iCBD-Replication Documentation

Release 1.0.0

Luis Silva

Nov 19, 2018

ICBD REPLICATION MODULE

This site covers iCBD-Replication usage & API documentation. For basic info on what iCBD-rep is, including its public changelog & how the project is maintained, please see the git repo.

CHAPTER

ONE

API DOCUMENTATION

We maintain a set of API documentation, autogenerated from the python source code's docstrings (which are typically very thorough.) and for the RESTfull API (TODO: FUTURE)

1.1 icbdrep.ImageRepo module

class `icbdrep.ImageRepo.`**`ImageRepo`**(*config*)

Bases: `object`

`addImage`(*image_name: str*)

Add an image name to the repository And checks if in that directory are already present some snapshots

Args: image_name: name of the image to be added

Returns: None

Raises: DirNotFoundException, BTRFSPathNotFoundException, ImageAlreadyExistsException

`addSnapshot`(*image_name: str*, *snap_number: str*) → None

Add a snapshot to a image

Args: image_name: the name of the image to receive a snapshot snap_number: the snapshot

Returns: None

Raises: BTRFSSubvolumeNotFoundException, SnapshotAlreadyExistsException

`deleteImage`(*image_name: str*) → None

Deletes a given image from the repository

Args: image_name: the name of the image to be deleted

Returns: None

Raises: ImageNotFoundException

`deleteSnapshot`(*image_name: str*, *snap_number: str*) → lib.icbdSnapshot.icbdSnapshot

Deletes a given snapshot of an image

Args: image_name: the image to which the snapshot refers to snap_number: the snapshot number

Returns: None

Raises: SnapshotNotFoundException

`getImagelist`() → typing.List[str]

Get the list of the VM images present in the repo

Returns: a list of strings with the images names

getImagepath(*image_name: str*) → str
Returns the path to the given image.

Args: image_name: the name of the image

Returns: a string with the path to the image

Raises: ImageNotFoundException

getLastSnapshot(*image_name: str*) → lib.icbdSnapshot.icbdSnapshot
Get the last snapshot from the given image.

Args: image_name: name of the image

Returns: an obj icbdSnapshot

Raises: ImageNotFoundException

getSnapshot(*image_name: str*, *snap_number: str*) → lib.icbdSnapshot.icbdSnapshot
Gets a specific snapshot given its number and the image name

Args: image_name: the name of the image snap_number: the number of the snapshot

Returns: an icbdSnapshot object

Raises: SnapshotNotFoundException

getSnapshotlist(*image_name: str*) → typing.List[lib.icbdSnapshot.icbdSnapshot]
Get the list of snapshots present in the repo for the given image. If there are no snapshots it returns a empty list.

Args: image_name: The image name that contains the snapshots

Returns: a list with the snapshots present in the repo

Raises: ImageNotFoundException

hasImage(*image_name: str*) → bool
Check if a given image name is present in the repository

Args: image_name: the image name to be checked

Returns: True if present, otherwise False

hasSnapshot(*image_name: str*, *snap_number: str*) → bool
Check if a snapshot is present in the given image

Args: image_name: the name of the image that should contain the snapshot snap_number: the snapshot

Returns: True if the snapshot is present, otherwise False

1.2 icbdrep.KeepAlive module

class `icbdrep.KeepAlive.`**KeepAlive**(*interval=10*, *tries_num=3*)
Bases: `threading.Thread`

keepAlive(*pyro_bind: bool*) → None
Check a replica state and updates NS if needed.

Args: pyro_bind: boolean True to use of the _pyroBind or False to use the ping method

Returns: None

run()

The main method of the class. This is triggered in the thread.start() call

Returns: None

stopKeepAlive() → None

Stop the execution of the keep alive thread. This should be part of the shutdown process.

