MCVirt Documentation

Release 4.0.1

I.T. Dev Ltd

July 12, 2016

Contents

CHAPTER 1

MCVirt - Managed Consistent Virtualisation

MCVirt `(em-see-vert)` - Command line virtual machine management utility.

CHAPTER 2

Description

MCVirt is a utility for managing virtual machines, supporting the following technologies:

- Ubuntu 14.04 LTS.
- KVM virtualisation.
- Clustering with optional DRBD support.

MCVirt is implemented in Python, using the the libvirt virtualisation library.

CHAPTER 3

Getting started

MCVirt must currently be built from source into a deb package, using the build script. The package and dependencies can then be installed:

```
$ ./build.sh
$ sudo dpkg -i mcvirt_0.10_all.deb
$ sudo apt-get -f install
```

See the installation guide for other dependencies and system configuration.

Start the MCVirt nameserver and daemon by running:

```
$ sudo service mcvirt-ns start
$ sudo service mcvirtd start
```

Most commands require a username and password to the MCVirt daemon. During installation a superuser is created with username `mjc` and password `pass` - see permissions for information on creating new users.

Configure the volume group that MCVirt will use to store virtual machine data:

```
$ sudo mcvirt node --set-vm-vg <Volume Group>
```

See the configuration guide for further node configuration steps.

Create a VM:

```
$ sudo mcvirt create --cpu-count 1 --memory 512 --disk-size 8000 test-vm
```

See the create/remove VMs, cluster, permissions and modifying VMs guides for further administrative instructions.

Start the VM:

```
$ sudo mcvirt start test-vm
```

See the controlling VMs guide for further user instructions.

Note: Username and password can be provided on the command line with the `--username` and `--password` options to instead of typing them in after every command.

For information on developing on MCVirt, see the development documentation.

See the guide index for a full list of manuals

For more information, please feel free to contact us

CHAPTER 4

LICENSE

MCVirt is licensed under GPL v2. For more information, please see LICENSE

CHAPTER 5

COPYRIGHT

5.1 Clustering

Nodes running MCVirt can be joined together in a cluster - this allows the synchronization of VM/global configurations.

5.1.1 Viewing the status of a cluster

To view the status of the cluster, run the following on an MCVirt node:

```
    mcvirt info
```

This will show the cluster nodes, IP addresses, and status.

5.1.2 Adding a new node

It is best to join a blank node (containing a default configuration without any VMs) to a cluster.

When a new node is connected to a cluster, the configuration from the present nodes in the cluster (e.g. users, permissions, networks etc.) are pushed to the new node and any existing configuration is replaced.

Note: Always run the `mcvirt cluster add` command from the source machine, containing VMs, connecting to a remote node that is blank.

The new node must be configured on separate network/VLAN for MCVirt cluster communication.

The IP address that MCVirt clustering/DRBD communications will be performed over must be configured by performing the following on both nodes:

```
mcvirt node --set-ip-address <Node cluster IP address>
```

This configuration can be retrieved by running `mcvirt info`.

Joining the node to the cluster

Note: The following can only be performed by a superuser.

1. From the remote node, run:

```
mcvirt cluster get-connect-string
```

The connect string will be displayed

2. From the source node, run:

```
mcvirt cluster add-node --connect-string <connect string>
```

where `<connect string>` is the string printed out in step 1.

3. The local node will connect to the remote node, ensure it is suitable as a remote node, setup authentication between the nodes and copy the local permissions/network/virtual machine configurations to the remote node. **Note:** All existing data on the remote node will be removed.

5.1.3 Removing a node from the cluster

Note: The following can only be performed by a superuser.

To the remove a node from the cluster, run:

```
mcvirt cluster remove-node --node <Remote Node Name>
```

5.1.4 Get Cluster information

- In order to view status information about the cluster, use the 'info' parameter for MCVirt, without specifying a VM name:

```
mcvirt info
```

5.1.5 Virtual machine migration

- VMs that use DRBD-based storage can be migrated to the other node in the cluster, whilst the VM is powered off, using:

```
mcvirt migrate --node <Destination node> <VM Name>
```

- Additional parameters are available to aid the migration and minimise downtime:
  - `--wait-for-shutdown`, which will cause the migration command to poll the running state of the VM and migrate once the VM is in a powered off state, allowing the user to shutdown the VM from within the guest operating system.
  - `--start-after-migration`, which starts the VM immediately after the migration has finished
  - `--online`, which will perform online migration. Note: these cannot be used with either of the previous arguments.

5.2 DRBD

DRBD is used by MCVirt to use replicate storage across a 2-node cluster.

Once DRBD is configured and the node is in a cluster, 'DRBD' can be specified as the storage type when creating a VM, which allows the VM to be migrated between nodes.

5.2.1 Configuring DRBD

1. Ensure the package `drbd8-utils` is installed on both of the nodes in the cluster
2. DRBD data will be transmitted over the 'cluster' address. Ensure that this has been set and that the network is segemneted from other network traffic (e.g. by using VLANs).
3. Perform the following MCVirt command to configure DRBD:

```
mcvirt drbd --enable
```

5.2.2 DRBD verification

MCVirt has the ability to start/monitor DRBD verifications (See the DRBD documentation).

The verification can be performed by using:

```
mcvirt verify <--all>|<VM Name>
```

This will perform a verification of the specified VM (or all of the DRBD-backed VMs, if '–all' is specified). Once the verification is complete, an exception is thrown if any of the verifications fail.

The status of the latest verification is captured and will stop users from starting/migrating the VM.

If the verification fails:

- The DRBD volume must be resynced (for more information, see the DRBD documentation for re-syncing).
- Once this is complete, perform another MCVirt verification to mark the VM as in-sync, which will lift the limitations.

5.3 Troubleshooting

5.3.1 Failures during VM creation/deletion

When a VM is created, the following order is performed:

1. The VM is created, configured with the name, memory allocation and number of CPU cores
2. The VM is then created on the remote node
3. The VM is then registered with LibVirt on the local node
4. The hard drive for the VM is created. (For DRBD-backed storage, the storage is created on both nodes and synced)
5. Any network adapters are added to the VM

If a failure of occurs during steps 4/5, the VM will still exist after the failure. The user should be able to see the VM, using `mcvirt list`.

The user can re-create the disks/network adapters as necessary, using the `mcvirt update` command, using `mcvirt info <VM Name>` to monitor the virtual hardware that is attached to the VM.

5.4 Configuration

5.4.1 Configure Network

Remove default network

- By default, libvirt configures a default network, 'default'.
- The 'default' network is attached to a private network, which provides NAT routing through the node's physical interfaces.
- If you wish to use bridging, the default network can be removed by performing the following:

```
mcvirt network delete default
```

Creating/Removing Networks

- Networks provide bridges between physical/bridge interfaces and virtual machines.
- To create a bridge network on the node, an additional network interface will need to be created on the node
- This will generally be placed in */etc/network/interfaces*

The following example should help with creating this interface:

```
auto vmbr0
iface vmbr0 inet manual
  bridge_ports <Physical interface>
  bridge_stp off
  bridge_fd 0
```

Where *<Physical interface>* is the name of the interface that the bridge should be bridged with, e.g. 'eth0'

- To create a network on the node, perform the following as a superuser:

```
mcvirt network create <Network name> --interface <Bridge interface>
```

- Assuming that there are not any VMs connected to a network, they can be removed using:

```
mcvirt network delete <Network name>
```

5.4.2 Configure MCVirt

- The first time MCVirt is run, it creates a configuration file for itself, found in **/var/lib/mcvirt/<Hostname>/config.json**.
- The volume group, in which VM data will be stored as logical volumes, must be setup using:

```
mcvirt node --set-vm-vg <Volume Group>
```

- The cluster IP address must be configured if the node will be used in a cluster (See the Cluster documentation):

```
mcvirt node --set-ip-address <Cluster IP Address>
```

- In order for the MCVirt client to connect to the daemon, the hosts file at `/etc/hosts` must edited by changing the line:

```
127.0.0.1    <hostname>
```

to:

```
<Cluster IP Address>    <hostname>
```

5.5 Controlling VMs

All commands must be performed on the MCVirt node, which can be accessed via SSH.

5.5.1 Start VM

- Use the MCVirt utility to start VMs:

```
mcvirt start <VM name>
```

5.5.2 Stop VM

- Use the MCVirt utility to stop VMs:

```
mcvirt stop <VM name>
```

5.5.3 Reset VM

- Use the MCVirt utility to reset VMs:

```
mcvirt reset <VM Name>
```

- Only a super user can reset a VM. Normal users can stop and start the VM.

5.5.4 Get VM information

- In order to view information about a VM, use the 'info' parameter for MCVirt:

```
mcvirt info <VM Name>
```

- Example output:

```
<Username>@node:~# mcvirt info test-vm
Name              | test-vm
CPU Cores         | 1
Memory Allocation | 512MB
State             | Running
ISO location      | /var/lib/mcvirt/iso/ubuntu-12.04-server-amd64.iso
-- Disk ID --     | -- Disk Size --
1                 | 8GB
-- MAC Address -- | -- Network --
52:54:00:2b:8a:a1 | Production
-- Group --       | -- Users --
owner             | mc
user              | nd
```

5.5.5 Listing virtual machines

- In order to list the virtual machines on a node, run the following:

```
mcvirt list
```

- This will provide the names of the virtual machines and their current state (running/stopped)

5.5.6 Connect to VNC

- By default, VMs are started with a VNC console, for which the port is automatically generated.
- The default listening IP address is 127.0.0.1, meaning that it can only be accessed from the node itself.
- To manually gain access to a VNC console, ssh to the node, forwarding the port:
  1. Determine the port that the VM is listening on:

```
mcvirt info <VM Name> --vnc-port
5904
```

  2. SSH onto the node, forwarding the port provided in the previous step (5904 in this case)
     - The local port can be any available port. In this example, 1232 is used:

```
ssh <Username>@<Node> -L 1232:127.0.0.1:5904
```

     - For putty, use the tunnels configuration under **Connection -> SSH -> Tunnels**, where the source port is the local port and the destination is 127.0.0.1:<VNC Port>
  3. Use an VNC client to connect to 127.0.0.1:1232 on your local PC

5.5.7 Removing VNC display

- By disabling the VNC display, a greater VM performance may be achieved.
- Power off the VM
- Perform:

```
virsh edit <VM Name>
```

