VM_disk.py

1. create_extra_disk_image

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM
disk_name	String	Name of the extra disk
size	Int	Required size of the disk. This size is in GB.
datastore	Object	It contains all the details of the selected datastore

2. get_extra_disk_location

Parameters	**Data Type**	**Description**
datastore_id	Int	This belongs to the datastore id where the disk is located
vm_identity	String	This is the name of the VM, stored in the database
disk_name	String	Name of the disk whose location has to be found
get_disk_size	Boolean	

This is called MODEL by admin_model and admin_vm_model. This function returns the location of the extra disk (IF it is there in the extra disk location) plus the size of the disk

Location in controller machine is :
/mnt/datastore/vm_extra_disks/<diskname>.qcow2
Location in FILER is : /baadal/data/vm_extra_disks/<diskname>.qcow2

Return values:

Values	**Data Type**	**Description**
disk_image_path	String	This is system mount point (/mnt/datastore/vm_extra_disks/<diskname>.qcow2)
image_present	boolean	whether extra disk image is present at above path or not
disk_size	Int	size of the disk if it is present else 0

2. attach_disk

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM
disk_name	String	Name of the disk
hostip	String	IP address of the host in which that VM resides or new VM will reside
already_attached_disks	Int	ID of already attached disk to the VM
new_vm	Boolean	The VM (to which extra disk is attaching) is new

		or old one.

This function is called from MODEL (admin and common) and vm_helper. This function searches the domain using “vm_details” then searches the extra disk location of that VM. And then attach that disk to the VM.

Return Values : Int type = 0

4. serve_extra_disk_request

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM
disk_size	Int	Required size of the disk. This size is in GB. This parameter is passed on to “create_extra_disk_image
host_ip	String	IP address of the host in which that VM resides or new VM will reside
new_vm	boolean	The VM (to which extra disk is attaching) is new or old one.

It acts as a serving function for taking extra disk request from vm_helper. It chooses an appropriate datastore, then it creates extra disk image. And if creation is successful then it attaches that disk image to the VM

Return Values : Boolean type
False : If it serves extra disk request SUCCESSFULLY
True : If it FAILS

5. attach_extra_disk

Parameters	Data Type	Description
parameters	List	It has 2 values in it 1. VM ID 2. Disk Size

It takes the request with vm_id (to which extra disk has to be attached) plus disk size. Then it sends details to "serve_extra_disk_request" which attaches the disk to the VM (corresponding to that vm_id).

Return values:
If successful then return value is : {TASK_QUEUE_STATUS_SUCCESS (=3) with the message : "Attached extra disk successfully"}

If failed : {TASK_QUEUE_STATUS_FAILED (=4) with the message : "Your request for additional HDD could not be completed at this moment. Check logs."}

VM_network.py

1. get_private_ip_mac

Parameters	Data Type	Description
security_domain_id	Int	This belongs to the security domain to which that private ip belongs. security domains are : Research, Private, Infrastructure

First it searches the "**vlan**" corresponding to the security_domain_id. Then it searches the **"first" private_ip_pool object** from private_ip_pool table (in DB) which has no VM, no host and which belongs to the "**vlan**".

Return Values :

Values	Data Type	Description
private_ip	String	Private IP address from the “first” private_ip_pool object
mac_addr	String	MAC address from the “first” private_ip_pool object
vlan_name	String	vlan name corresponding to that private ip
vlan_tag	String	vlan tag corresponding to that private ip

2. choose_mac_ip

Parameters	Data Type	Description
vm_properties	List	It contains ● private_ip (String type) : IP that has to be given to new VM installed ● mac_addr (String type) : mac address that has to be given to new VM ● vlan_name (String type) : Name of the virtual lan under to that IP belongs ● vlan_tag (String

		type) ● public_ip_req (boolean type) : Whether public ip is requested or not for that new VM ● public_ip (String type) : public ip address that has to be assigned to the VM

It first gets the values from “get_private_ip_mac” and then updates the values of its parameter “vm_properties”.
Then it checks if public ip is requested or not. it requested then it assigned the public ip to the VM from the public ip pool. Else “Not Assigned” value is given to “public_ip” of vm_properties

Return Values : No return value

3. choose_mac_ip_vncport

Parameters	**Data Type**	**Description**
vm_properties	List	same as in “choose_mac_ip” plus ● vnc_port (Int type) : This is the vnc port which is assigned to the new VM

It first sends vm_properties parameter to the “choose_mac_ip” where the values gets updated. then it searches a random unassigned vnc_port value from the pool of vnc_ports and assigned it to the “vnc_port” of vm_properties.

Return Values : No return value

4. set_portgroup_in_vm

Parameters	**Data Type**	**Description**
domain	String	It is the name of the domain (VM)
portgroup	String	this belongs to the value of “portgroup” in XML of VM under “//interface/source”
host_ip	String	Ip address of the host under which the VM will be residing
vlan_tag	String	Virtual lan tag of vlan of the VM

It first enters the host using host_ip the n searched for the VM using “domain”, then converts the domain into xml object. Then it searched for the network element in that xml tree (//interface/source).
In the xml tree if “network” is not -1 then it sets the ‘portgroup’ to **portgroup**(parameter).
else if “bridge” is not -1 then it extracts ‘vlan’ and ‘tag’ from the sub tree element and sets the ‘tag’ to **“vlan_tag”** (parameter).

Return values : No return value

VM_utilization.py

1. get_dom_mem_usage

Parameters	Data Type	Description
dom_name	String	Name of the VM
host_ip	String	IP address of the host under which that VM resides

VMs run on host as individual processes. Memory utilization of the process is derived. It calls “execute_remote_cmd” of helper.py in which it sends the host_ip and command (using dom_name) and retreives the memory (RAM) usage of the domain.

Return values : Int type
RAM size in bytes.

2. get_dom_nw_usage

Parameters	Data Type	Description
dom_obj	Object	libvirt domain object. it contains all the details of the VM

It searches all the devices in the XML of domain (devices/interface/target). then for all devices it sums up the transmitted and received bytes.

Return Values :

Values	Data Type	Description
rx	Int	Received bytes to all the network interface devices of the VM
tx	Int	Transmitted bytes of all

		the network interface devices of the VM

3. get_dom_disk_usage

Parameters	**Data Type**	**Description**
dom_obj	Object	libvirt domain object. it contains all the details of the VM

It searched in the XML of the domain (devices/interface/target). Then sums up all the bytes read and write from and to all the devices.