Returns: None

1.3 icbdrep.MasterNode module

class icbdrep.MasterNode.**MasterNode**(*node_config*, *ns_config*, *interactive_mode_flag: bool*)

Bases: `threading.Thread`

addImage(*image_name: str*, *node: int*) → None

Add an image to the node repository

Args: image_name: the name of the image to be added node: the node where the image will be added

Returns: Node

delete_snapshot(*image_name: str*, *snap_number: str*, *node: int*) → None

Deletes a snapshot from a given image in a node.

Args: image_name: the image name snap_number: the snapshot number node: the node to do the deletion

Returns: None

exeCommand(*line: str*) → None

Receives a command line and interprets the content. Separating the various fields of the string into arguments, and calls the appropriated function.

Args: line: a line with the command to execute

Returns: None

getReplicasFromNS() -> *(<class 'int'>, typing.Dict[int, Pyro4.core.Proxy])*

Get a list of the replicas present in the system (Name Server) and saves them to the replicas proxy list

Returns: the number of found replicas

interactiveMode() → None

When in interactive mode, the server runs with a prompt, so that individual commands can be typed in

Returns: None

listImages(*node: int*) → None

List the collection of images available in a node.

Args: node: The node to list. (Master or one of the Replicas)

Returns: None

listReplicas() → None

List the replicas present in the system and prints to the console.

Returns: None

listSnapshots(*node: int*, *image_name: str*) → None

List the colection of snapshots of a given image in a node.

Args: node: The node to list (Master or one of the replicas) image_name: The image the snapshots refer to

Returns: None

registerInNS() → Pyro4.core.Daemon
Register the server in the Name Server

Returns: the registered daemon

run()
The main method of the class. This is triggered in the thread.start() call

Returns: None

send(*node: int*, *image_name: str*, *snapshot_number: str*, *blocking: bool*, *ssh: bool = False*, *compression: str = None*) → None
Send Command - Instructs the replica to listen for a transfer, and sends the snapshot in the btrfs path

Args: node: the number of the node image_name: the name of the image snapshot_number: the number of the image blocking: if the function should block

Returns: None

stopMaster() → None
WARNING!! Don't use this! Only for testing and should be deprecated!

Returns: None

1.4 icbdrep.NameServer module

class `icbdrep.NameServer.`**NameServer**(*config*)
Bases: `threading.Thread`

run()
The main method of the class. This is triggered in the thread.start() call

Returns: None

stopNS() → None
This function closes both the broadcast and name servers. This is called in the shutdown procedure.

Returns: None

1.5 icbdrep.ReplicaNode module

class `icbdrep.ReplicaNode.`**ReplicaNode**(*rep_id: int*, *node_config*, *ns_config*)
Bases: `object`

addImage(*image_name: str*) → bool
Add an image to the node's repository

Args: image_name: the name of the image to be added.

Returns: a boolean with the sucess of the operation

deleteSnapshot(*image_name: str*, *snap_number: str*) → lib.icbdSnapshot.icbdSnapshot
Delete a snapshot from the repo and FS

Args: image_name: the name of the image snap_number: the number of the snapshot

Returns: the snapshot which as deleted

getImagesList() → typing.List[str]
Get the list of images present in the replica

Returns: a list of strings

getLastSnapshot(*image_name: str*) → lib.icbdSnapshot.icbdSnapshot
Return the last snapshot of the given image.

Args: image_name: the name of the image

Returns: an obj icbdSnapshot

getName() → str
Get the replica name

Returns: a string with the name

getReplicaBtrfsAddress() → typing.Tuple[str, int]
Return the IP and PORT address for the btrfs transfer.

Returns: A tuple with an IP and PORT

getReplicaID() → int
Get the replica ID number. This should be a integer that originates from the

Returns: the replica ID

getSnapshotList(*image_name: str*) → typing.List[lib.icbdSnapshot.icbdSnapshot]
Return the list of snapshots stored in the repo for the given image name. Case there are no snapshots the list returned is empty. Case the image in args ins't in the repo return None.

Args: image_name: Image name to get the snapshot list.