- Remove the `<graphics type='vnc'...>...</graphics>` lines from the configuration.
- Save the configuration and start the VM
- This can only be performed by root

5.5.8 Monitoring Resources

- To monitor resources, the following commands are available that can be run from an SSH console:
  - top - monitor CPU/memory usages by processes
  - iftop - monitor network usage
  - iotop - monitor disk usages

5.5.9 Back up VM

MCVirt can provide access to snapshots of the raw volumes of VM disks, allowing a superuser to backup the data

1. To create a snapshot, perform the following:

```
mcvirt backup --create-snapshot --disk-id <Disk ID> <VM Name>
```

2. The returned path provides access to the disk at the time that the snapshot was created

Warning: The snapshot is 500MB in size, meaning that once the VM has changed 500MB of space on the disk, the VM will no longer be able to write to its disk

3. Once the data has been backed up, the snapshot can be removed by performing:

```
mcvirt backup --delete-snapshot --disk-id <Disk ID> <VM Name>
```

- This can only be performed by a superuser

5.6 Create/Remove VMs

- All commands must be performed on the MCVirt node, which can be accessed via SSH using LDAP credentials.
- You must be a superuser to create and remove VMs

5.6.1 Create VM

- Use the MCVirt utility to create VMs:

```
mcvirt create '<VM Name>'
```

- The following parameters are available:
  - **–memory** - Amount of memory to allocate to the VM (MB) (required)
  - **–cpu-count** - Number of vCPUs to be allocated to the VM (required)
  - **–disk-size** - Size of initial disk to be added to the VM (MB) (optional)
  - **–network** - Provide the name of a network to be attached to the VM. (optional)
    - This can be called as multiple times.
    - A separate network interface is added to the VM for each network.
    - A network can be specified multiple times to create multiple adapters connected to the same network.
  - **–storage-type** - Storage backing type - either `Local` or `DRBD`.
  - **–nodes** - Specifies the nodes that the VM will be hosted on, if a DRBD storage-type is specified and there are more than 2 nodes in the cluster.
  - **–driver** - The virtual disk driver to use. If this is not specified then MCVirt will select the most appropriate driver (optional)

5.6.2 Cloning a VM

Cloning/duplicating a VM will create an identical replica of the VM.

Although both cloning and duplicating initially may appear to provide the same functionality, there are core differences, based on how they work, which should be noted to decide which function to use.

Both cloning and duplicating a VM can be performed by an **owner** of a VM.

5.6.3 Cloning

- The hard disk for the VM is **snapshotted**, which means the VM is cloned very quickly
- Cloning VMs is not support for DRBD-backed VMs
- Some restrictions are imposed on both the parent and clone, due to the way that the storage is cloned:
  - Parent VMs cannot be:
    - Started
    - Resize (HDDs)
    - Deleted
  - VM Clones cannot be:
    - Resized
    - Cloned
  - **Note:** All restrictions are lifted once all VM clones have been removed.

A VM can be cloned by performing the following:

```
mcvirt clone --template <Source VM Name> <Target VM Name>
```

5.6.4 Duplicating

- Duplicating produces a new VM that is a completely separate entity to the source, meaning that no restrictions are imposed on either VM
- Duplicating a VM will copy the entire VM hard drive, which takes longer than cloning a VM

A VM can be duplicated by performing the following:

```
mcvirt duplicate --template <Source VM Name> <Target VM Name>
```

5.6.5 Removing VM

- Ensure that the VM is stopped.
- Use the MCVirt utility to remove the VM:

```
mcvirt delete <VM Name>
```

- Without any parameters, the VM will simply be 'unregistered' from the node.
- To remove all data associated with the VM, supply the parameter **–remove-data**
- Only a superuser can delete a VM

5.7 Development

5.7.1 Coding Standards

The MCVirt code base follows the python PEP 8 coding standards, with a line length limit of 100 characters.

All code changes must comply with this coding standard and are checked by continuous integration.

The PEP 8 code checker can be installed using:

```
sudo apt-get install pep8
```

Run the checks using:

```
pep8
```

5.7.2 Automated Tests

There is a collection of unit tests for MCVirt, which can be run as follows:

```
python /usr/lib/mcvirt/test/run_tests.py
```

Before runing the tests ensure that the `mcvirt-ns` service is running on all nodes in the cluster, and that `mcvirtd` is running on all nodes except the one the tests are being run on (since `mcvirtd` is started when the tests are run)

5.7.3 Manual Test Procedure

This test procedure is designed to compliment the automated unit tests and should be performed prior to making a new release.

- Make sure the `mcvirt-ns` and `mcvirtd` daemons are started
- Create a VM called 'test-vm'
- Run `mcvirt list` and check that 'test-vm' is shown in the list, and that its state is 'STOPPED'
  - If the node is part of a cluster, run `mcvirt list` on another node in the cluster, and check that 'test-vm' is listed
- Start 'test-vm'. Run `mcvirt list` again and check that its state is now 'RUNNING'
  - Run `mcvirt list` on a remote node to check the state of 'test-vm' if the node is part of a cluster
- Try to delete 'test-vm' and check that an error is shown saying 'Can't delete a running VM'
- Stop 'test-vm', and try to delete it again. Check that it is no longer shown in the output of `mcvirt list`
  - If the node is part of a cluster, confirm 'test-vm' has been deleted on a remote node too
- If the node is part of a cluster:
  - Make sure DRBD is enabled by running `mcvirt drbd --enable`
  - Create a new VM called 'cluster-vm', specifying the storage type as 'Drbd'
  - Start 'cluster-vm'
  - Test online migration of VMs by running `mcvirt migrate --online --node <remote node> cluster-vm`

  - Run `mcvirt list` on the local and remote nodes to check that 'cluster-vm' is now registered on the remote node

5.8 Installation

5.8.1 Install Operating System

- MCVirt is currently built to support Ubuntu 14.04 with native versions of dependencies.
- When installing the operating system, create the following logical volumes:
  - Root - Create a 50GB partition using ext4. This is used for the operating system, MCVirt configurations and ISO images
  - SWAP - leave the suggested SWAP volume unaltered
- Virtual machine storage will be created as additional volumes in the volume group.

5.8.2 Building the package

- Ensure the build dependencies are installed: `dpkg, python-docutils`
- Clone the repository with: `git clone https://github.com/ITDevLtd/MCVirt`
- From within the root of the working copy, run build.sh

5.8.3 Installing Package

To install the package, run:

```
$ sudo dpkg -i mcvirt_X.XX_all.deb
$ sudo apt-get -f install
```

MCVirt uses a customised version of Pyro, which can be installed by running:

```
$ git clone https://github.com/MatthewJohn/Pyro4
$ cd Pyro4
$ sudo pip install .
```

You may need to install *pip* by running *sudo apt-get install python-pip*.

5.9 User Guides

5.9.1 Installation

- Installation - Procedure for building the MCVirt package, setting up a node with MCVirt and performing initial configuration

5.9.2 Configuration

- Configuration - Procedures for performing initial configurations of an MCVirt installation

5.9.3 Administration

- Permissions - Procedures for configuring permissions within MCVirt
- Cluster - Procedures for configuring an MCVirt Cluster

5.9.4 Usage

- Create/Remove VMs - Procedures for creating and deleting virtual machines
- Controlling VMs - Instructions for using the MCVirt script and controlling virtual machines
- Modifying VMs - Procedures for making changes to virtual machine configurations

5.9.5 Development

- Development - Information about performing development on MCVirt

5.10 Modifying VMs

5.10.1 Increase Disk Size

- Power off the VM
- Use MCVirt to increase the size of the disk - you will need to find the disk ID, which can be found by looking at the VM configuration (in most cases where a VM has one disk attached to it, it should be 1):

```
mcvirt update --increase-disk <Amount to increase (MB)> --disk-id <Disk Id> <VM Name>
```

5.10.2 Change Memory/CPU Allocation

- Update the VM memory allocation and virtual CPU count using the following:

```
mcvirt update --memory <New Memory Allocation (MB)> <VM Name>
mcvirt update --cpu-count <New CPU count> <VM Name>
```

- The changes will take affect the next time the VM is booted. If the VM is running, it will need to be powered off and started again.

5.10.3 Add Additional Disk

- Use the following MCVirt command to add an additional disk to a VM:

```
mcvirt update --add-disk <Size of disk (MB)> <VM Name>
```

- The device will be attached to the VM the next time it's booted. If the VM is running, it will need to be powered off and started again.

5.10.4 Add/Remove Network Adapter

- Use the following MCVirt command to add/remove network adapters to/from a VM
- Add an adapter:

```
mcvirt update --add-network <Network Name> <VM Name>
```

- Remove an adapter:

```
mcvirt update --remove-network '<NIC MAC Address>' <VM Name>
```

- Use the formatting '00:11:22:33:44:55' for the MAC address
- The device will altered the next time the VM is booted. If the VM is running, it will need to be powered off and started again.

5.10.5 Attaching ISO

- ISO images can be attached to the cdrom drive of a VM whilst booting the VM
- Use the MCVirt utility to start the VM, using the '–iso' parameter to define the ISO image to be attached to the VM:

```
mcvirt start <VM Name> --iso <Name of ISO file>
```

- The ISO file must be stored within /var/lib/mcvirt/iso.

VM Locking

VMs can be locked by superusers, which stops them from being started, stopped or migrated

- To lock a VM:

```
mcvirt lock --lock <VM Name>
```

- To unlock a VM:

```
mcvirt lock --unlock <VM Name>
```

- Users can check the lock status of a VM by running:

```
mcvirt lock --check-lock <VM Name>
```

5.11 Permissions

5.11.1 Superusers

- To run MCVirt commands as a superuser you must either:
  - Have your username included in the superusers section in the configuration file.
- Superusers can be added/removed using the following:

```
mcvirt permission --add-superuser=<username>
mcvirt permission --delete-superuser=<username>
```

5.11.2 Managing users

- To create a new user, perform the following as a superuser:

```
mcvirt user create <new username>
```

  The password for the new user can be provided interactively, passed on the command line with `--user-password <new password>`, or generated automatically with `--generate-password`. The generated password will be displayed when the user is created.

- To remove a user, perform the following as a superuser:

```
mcvirt user remove <user>
```

- To change your password, perform the following:

```
mcvirt user change-password
```

  The new password can be provided interactively or on the command line with `--new-password <new password>`. **Note:** Superusers can change the password of any other user by running `mcvirt user change-password --target-user <other user>`.

