc/unmount-image

Request Method:POST

Request headers:-

Authorization:Token ztDsT6Ou4Y4ppemlYY6oOMF8O22yRU2ez8c9Y/6Ezv0=

Content-Type:application/json

Payload:-

{"id":"2EE8E2F3-7F69-480B-B730-B932BFAE7CFF"}

Response:-

{"created\_date":1452709800000,"edited\_by\_user\_id":"admin","edited\_date":1452763821313,"id":"2EE8E2F3-7F69-480B-B730-B932BFAE7CFF","image\_name":"10.35.35.182","image\_deployments":"BareMetal","deleted":false,"image\_upload\_status":"success"}

g. Fetch list for bare metal live:-

Request URL:https://10.35.35.133/v1/images?deploymentType=BareMetalLive

Request Method:GET

Request headers:-

Authorization:Token ztDsT6Ou4Y4ppemlYY6oOMF8O22yRU2ez8c9Y/6Ezv0=

Content-Type:application/json

Response:-

{"images":[{"created\_date":"2016-01-14","edited\_by\_user\_id":"admin","edited\_date":"2016-01-14","id":"2EE8E2F3-7F69-480B-B730-B932BFAE7CFF","image\_name":"10.35.35.182","image\_deployments":"BareMetal","deleted":false,"trust\_policy\_id":"caa3e2d7-4a47-4189-ac81-03b76b798fcf","uploads\_count":0,"policy\_name":"A1\_renamed","image\_upload\_status":"Complete"},{"created\_date":"2016-01-14","edited\_date":"2016-01-14","id":"79518061-065C-4B97-B0AE-AAE9400458CE","image\_name":"10.35.35.182","image\_deployments":"BareMetal","deleted":false,"uploads\_count":0,"policy\_name":"-","image\_upload\_status":"Complete"},{"created\_date":"2016-01-14","edited\_date":"2016-01-14","id":"5F9BED84-C758-42F4-9456-8245C95A057F","image\_deployments":"BareMetal","deleted":false,"uploads\_count":0,"policy\_name":"-","image\_upload\_status":"Complete"},{"created\_date":"2016-01-14","edited\_by\_user\_id":"admin","edited\_date":"2016-01-14","id":"E1882707-A774-42E9-A8CC-AA817439B88E","image\_name":"10.35.35.182","image\_deployments":"BareMetal","mounted\_by\_user\_id":"admin","deleted":false,"trust\_policy\_draft\_id":"625455bf-915a-4c1a-94aa-4f088cf5fd90","uploads\_count":0,"policy\_name":"A1","image\_upload\_status":"Complete"}]}