Returns: a list with the snapshots.

ping() → str
Responds to a ping request with "pong"

Returns: "pong"

poisonPill() → None
Shutdown message to the replica

Returns: None

prepareReceive(*image_name: str*, *snap_number: str*) → bool
This function should precede the receive() call. Checks if the node wants the image in question or if the snapshot is already present.

Args: image_name: the name of the image snap_number: the name of the snap

Returns: a bool that indicates if the replica will accept the receive

receive(*image_name: str*, *snap_number: str*, *compression: str = None*)
Receives a snapshot

Returns: None

1.6 icbdrep.icbdrepd module

1.7 lib.benchmarkinglib module

class `lib.benchmarkinglib.`**`Benchmark`**(*name*)

Bases: `object`

`addRun`(*run: lib.benchmarkinglib.Run*)

`get_name`()

`mean`()

`median`()

`stdev`()

class `lib.benchmarkinglib.`**`Run`**(*interfaceName*, *runNumber=-1*, *imageName='default'*)

Bases: `object`

`getBtrfsTransferBytes`()

Returns:

`getBtrfsTransferPackets`()

Returns:

`getBtrfsTransferRuntime`()

Returns:

`getGlobalTransferRuntime`()

Returns:

`getIcbdBootTransferBytes`()

Returns:

`getIcbdBootTransferPackets`()

Returns:

`getIcbdBootTransferRuntime`()

Returns:

`getIscsiTargetTransferBytes`()

Returns:

`getIscsiTargetTransferPackets`()

Returns:

`getIscsiTargetTransferRuntime`()

Returns:

`startTimmer`(*transferType*)

Start a timmer for one of the transfer counters.

Args: transferType: the type of the transfer to start counting time

Returns: call the appropriated function

`stopTimmer`(*transferType*)

Stop a timmer for one of the transfer counters.

Args: transferType: the type of the transfer to start counting time

Returns: call the appropriated function

class `lib.benchmarkinglib.`**`linuxNetworkTraffic`**

Bases: `object`

static getInterfaceStats(*interfaceName*)

Args: interfaceName:

Returns:

1.8 lib.btrfslib module

class `lib.btrfslib.`**`BtrfsFsCheck`**

Bases: `object`

static isBtrfsPath(*path: str*)

Check if a given path is in fact present in a BTRFS tree

!!Caution!! : This function does not takes into account the fact that the path might not be a valid one.

Args: path: the path to be checked

Returns: true if present, otherwise falses

static isBtrfsSubvolume(*path: str*)

Check if the given path is a BTRFS subvolume / snapshot.

Args: path: the path to be checked

Returns: True if a subvolume, otherwise false

static searchForSnapshots(*path: str*) → typing.List[str]

Search the directory , and gets the snapshots that are already present

Args: path: the directory to be searched

Returns: a List with the name of the snapshot

class `lib.btrfslib.`**`BtrfsTool`**

Bases: `object`

static delete(*path: str*) → None

Wrapper for the BTRFS Tools subvolume delete command.

The method receives a path and calls the btrfs subvolume delete for that path.

Args: path: the path to the subvolume to delete

Returns: None

static receive(*dst_path: str*, *src_port: int*, *compression: str = None*)

Wrapper for the BTRFS Tools receive() command.

This method opens a socket and listens for a connection Then receives a snapshot and redirect it to the stdin of the BTRFS receive

Args: dst_path: the path of the image to place the snapshot src_port: the port to listening for the transfer

Returns: None

static send(*src_path: str*, *dst_ip: str*, *dst_port: int*, *parent: str = None*, *compression: str = None*)

Wrapper for the BTRFS Tools send() command.

This method is BLOCKING, it will wait for the conclusion of the send command. It uses regular sockets to send to an endpoint the data from the snapshot.