- In MCVirt, 'users' are able to start/stop VMs
- To view the current permissions on a VM, including users and owners of a VM, run:

```
mcvirt info <VM Name>
```

- To add a user to VM, perform the following:

```
mcvirt permission --add-user <Username> <VM Name>
```

- To remove a user, perform the following:

```
mcvirt permission --delete-user <Username> <VM Name>
```

- **Owners** of a VM are able to manage the **users** of a VM.

5.11.3 Managing owners

- VM owners have the same permissions as users, except they are also able to manage the users of the VM
- To add an owner to VM, perform the following:

```
mcvirt permission --add-owner <Username> <VM Name>
```

- To remove an owner, perform the following:

```
mcvirt permission --delete-owner <Username> <VM Name>
```

- Only superusers are able to manage the **owners** of a VM.

CHAPTER 6

API Documentation

Contents:

6.1 mcvirt package

6.1.1 Subpackages

mcvirt.auth package

Submodules

mcvirt.auth.auth module

Provide auth class for managing permissions.

class mcvirt.auth.auth.**Auth**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Provides authentication and permissions for performing functions within MCVirt.

add_superuser(*user_object*, *ignore_duplicate=False*)
Add a new superuser.

add_user_permission_group(**args*, ***kwargs*)

assert_permission(*permission_enum*, *vm_object=None*)
Use check_permission function to determine if a user has a given permission and throws an exception if the permission is not present.

assert_user_type(**user_type_names*)
Ensure that the currently logged in user is of a specified type.

check_permission(*permission_enum*, *vm_object=None*, *user_object=None*)
Check that the user has a given permission, either globally through MCVirt or for a given VM.

check_permission_in_config(*permission_config*, *user*, *permission_enum*)
Read permissions config and determines if a user has a given permission.

static check_root_privileges()
Ensure that the user is either running as root or using sudo.

check_user_type(**user_type_names*)
Check that the currently logged-in user is of a specified type.

copy_permissions(*source_vm*, *dest_vm*)
Copy the permissions from a given VM to this VM. This functionality is used whilst cloning a VM

delete_superuser(*user_object*)
Remove a superuser.

delete_user_permission_group(**args*, ***kwargs*)

get_permission_groups()
Return list of user groups.

get_superusers()
Return a list of superusers

get_users_in_permission_group(*permission_group*, *vm_object=None*)
Obtain a list of users in a given group, either in the global permissions or for a specific VM.

is_superuser()
Determine if the current user is a superuser of MCVirt.

mcvirt.auth.cluster_user module

Provide class for managing cluster users.

class mcvirt.auth.cluster_user.**ClusterUser**(*username*)
Bases: *mcvirt.auth.user_base.UserBase*

User type for cluster daemon users.

CAN_GENERATE = True

CLUSTER_USER = True

DISTRIBUTED = False

USER_PREFIX = 'mcv-cluster-'

allow_proxy_user
Connection users can proxy for another user.

static get_default_config()
Return the default user config.

node
Return the node that the user is used for

update_host(*host*)
Update the host associated with the user.

mcvirt.auth.connection_user module

Provide class for managing connection users.

class mcvirt.auth.connection_user.**ConnectionUser**(*username*)
Bases: *mcvirt.auth.user_base.UserBase*

User type for initial connection users

CAN_GENERATE = True

CLUSTER_USER = True

DISTRIBUTED = False

PERMISSIONS = [<Mock id='140115252141456'>]

USER_PREFIX = 'mcv-connection-'

allow_proxy_user
Connection users can proxy for another user.

create_cluster_user(*host*)
Create a cluster user for the remote node.

mcvirt.auth.factory module

Provide factory class to create/obtain users.

class mcvirt.auth.factory.**Factory**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Class for obtaining user objects

USER_CLASS
alias of UserBase

add_config(*username*, *user_config*)
Add a user config to the local node.

authenticate(*username*, *password*)
Attempt to authenticate a user, using username/password.

create(*username*, *password*, *user_type=<class 'mcvirt.auth.user.User'>*)
Create a user.

ensure_valid_user_type(*user_type*)
Ensure that a given user_type is valid.

generate_user(*user_type*)
Remove any existing connection user and generates credentials for a new connection user.

get_all_user_objects(*user_class=None*)
Return the user objects for all users, optionally filtered by user type.

get_all_users()
Return all the users, excluding built-in users.

get_cluster_user_by_node(*node*)
Obtain a cluster user for a given node

get_user_by_username(*username*)
Obtain a user object for the given username.

get_user_types()
Return the available user classes.

mcvirt.auth.permissions module

Provide permission enum and permission group definitions.

mcvirt.auth.session module

Provide class for managing authentication sessions.

class mcvirt.auth.session.**Session**
Bases: object

Handle daemon user sessions.

USER_SESSIONS = {}

authenticate_session(*username*, *session*)
Authenticate user session.

authenticate_user(*username*, *password*)
Authenticate using username/password and store session

get_current_user_object()
Return the current user object, based on pyro session.

get_proxy_user_object()
Return the user that is being proxied as.

mcvirt.auth.user module

Provide class for regular MCVirt interactive users

class mcvirt.auth.user.**User**(*username*)
Bases: *mcvirt.auth.user_base.UserBase*

Provides an interaction with the local user backend

set_password(*new_password*)
Change the current user's password.

mcvirt.auth.user_base module

Provide a base class for user objects.

class mcvirt.auth.user_base.**UserBase**(*username*)
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Base object for users (both user and automated).

CAN_GENERATE = False

CLUSTER_USER = False

DISTRIBUTED = True

PERMISSIONS = []

USER_PREFIX = None

allow_proxy_user
Connection users can proxy for another user.

delete(**args*, ***kwargs*)

static generate_password(*length*, *numeric_only=False*)
Return a randomly generated password

get_config()
Return the configuration of the user.

static get_default_config()
Return the default configuration for the user type.

get_user_type()
Return the user type of the user

get_username()
Return the username of the current user

Module contents

mcvirt.client package

Submodules

mcvirt.client.rpc module

Provide class for connecting to RPC daemon

class mcvirt.client.rpc.**Connection**(*username=None*, *password=None*, *session_id=None*, *host=None*, *ignore_cluster=False*)
Bases: object

Connection class, providing connections to the Pyro MCVirt daemon

NS_PORT = 9090

SESSION_OBJECT = 'session'

annotate_object(*object_ref*)
Add authentication attributes to remote object

get_connection(*object_name*, *password=None*)
Obtain a connection from pyro for a given object

ignore_cluster()
Set flag to ignore cluster

ignore_drbd()
Set flag to ignore DRBD

session_id
Property for the session ID

Module contents

mcvirt.cluster package

Submodules

mcvirt.cluster.cluster module

Provide cluster classes

class mcvirt.cluster.cluster.**Cluster**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Class to perform node management within the MCVirt cluster

add_node(**args*, ***kwargs*)

add_node_configuration(**args*, ***kwargs*)

check_ip_configuration()
Perform various checks to ensure that the IP configuration is such that is suitable to be part of a cluster

check_node_exists(*node_name*)
Determine if a node is already present in the cluster

check_node_versions()
Ensure that all nodes in the cluster are connected and checks the node Status

check_remote_machine(*remote_connection*)
Perform checks on the remote node to ensure that there will be no object conflicts when syncing the Network and VM configurations

ensure_node_exists(*node*)
Check if node exists and throws exception if it does not

generate_connection_info()
Generate required information to connect to this node from a remote node

get_cluster_config()
Get the MCVirt cluster configuration

get_cluster_ip_address()
Return the cluster IP address of the local node

get_connection_string()
Generate a string to connect to this node from a remote cluster

get_node_config(*node*)
Return the configuration for a node

get_nodes(*return_all=False*)
Return an array of node configurations

get_remote_node(*node*, *ignore_cluster_master=False*)
Obtain a Remote object for a node, caching the object

print_info()
Print information about the nodes in the cluster

remove_node(**args*, ***kwargs*)

remove_node_configuration(*node_name*)
Remove an MCVirt node from the configuration and regenerates authorized_keys file

remove_node_ssl_certificates(*remote_node*)
Exposed method for _remove_node_ssl_certificates

run_remote_command(*callback_method*, *nodes=None*, *args=[]*, *kwargs={}*)
Run a remote command on all (or a given list of) remote nodes

sync_networks(*remote_object*)
Add the local networks to the remote node

sync_permissions(*remote_object*)
Duplicate the global permissions on the local node onto the remote node

sync_users(*remote_node*)
Synchronise the local users with the remote node

sync_virtual_machines(*remote_object*)
Duplicate the VM configurations on the local node onto the remote node

mcvirt.cluster.remote module

Provide interface for RPC to cluster nodes

class mcvirt.cluster.remote.**Node**(*name*, *node_config*)
Bases: *mcvirt.client.rpc.Connection*

A class to perform remote commands on MCVirt nodes

Module contents

mcvirt.iso package

Submodules

mcvirt.iso.factory module

Provide factory class for ISO

class mcvirt.iso.factory.**Factory**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Class for obtaining ISO objects

ISO_CLASS
alias of Iso

add_from_url(**args*, ***kwargs*)

add_iso(*path*)
Copy an ISO to ISOs directory

add_iso_from_stream(**args*, ***kwargs*)

get_iso_by_name(*iso_name*)
Create and register Iso object

get_iso_list()
Return a user-readable list of ISOs

get_isos()
Return a list of a ISOs

class mcvirt.iso.factory.**IsoWriter**(*temp_file*, *factory*, *temp_directory*, *path*)
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Provide an interface for writing ISOs

write_data(*data*)
Write data to the ISO file

write_end()
End writing object, close FH and import ISO

mcvirt.iso.iso module

Provide class for managing ISO files

class mcvirt.iso.iso.**Iso**(*name*)
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Provides management of ISOs for use in MCVirt

delete()
Delete an ISO

static get_filename_from_path(*path*, *append_iso=True*)
Return filename part of path

get_name()
Return the name of the ISO

get_path()
Return the full path of the ISO

in_use
Determine if the ISO is currently in use by a VM

static overwrite_check(*filename*, *path*)
Check if a file already exists at path. Ask user whether they want to overwrite. Returns True if they will overwrite, False otherwise

set_iso_permissions()
Set permissions to 644

Module contents

mcvirt.node package

Subpackages

mcvirt.node.network package

Submodules

mcvirt.node.network.factory module Provide class for generating network objects

class mcvirt.node.network.factory.**Factory**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Class for obtaining network objects