Return values :

Values	**Data Type**	**Description**
bytesr	Int	total bytes read from disk
bytesw	Int	total bytes write to the disk

4. get_current_dom_resource_usage

Parameters	**Data Type**	**Description**
dom_obj	Object	libvirt domain object. it contains all the details of the VM
host_ip	String	IP address of the host under which that VM resides

It uses above 3 functions and creates domain usage statistics.

Return values: Object type
dom_stats : This is a list which contains following values

- rx : bytes received (net usage)
- tx : bytes transmitted (net usage)
- diskr : bytes read from disk (disk usage)
- diskw : bytes write to the disk (disk usage)
- memory : RAM usage in bytes
- cputime : cpu time given to the domain. This is extracted from dom_obj.info
- cpus : Count of vCPUs assigned to the VM, extracted from dom_obj.info

5. get_actual_usage

Parameters	**Data Type**	**Description**
dom_obj	Object	libvirt domain object. it contains all the details of the VM
host_ip	String	IP address of the host under which that VM resides

It uses **"get_current_dom_resource_usage"** to get the current domain statistics. then it calculates the actual cpu usage by using previous dom stats and current domain stats.

Return Value: List type
usage : This is a list contains following values

- ram : RAM usage in bytes
- cpu : Actual CPU usage by using previous stats and current stats
- tx : Network transmitted bytes
- rx : Network received bytes
- dr : Bytes read from the disk

- dw : Bytes written to the disk

VM_snapshot.py

1. vm_has_snapshots

Parameters	**Data Type**	**Description**
vm_id	Int	ID of the VM of which snapshot has to be taken

It tells whether there is a snapshot exists corresponding to the VM with ID=vm_id.

Return Value : Boolean type
True : if snapshot exists
False : if snapshot does not exists

2. snapshot

Parameters	**Data Type**	**Description**
parameters	List	It contains vm_id, snapshot_type.

As name suggests, this function is used to take the snapshot of the VM. It extracts the vm details from DB using vm_id. Then it checks whether the VM is pingable or not. If not pingable then it does nothing. But if it is pingable then it takes the snapshot of that VM and inserts the value of that snapshot with VM details in snapshot table in DB.

Return values:
If successful then return value is : {TASK_QUEUE_STATUS_SUCCESS (=3) with the message : "Snapshotted successfully"}

If failed : {TASK_QUEUE_STATUS_FAILED (=4) with the message : “Unable to ping VM before snapshotting : <private ip of VM>"}

3. revert

Parameters	**Data Type**	**Description**
parameters	List	It contains vm_id, snapshot_id

It uses vm_id to get the details of the VM, snapshot_id to get the snapshot details. Then get the domain object of the VM using VM name and snapshot object of the snapshot using snapshot name. then reverts the domain to the snapshot using the snapshot object.

Return values:
If successful then return value is : {TASK_QUEUE_STATUS_SUCCESS (=3) with the message : "Reverted to snapshot successfully"}

If failed : {TASK_QUEUE_STATUS_FAILED (=4) with the message : “Task Status: FAILED Error: <EXCEPTION MESSAGE>"}

4. delete_snapshot

Parameters	**Data Type**	**Description**
parameters	List	It contains vm_id, snapshot_id

As name suggests, it uses to delete the snapshot corresponding to the VM of ID=vm_id. It retrieves snapshot object using snapshot_id then deleted the snapshot. And also deletes the entry from snapshot table corresponding to snapshot_id too.

Return values:
If successful then return value is : {TASK_QUEUE_STATUS_SUCCESS (=3) with the message : "Deleted snapshot successfully"}

If failed : {TASK_QUEUE_STATUS_FAILED (=4) with the message : “Task Status: FAILED Error: <EXCEPTION MESSAGE>"}

VM_migration.py

1. migrate_domain_with_snapshots

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM
destination_host_ip	String	IP address of the host where the VM will be migrated
domain	Object	VM object, it contains all the information of the VM
domain_snapshots_list	List	List of snapshot names corresponding to a VM
current_snapshot_name	String	Name of the current snapshot which is to be taken
flags	Boolean	This indicates the type of migration. Few types can be LIVE, PAUSED, OFFLINE etc.
vm_backup_during_migration	String	Path to directory at which snapshots of the VM are saved. Path is

		/mnt/datastore/vm_migration_data/

This function migrates the snapshots of the VM which is to be migrated.
First it dumps the XML of all the snapshots of the VM to the FILER at path: /mnt/datastore/vm_migration_data/. Then it deleted all the snapshots of the VM from the current host. Then it redefines all the snapshots in destination host. And then takes the current snapshot of the VM in destination host

Return Values : NULL

2. clean_migration_directory

Parameters	**Data Type**	**Description**
vm_backup_during_migration	String	Path to directory at which snapshots of the VM are saved. Path is /mnt/datastore/vm_migration_data/

It deletes directory created for storing dumpxml of vm snapshots. Path is /mnt/datastore/vm_migration_data/

Return Values : NULL

3. undo_migration

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM
domain_snapshots_list	List	List of snapshot names corresponding to a VM

current_snapshot_name	String	Name of the current snapshot which is to be taken
vm_backup_during_migration	String	Path to directory at which snapshots of the VM are saved. Path is /mnt/datastore/vm_migration_data/

It simply redefines the snapshots of the VM on the source host so that its migration could be cancelled. Then it take current snapshot of the VM on source only and deletes the dump XML from the migration directory

Return Values : NULL

4. migrate_domain

Parameters	**Data Type**	**Description**
vm_id	Int	ID of the VM which to be migrated
destination_host_id	Int	ID of the host to which VM will be migrated
live_migration	Boolean	It shows whether the VM will be migrated in active(LIVE) state or not

It gets the vm details by using vm_id. Then gets the ip address of the host by using destination_host_id. If host id is null then it searches the best host (best host means host with enough RAM and CPU available). After that it sets the flag for the type of migration (LIVE, OFFLINE etc). Then it collects the snapshots of the VM and migrate them first to the destination host. And then it migrates the VM.