Args: src_path: the path of the snapshot to be send dst_ip: the IP of the destiny socket dst_port: the Port the destiny is listening

Returns: None

static sendNonBlock(*src_path: str*, *dst_ip: str*, *dst_port: int*, *parent: str = None*, *compression: str = None*)

Wrapper for the BTRFS Tools send() command.

This method is NON BLOCKING, it will NOT wait for the conclusion of the send command. It uses regular sockets to send to an endpoint the data from the snapshot.

Args: src_path: the path of the snapshot to be send dst_ip: the IP of the destiny socket dst_port: the Port the destiny is listening

Returns: None

static sendSSH(*src_path: str*, *dst_ip: str*, *dst_port: int*, *parent: str = None*, *compression: str = None*)

Wrapper for the BTRFS Tools send() command.

This method is BLOCKING, it will wait for the conclusion of the send command. It uses regular sockets to send to an endpoint the data from the snapshot.

Args: src_path: the path of the snapshot to be send dst_ip: the IP of the destiny socket dst_port: the Port the destiny is listening

Returns: None

static setReadOnly(*path: str*, *state: bool*) → None

Wrapper for the BTRFS Tools property set read only command.

This method sets the the read only property for the given subvolume in the path.

Args: path: the path to the subvolume state: a boolean of the state of the read only

Returns: None

1.9 lib.compressionlib module

class `lib.compressionlib.`**compressionLib**

Bases: `object`

static checkCompression(*compression: str*) → bool

Check if the given compression algorithm is available to use in the lib.

Args: compression: A string with the algorithm to check

Returns: A bool representing the availability of the chosen algo.

class `lib.compressionlib.`**g_snappy**

Bases: `object`

static compressStream(*in_stream*, *out_stream*, *blocksize=65536*) → None

Uses the Google snappy compress function to compress a stream of bytes.

Takes an incoming file-like object and an outgoing file-like object, reads data from "in_stream", compresses it, and writes it to "out_stream". "in_stream" should support the read method, and "out_stream" should support the write method.

Args: in_stream: a stream of bytes out_stream: a compressed stream blocksize: [optional] the size used for the buffer in bytes

Returns: None

static compress_native(*in_stream*, *out_stream*, *blocksize=65536*) → None

Wrapper for the snappy native stream compression

Args: in_stream: a stream of bytes out_stream: a compressed stream blocksize: [optional] the size used for the buffer in bytes

Returns:

static decompressStream(*in_stream*, *out_stream*, *blocksize=65536*) → None

Uses the Google snappy decompress function to handle a compressed stream.

Takes an incoming file-like object and an outgoing file-like object, reads data from "in_stream", decompresses it, and writes it to "out_stream". "in_stream" should support the read method, and "out_stream" should support the write method.

Args: in_stream: a compressed stream out_stream: the original stream of bytes blocksize: [optional] the size used for the buffer in bytes

Returns:None

static decompress_native(*in_stream*, *out_stream*, *blocksize=65536*) → None

Wrapper for the snappy native stream decompression

Args: in_stream: a compressed stream out_stream: the original stream of bytes blocksize: [optional] the size used for the buffer in bytes

Returns:

class `lib.compressionlib.`**lz4**

Bases: `object`

static compressStream(*in_stream*, *out_stream*) → None

Uses the lz4 compress function to compress a stream of bytes

Takes an incoming file-like object and an outgoing file-like object, reads data from "in_stream", compresses it, and writes it to "out_stream". "in_stream" should support the read method, and "out_stream" should support the write method.

Args: in_stream: a bytes input stream to be compressed out_stream: the compressed stream

Returns: None

static decompressStream(*in_stream*, *out_stream*) → None

Uses the lz4 decompress function to decompress a stream of bytes

Takes an incoming file-like object and an outgoing file-like object, reads data from "in_stream", decompresses it, and writes it to "out_stream". "in_stream" should support the read method, and "out_stream" should support the write method.

