**SAL REST API Overview**

**Version History**

| Version Number | Author | Description |
| --- | --- | --- |
| 1.0 | [Rahul katariya](mailto:rahul.katariya@primaryio.com) | Initial high level SAL API description with request and response structure. This document is intended to assist other modules with APIs for sending and retrieving data to/from the SAL. |

**1.0 Overview**

The SAL (Storage Abstraction Layer) REST server provides an interface for writing and retrieving VM datablocks to/from the cloud through REST APIs. Currently, it supports writing and retrieving data to/from only IBM cloud. Gin Web Framework is used internally to design & develop SAL REST APIs. SAL server is a stateless server which uploads and retrieves the data, it does not store any metadata information. If the client does not get any acknowledgement of any upload request, then the client needs to send the upload request again.

SAL REST API:

* Is organized around REST.
* Accepts JSON-encoded request bodies.
* Returns JSON-encoded responses.
* Uses standard HTTP response codes.

SAL REST APIs lets you:

* list protected VMs
* store/retrieve VMDK file into/from the cloud
* store/retrieve specific blocks of VM. It also supports versioning.

Authorization:

* TBD (\*To be determined). Currently, there is no authorization required.

**1.1 Upload VMDK file:**

For uploading a VMDK file of filesize >10MB, client needs to execute three steps. First step is to initiate the upload process of the file by using REST API **startUploadFileObj**. REST API server returns upload-id. This upload-id will be used for further communication. File to be uploaded is divided into 4MB/8MB partsize (block size). And all parts will be uploaded by using REST API  **uploadPartFileObj**. Note that the last part which is uploaded might be less than 4MN/8MB size. **uploadPartFileObj** REST API will return completion-id-number (Etag) for the part.

This completion-id-number (Etag) related to the part needs to be saved by the client. After aloof the completion-id-nuber received by the client, client will send all of these completion-id-numbers (Etags) to server using API **completeUploadFileObj**. Server checks all completion-id-number(s) sent by the client and then combines all parts into one file.

Following 3 APIs will be used for uploading VMDK file:

* startUploadFileObj
* uploadPartFileObj
* completeUploadFileObj



**1.2 VAIO Communication:**

From the VAIO intercept driver, modified blocks are sent to the receiver which are forwarded to the SAL server. The SAL server maintains different versions of the modified datablock of size (4MB/8MB) of the VMDK file. Client can retrieve the specific version of the datablock by providing UTC time.

Following 2 APIs will be used to send and retrieve block data to/from the SAL server.

* addVaioObj
* getVaioObj



**2.0 REST APIs**

Following REST API endpoints are implemented and exported by SAL server:

* listProtectedVms : to list VM Buckets.
* createVMBucket : to create VM Bucket with versioning enabled.
* deleteVMBucket : to delete VM Bucket.
* addVaioObj : to store VAIO Object of specific block number of specific

VMDK file.

* getVaioObj : to retrieve a VAIO Object of a specific block number of a

specific VMDK file at specific UTC time.

* startUploadFileObj : first step which needs to be executed to upload a

file/VMDK of a VM

* uploadPartFileObj : second step which needs to be executed to upload parts

of file/VMDK.

* completeUploadFileObj : last step which needs to be executed to inform

SERVER that all parts are uploaded.

**2.1 listProtectedVms**:

This is a GET request that retrieves the list of protected VMs. This request needs to be sent using JSON.

Sample URL:

http://<Sal\_Server\_Ip>:8080/listProtectedVms

Following is the Request structure for ListProtectedVms API:

N/A

Following is the Response structure for ListProtectedVms API:

| Response Structure |
| --- |
| Array of string(s) |

Example:

# curl <http://172.16.99.24:8080/listProtectedVms>

$ curl http://172.16.99.24:8080/listProtectedVms

["testVM1","testVM2","testVM3","testVM4","testVM5","testVM6","testVM10","tprkVM"]

**2.2 createVMBucket**:

This is a POST request that creates a bucket for the VM to be protected. This request needs to be sent using JSON.

Sample URL command using curl tool:

curl -X POST -H "Content-Type: application/json" -d '{"vmbucketname": "rahulk-testVM9"}' http://<sal\_server\_ip>:8080/createVMBucket

Following is the Request structure for createVMBucket:

| Parameter Name | Type |
| --- | --- |
| vmbucketname | string |

Following is the Response structure for createVMBucket API:

| Response Structure | Description |
| --- | --- |
| string | Success message or error message |

Note: Response structure would be modified later for error handling.

**2.4 addVaioObj**:

This is a POST request that sends a VAIO data object to the SAL server. This request needs to be sent using JSON.

It assumes VMbucket is already created of vmbucketname.This request needs to be sent using JSON.

<VmdkName>\_<BlockNumber> object is created inside bucket <VmBucketName> and Blockdata is written inside it. If the block already exists, then a new version of the object is created.

If addVaioObj API is called with bucketName "VaioTestVM" with VMDKName "VaioTestVM.vmdk" with BlockNumber "100", then "VaioTestVM.vmdk\_100" object is created inside VMbucket "VaioTestVM".

Following is the Request structure for addVaioObj API:

| Parameter Name | Type | Description |
| --- | --- | --- |
| vmbucketname | string | the VMName for which blockdata to be written |
| vmdkname | string | VMDKName of the VM |
| blocknumber | int | BlockNumber of VMDK to be stored in SAL |
| blockData | byte[] | array of bytes which is actual block data to be written |

Following is the Response structure for addVaioObj API:

| Response Structure | Description |
| --- | --- |
| string | Success message or error message |

Note: Response structure would be modified later for error handling.

Sample code in go language:

=================================================

addVaioDataRequestObj := addVaioDataRequest{

VmBucketName: vmbucketname,

VmdkName: vmdkname,

BlockNumber: blocknumber,

BlockData: blockdata,

}

jsonValue, \_ := json.Marshal(addVaioDataRequestObj)

resp, err := http.Post("http://localhost:8080/addVaioObj","application/json",bytes.NewBuffer(jsonValue))

=================================================

**2.5 getVaioObj**:

This is a GET request that retrieves the blockdata of specific version from the object storage of vmbucket. This request needs to be sent using JSON. This API returns the latest datablock if UTC time is not provided. If UTC time is provided, then the block version at the time of UTC time will be returned.

Following is the Request structure for getVaioObj API:

| Parameter Name | Type | Description |
| --- | --- | --- |
| vmbucketname | string | the VMName for which blockdata to be written |
| vmdkname | string | VMDKName of the VM |
| blocknumber | int | BlockNumber of VMDK to be stored in SAL |
| utctime | string | Optional parameter. UTC time should be in following format “YYYY-MM-DD HH:MM:SS.0 +0000 UTC”  example:  "2022-09-19 09:43:58.0 +0000 UTC". If this parameter is not provided, latest datablock will be returned, otherwise latest datablock at the UTC time will be returned. |

Following is the Response structure for getVaioObj API:

| Response Structure | Description |
| --- | --- |
| string | Datablock or error message |

Note: Response structure would be modified later for error handling.

**2.6 startUploadFileObj**:

This is a first step which needs to be executed to upload a file/VMDK of a VM. This is a POST request that starts uploading of a file/VMDK file. This request needs to be sent using JSON.

Following is the Request structure for startUploadFileObj API:

| Parameter Name | Type | Description |
| --- | --- | --- |
| vmbucketname | string | the VMName for which blockdata to be written |
| vmdkname | string | VMDKName of the VM |

Following is the Response structure for startUploadFileObj API:

| Response Structure | Type | Description |
| --- | --- | --- |
| uploadId | string | Upload-id which needs to be provided in the sub-sequent part upload requests |

Note: Response structure would be modified later for error handling.

**2.7 uploadPartFileObj**:

This is a second step which needs to be executed to upload parts

of file/VMDK. This is a POST request which sends data with the upload-id. Data needs to be sent in 4MB/8MB (blocksize) partsize. Only the last part could be less than 4MB/8MB partsize. This request needs to be sent using JSON.

Following is the Request structure for uploadPartFileObj API:

| Parameter Name | Type | Description |
| --- | --- | --- |
| vmbucketname | string | the VMName for which blockdata to be written |
| vmdkname | string | VMDKName of the VM |
| partnumber | int | PartNumber of VMDK to be stored in SAL. PartNumber is a partnumber of the VMDK file which needs to be uploaded. |
| uploadId | string | Upload-id which needs to be provided in the sub-sequent part upload requests |
| filebytes | byte[] | Filebytes data of specific part number which needs to be written. |

Following is the Response structure for uploadPartFileObj API:

| Response Structure | Type | Description |
| --- | --- | --- |
| completedParts | s3.CompletedPart | Uploaded-part-id specific to part which is uploaded |

Note: Response structure would be modified later for error handling.

**2.8 completeUploadFileObj**:

This is a last step which needs to be executed to inform the SAL SERVER that all parts are uploaded. This is a POST request that informs the server that all the parts are already sent and the server can combine all parts into one file. ETag and PartNumber of structure s3.CompletedPart needs to be sent iin the request structure which are related to parts which are already uploaded. Server verifies ETAG and partnumber and then combines all parts into one file. This request needs to be sent using JSON.

Following is the Request structure for completeUploadFileObj API:

| Parameter Name | Type | Description |
| --- | --- | --- |
| vmbucketname | string | the VMName for which blockdata to be written |
| vmdkname | string | VMDKName of the VM |
| uploadId | string | Upload-id which needs to be provided in the sub-sequent part upload requests |
| completedParts | s3.CompletedPart[] | Array of uploaded part return ids [Etag and partnumber] |

Following is the Response structure for completeUploadFileObj API:

| Response Structure | Type | Description |
| --- | --- | --- |
| data | s3.CompleteMultipartUploadOutput | Id related to completed multipart upload. Currently the response structure is of no use. This will be modified later. |

Note: Response structure would be modified later for error handling.