Return Values :
If successful then return value is : {TASK_QUEUE_STATUS_SUCCESS (=3) with the message : "<VM name> is migrated successfully"}

If failed : {TASK_QUEUE_STATUS_FAILED (=4) with the message : “Task Status: FAILED Error: <EXCEPTION MESSAGE>"}

5. migrate

Parameters	**Data Type**	**Description**
parameters	List	It has 3 values. **vm_id (Int Type)**: ID of VM which is to be migrated. **destination_host (Int type)** : ID of the host where the VM has to be migrated. **live_migration(boolean type) :** Migration of VM in active state of not

This is the main function called from outside. It decides the type of migration initially. Then it calls the migrate_domains with vm_id, destination id and flag

Return Values :
Return value **“migrate_domain”**

6. migrate_clone_to_new_host

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM

cloned_vm_details	Object	Details of cloned VM of the actual VM
new_host_id_for_cloned_vm	Int	ID of the host where the cloned VM will be transferred
vm_properties	List	It has several values related to VM. In this function **'host'** value will get updated to 'new_host_id_for_cloned_vm'

It searched the ip of the new host on the basis of new host id. It migrates the cloned VM to the new host. and then update the records of cloned vm details and vm properties [host].

Return values : Boolean type
If migration successful then TRUE
Else FALSE

VM_security_domain.py

1. update_security_domain

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM
security_domain_id	Int	ID of the security domain under which private ip of VM lies
xmlDesc	XML object	It is the XML tree of the VM

It fetches the private ip information by using 'get_private_ip_mac' of vm_network.py. Then sets the address in XML tree of VM with vlan name (extracted from private ip info). Then updates the vm_data and private_ip_pool table in DB

Return Values : String Type
String form of XML tree

VM_clone.py

1. get_clone_properties

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM
cloned_vm_details	Object	Details of cloned VM of the actual VM
vm_properties	List	It has several values related to VM. It has following values in it: ● datastore (Object type) : Datastore where VM resides ● security_domain (Int type) : ID of security domain of private ip of VM ● public_ip_req(boolean type) : Whether public ip is requested or not ● mac_addr(String type) : Mac address of the VM ● private_ip(String

		type) : Private ip of the VM ● vnc_port(Int type) : VNC port value of the VM ● template(Int type) : Template ID of the VM ● vm_host_details(Object type) : Host details ● host(Int type) : host Id

First it finds mac address, ip address, vnc port for the cloned VM, Template and host of parent vm. Then it creates a directory for the cloned VM (/mnt/datastore/vm_images). Then it creates a folder for additional disks of the cloned VM (/mnt/datastore/vm_extra_disks).

Return Values : String type
clone_file_parameters : This string contains the cloned VM image path and extra disk path (if exists).

2. clone

Parameters	**Data Type**	**Description**
vmid	Int	ID of the VM of which clone has to be done

It fetches the vm details using vmid. Then calls get_clone_details to get the clone file parameter. Then it checks the host if it has enough RAM and CPU to support the clones VM. If it has then VM is cloned in the same host else exception is raised that host is exhausted…..

After it is cloned in the same host, new host id is searched again. If new id is not same as current host id then this clone is migrated to the new host.

Return values: String type
If successful then return value is : {TASK_QUEUE_STATUS_SUCCESS (=3) with any one message out of 5 following message :

- "Task Status: SUCCESS Message: Cloned successfully."}
- "Task Status: SUCCESS Message: Cloned successfully. Found new host and migrated successfully."}
- "Task Status: SUCCESS Message: Cloned successfully.Found new host but not migrated successfully."}
- "Task Status: SUCCESS Message: Cloned successfully. New host selected to migrate cloned vm is same as the host on which it currently resides."}
- "Task Status: SUCCESS Message: Cloned successfully. Could not find host to migrate cloned vm."}

If failed : {TASK_QUEUE_STATUS_FAILED (=4) with the message : “Task Status: FAILED Error: <EXCEPTION MESSAGE>"}

VM_helper.py

1. choose_datastore : No Parameter

It is used to choose datastore from a list of available datastores.

Return values: Object type
first_elts : It is the datastore object

2. allocate_vm_properties

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM

It allocates vm properties (datastore, host, ip address, mac address, vnc port, ram, vcpus).

Return values : List Type
vm_properties : It contains all the properties of the VM

3. create_vm_image

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM
datastore	Object	It contains all the information of selected datastore

It creates directory for the VM at /mnt/datastore/vm_image. Then it searches for the location of template image that the user has requested for its VM using datastore parameter.

Return values :

Values	**Data Type**	**Description**
template	Object	It contains all the information related to the template of the VM
vm_image_name	String	This is the path to the image name (qcow2 image)

4. get_install_command

Parameters	Data Type	Description
vm_details	Object	It contains all the details of a VM
vm_image_location	String	Path to the qcow2 image of the VM (/mnt/datastore/vm_images)
vm_properties	List	It contains all the properties of the VM.

This function takes the values from its parameters and creates VM installation command.

Return Values : String type
install_command : This is a string which is a virt-install command.

5. generate_xml

Parameters	Data Type	Description
diskpath	String	This is the path of the disk of the VM kept at /mnt/datastore/vm_extra_disks/ .
target_disk	String	This is the name of the disk named as “vd*”

This function puts the disk path and target disk name in the appropriate place in XML

Return Values : XML object type
It returns the XML tree (root)

6. launch_vm_on_host

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM
vm_image_location	String	Path to the vm image. /mnt/datastore/vm_images
vm_properties	List	It contains all the Information related to the VM

This function first creates the installation command using its all 3 parameters. Then it fetches the host_ip by using host value of vm_properties. Then it calls the execution of this insllation command on host using host ip. Then it checks if VM(user) has also requested for extra disk. If yes then it calls **serve_extra_disk_request** function of vm_disk.py.

Return values : String type
attach_disk_status_message : “Attached extra disk successfully.” (on successfully serving extra disk request)

OR

“Attached extra disk failed.” (on failing serving extra disk request)

7. check_if_vm_defined

Parameters	**Data Type**	**Description**
hostip	String	IP address of the host
vmname	String	Name of the VM about which this function will

		check

This function checks whether the VM with vmname is already defined in the host with IP=hostip.

Return values : Boolean type
vm_defined : True, if vm defined. Else False

8. free_vm_properties

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM
vm_properties	List	It contains all the Information related to the VM

This function deletes the VM from the host. Deletes the VM entry from the database. Deletes the disk and image of the VM from the vm disk and image location.