Args: in_stream: a compressed stream out_stream: the original bytes

Returns: None

class `lib.compressionlib.`**z_lib**

Bases: `object`

static compress2(*in_stream*, *out_stream*)

!!IN TESTING!! !!DONT USE THIS!!

Args: in_stream: out_stream:

Returns:

static compressStream(*in_stream*, *out_stream*, *blocksize=32768*) → None

Uses the zlib compress function to compress a stream of bytes.

Takes an incoming file-like object and an outgoing file-like object, reads data from "in_stream", compresses it, and writes it to "out_stream". "in_stream" should support the read method, and "out_stream" should support the write method.

Args: in_stream: a stream of bytes out_stream: a compressed stream blocksize: [optional] the size used for the buffer in bytes

Returns: None

static decompress2(*in_stream*, *out_stream*)

!!IN TESTING!! !!DONT USE THIS!!

Args: in_stream: out_stream:

Returns:

static decompressStream(*in_stream*, *out_stream*, *blocksize=32768*) → None

Uses the zlib decompress function to handle a compressed stream.

Takes an incoming file-like object and an outgoing file-like object, reads data from "in_stream", decompresses it, and writes it to "out_stream". "in_stream" should support the read method, and "out_stream" should support the write method.

Args: in_stream: a compressed stream out_stream: the original stream of bytes blocksize: [optional] the size used for the buffer in bytes

Returns: None

1.10 lib.icbdSnapshot module

class lib.icbdSnapshot.**icbdSnapshot**(*mount_point: str*, *image_name: str*, *snapshot_number: str*, *icbd_boot_package_path: str*, *iscsi_target_folder: str*)

Bases: `object`

getICBDBootPackagePath()

Get a string with the full path to the iCBD Boot Package of the Image.

Returns: a string with the path

getISCSITarget()

Get a string with the path to the ISCSI target for this snapshot.

Returns: a string with the path

getImagePath() → str

Get a string with the formatted path, but without the snapshot number. This should be used as a destiny path

Returns: a string with the path in the format {/mountpoint/imagename}

getMountpointPath() → str

Get a string with only the mount point of the snapshot

Returns: the mountpoint

getPath() → str

Get a string with the full path of the snapshot, including the mountpoint and image name. Format: {mountpoint/imagename/snapshotnumber}

Returns: a string with the path

1.11 lib.restapilib module

class `lib.restapilib.`**`RestAPI`**(*port: int = 5009*)

Bases: `object`

iCBD-Replication Rest API Class

This instantiate the micro-framework Flack to provide a simple HTTP API for interacting with the system.

Note that every communication with this API uses JSON files. Responses are in JSON and an example can be found in the documentation of each method.

`api = <flask_restful.Api object>`

`app = <Flask 'lib.restapilib'>`

`deleteImageVersion`(*replica*, *imi*, *version*)

Delete a version of an iMI in a Replica Json response example:

Endpoint path : <IP>:<Port>/api/replicas/<replica>/imis/<imi>/versions/<version>/delete/

Returns:

`listImageVersionsByReplica`(*replica*, *imi*)

List the version of an iMI that are present in a Replica Json response example:

Endpoint path : <IP>:<Port>/api/replicas/<replica>/imis/<imi>/versions

Returns:

`listImagesByReplica`(*replica*)

List all iMIS present in a replica. Json response example:

Endpoint path : <IP>:<Port>/api/replicas/<replica>/imis

Returns:

`listReplicas`()

List all the replicas registered in the system. Json response example:

Endpoint path : <IP>:<Port>/api/replicas

Returns:

`listSystemImages`()

List all the iMIs present in the Master Node This will list all iMIs available to be transfered to any replica.