OBJECT_TYPE = 'network'

check_exists(*name*)
Check if a network exists

create(**args*, ***kwargs*)

ensure_exists(*name*)
Ensure network exists

get_all_network_names()
Return a list of network names

get_all_network_objects()
Return all network objects

get_network_by_name(*network_name*)
Return a network object of the network for a given name.

get_network_list_table()
Return a table of networks registered on the node

interface_exists(*interface*)
Public method for to determine if an interface exists

mcvirt.node.network.network module Provide interface to libvirt network objects

class mcvirt.node.network.network.**Network**(*name*)
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Provides an interface to LibVirt networks

delete(**args*, ***kwargs*)

get_adapter()
Return the name of the physical bridge adapter for the network

get_name()
Return the name of the network

static get_network_config()
Return the network configuration for the node

Module contents

Submodules

mcvirt.node.drbd module

Provides interface to mange the DRBD installation.

class mcvirt.node.drbd.**Drbd**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Performs configuration of DRBD on the node

CLUSTER_SIZE = 2

CONFIG_DIRECTORY = '/etc/drbd.d'

DrbdADM = '/sbin/drbdadm'

GLOBAL_CONFIG = '/etc/drbd.d/global_common.conf'

GLOBAL_CONFIG_TEMPLATE = '/usr/lib/mcvirt/templates/drbd_global.conf'

adjust_drbd_config(*resource='all'*)
Perform a Drbd adjust, which updates the Drbd running configuration

enable(**args*, ***kwargs*)

ensure_installed()
Ensure that Drbd is installed on the node

generate_config()
Generate the Drbd configuration

generate_secret()
Generate a random secret for Drbd

get_all_drbd_hard_drive_object(*include_remote=False*)
Obtain all hard drive objects that are backed by DRBD

get_config()
Return the global Drbd configuration

static get_default_config()
Return the default configuration for DRBD

get_used_drbd_minors()
Return a list of used Drbd minor IDs

get_used_drbd_ports()
Return a list of used Drbd ports

is_enabled()
Determine whether Drbd is enabled on the node or not

is_installed()
Determine if the 'drbdadm' command is present to determine if the 'drbd8-utils' package is installed

list()
List the Drbd volumes and statuses

set_secret(*secret*)
Set the Drbd configuration in the global MCVirt config file

mcvirt.node.libvirt_config module

Provide class to configure libvirtd

class mcvirt.node.libvirt_config.**LibvirtConfig**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Provides configuration for libvirtd

CONFIG_FILE = '/etc/libvirt/libvirtd.conf'

CONFIG_TEMPLATE = '/usr/lib/mcvirt/templates/libvirtd.conf'

DEFAULT_CONFIG = '\n# Defaults for libvirtd initscript (/etc/init.d/libvirtd)\n# This is a POSIX shell fragment\n\n# Start

DEFAULT_FILE = '/etc/default/%s'

generate_config()
Generate the libvirtd configuration

get_config()
Create the configuration for libvirt

get_service_name()
Locate the libvirt service

mcvirt.node.node module

Perform configurations for local node.

class mcvirt.node.node.**Node**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Provides methods to configure the local node.

clear_method_lock()
Force clear a method lock to escape deadlock

get_version()
Returns the version of the running daemon

is_volume_group_set()
Determine if the volume group has been configured on the node

set_cluster_ip_address(**args*, ***kwargs*)

set_storage_volume_group(**args*, ***kwargs*)

Module contents

mcvirt.rpc package

Submodules

mcvirt.rpc.certificate_generator module

Provides class to generate and manage SSL certificates

class mcvirt.rpc.certificate_generator.**CertificateGenerator**(*server=None*, *remote=False*)
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Class for providing SSL socket wrappers for Pyro. Since the MCVirt isn't available for 2/3 of the time that this is used (NS and CLI), all methods are static and paths are calculated manually. @TODO Fix this in future - create MCVirt config class.

OPENSSL = '/usr/bin/openssl'

add_public_key(*key*)
Add the public key for a remote node

ca_key_file
Return/generate the CA prviate key.

ca_pub_file
Return/generate the CA pub file

check_certificates(*check_client=True*)
Ensure that the required certificates are available to start the daemon and connect to the local host

client_csr
Return the client CSR

client_key_file
Obtain the private key for the client key

client_pub_file
Return/generate the client public file, used for connecting to the libvirt daemon

generate_csr()
Generate a certificate request for the remote server

get_ca_contents()
Return the contents of the local CA certificate

is_local
Determine if the server is the local machine

remote_ssl_directory
Return the 'remote' subdirectory of server, used for storing certificates that are used by a remote server.

remove_certificates()
Remove a certificate directory for a node

server_key_file
Obtain the server private key file

server_pub_file
Obtain the server public key file

sign_csr(*csr*)
Sign the CSR for a remote SSL certificate.

ssl_base_directory
Return the base SSL directory for the node.

ssl_directory
Return the SSL directory for the server

ssl_dn
"Return the certificate DN is openssl argument format.

ssl_subj
Return the SSL DN in regular format

mcvirt.rpc.certificate_generator_factory module

Provides an interface to obtain certificate generator objects

class mcvirt.rpc.certificate_generator_factory.**CertificateGeneratorFactory**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Provides an interface to obtain certificate generator objects

get_cert_generator(*server*, *remote=False*)
Obtain a certificate generator object for a given server

mcvirt.rpc.constants module

Provides constants for the RPC daemon

class mcvirt.rpc.constants.**Annotations**
Bases: object

Pyro annotation names @TODO Move to main MCVIrt constants class

CLUSTER_MASTER = 'CLMA'

HAS_LOCK = 'HASL'

IGNORE_CLUSTER = 'IGCL'

IGNORE_Drbd = 'IGDR'

PASSWORD = 'PASS'

PROXY_USER = 'ALTU'

SESSION_ID = 'SEID'

USERNAME = 'USER'

mcvirt.rpc.daemon_lock module

Provides a locking mechanism for the MCVirt daemon

class mcvirt.rpc.daemon_lock.**DaemonLock**(*timeout=2*)
Bases: object

Provides a lock for the MCVirt daemon

LOCK = None

mcvirt.rpc.lock module

Provides classes for locking the MCVirt daemon whilst a function is being performed

class mcvirt.rpc.lock.**MethodLock**
Bases: object

Class for storing/generating/obtaining a lock object

classmethod get_lock()
Obtain the lock object and return

mcvirt.rpc.lock.**locking_method**(*object_type=None*, *instance_method=True*)
Provide a decorator method for locking the node whilst performing the method

mcvirt.rpc.name_server module

Thread for running the name server

class mcvirt.rpc.name_server.**NameServer**
Bases: object

Thread for running the name server

start()
Start the Pyro name server

mcvirt.rpc.pyro_object module

Base class for providing Pyro-based methods for objects

class mcvirt.rpc.pyro_object.**PyroObject**
Bases: object

Base class for providing Pyro-based methods for objects

mcvirt.rpc.rpc_daemon module

Provide class for RPC daemon.

class mcvirt.rpc.rpc_daemon.**BaseRpcDaemon**(**args*, ***kwargs*)
Bases: Pyro4.core.Daemon

Override Pyro daemon to add authentication checks and MCVirt integration

validateHandshake(*conn*, *data*)
Perform authentication on new connections

class mcvirt.rpc.rpc_daemon.**DaemonSession**
Bases: object

Class for allowing client to obtain the session ID

get_session_id()
Return the client's current session ID

class mcvirt.rpc.rpc_daemon.**RpcNSMixinDaemon**
Bases: object

Wrapper for the daemon. Required since the Pyro daemon class overrides get/setattr and other built-in object methods

DAEMON = None

obtain_connection()
Attempt to obtain a connection to the name server.

register(*obj_or_class*, *objectId*, **args*, ***kwargs*)
Override register to register object with NS.

register_factories()
Register base MCVirt factories with RPC daemon

shutdown(*signum*, *frame*)
Shutdown Pyro Daemon

start(**args*, ***kwargs*)
Start the Pyro daemon

mcvirt.rpc.ssl_socket module

Provides methods for wrapping Pyro methods with SSL

class mcvirt.rpc.ssl_socket.**SSLSocket**
Bases: object

Provides methods for wrapping Pyro methods with SSL

static create_broadcast_ssl_socket(**args*, ***kwargs*)
Override the Pyro createBroadcastSocket method and wrap with SSL

static create_ssl_socket(**args*, ***kwargs*)
Override the Pyro createSocket method and wrap with SSL

static wrap_socket(*socket_object*, **args*, ***kwargs*)
Wrap a Pyro socket connection with SSL

Module contents

mcvirt.test package

Subpackages

mcvirt.test.lock package

Submodules

mcvirt.test.lock.lock_tests module

class mcvirt.test.lock.lock_tests.**LockTests**(*methodName='runTest'*)

Bases: *mcvirt.test.test_base.TestBase*

Provide unit tests for the functionality provided by the node subparser

static suite()

Return a test suite

test_method_lock_escape_return()

Test whether locks can be cleared and clear_method_lock returns accurateley

test_method_lock_rpc()

Test whether locks can be cleared over the RPC

Module contents

mcvirt.test.node package

Submodules

mcvirt.test.node.network_tests module

class mcvirt.test.node.network_tests.**NetworkTests**(*methodName='runTest'*)

Bases: *mcvirt.test.test_base.TestBase*

Test suite for performing tests on the network class

static suite()

Return a test suite of the network tests

test_create()

Test the creation of network through the argument parser

test_delete()

Test deleting a network through the argument parser

test_delete_non_existent()

Attempt to delete a non-existent network

test_delete_utilized()

Attempt to remove a network that is in use by a VM

test_duplicate_name_create()

Test attempting to create a network with a duplicate name through the argument parser

test_list()
Attempt to use the parser to list the networks

mcvirt.test.node.node_tests module

class mcvirt.test.node.node_tests.**NodeTests**(*methodName='runTest'*)
Bases: *mcvirt.test.test_base.TestBase*

Provide unit tests for the functionality provided by the node subparser

setUp()
Create various objects and deletes any test VMs

static suite()
Return a test suite

tearDown()
Reset any values changed to the MCVirt config

test_set_invalid_ip_address()
Test the validity checks for IP addresses

test_set_invalid_volume_group()
Test the validity checks for volume group name

test_set_ip_address()
Change the cluster IP address using the argument parser

test_set_volume_group()
Change the cluster IP address using the argument parser

Module contents

mcvirt.test.virtual_machine package

Subpackages

mcvirt.test.virtual_machine.hard_drive package

Submodules

mcvirt.test.virtual_machine.hard_drive.drbd_tests module

class mcvirt.test.virtual_machine.hard_drive.drbd_tests.**DrbdTests**(*methodName='runTest'*)
Bases: *mcvirt.test.test_base.TestBase*

Provides unit tests for the Drbd hard drive class

static suite()
Return a test suite of the Virtual Machine tests

test_verify(**args*)

Module contents

Submodules

mcvirt.test.virtual_machine.online_migrate_tests module

class mcvirt.test.virtual_machine.online_migrate_tests.**LibvirtConnectorUnitTest**
Bases: *mcvirt.libvirt_connector.LibvirtConnector*

Override LibvirtConnector class to provide ability to cause connection errors whilst connecting to remote libvirt instances

get_connection(*server=None*)

exception mcvirt.test.virtual_machine.online_migrate_tests.**LibvirtFailureSimulationException**
Bases: *mcvirt.exceptions.MCVirtException*

A libvirt command has been simulated to fail

class mcvirt.test.virtual_machine.online_migrate_tests.**OnlineMigrateTests**(*methodName='runTest'*)
Bases: *mcvirt.test.test_base.TestBase*

Provides unit tests for the onlineMigrate function

RPC_DAEMON = None

setUp()
Create various objects and deletes any test VMs

static suite()
Return a test suite of the online migrate tests

tearDown()
Stops and tears down any test VMs

test_migrate(**args*)

test_migrate_drbd_not_connected(**args*)

test_migrate_inappropriate_node(**args*)

test_migrate_invalid_iso(**args*)

test_migrate_invalid_network(**args*)

test_migrate_invalid_node(**args*)

test_migrate_libvirt_connection_failure(**args*)

test_migrate_locked(**args*)

test_migrate_post_migration_libvirt_failure(**args*)

test_migrate_pre_migration_libvirt_failure(**args*)

test_migrate_stopped_vm(**args*)

test_migrate_unregistered(**args*)

class mcvirt.test.virtual_machine.online_migrate_tests.**VirtualMachineFactoryUnitTest**
Bases: *mcvirt.virtual_machine.factory.Factory*

getVirtualMachineByName(*vm_name*)
Obtain a VM object, based on VM name

class mcvirt.test.virtual_machine.online_migrate_tests.**VirtualMachineLibvirtFail**(*virtual_machine_name*)
Bases: *mcvirt.virtual_machine.virtual_machine.VirtualMachine*

Override the VirtulMachine class to add overrides for simulating libvirt failures.

LIBVIRT_FAILURE_MODE

mcvirt.test.virtual_machine.virtual_machine_tests module

class mcvirt.test.virtual_machine.virtual_machine_tests.**VirtualMachineTests**(*methodName='runTest'*)

Bases: *mcvirt.test.test_base.TestBase*

Provide unit tests for the VirtualMachine class

static suite()
Return a test suite of the Virtual Machine tests

test_clone_drbd(**args*)

test_clone_local()
Test the VM cloning in MCVirt using the argument parser

test_create(*storage_type*)
Test the creation of VMs through the argument parser

test_create_alternative_driver()
Create VMs using alternative hard drive drivers

test_create_drbd(**args*)

test_create_drbd_not_enabled(**args*)

test_create_duplicate()
Attempt to create two VMs with the same name

test_create_local()
Perform the test_create test with Local storage

test_delete(*storage_type*)
Test the deletion of a VM through the argument parser

test_delete_drbd(**args*)

test_delete_local()
Perform the test_delete test with Local storage

test_duplicate(*storage_type*)
Attempt to duplicate a VM using the argument parser and perform tests on the parent and duplicate VM

test_duplicate_drbd(**args*)

test_duplicate_local()
Perform test_duplicate test with Local storage

test_invalid_name()
Attempt to create a virtual machine with an invalid name

test_invalid_network_name()
Attempt to create a VM using a network that does not exist

test_live_iso_change()
Change the ISO attached to a running VM

test_lock()
Exercise VM locking

test_offline_migrate(**args*)

test_reset()
Reset a running VM

test_reset_stopped_vm()
Attempt to reset a stopped VM

test_start(*storage_type*)
Test starting VMs through the argument parser

test_start_drbd(**args*)

test_start_local()
Perform the test_start test with Local storage

test_start_running_vm()
Attempt to start a running VM

test_stop(*storage_type*)
Test stopping VMs through the argument parser

test_stop_drbd(**args*)

test_stop_local()
Perform the test_stop test with Local storage

test_stop_stopped_vm()
Attempt to stop an already stopped VM

test_unspecified_storage_type_drbd(**args*)

test_unspecified_storage_type_local(**args*)

test_vm_directory_already_exists()
Attempt to create a VM whilst the directory for the VM already exists

Module contents

Submodules

mcvirt.test.auth_tests module

class mcvirt.test.auth_tests.**AuthTests**(*methodName='runTest'*)
Bases: *mcvirt.test.test_base.TestBase*

Provides unit tests for the Auth class

TEST_PASSWORD = 'test-password'

TEST_USERNAME = 'test-user'

TEST_USERNAME_ALTERNATIVE = 'user-to-delete'

create_test_user(*username*, *password*)
Create a test user, annotate the user object and return it

parse_command(*command*, *username*, *password*)
Parse the specified command with the specified credentials

setUp()
Set up a test user

static suite()
Returns a test suite of the Auth tests

tearDown()
Remove the test user

test_add_delete_superuser()
Add/delete a user to/from the superuser role

test_add_duplicate_superuser()
Attempts to add a superuser twice

test_add_new_user()
Create a new user through the parser

test_add_remove_user_permission()
Add a user to a virtual machine, using the argument parser

test_attempt_add_superuser_to_vm()
Attempts to add a user as a superuser to a VM

test_change_password()
Change the password of a user through the parser

test_delete_non_existant_superuser()
Attempts to remove a non-existent user from the superuser group

test_remove_user_account()
Delete a user through the parser

mcvirt.test.run_tests module

mcvirt.test.test_base module

class mcvirt.test.test_base.**TestBase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Provide base test case, with constructor/destructor for providing access to the parser and RPC

RPC_PASSWORD = 'pass'

RPC_USERNAME = 'mjc'

create_vm(*vm_name*, *storage_type*)
Create a test VM, annotate object and ensure it exists

setUp()
Obtain connections to the daemon and create various member variables.

stop_and_delete(*vm_name*)
Stop and remove a virtual machine

tearDown()
Destroy stored objects.

mcvirt.test.test_base.**skip_drbd**(*required*)

mcvirt.test.unit_test_bootstrap module

class mcvirt.test.unit_test_bootstrap.**UnitTestBootstrap**
Bases: object

Bootstrap daemon with unit tests

daemon_loop_condition()
Provide a condition for the daemon loop

start()
Start the daemon, run the unit tests and tear down

mcvirt.test.update_tests module

class mcvirt.test.update_tests.**UpdateTests**(*methodName='runTest'*)
Bases: *mcvirt.test.test_base.TestBase*

Provide unit tests for the functionality provided by the update subparser

setUp()
Create network adapter factory

static suite()
Return a test suite

tearDown()
Tear down network adapter factory

test_remove_network()
Remove a network interface from a VM, using the parser

test_remove_network_non_existant()
Attempt to remove a network interface from a VM that doesn't exist

mcvirt.test.validation_tests module

class mcvirt.test.validation_tests.**ValidationTests**(*methodName='runTest'*)
Bases: *mcvirt.test.test_base.TestBase*

Provides unit tests for validation

static suite()
Return a test suite of validation tests

test_boolean()
Test the validation of booleans

test_create_network()
Test creating a netork with an invalid name to check that network creation uses ArgumentValidator

test_create_vm()
Test an invalid VM name to check that VM creation uses ArgumentValidator

test_drbd_resource()

test_hostnames()
Test the validation of hostnames

test_integer()
Test the validation of integers

test_network_names()
Test the validation of network names

test_pos_integer()
Test the validation of positive integers

test_validity(*validator*, *valid_list*, *invalid_list*, *expected_exception=<type 'exceptions.TypeError'>*)
Use the provided validator function to test each string in valid_list and invalid_list, failing the test if

expected_exception is raised for anything in valid_list, and failing if the exception is NOT raised for anything in invalid_list

Module contents

mcvirt.virtual_machine package

Subpackages

mcvirt.virtual_machine.hard_drive package

Submodules

mcvirt.virtual_machine.hard_drive.base module Provide base operations to manage all hard drives, used by VMs

class mcvirt.virtual_machine.hard_drive.base.**Base**(*vm_object*, *disk_id=None*, *driver=None*)

Bases: *mcvirt.rpc.pyro_object.PyroObject*

Provides base operations to manage all hard drives, used by VMs

DEFAULT_DRIVER

MAXIMUM_DEVICES = 1

SNAPSHOT_SIZE = '500M'

SNAPSHOT_SUFFIX = '_snapshot'

activateDisk()
Activates the storage volume

activateLogicalVolume(**args*, ***kwargs*)

addToVirtualMachine(**args*, ***kwargs*)

clone(*destination_vm_object*)
Clone a VM, using snapshotting, attaching it to the new VM object

config_properties
Return the disk object config items

create()
Creates a new disk image, attaches the disk to the VM and records the disk in the VM configuration

createBackupSnapshot()
Creates a snapshot of the logical volume for backing up and locks the VM

createLogicalVolume(**args*, ***kwargs*)

deactivateDisk()
Deactivates the storage volume

delete()
Delete the logical volume for the disk

deleteBackupSnapshot()
Deletes the backup snapshot for the disk and unlocks the VM

disk_id
Return the disk ID of the current disk, generating a new one if there is not already one present

driver
Return the disk drive driver name

duplicate(*destination_vm_object*)
Clone the hard drive and attach it to the new VM object

getDiskConfig()
Returns the disk configuration for the hard drive

getDiskPath()
Exposed method for _getDiskPath

getSize()
Gets the size of the disk (in MB)

get_remote_object(*node_name=None*, *remote_node=None*, *registered=True*)
Obtain an instance of the current hard drive object on a remote node

get_type()
Return the type of storage for the hard drive

increaseSize(**args*, ***kwargs*)

static isAvailable(*pyro_object*)
Returns whether the storage type is available on the node

move(*destination_node*, *source_node*)
Moves the storage to another node in the cluster

postOnlineMigration()
Performs post tasks after a VM has performed an online migration

preMigrationChecks(*destination_node*)
Determines if the disk is in a state to allow the attached VM to be migrated to another node

preOnlineMigration()
Performs required tasks in order for the underlying VM to perform an online migration

removeFromVirtualMachine(**args*, ***kwargs*)

removeLogicalVolume(**args*, ***kwargs*)

zeroLogicalVolume(**args*, ***kwargs*)

mcvirt.virtual_machine.hard_drive.drbd module

class mcvirt.virtual_machine.hard_drive.drbd.**Drbd**(*drbd_minor=None*, *drbd_port=None*, **args*, ***kwargs*)
Bases: *mcvirt.virtual_machine.hard_drive.base.Base*

Provides operations to manage Drbd-backed hard drives, used by VMs

CACHE_MODE = 'none'

CREATE_PROGRESS

Drbd_CONFIG_TEMPLATE = '/usr/lib/mcvirt/templates/drbd_resource.conf'

Drbd_META_SUFFIX = 'meta'

Drbd_RAW_SUFFIX = 'raw'

Drbd_STATES = {'CONNECTION': {'CONNECTED': [<Mock id='140115250882704'>, <Mock id='140115242915472'>

INITIAL_MINOR = 1

INITIAL_PORT = 7789

activateDisk()
Ensure that the disk is ready to be used by a VM on the local node

config_properties
Return the disk object config items

create(*size*)
Creates a new hard drive, attaches the disk to the VM and records the disk in the VM configuration

deactivateDisk()
Marks Drbd volume as secondary

drbdConnect(**args*, ***kwargs*)

drbdDisconnect(**args*, ***kwargs*)

drbdDown(**args*, ***kwargs*)

drbdGetConnectionState()
Provide an exposed method for _drbdGetConnectionState

drbdGetDiskState()
Provide an exposed method for drbdGetDiskState

drbdSetPrimary(**args*, ***kwargs*)

drbdSetSecondary(**args*, ***kwargs*)

drbdUp(**args*, ***kwargs*)

drbd_minor
Returns the Drbd port assigned to the hard drive

drbd_port
Returns the Drbd port assigned to the hard drive

generateDrbdConfig(**args*, ***kwargs*)

getSize()
Gets the size of the disk (in MB)

initialiseMetaData(**args*, ***kwargs*)

static isAvailable(*pyro_object*)
Determine if Drbd is available on the node

move(*destination_node*, *source_node*)
Replaces a remote node for the Drbd volume with a new node and syncs the data

postOnlineMigration()
Performs post tasks after a VM has performed an online migration

preMigrationChecks()
Ensures that the Drbd state of the disk is in a state suitable for migration

preOnlineMigration(*destination_node*)
Performs required tasks in order for the underlying VM to perform an online migration

removeDrbdConfig(**args*, ***kwargs*)

resource_name
Returns the Drbd resource name for the hard drive object

setSyncState(**args*, ***kwargs*)

setTwoPrimariesConfig(**args*, ***kwargs*)

verify()
Performs a verification of a Drbd hard drive

mcvirt.virtual_machine.hard_drive.factory module

class mcvirt.virtual_machine.hard_drive.factory.**Factory**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Provides a factory for creating hard drive/hard drive config objects

DEFAULT_STORAGE_TYPE = 'Local'

HARD_DRIVE_CLASS
alias of Base

OBJECT_TYPE = 'hard disk'

STORAGE_TYPES = [<class 'mcvirt.virtual_machine.hard_drive.local.Local'>, <class 'mcvirt.virtual_machine.hard_drive

create(**args*, ***kwargs*)

getClass(*storage_type*)
Obtains the hard drive class for a given storage type

getDrbdObjectByResourceName(*resource_name*)
Obtains a hard drive object for a Drbd drive, based on the resource name

getObject(*vm_object*, *disk_id*, ***config*)
Returns the storage object for a given disk

getStorageTypes()
Returns the available storage types that MCVirt provides

mcvirt.virtual_machine.hard_drive.local module

class mcvirt.virtual_machine.hard_drive.local.**Local**(**args*, ***kwargs*)
Bases: *mcvirt.virtual_machine.hard_drive.base.Base*

Provides operations to manage local hard drives, used by VMs

CACHE_MODE = 'directsync'

MAXIMUM_DEVICES = 4

activateDisk()
Starts the disk logical volume

clone(*destination_vm_object*)
Clone a VM, using snapshotting, attaching it to the new VM object

create(*size*)
Creates a new disk image, attaches the disk to the VM and records the disk in the VM configuration

deactivateDisk()
Deactivates the disk loglcal volume

getSize()
Gets the size of the disk (in MB)

increaseSize(**args*, ***kwargs*)

static isAvailable(*pyro_object*)
Determine if local storage is available on the node

preMigrationChecks()
Perform pre-migration checks

Module contents

mcvirt.virtual_machine.network_adapter package

Submodules

mcvirt.virtual_machine.network_adapter.factory module

class mcvirt.virtual_machine.network_adapter.factory.**Factory**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Factory method to create/obtain network adapter instances

NETWORK_ADAPTER_CLASS
alias of NetworkAdapter

OBJECT_TYPE = 'network adapter'

create(**args*, ***kwargs*)

getNetworkAdapterByMacAdress(*virtual_machine*, *mac_address*)
Returns the network adapter by a given MAC address

getNetworkAdaptersByVirtualMachine(*virtual_machine*)
Returns an array of network interface objects for each of the interfaces attached to the VM

mcvirt.virtual_machine.network_adapter.network_adapter module Provide class for network adapters.

class mcvirt.virtual_machine.network_adapter.network_adapter.**NetworkAdapter**(*mac_address*, *vm_object*)
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Provides operations to network interfaces attached to a VM

delete(**args*, ***kwargs*)

static generateMacAddress()
Generates a random MAC address for new VM network interfaces

getConnectedNetwork()
Returns the network that a given interface is connected to

getLibvirtConfig()
Returns a dict of the LibVirt configuration for the network interface

getMacAddress()
Returns the MAC address of the current network object

get_config()
Returns a dict of the MCVirt configuration for the network interface

Module contents

Submodules

mcvirt.virtual_machine.disk_drive module

class mcvirt.virtual_machine.disk_drive.**DiskDrive**(*vm_object*)
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Provides operations to manage the disk drive attached to a VM

attachISO(*iso_object*, *live=False*)
Attaches an ISO image to the disk drive of the VM

getCurrentDisk()
Returns the path of the disk currently attached to the VM

preOnlineMigrationChecks(*destination_node_name*)
Performs pre-online-migration checks

removeISO()
Removes ISO attached to the disk drive of a VM

mcvirt.virtual_machine.factory module

class mcvirt.virtual_machine.factory.**Factory**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Class for obtaining virtual machine objects

OBJECT_TYPE = 'virtual machine'

VIRTUAL_MACHINE_CLASS
alias of VirtualMachine

checkName(*name*, *ignore_exists=False*)

check_exists(*vm_name*)
Determines if a VM exists, given a name

create(**args*, ***kwargs*)

getAllVirtualMachines()
Return objects for all virtual machines

getAllVmNames(*node=None*)
Returns a list of all VMs within the cluster or those registered on a specific node

getVirtualMachineByName(*vm_name*)
Obtain a VM object, based on VM name

listVms(**args*, ***kwargs*)

mcvirt.virtual_machine.virtual_machine module

Provides virtual machine class.

class mcvirt.virtual_machine.virtual_machine.**VirtualMachine**(*virtual_machine_factory*, *name*)
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Provides operations to manage a LibVirt virtual machine.

OBJECT_TYPE = 'virtual machine'

clone(**args*, ***kwargs*)

delete(**args*, ***kwargs*)

duplicate(**args*, ***kwargs*)

editConfig(**args*, ***kwargs*)

ensureRegistered()
Ensures that the VM is registered

ensureRegisteredLocally()
Ensures that the VM is registered locally, otherwise an exception is thrown

ensureUnlocked()
Ensures that the VM is in an unlocked state

getAvailableNodes()
Returns the nodes that the VM can be run on

getCPU()
Returns the number of CPU cores attached to the VM

getCloneChildren()
Returns the VMs that have been cloned from the VM

getCloneParent()
Determines if a VM is a clone of another VM

getHardDriveObjects()
Returns an array of disk objects for the disks attached to the VM

getInfo()
Gets information about the current VM

getLibvirtConfig()
Returns an XML object of the libvirt configuration for the domain

getLockState()

getNode()
Returns the node that the VM is registered on

getPowerState()

getRAM()
Returns the amount of memory attached the VM

getStorageType()
Returns the storage type of the VM

getVncPort()
Returns the port used by the VNC display for the VM

get_config_object()
Return the configuration object for the VM

get_disk_drive()
Returns a disk drive object for the VM

get_libvirt_xml()
Obtain domain XML from libvirt

get_name()
Return the name of the VM

get_remote_object()
Return a instance of the virtual machine object on the machine that the VM is registered

isRegistered()
Returns true if the VM is registered on a node

isRegisteredLocally()
Returns true if the VM is registered on the local node

isRegisteredRemotely()
Returns true if the VM is registered on a remote node

move(**args*, ***kwargs*)

offlineMigrate(**args*, ***kwargs*)

onlineMigrate(**args*, ***kwargs*)

register(**args*, ***kwargs*)

reset(**args*, ***kwargs*)

setBootOrder(*boot_devices*)
Sets the boot devices and the order in which devices are booted from

setLockState(*lock_status*)

setNode(*node*)

setNodeRemote(**args*, ***kwargs*)

start(**args*, ***kwargs*)

stop(**args*, ***kwargs*)

unregister(**args*, ***kwargs*)

updateCPU(**args*, ***kwargs*)

updateRAM(**args*, ***kwargs*)

update_config(*attribute_path*, *value*, *reason*)
Updates a VM configuration attribute and replicates change across all nodes

mcvirt.virtual_machine.virtual_machine_config module

class mcvirt.virtual_machine.virtual_machine_config.**VirtualMachineConfig**(*vm_object*)
Bases: *mcvirt.config_file.ConfigFile*

Provides operations to obtain and set the MCVirt configuration for a VM

static create(*vm_name*, *available_nodes*, *cpu_cores*, *memory_allocation*)
Creates a basic VM configuration for new VMs

static get_config_path(*vm_name*)
Provides the path of the VM-spefic configuration file

Module contents

6.1.2 Submodules

6.1.3 mcvirt.argument_validator module

Argument validators.

class mcvirt.argument_validator.**ArgumentValidator**
Bases: object

Provide methods to validate argument values

static validate_boolean(*variable*)
Ensure variable is a boolean

static validate_drbd_resource(*variable*)
Validate DRBD resource name

static validate_hostname(*hostname*)
Validate a hostname

static validate_integer(*value*)
Validate integer

static validate_network_name(*name*)
Validate the name of a network

static validate_positive_integer(*value*)
Validate that a given variable is a positive integer

6.1.4 mcvirt.bash-complete module

6.1.5 mcvirt.config_file module

Provide base class for configuration files

class mcvirt.config_file.**ConfigFile**
Bases: object

Provides operations to obtain and set the MCVirt configuration for a VM

CURRENT_VERSION = 4

GIT = '/usr/bin/git'

static create()
Creates a basic VM configuration for new VMs

getPermissionConfig()

get_config()
Load the VM configuration from disk and returns the parsed JSON.

static get_config_path(*vm_name*)
Provide the path of the VM-specific configuration file

gitAdd(*message=''*)
Commits changes to an added or modified configuration file

gitRemove(*message=''*)
Removes and commits a configuration file

setConfigPermissions()
Sets file permissions for config directories

update_config(*callback_function*, *reason=''*)
Write a provided configuration back to the configuration file.

upgrade()
Performs an upgrade of the config file

6.1.6 mcvirt.constants module

Provide constants used throughout MCVirt.

class mcvirt.constants.**DirectoryLocation**
Bases: object

Provides directory/file path constants.

BASE_STORAGE_DIR = '/var/lib/mcvirt'

BASE_VM_STORAGE_DIR = '/var/lib/mcvirt/build-4189623-project-52530-mcvirt/vm'

ISO_STORAGE_DIR = '/var/lib/mcvirt/build-4189623-project-52530-mcvirt/iso'

LOCK_FILE = '/var/run/lock/mcvirt/lock'

LOCK_FILE_DIR = '/var/run/lock/mcvirt'

LOG_FILE = '/var/log/mcvirt.log'

NODE_STORAGE_DIR = '/var/lib/mcvirt/build-4189623-project-52530-mcvirt'

TEMPLATE_DIR = '/usr/lib/mcvirt/templates'

6.1.7 mcvirt.exceptions module

Provide access to all MCVirt exceptions.

exception mcvirt.exceptions.**ArgumentParserException**
Bases: *mcvirt.exceptions.MCVirtException*

An invalid argument was provided

exception mcvirt.exceptions.**AttributeAlreadyChanged**
Bases: *mcvirt.exceptions.MCVirtException*

Attribute, user is trying to change, has already changed

exception mcvirt.exceptions.**AuthenticationError**
Bases: *mcvirt.exceptions.MCVirtException*

Incorrect credentials

exception mcvirt.exceptions.**BackupSnapshotAlreadyExistsException**
Bases: *mcvirt.exceptions.MCVirtException*

The backup snapshot for the logical volume already exists

exception mcvirt.exceptions.**BackupSnapshotDoesNotExistException**
Bases: *mcvirt.exceptions.MCVirtException*

The backup snapshot for the logical volume does not exist

exception mcvirt.exceptions.**BlankPasswordException**
Bases: *mcvirt.exceptions.MCVirtException*

The provided password is blank

exception mcvirt.exceptions.**CACertificateAlreadyExists**
Bases: *mcvirt.exceptions.MCVirtException*

CA file for server already exists

exception mcvirt.exceptions.**CACertificateNotFoundException**
Bases: *mcvirt.exceptions.MCVirtException*

CA certificate for host could not be found

exception mcvirt.exceptions.**CAFileAlreadyExists**
Bases: *mcvirt.exceptions.MCVirtException*

The CA file already exists.

exception mcvirt.exceptions.**CannotCloneDrbdBasedVmsException**
Bases: *mcvirt.exceptions.MCVirtException*

Cannot clone Drbd-based VMs

exception mcvirt.exceptions.**CannotDeleteClonedVmException**
Bases: *mcvirt.exceptions.MCVirtException*

Cannot delete a cloned VM

exception mcvirt.exceptions.**CannotMigrateLocalDiskException**
Bases: *mcvirt.exceptions.MCVirtException*

Local disks cannot be migrated

exception mcvirt.exceptions.**CannotStartClonedVmException**
Bases: *mcvirt.exceptions.MCVirtException*

Cloned VMs cannot be started

exception mcvirt.exceptions.**ClusterNotInitialisedException**
Bases: *mcvirt.exceptions.MCVirtException*

The cluster has not been initialised, so cannot connect to the remote node

exception mcvirt.exceptions.**ConfigFileCouldNotBeFoundException**
Bases: *mcvirt.exceptions.MCVirtException*

Config file could not be found

exception mcvirt.exceptions.**ConnectionFailureToRemoteLibvirtInstance**
Bases: *mcvirt.exceptions.MCVirtException*

Connection failure whilst attempting to obtain a remote libvirt connection

exception mcvirt.exceptions.**CouldNotConnectToNodeException**
Bases: *mcvirt.exceptions.MCVirtException*

Could not connect to remove cluster node

exception mcvirt.exceptions.**CurrentUserError**
Bases: *mcvirt.exceptions.MCVirtException*

Error whilst obtaining current pyro user

exception mcvirt.exceptions.**DiskAlreadyExistsException**
Bases: *mcvirt.exceptions.MCVirtException*

The disk already exists

exception mcvirt.exceptions.**DrbdAlreadyEnabled**
Bases: *mcvirt.exceptions.MCVirtException*

Drbd has already been enabled on this node

exception mcvirt.exceptions.**DrbdBlockDeviceDoesNotExistException**
Bases: *mcvirt.exceptions.MCVirtException*

Drbd block device does not exist

exception mcvirt.exceptions.**DrbdNotEnabledOnNode**
Bases: *mcvirt.exceptions.MCVirtException*

Drbd volumes cannot be created on a node that has not been configured to use Drbd

exception mcvirt.exceptions.**DrbdNotInstalledException**
Bases: *mcvirt.exceptions.MCVirtException*

Drbd is not installed

exception mcvirt.exceptions.**DrbdStateException**
Bases: *mcvirt.exceptions.MCVirtException*

The Drbd state is not OK

exception mcvirt.exceptions.**DrbdVolumeNotInSyncException**
Bases: *mcvirt.exceptions.MCVirtException*

The last Drbd verification of the volume failed

exception mcvirt.exceptions.**DuplicatePermissionException**
Bases: *mcvirt.exceptions.MCVirtException*

User already exists in group

exception mcvirt.exceptions.**ExternalStorageCommandErrorException**
Bases: *mcvirt.exceptions.MCVirtException*

An error occurred whilst performing an external command

exception mcvirt.exceptions.**FailedToRemoveFileException**
Bases: *mcvirt.exceptions.MCVirtException*

A failure occurred whilst trying to remove an ISO

exception mcvirt.exceptions.**HardDriveDoesNotExistException**
Bases: *mcvirt.exceptions.MCVirtException*

The given hard drive does not exist

exception mcvirt.exceptions.**InaccessibleNodeException**
Bases: *mcvirt.exceptions.MCVirtException*, Pyro4.errors.SecurityError

Unable to connect to node in the cluster

exception mcvirt.exceptions.**IncorrectCredentials**
Bases: *mcvirt.exceptions.MCVirtException*

The supplied credentials are incorrect

exception mcvirt.exceptions.**InsufficientPermissionsException**
Bases: *mcvirt.exceptions.MCVirtException*

User does not have the required permission

exception mcvirt.exceptions.**InterfaceDoesNotExist**
Bases: *mcvirt.exceptions.MCVirtException*

Physical interface does not exist

exception mcvirt.exceptions.**InvalidArgumentException**
Bases: *mcvirt.exceptions.MCVirtException*

Argument given is not valid

exception mcvirt.exceptions.**InvalidConnectionString**
Bases: *mcvirt.exceptions.MCVirtException*

Connection string is invalid

exception mcvirt.exceptions.**InvalidIPAddressException**
Bases: *mcvirt.exceptions.MCVirtException*

The specified IP address is invalid

exception mcvirt.exceptions.**InvalidISOPathException**
Bases: *mcvirt.exceptions.MCVirtException*

ISO to add does not exist

exception mcvirt.exceptions.**InvalidNodesException**
Bases: *mcvirt.exceptions.MCVirtException*

The nodes passed is invalid

exception mcvirt.exceptions.**InvalidPermissionGroupException**
Bases: *mcvirt.exceptions.MCVirtException*

Attempted to perform actions on an invalid permission group

exception mcvirt.exceptions.**InvalidUserTypeException**
Bases: *mcvirt.exceptions.MCVirtException*

An invalid user type was specified.

exception mcvirt.exceptions.**InvalidUsernameException**
Bases: *mcvirt.exceptions.MCVirtException*

Username is within a reserved namespace

exception mcvirt.exceptions.**InvalidVirtualMachineNameException**
Bases: *mcvirt.exceptions.MCVirtException*

VM is being created with an invalid name

exception mcvirt.exceptions.**InvalidVolumeGroupNameException**
Bases: *mcvirt.exceptions.MCVirtException*

The specified name of the volume group is invalid

exception mcvirt.exceptions.**IsoAlreadyExistsException**
Bases: *mcvirt.exceptions.MCVirtException*

An ISO with the same name already exists

exception mcvirt.exceptions.**IsoInUseException**
Bases: *mcvirt.exceptions.MCVirtException*

The ISO is in use, so cannot be removed

exception mcvirt.exceptions.**IsoNotPresentOnDestinationNodeException**
Bases: *mcvirt.exceptions.MCVirtException*

ISO attached to VM does not exist on destination node whilst performing a migration]

exception mcvirt.exceptions.**LibVirtConnectionException**
Bases: *mcvirt.exceptions.MCVirtException*

An error ocurred whilst connecting to LibVirt

exception mcvirt.exceptions.**LibvirtException**
Bases: *mcvirt.exceptions.MCVirtException*

Issue with performing libvirt command

exception mcvirt.exceptions.**LibvirtNotInstalledException**
Bases: *mcvirt.exceptions.MCVirtException*

Libvirt does not appear to be installed

exception mcvirt.exceptions.**LogicalVolumeDoesNotExistException**
Bases: *mcvirt.exceptions.MCVirtException*

A required logical volume does not exist

exception mcvirt.exceptions.**MCVirtCommandException**
Bases: *mcvirt.exceptions.MCVirtException*

Provides an exception to be thrown after errors whilst calling external commands

exception mcvirt.exceptions.**MCVirtException**
Bases: exceptions.Exception

Provides an exception to be thrown for errors in MCVirt

exception mcvirt.exceptions.**MCVirtLockException**
Bases: *mcvirt.exceptions.MCVirtException*

A lock has already been found

exception mcvirt.exceptions.**MigrationFailureExcpetion**
Bases: *mcvirt.exceptions.MCVirtException*

A Libvirt Exception occurred whilst performing a migration

exception mcvirt.exceptions.**MissingConfigurationException**
Bases: *mcvirt.exceptions.MCVirtException*

Configuration is missing

exception mcvirt.exceptions.**MustGenerateCertificateException**
Bases: *mcvirt.exceptions.MCVirtException*

The certificate cannot be manually added and must be generated

exception mcvirt.exceptions.**NameNotSpecifiedException**
Bases: *mcvirt.exceptions.MCVirtException*

A name has not been specified and cannot be determined by the path/URL

exception mcvirt.exceptions.**NetworkAdapterDoesNotExistException**
Bases: *mcvirt.exceptions.MCVirtException*

The network adapter does not exist

exception mcvirt.exceptions.**NetworkAlreadyExistsException**
Bases: *mcvirt.exceptions.MCVirtException*

Network already exists with the same name

exception mcvirt.exceptions.**NetworkDoesNotExistException**
Bases: *mcvirt.exceptions.MCVirtException*

Network does not exist

exception mcvirt.exceptions.**NetworkUtilizedException**
Bases: *mcvirt.exceptions.MCVirtException*

Network is utilized by virtual machines

exception mcvirt.exceptions.**NodeAlreadyPresent**
Bases: *mcvirt.exceptions.MCVirtException*

Node being added is already connected to cluster

exception mcvirt.exceptions.**NodeAuthenticationException**
Bases: *mcvirt.exceptions.MCVirtException*

Incorrect password supplied for remote node

exception mcvirt.exceptions.**NodeDoesNotExistException**
Bases: *mcvirt.exceptions.MCVirtException*

The node does not exist

exception mcvirt.exceptions.**NodeVersionMismatch**
Bases: Pyro4.errors.SecurityError

A node is running a different version of MCVirt

exception mcvirt.exceptions.**OpenSSLNotFoundException**
Bases: *mcvirt.exceptions.MCVirtException*

The OpenSSL executable could not be found

exception mcvirt.exceptions.**PasswordsDoNotMatchException**
Bases: *mcvirt.exceptions.MCVirtException*

The new passwords do not match

exception mcvirt.exceptions.**ReachedMaximumStorageDevicesException**
Bases: *mcvirt.exceptions.MCVirtException*

Reached the limit to number of hard disks attached to VM

exception mcvirt.exceptions.**RemoteCommandExecutionFailedException**
Bases: *mcvirt.exceptions.MCVirtException*

A remote command execution fails

exception mcvirt.exceptions.**RemoteNodeLockedException**
Bases: *mcvirt.exceptions.MCVirtException*

Remote node is locked

exception mcvirt.exceptions.**RemoteObjectConflict**
Bases: *mcvirt.exceptions.MCVirtException*

The remote node contains an object that will cause conflict when syncing

exception mcvirt.exceptions.**StorageTypeNotSpecified**
Bases: *mcvirt.exceptions.MCVirtException*

Storage type has not been specified

exception mcvirt.exceptions.**StorageTypesCannotBeMixedException**
Bases: *mcvirt.exceptions.MCVirtException*

Storage types cannot be mixed within a single VM

exception mcvirt.exceptions.**UnknownRemoteCommandException**
Bases: *mcvirt.exceptions.MCVirtException*

An unknown command was passed to the remote machine

exception mcvirt.exceptions.**UnknownStorageTypeException**
Bases: *mcvirt.exceptions.MCVirtException*

An hard drive object with an unknown disk type has been initialised

exception mcvirt.exceptions.**UnprivilegedUserException**
Bases: *mcvirt.exceptions.MCVirtException*

Unprivileged user running executable

exception mcvirt.exceptions.**UnsuitableNodeException**
Bases: *mcvirt.exceptions.MCVirtException*

The node is unsuitable to run the VM

exception mcvirt.exceptions.**UserAlreadyExistsException**
Bases: *mcvirt.exceptions.MCVirtException*

The given user already exists.

exception mcvirt.exceptions.**UserDoesNotExistException**
Bases: *mcvirt.exceptions.MCVirtException*

The specified user does not exist

exception mcvirt.exceptions.**UserNotPresentInGroup**
Bases: *mcvirt.exceptions.MCVirtException*

User to be removed from group is not in the group

exception mcvirt.exceptions.**VirtualMachineDoesNotExistException**
Bases: *mcvirt.exceptions.MCVirtException*

Virtual machine does not exist

exception mcvirt.exceptions.**VirtualMachineLockException**
Bases: *mcvirt.exceptions.MCVirtException*

Lock cannot be set to the current lock state

exception mcvirt.exceptions.**VmAlreadyExistsException**
Bases: *mcvirt.exceptions.MCVirtException*

VM is being created with a duplicate name

exception mcvirt.exceptions.**VmAlreadyRegisteredException**
Bases: *mcvirt.exceptions.MCVirtException*

VM is already registered on a node

exception mcvirt.exceptions.**VmAlreadyStartedException**
Bases: *mcvirt.exceptions.MCVirtException*

VM is already started when attempting to start it

exception mcvirt.exceptions.**VmAlreadyStoppedException**
Bases: *mcvirt.exceptions.MCVirtException*

VM is already stopped when attempting to stop it

exception mcvirt.exceptions.**VmDirectoryAlreadyExistsException**
Bases: *mcvirt.exceptions.MCVirtException*

Directory for a VM already exists

exception mcvirt.exceptions.**VmIsCloneException**
Bases: *mcvirt.exceptions.MCVirtException*

VM is a clone

exception mcvirt.exceptions.**VmNotRegistered**
Bases: *mcvirt.exceptions.MCVirtException*

The virtual machine is not currently registered on a node

exception mcvirt.exceptions.**VmRegisteredElsewhereException**
Bases: *mcvirt.exceptions.MCVirtException*

Attempt to perform an action on a VM registered on another node

exception mcvirt.exceptions.**VmRunningException**
Bases: *mcvirt.exceptions.MCVirtException*

An offline migration can only be performed on a powered off VM

exception mcvirt.exceptions.**VmStoppedException**
Bases: *mcvirt.exceptions.MCVirtException*

An online migraiton can only be performed on a powered on VM

exception mcvirt.exceptions.**VncNotEnabledException**
Bases: *mcvirt.exceptions.MCVirtException*

VNC is not enabled on the VM

mcvirt.exceptions.**exception_class**
alias of *InaccessibleNodeException*

6.1.8 mcvirt.libvirt_connector module

class mcvirt.libvirt_connector.**LibvirtConnector**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

Obtains/manages Libvirt connections

get_connection(*server=None*)
Obtains a Libvirt connection for a given server

6.1.9 mcvirt.logger module

class mcvirt.logger.**LogItem**(*method*, *user*, *object_name*, *object_type*)
Bases: object

description

finish_error(*exception*)

finish_error_unknown(*exception*)

finish_success()

start()

class mcvirt.logger.**LogState**
Bases: object

FAILED = {'status': 3, 'name': 'FAILED'}

QUEUED = {'status': 0, 'name': 'QUEUED'}

RUNNING = {'status': 1, 'name': 'RUNNING'}

SUCCESS = {'status': 2, 'name': 'SUCCESS'}

class mcvirt.logger.**Logger**
Bases: *mcvirt.rpc.pyro_object.PyroObject*

LOGS = []

create_log(*method*, *user*, *object_name*, *object_type*)

get_logs(*start_log=None*, *back=0*, *newer=False*)

mcvirt.logger.**getLogNames**(*callback*, *instance_method*, *object_type*, *args*, *kwargs*)
Attempts to determine object name and object type, based on method

6.1.10 mcvirt.mcvirt-drbd module

6.1.11 mcvirt.mcvirt_config module

class mcvirt.mcvirt_config.**MCVirtConfig**
Bases: *mcvirt.config_file.ConfigFile*

Provides operations to obtain and set the MCVirt configuration for a VM

create()
Create a basic VM configuration for new VMs

getListenAddress()
Return the address that should be used for listening for connections - the stored IP address, if configured, else all interfaces

6.1.12 mcvirt.parser module

Provides argument parser.

class mcvirt.parser.**Parser**(*verbose=True*)
Bases: object

Provides an argument parser for MCVirt.

parse_arguments(*script_args=None*)
Parse arguments and performs actions based on the arguments.

print_status(*status*)
Print if the user has specified that the parser should print statuses.

class mcvirt.parser.**ThrowingArgumentParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *version=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*)
Bases: argparse.ArgumentParser

Override the ArgumentParser class, in order to change the handling of errors.

error(*message*)
Override the error function.

6.1.13 mcvirt.syslogger module

class mcvirt.syslogger.**Syslogger**
Bases: object

Provide interface for logging to log file

LOGGER_INSTANCE = None

static get_log_level()
Return the log level, set either by environmental variable or configuration in MCVirt config

static logger()
Obtain logger instance if not already create, else return cached object

6.1.14 mcvirt.system module

class mcvirt.system.**System**
Bases: object

static getNewPassword()
Prompts the user for a new password, throwing an exception is the password is not repeated correctly

static getUserInput(*display_text*, *password=False*)
Prompts the user for input

static runCommand(*command_args*, *raise_exception_on_failure=True*, *cwd=None*)
Runs system command, throwing an exception if the exit code is not 0

6.1.15 mcvirt.utils module

mcvirt.utils.**get_all_submodules**(*target_class*)
Returns all inheriting classes, recursively

mcvirt.utils.**get_hostname**()
Returns the hostname of the system

6.1.16 mcvirt.version module

6.1.17 Module contents

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

m

A

B

C

D

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

Z