Return value : NULL

9. update_db_after_vm_installation

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM
vm_properties	List	It contains all the Information related to the VM

parent_id	Int	

10.install

Parameters	**Data Type**	**Description**
parameters	List	This contains all the values required by the VM installation command. It has following values in it: ● vm_id (Int Type) : ID of the VM which is going to install

This function first fetches the vm details from DB using vm_id. Then it calls allocate_vm_properties function which sets the vm_properties by using the values from vm details. Then it creates vm image. then launches the vm on host. After vm is installed successfully, it updates the DB.

Return values : String type
If successful then return value is : {TASK_QUEUE_STATUS_SUCCESS (=3) with the message : "VM is installed successfully"}

If failed : {TASK_QUEUE_STATUS_FAILED (=4) with the message : “Task Status: FAILED Error: <EXCEPTION MESSAGE>"}

11.start

Parameters	**Data Type**	**Description**
parameters	List	This contains all the values required by the VM start command. It has following values in it: ● vm_id (Int Type) : ID of the VM which is going to start

First it fetches the vm details from the DB using vm_id. Then it connects to the host using the host ip from vm details. And starts the VM and updates the vm status to running.

Return values : String type
If successful then return value is : {TASK_QUEUE_STATUS_SUCCESS (=3) with the message : "<VM_NAME> is started successfully"}

If failed : {TASK_QUEUE_STATUS_FAILED (=4) with the message : <EXCEPTION MESSAGE>"}

12.suspend

Parameters	**Data Type**	**Description**
parameters	List	This contains all the values required by the VM suspend command. It has following values in it: ● vm_id (Int Type) :

		ID of the VM which is going to suspend

First it fetches the vm details from the DB using vm_id. Then it connects to the host using the host ip from vm details and suspends the VM and updates the DB with vm is suspend

Return values : String type
If successful then return value is : {TASK_QUEUE_STATUS_SUCCESS (=3) with the message : "<VM_NAME> is suspend successfully"}

If failed : {TASK_QUEUE_STATUS_FAILED (=4) with the message : <EXCEPTION MESSAGE>"}

13.resume

Parameters	**Data Type**	**Description**
parameters	List	This contains all the values required by the VM resume command. It has following values in it: ● vm_id (Int Type) : ID of the VM which is going to resume

First it fetches the vm details from the DB using vm_id. Then it connects to the host using the host ip from vm details and suspends the VM and updates the DB with vm is running

Return values : String type

If successful then return value is : {TASK_QUEUE_STATUS_SUCCESS (=3) with the message : "<VM_NAME> is resumed successfully"}

If failed : {TASK_QUEUE_STATUS_FAILED (=4) with the message : <EXCEPTION MESSAGE>"}

14.destroy

Parameters	**Data Type**	**Description**
parameters	List	This contains all the values required by the VM destroy command. It has following values in it: ● vm_id (Int Type) : ID of the VM which is going to destroy

It forcefully shut down the VM. First it fetches the vm details from the DB using vm_id. Then it connects to the host using the host ip from vm details and suspends the VM and updates the DB with vm is shut down

Return values : String type
If successful then return value is : {TASK_QUEUE_STATUS_SUCCESS (=3) with the message : "<VM_NAME> is destroyed successfully"}

If failed : {TASK_QUEUE_STATUS_FAILED (=4) with the message : <EXCEPTION MESSAGE>"}

15.delete

Parameters	**Data Type**	**Description**
parameters	List	This contains all the values required by the VM deletes command. It has following values in it: ● vm_id (Int Type) : ID of the VM which is going to delete

First it fetches the vm details from the DB using vm_id. Then it connects to the host using the host ip from vm details and suspends the VM and removes the VM entry from the DB

Return values : String type
If successful then return value is : {TASK_QUEUE_STATUS_SUCCESS (=3) with the message : "<VM_NAME> is deleted successfully"}

If failed : {TASK_QUEUE_STATUS_FAILED (=4) with the message : <EXCEPTION MESSAGE>"}

16.clean_up_database_after_vm_deletion

Parameters	**Data Type**	**Description**
vm_details	Object	It contains all the details of a VM

When VM delete command is fired, it calls this function to delete VM image, vm disk etc from the filer.

Return values : No return value

17.edit_vm_config

Parameters	**Data Type**	**Description**
parameters	List	This contains all the values required by the VM deletes command. It has following values in it: ● vm_id (Int Type) : ID of the VM ● vcpus (Int type) : Count of cpus given to the VM ● ram (Int type) : Count(in GB) of RAM given to VM ● public_ip (String type) : Public ip assigned to the VM ● security_domain (Int type) : ID of the security domain under which VM lies

First it fetches the vm details from the DB using vm_id. Then it connects to the host using the host ip from vm details. Then it edits the vcpu, ram, public_ip and security domain (whichever is there in the parameters) as per request from the user(admin, faculty, org-admin).

Return values : String type
If successful then return value is : {TASK_QUEUE_STATUS_SUCCESS (=3) with the message :“Edited <resource name (which is requested to edit)> successfully.”}
Above message has multiple appends of Edited messages if multiple resource edit requested.

If exception : {TASK_QUEUE_STATUS_FAILED (=4) with the message : <EXCEPTION MESSAGE>"}

18.get_vm_image_location

Parameters	**Data Type**	**Description**
datastore_id	Int	ID of the datastore where the VM resides
vm_identity	String	Name of the VM

This function fetches the datastore by using datastore _id. then creates the path string to the VM using datastore and vm name

Return values : string type

Values	**Data Type**	**Description**
vm_image_name	String	this is the path to the qcow2 image of the VM. /mnt/datastore/vm_images/<vm_name>.qcow2
image_present	Boolean	If VM present at the location or not

19.launch_existing_vm_image

Parameters	Data Type	Description
vm_details	Object	It contains all the details of a VM

This function as name suggests launches the VM on the host. It takes out the vm installation related values from vm_details then it calls launch_vm_on_host function which ultimately launches the image on appropriate host. And at last updates the vm values in DB

Return values : No return values

vm_rrd.py

1. update_vm_rrd

Parameters	Data Type	Description
dom	Object	It is the libvirt domain object
active_dom_ids	List	List of all active domain ids
host_ip	String	IP address of the host

this function searches the dom.ID() in the list of active domain ids. If found then updates the RRD file of that VM in the filer

Return value : No return value

Host Operation

host_power.py

1. host_power_operation : No parameter

This function checks for the live host in which no vm is running. If it found such host then it turn off the extra hosts and turn on if host is required.

Return values : No value

2. host_power_up

Parameters	Data Type	Description
host_data	Object	it contains all host related data in it

As name suggests this function powers up the host which is requested in the host_data.

Return values : No value

3. host_power_down

Parameters	Data Type	Description
host_data	Object	it contains all host related data in it

As name suggests this function shutdown the host which is requested in the host_data

Return values : No value

host_resource_details.py

1. get_host_cpu

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

This function fetches the count of CPU cores of the host by using IP address of the host

Return values : Int Type
Count of the CPU cores of the host

2. get_host_ram

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

This function fetches the RAM(in GB) of the host by using IP address of the host

Return values : Int Type
Count of the RAM (GB) of the host

3. get_host_hdd

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

This function fetches disk size(in GB) of the host by using IP address of the host

Return values : Int Type

Size of the disk (GB)

host_utilization.py

1. get_host_cpu_usage

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

This function captures the cpu usage of the host by using “iostat” command.

Return value : float type
It returns the sum of the % of CPU usage by (user + nice + system)

2. get_host_disk_usage

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

This function captures the input/output statistics of the disk of the host by using “iostat” command.

Return value : float type
It returns the sum of the rate of KB read and KB write on disk

3. get_host_mem_usage

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

This function captures memory usage of the host by using “top” command.

Return value : Int type
It return the value of RAM in KB

4. get_host_nw_usage

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

This function captures the network usage of the host by using “ifconfig” command. Usage includes bytes read and write on the network

Return value :

Value	**Data Type**	**Description**
rx	Int	bytes read on the network
tx	Int	bytes write on the network

5. get_host_resources_usage

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

This is the main function which is call from outside to capture host resource usage statistics. It uses above functions to get various resources usage of the host.

Return value : List type
host_usage : This is a list which contains usage of all resources by the host

host_vms.py

1. respawn_dangling_vms

Parameters	**Data Type**	**Description**
host_id	Int	ID of the host

This function fetches the data of all the VM hosted on down host with Id=host_id. And respawn all the VMs on other appropriate hosts. Then revert the VM to its recent snapshot if snapshot exists for that vm

Return values : No return value

2. has_running_vm

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

this function checks that the host with IP=host_ip has running VM in it or not

Return value : Boolean type
True : if running vm exists
False : if not exists

3. delete_orhan_vm

Parameters	**Data Type**	**Description**

vm_name	String	Name of the orphan VM
host_id	Int	ID of the host

vm_name is the name of the orphan vm. This function deletes the orphan vm from host. first it shut it down and then removes it from the host
Return value : No return value

4. migrate_all_vms_from_host

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

This function is use to migrate all the inactive and active VMs from host with IP=host_ip to other host. It adds migration of all the vms to the task queue which later on execute by task scheduler of baadal.

Return value : No return value

host_helper.py

1. find_new_host

Parameters	**Data Type**	**Description**
RAM	Int	This is the RAM value required by the VM
vCPU	Int	This is the count of CPU required by the VM

This function is called when there is a VM going to launch on some host. This function is called to search the appropriate host which has enough amount of

memory and cpu cores left to support VM with memory value=RAM and cpu value=vCPU.

Return values : Int Type
host id : This function returns the host id of the first best suited host

2. host_resources_used

Parameters	**Data Type**	**Description**
host_id	Int	ID of the host

This function is used to fetch the memory and cpu already used by other processes in a host.

Return value :

Value	**Data Type**	**Description**
RAM	Int	Used RAM value in GB
CPU	Int	Used CPU core count

3. check_host_status

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

This function checks if the host is pingable or not by pinging IP address of the host

Return value : Int type

If pingable : Return value is HOST_STATUS_UP=1
If not pingable : Return value is HOST_STATUS_DOWN=0

4. is_host_available

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

It checks the availability of host by executing the “pwd” command on the host.

Return value : boolean type
True : if host is available
False : if not available

5. get_host_mac_address

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

This function fetches the mac address of the host by using hosts ip address and executing ifconfig command in it.

Return value : String type
mac_addr : MAC address of the host

6. get_host_type

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

This function checks the type of host, if it is virtual machine or physical machine.

Return value : String type
HOST_TYPE_VIRTUAL (=”Virtual”) : if host is a virtual machine.
HOST_TYPE_PHYSICAL (=”Physical”) : if host is a physical machine.

7. check_host_service_status

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

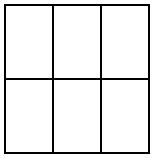
This function checks the libvirtd service and OVS switch service are running on the host or not.

Return value : boolean type
True : If both services are running on the host
False : If any one of them is not running

8. host_status_sanity_check : No parameter

This function checks the status of all the hosts of the baadal system. If host status is shutdown then it calls the respawn_dangling_vms function.

Return value : No return value



9. get_host_domains

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

This function Establishes a read only remote connection to libvirtd and Finds out all domains running and not running on that host

Return value : List type
List of all the domains of that host with ip address=host_ip

10. add_migrate_task_to_queue

Parameters	**Data Type**	**Description**
vm_id	Int	ID of the VM which is going to be migrated
dest_host_id	Int	ID of the destination host to which VM will be migrated
live_migration	boolean	Whether VM is migrated in running condition or not

This function is used to add migration task in the task queue. this queue is executed by the task_scheduler.

Return value : No return value

11. get_active_hosts : No parameter

This function checks the host table in the database for all the host those are in power up.

Return value : List type
It returns the list of all the hosts whose status is UP. Each element in the list is an object of host which contains all the details related to that host

host_networking_graph.py

1. get_latency_btw_hosts

Parameters	**Data Type**	**Description**
next_host_ip	String	IP address of the other host to which latency will be calculated
host_ip	String	IP address of the host from which latency will be calculated

This function uses “netperf” command to calculate the latency between two hosts from the host with IP=host_ip.

Return value : String type
Latency is calculated in form of float type (with decimal up to two points, 0.2f). But it is converted to string then returns.

2. get_bandwidth_of_host

Parameters	**Data Type**	**Description**

host_ip	String	IP address of the host

This function uses “lshw” command to calculate the bandwidth of the host.

Return value : String type
It returns the bandwidth in terms of integer value converted to string, with Gbit/s. for example “1Gbit/s”

3. get_throughput_btw_hosts

Parameters	**Data Type**	**Description**
next_host_ip	String	IP address of the host to which throughput has to be calculated
host_ip	String	IP address of the host from which throughput will be calculated

This function uses “netperf” command to calculate the throughput from host(host_ip) to next host.

Return value : String type
Return value is the throughput in Megabits. for example, “1M”

4. generate_axis

Parameters	**Data Type**	**Description**
No parameter		

This function is a recursive function. It randomly chooses X and Y coordinate from the range of 0-20. And put them in a sublist, like [X1,Y1,X2,Y2..] and so on.

Return value : List type
It return a List of X and Y coordinate in form of [X1,Y1,X2,Y2….]

5. check_sublist

Parameters	**Data Type**	**Description**
search1	Int	This is the value of X coordinate
search2	Int	This is the value of Y coordinate
sublist	List	List of X and Y, [X1,Y1,X2,Y2….]

This function checks whether X and Y are there in the sublist.

Return value : Int type
Returns 1 if found

6. node_attribute

Parameters	**Data Type**	**Description**
i	Int	Basically this represents the ID for the Node (host)
host_name	String	Name of the host
host_ip	String	IP address of the host

This function sets the attribute of the node(host). It does following thing in sequence:

- Get the bandwidth of the host.
- creates a list node_data. and sets the “id” = i
- creates a list of X and Y coordinate by calling generate_axis()
- sets ‘x’ and ‘y’ of nde_data to X and Y coordinate from list
- sets ‘label’ of node_data to the bandwidth of the host
- sets ‘size’ to 4

Return value : No return value

7. edge_attribute

Parameters	**Data Type**	**Description**
next_host_ip	String	IP address of the other host
host_ip	String	IP address of the host
k	Int	ID of the edge
i	Int	Source of the edge
j	Int	Destination of the edge

This function sets the attribute of the edge(between hosts). It does following thing in sequence:

- first it gets the latency between the hosts
- gets the throughput between hosts
- make a list edge_data and sets ‘id’ to k (in string format)
- sets ‘source’ to i
- sets ‘target’ to j
- sets ‘label’ to latency : bandwidth

Return value : No value

8. collect_data_from_host

Parameters	Data Type	Description
host_ip_list	List	List of the active Host IPs
host_name_list	List	List of the active Host names

this function uses above functions, creates node_attribute and edge_attribute and creates the graph using web2py.

Return value : No value

host_rrd.py

1. update_host_rrd

Parameters	Data Type	Description
host_ip	String	IP address of the host

this function fetches the rrd data from get_rrd_file() of rrd_graph.py. then fetches the host resource usage and updates its rrd file

Return value : No return value

helper.py

1. get_context_path : No parameter

Return value : String type

This function return the path to the file

2. get_config_file : No parameter

Return value : Parsed object

This function parses the config file (static/baadalapp.cfg) for the constants which are defined in the baadalapp.cfg file.

3. get_datetime : No parameter

Return value : DateTime type

It return the current time

4. get_constant

Parameters	**Data Type**	**Description**
constant_name	String	Name of the constant

This function searches the constant_name string in DB (constants table) and fetches the value corresponding to that constant

Return value : String type

Value of the constant

5. update_constant

Parameters	**Data Type**	**Description**
constant_name	String	Name of the constant
constant_value	String	Value of the constant

this function updates the constant table in DB with constant value corresponding to the constant name.

Return value : No return value

6. execute_remote_cmd

Parameters	**Data Type**	**Description**
machine_ip	String	IP address of the host
user_name	String	User name of the host machine. this is “root”
command	String	this is the command which to be executed on the host machine
password	String	Password o the username of the host. Passwordless ssh is used so password can be none for the “root”
ret_list	boolean	

this function establishes ssh connection with the host using paramiko API. then executes the command on the host remotely and fetches the output.

Return value : Any type
Return value depends on the command nature

7. execute_remote_bulk_cmd

Parameters	**Data Type**	**Description**

machine_ip	String	IP address of the host
user_name	String	User name of the host machine. this is “root”
command	String	this is the command which to be executed on the host machine
password	String	Password o the username of the host. Passwordless ssh is used so password can be none for the “root”

Does same as above function. It invokes the shell of host

Return value : Any type
Return value depends on the command nature

8. log_exception

Parameters	**Data Type**	**Description**
message	String	the message related to exception
log_handler	handler	describes the handler for the log

this function logs the exception of the system in log files

Return value : String type
Returns the Traceback path of the exception

class IS_MAC_ADDRESS
this class is use as a validator to check if a string is a valid mac address

network_helper.py

1. is_valid_ipv4

Parameters	**Data Type**	**Description**
value	String	IPV4 address

This checks whether ‘value’ is a valid 32 bit IP address or not.

Return value : boolean type
True : if valid IPv4 address
False : If not

2. validate_ip_range

Parameters	**Data Type**	**Description**
ipFrom	String	Starting IP address of the range
ipTo	String	Ending IP address of the range

This function validates if the range given is valid or not. Valid means ipFrom should be smaller than ipTo in terms of IP address format

Return value : Boolean type
True : if valid range
False :If not

3. get_ips_in_range

Parameters	Data Type	Description
ipFrom	String	Starting IP address of the range
ipTo	String	Ending IP address of the range

this function creates a list of all the IPs which are between ipFrom and ipTo including these two also

Return value : List type
List of IP addresses

4. generate_random_mac : No parameter

This function generates the random 48 bit MAC address.

Return value : String type
MAC address

5. create_dhcp_bulk_entry

Parameters	Data Type	Description
dhcp_info_list	List	dhcp information from config file (static/baadalapp.cfg)

It Creates bulk entry into DHCP. It gets list of tuple containing (host_name, mac_addr, ip_addr) and make entry into /etc/dhcp/dhcpd.conf file and restarts the isc-dhcp-server to update the entry

Return value : No return value

6. create_dhcp_entry

Parameters	**Data Type**	**Description**
host_name	String	Name of the host
mac_addr	String	MAC address of the host
ip_addr	String	IP of the host

this does same as above function, but it creates a single entry in the dhcpd.conf file

Return value : No return value

7. remove_dhcp_entry

Parameters	**Data Type**	**Description**
host_name	String	Name of the host
mac_addr	String	MAC address of the host
ip_addr	String	IP of the host

Does same as above function but use to create entry on remote machine.
Return value : No return value

8. is_pingable

Parameters	**Data Type**	**Description**
ip	String	ip address of system

Checks if ip is pingable or not

Return value : boolean type
True : If pingable
False : If not

nat_mapper.py

1. get_nat_details : No parameter

This function fetches the nat data from config file (static/baadalapp.cfg)
Return value :

Value	**Data Type**	**Description**
nat_type	String	Type of nat. Here it is software nat
nat_ip	String	IP of nat server
nat_user	String	user of nat server. “root”

2. get_ip_tables_command

Parameters	**Data Type**	**Description**
command_type	String	Command related to nat
source_ip	String	IP of source
destination_ip	String	IP of destination
source_port	Int	source port
destination_port	Int	destination port

This function Construct command string or updating IP tables. command string created by this function uses iptables command.

Return value : String type
Command string

3. get_interfaces_command

Parameters	**Data Type**	**Description**
command_type	String	Command related to nat
source_ip	String	IP of source

This function Construct command for updating interfaces file/

Return value : string type
returns the interface command

4. create_mapping

Parameters	**Data Type**	**Description**
command_type	String	Command related to nat
source_ip	String	IP of source
destination_ip	String	IP of destination
source_port	Int	source port
destination_port	Int	destination port

This function is to create mappings in NAT. If NAT type is software_nat then for creating public - private IP mapping:

- Create the alias on NAT that will listen on the public IP.
- Create rules in iptables to forward the traffic coming on public IP to private IP and vice versa.

And for providing VNC access:

- Check if the NAT is listening on the public IP to be used for VNC access.

- If NAT is not listening on the desired public IP, create an alias to listen on that IP.
- Create rules in iptables to forward the VNC traffic to the host.

Return value : No return value

5. remove_mapping

Parameters	**Data Type**	**Description**
source_ip	String	IP of source
destination_ip	String	IP of destination
source_port	Int	source port
destination_port	Int	destination port

This function to remove mapping from NAT. If NAT type is software_nat then for removing public IP - private IP mapping:

- Remove the alias on NAT listening on the public IP.
- Delete rules from iptables to forward the traffic coming on public IP to private IP and vice versa.
- Flush the entries from DB and make public IP free.

And for revoking VNC access:

- Delete rules from iptables to forward the VNC traffic to the host.
- Update DB to make the VNC access inactive.

Return value : No return value

6. clear_all_nat_mappings

Parameters	**Data Type**	**Description**

db	Object	this object points to the database

This function is use to flush all mappings from NAT. For all public IP - private IP mappings, Delete aliases. It flushes all rules from iptables and then updates DB.

Return value : No return value

7. clear_all_timedout_vnc_mappings : No parameter

This function is to delete all timed-out VNC mappings from NAT. First it get all active VNC mappings from DB. Then it deletes the VNC mapping from NAT if the duration of access has past its requested time duration, then deletes rules from iptables on NAT box. It updates the DB for each VNC access.

Return value : No return value

8. create_vnc_mapping_in_nat

Parameters	**Data Type**	**Description**
vm_id	Int	ID of the VM

This function is use to create mapping in NAT for VNC access for the VM. It inserts the vnc port for the VM in DB.

Return value : No return value

9. create_public_ip_mapping_in_nat

Parameters	**Data Type**	**Description**
vm_id	Int	ID of the VM

This function to create mapping in NAT for public IP - private IP for the VM with id=vm_id. Then it updates DB also

Return value : No return value

10.remove_vnc_mapping_from_nat

Parameters	**Data Type**	**Description**
vm_id	Int	ID of the VM

This function to remove mapping for VNC access from NAT of the VM.

Return value : No return value

11.remove_public_ip_mapping_from_nat

Parameters	**Data Type**	**Description**
vm_id	Int	ID of the VM

This function to remove public IP - private IP mapping from NAT of the VM. Then it updates the DB.

Return value : No return value

maintenance.py

1. snapshot_and_destroy

Parameters	**Data Type**	**Description**
vm_id	Int	ID of the VM
vm_identity	String	Name of the VM
vm_status	Int	Status of VM. whether it

		is running, shutdown etc

This function takes the snapshot and destroys the VM. If VM is running then it takes the snapshot first and then destroys it, it VM is not in running condition then it directly destroys it

Return value : No return value

2. shutdown_baadal

This function shutdown the baadal process. It fetches all the VM details from the DB and takes snapshot of all of them and destroys them.

Return value : No return value

3. shutdown_host

Parameters	**Data Type**	**Description**
host_id_list	List	It is a list of host IDs(Int type)

This function fetches all the VMs from all the hosts whose ids are there in the list (host_id_list). And takes the snapshot of all of them and destroys them

Return value : No return value

4. revert_and_resume

Parameters	**Data Type**	**Description**
vm_id	Int	ID of the VM
vm_identity	String	Name of the VM

This function fetches all the snapshots of the vM (with ID=vm_id) from the DB. Revert back to the snapshot and delete it.

Return value : No return value

5. bootup_baadal

This function fetches all the VMs except those whose status is not known and those which are still in task queue. Then it calls revert_and_resume for all those VMs.

Return value : No return value

6. bootup_host

Parameters	**Data Type**	**Description**
host_id_list	List	It is a list of host IDs(Int type)

This function fetches all the VMs which are hosted by those host whose ids is there in the host_id_list and except those whose status is not known and those which are still in task queue. Then it calls revert_and_resume for all those VMs.

Return value : No return value

class Worker

- **__init__() :** Thread executing tasks from a given tasks queue
  Parameter :
  1) tasks : It is a task from the task queue
- **run() :** Putting the thread in running

class ThreadPool

- **__init__() :** Pool of threads consuming tasks from a queue.
  Parameter :
    1) num_threads : Count of threads which need to be run by system
- **add_task() :** Add a task to the queue
- **wait_completion()** : Wait for completion of all the tasks in the queue

rrd_graph.py

1. get_rrd_file

Parameters	**Data Type**	**Description**
identity	String	this is the name of the rd file of VM or host

Return value : String type
this function returns the path to the .rd file. path is /mnt/datastore/vm_rds/<identity>.rrd

2. create_graph

Parameters	**Data Type**	**Description**
rrd_file_name	String	This is the name of the VM
graph_type	String	this is the type of the graph.Type means memory, cpu, disk etc
rrd_file_path	String	Path to the rrd file. /mnt/datastore/vm_rds/
graph_period	String	It stands for the time interval. hours, days, weeks, months, years

There are 5 types of periods for making graphs. Periods are mentioned in parameter tables. this function takes the rrd file of the host or vm from the filer and creates the graph bases on the values given in rrd file and given period.

Return value : boolean type
True : If graph file exists
False : If not exists

3. get_performance_graph

Parameters	**Data Type**	**Description**
graph_type	String	this is the type of the graph.Type means memory, cpu, disk etc
vm	String	Name of the VM
graph_period	String	It stands for the time interval. hours, days, weeks, months, years

This function is use to get the performance graph of the VM. First it gets the rrd file path from get_rrd_file() using vm name. If rrd file exists then it calls create_graph() to generate the graph at baadal/static/images/vm_graphs.

Return value : No return value

4. fetch_rrd_data

Parameters	**Data Type**	**Description**
rrd_file_name	String	This is the name of the VM

period	Int	Defines the graph period. default value is 2 (stands for 24 hours)

This function fetch RRD data to display in tabular format. It fetches data regarding cpu, ram, disk read, disk write, network read, network write from RRD file.

Return value :

Values	**Data Type**	**Description**
Memory	Float	This is the average value
CPU	Float	This is the average value
Disk Read	Float	This is the average value
Disk Write	Float	This is the average value
Network read	Float	This is the average value
Network write	Float	This is the average value

5. create_rrd

Parameters	**Data Type**	**Description**
rrd_file	String	Name of the RRD file

This function creates RRD file at the rrd file path /mnt/datastore/vm_rrds/

Return value : No return value

6. update_rrd

Parameters	Data Type	Description
host_ip	String	IP address of the host machine

This function first updates the rrd of host using host_ip by calling update_host_rrd() of host_rrd.py. Then it connects to the host and updates the rrds file of all the vms hosted by that host.

Return value : No return value

auth_user.py

1. register_callback

Parameters	Data Type	Description
form	Object	

Add membership of the user in the DB

Return value : No return value

2. login_callback

Parameters	Data Type	Description
form	Object	

This function first fetches the member whose membership id equals to the logged in user id. if member list is empty then it fetches the role from db using username of the authorized logged in user and add their membership in DB

Return value : No return value

3. login_ldap_callback

Parameters	**Data Type**	**Description**
form	Object	

It gets the username of the authorized logged in user. fetches the user info from user table of the DB whose last name is empty. then creates or updates the user info in the table

Return value : No return value

4. fetch_user_role

Parameters	**Data Type**	**Description**
username	String	Name of the user

this function checks the role of the user using username. and creates a list of roles.

Return value : List type
Returns the role list of the user

5. fetch_ldap_user

Parameters	**Data Type**	**Description**
username	String	Name of the user

this function searches the user in ldap. It finds role and organisation from ldap and set in db accordingly (current iitd ldap does not support this feature entirely). If user has super admin rights; it is added to separate organization

Return value : Object type
User_info : If user found in the ldap
None : if none found

6. create_or_update_user

Parameters	**Data Type**	**Description**
user_info	Object	Details of the user
update_session		

This method is called only when user logs in for the first time or when faculty name is verified on 'request VM' screen.

Return value : No return value

7. add_or_update_user_memberships

Parameters	**Data Type**	**Description**
user_id	Int	ID of the user who is logged in
roles	Int	Role id of the user
update_session		

This function adds the user membership in DB when a user or creates OR updates the membership when admin changes the membership of the user.

Return value : No return value

8. add_membership_db

Parameters	**Data Type**	**Description**
user_id	Int	ID of the user who is logged in
roles	Int	Role id of the user
update_session		

This function first finds the group_id for the given role. then add role to the current session.

Return value : No return value

9. is_auth_type_db : No parameter

It uses to check whether the authentication has to be done by DB (user table) or from ldap server.

Return value : boolean
True : if authentication is from DB
False : if from ldap etc (NOT from DB)

log_handler.py : It is a smart replacement for the standard RotatingFileHandler
ConcurrentRotatingFileHandler: This class is a log handler which is a drop-in replacement for the python standard log handler 'RotateFileHandler', the primary difference being that this handler will continue to write to the same file if the file cannot be rotated for some reason, whereas the RotatingFileHandler will strictly adhere to the maximum file size. Unfortunately, if you are using the RotatingFileHandler on Windows, you will find that once an attempted rotation

fails, all subsequent log messages are dropped. The other major advantage of this module is that multiple processes can safely write to a single log file. To put it another way: This module's top priority is preserving your log records, whereas the standard library attempts to limit disk usage, which can potentially drop log messages. If you are trying to determine which module to use, there are number of considerations: What is most important: strict disk space usage or preservation of log messages? What OSes are you supporting? Can you afford to have processes blocked by file locks?
Concurrent access is handled by using file locks, which should ensure that log messages are not dropped or clobbered. This means that a file lock is acquired and released for every log message that is written to disk

this API exposes two loggers.

1. logger : this is use to log the debug message, info message, exception message
2. rrd_logger : this is use to log the rrd related data to rrd file

load_balancer.py

1. get_host_domains_ID

Parameters	**Data Type**	**Description**
host_ip	String	IP address of the host

this function is use to get all the VM ids running on that host

Return value : List Type
domains : List of VM ids

2. get_migration_vector

Parameters	**Data Type**	**Description**
HMAX	Float	Max CPU usage
HMIN	Flaot	Min CPU usage

Return value : Float type
this function returns the average of HMAX and HMIN

3. load_balance : No Parameter

this function is called by task_scheduler or when a VM is created or deleted. First this function fetches all the active hosts. then it calculates the CPU usage of all the host. Identifies the host with max cpu usage , and for other hosts, it keeps their cpu usage and id in a list and sort that list in ascending order of the cpu usage. Then it fetches all the VM ids from the host having max cpu. And then searches the host with min cpu and enough ram for the the vms of the maxhost which has cpu usage closer to avg(HMAX,HMIN). If found that min host it migrates the VM to that host from maxhost. It keeps on running till it finds such VMs.

Return value: No return value