Json response example:

Endpoint path : <IP>:<Port>/api/master/imis

Returns:

`listSystemImagesVersions`(*imi*)

List all the versions of an iMIs present in the Master Node

Json response example:

Endpoint path : <IP>:<Port>/api/master/imis/<imi>/versions

Returns:

root()

Default root route endpoint. Mainly for testing

Endpoint path : <IP>:<Port>/api

Returns: a simple test string

sendImageVersionToReplica()

List all the versions of an iMIs present in the Master Node

Json response example:

Endpoint path : <IP>:<Port>/api/master/send?imi={imi}&version={version}&replica={replica}

Returns:

subscribeImage(*replica*, *imi*)

Replica subscribe to a iMI Json response example:

Endpoint path : <IP>:<Port>/api/replicas/<replica>/imis/subscribe/<imi>

Returns:

unsubscribeImage(*replica*, *imi*)

Replica unsubscribe to a iMI Json response example:

Endpoint path : <IP>:<Port>/api/replicas/<replica>/imis/unsubscribe/<imi>

Returns:

1.12 lib.serializerslib module

class `lib.serializerslib.`**icbdSnapshotSerializer**

Bases: `object`

static icbdSnapshot_class_to_dict(*obj: lib.icbdSnapshot.icbdSnapshot*)

static icbdSnapshot_dict_to_class(*class_name*, *dict*)

1.13 lib.sshlib module

class `lib.sshlib.`**sshTunnel**(*host*, *local_port*, *remote_port*)

Bases: `object`

createTunnel(*host*, *local_port*, *remote_port*)

1.14 lib.utillib module

class `lib.utillib.`**icbdUtil**

Bases: `object`

logHeading(*string*)

Big header for logger –["string"]——————

Args: string: a string to be placed inside the big header

Returns: the string encapsulated in the header

prettify(*obj*)
Return pretty representation of obj. Useful for debugging.

Args: obj: the object to prettify

Returns: a pretty representation of obj

1.15 exceptions.ImageRepoException module

exception `exceptions.ImageRepoException.`**BTRFSPathNotFoundException**(*message*)
Bases: `Exception`

Raise when a BTRFS Path is not in the File System

exception `exceptions.ImageRepoException.`**BTRFSSubvolumeNotFoundException**(*message*)
Bases: `Exception`

Raise when a BTRFS Subvolume is not in the File System

exception `exceptions.ImageRepoException.`**DirNotFoundException**(*message*)
Bases: `Exception`

Raise when a Directory is not in the File System

exception `exceptions.ImageRepoException.`**ImageAlreadyExistsException**(*message*)
Bases: `Exception`

Raise when a Images already is present in the repo

exception `exceptions.ImageRepoException.`**ImageNotFoundException**(*message*)
Bases: `Exception`

Raise when a Images is not found

exception `exceptions.ImageRepoException.`**SnapshotAlreadyExistsException**(*message*)
Bases: `Exception`

Raise when a Snapshot already is present in the repo

exception `exceptions.ImageRepoException.`**SnapshotNotFoundException**(*message*)
Bases: `Exception`

Raise when a Snapshot is not found

1.16 exceptions.ReplicasException module

exception `exceptions.ReplicasException.`**ReplicaNotFoundException**(*message*)
Bases: `Exception`

Raise when a replica is not found

1.17 tests.benchLibTests module

`tests.benchLibTests.`**dummyFunc**()

`tests.benchLibTests.`**main**()

`tests.benchLibTests.`**startCompleteRun**()

1.18 tests.pyroNSTests module

class `tests.pyroNSTests.`**`NamingTrasher`**(*nsuri*, *number*)

Bases: `threading.Thread`

`list`()

`listprefix`()

`listregex`()

`lookup`()

`register`()

`remove`()

`run`()

`tests.pyroNSTests.`**`main`**()

`tests.pyroNSTests.`**`randomname`**()

1.19 tests.utilTests module

class `tests.utilTests.`**`TestMount`**(*methodName='runTest'*)

Bases: `unittest.case.TestCase`

Our basic test class

`isBTRFS`(*path*, *assertVal*)

`isSubvolume`(*path*, *assertVal*)

`test_isBtrfsSet`()

`test_isSubvolumeSet`()

1.20 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX