UbiObjects

Modules

MySQLdb   md5   re   time
Utilities  os   string

Classes

__builtin__.object
  IPAddress
  MACAddr
  UbiKeys
  UbiNode
  UbiVirtualMachine
  UbiVirtualMachineSession

exceptions.Exception(exceptions.BaseException)
  UbiObjectException
    NoSuchObjectError
    RunOnNodeError
    SaveError
    VmCreationError

class IPAddress(__builtin__.object)
  Methods defined here:
  __add__(self, other)
  __init__(self, addr)
  __str__(self)
  Data descriptors defined here:
  __dict__
    dictionary for instance variables (if defined)
  __weakref__
    list of weak references to the object (if defined)

class MACAddr(__builtin__.object)
  Methods defined here:
  __add__(self, other)
  __init__(self, addr)
  __str__(self)
  Data descriptors defined here:
__dict__
dictionary for instance variables (if defined)

__weakref__
list of weak references to the object (if defined)

class NoSuchObjectError(UbiObjectException)
This will be raised if an object is unable to be fetched from the database

Method resolution order:
NoSuchObjectError
UbiObjectException
exceptions.Exception
exceptions.BaseException
__builtin__.object

Methods defined here:
__init__(self, type, id)
__str__(self)

Data descriptors inherited from UbiObjectException:
__weakref__
list of weak references to the object (if defined)

Data and other attributes inherited from exceptions.Exception:
__new__ = <built-in method __new__ of type object at 0x8141680>
T.__new__(S, ...) -> a new object with type S, a subtype of T

Methods inherited from exceptions.BaseException:
__delattr__(...)  
  x.__delattr__('name') <=> del x.name

__getattribute__(...)  
  x.__getattribute__('name') <=> x.name

__getitem__(...)  
  x.__getitem__(y) <=> x[y]

__getslice__(...)  
  x.__getslice__(i, j) <=> x[i:j]

Use of negative indices is not supported.

__reduce__(...)  
__repr__(...)  
  x.__repr__() <=> repr(x)
class RunOnNodeError(UbiObjectException)

If running a command on a node fails, this will be raised

Method resolution order:
RunOnNodeError
UbiObjectException
exceptions.Exception
exceptions.BaseException

Methods defined here:
__init__(self, cmd, node_id)
__str__(self)

Data descriptors inherited from UbiObjectException:
__weakref__
list of weak references to the object (if defined)

Data and other attributes inherited from exceptions.Exception:
__new__ = <built-in method __new__ of type object at 0x8141680>
T.__new__(S, ...) -> a new object with type S, a subtype of T

Methods inherited from exceptions.BaseException:
__delattr__(...)  
x.__delattr__('name') <=> del x.name
__getattr__(...)  
x.__getattr__('name') <=> x.name
__getitem__(...)  
x.__getitem__(y) <=> x[y]
__getslice__(...)  
x.__getslice__(i, j) <=> x[i:j]

Use of negative indices is not supported.
class SaveError(UbiObjectException)

This is the base exception for all exceptions raised by ubi objects
__getslice__(...)  
x. __getslice__(i, j) <=> x[i:j]  
Use of negative indices is not supported.

__reduce__(...)  
__repr__(...)  
x. __repr__() <=> repr(x)

__setattr__(...)  
x. __setattr__('name', value) <=> x.name = value

__setstate__(...)  
Data descriptors inherited from exceptions.BaseException:

__dict__
args
message  
exception message

class UbiKeys( builtin.object)  
The dumbest object ever.

Methods defined here:

__init__(self, bits=1024, progress_func=None, passphrase=None)  
Lets use the default NX keys for now. Eventually we should generate  
user-unique keys with paramiko.

Data descriptors defined here:

__dict__  
dictionary for instance variables (if defined)

__weakref__  
list of weak references to the object (if defined)

class UbiNode( builtin.object)  
An UbiNode corresponds to a physical machine which is able to have virtual  
machines running on it.

Methods defined here:

__init__(self)  
Creates an Ubi Node that is basically blank. Really never used by the cont

save(self)  
Save the object to the database

Static methods defined here:

Get(id)  
Given an id retrieves the corresponding object from the database
GetAll()
Gets all objects from the database

GetLeastLoaded()
Finds the node with the fewest number of virtual machines running on it.
Returns and UbiNode

Data descriptors defined here:
__dict__
dictionary for instance variables (if defined)
__weakref__
list of weak references to the object (if defined)

class UbiObjectException(exceptions.Exception)
This is the base exception for all exceptions raised by ubi objects

Method resolution order:
UbiObjectException
exceptions.Exception
exceptions.BaseException
__builtin___.object

Data descriptors defined here:
__weakref__
list of weak references to the object (if defined)

Methods inherited from exceptions.Exception:
__init__(...)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Data and other attributes inherited from exceptions.Exception:
__new__ = <built-in method __new__ of type object at 0x8141680>
T.__new__(S, ...) -> a new object with type S, a subtype of T

Methods inherited from exceptions.BaseException:
__delattr__(...)
x.__delattr__('name') <=> del x.name
__getattribute__(...)
x.__getattribute__('name') <=> x.name
__getitem__(...)
x.__getitem__(y) <=> x[y]
__getslice__(...)  
x. __getslice__ (i, j) <=> x[i:j]

Use of negative indices is not supported.

__reduce__(...)  
__repr__(...)  
x. __repr__ () <=> repr(x)

__setattr__(...)  
x. __setattr__ ('name', value) <=> x.name = value

__setstate__(...)  
__str__(...)  
x. __str__ () <=> str(x)

Data descriptors inherited from exceptions.BaseException:
__dict__
args
message
  exception message

class UbiVirtualMachine(__builtin.object)  
The abstract ubi-Desk virtual machine.

HACKish --currently coded as ssh calls to xen-tools/xm.  
Should be recoded for libvirt/XenAPI.

Methods defined here:
__init__(self, hostname, username)  
  Initializes a blank UbiVirtualMachine

boot(self, node_id=1)  
  Boots the VM

getSession(self)  
  Gets the current active session for this UbiVirtualMachine from the database.
  If there is no active session, it raises a NoSuchObjectError

getState(self, node_id=1)  
  Gets the state of the virtual machine by querying all of the nodes.
  Currently a bit of a hack, running this may alter the object.

isConnected(self, toPort=22, trackFile='/proc/net/ip_conntrack')  
  Checks to see whether the user is currently connected to their virtual machine.
  It checks this by checking the ipconntrack file to see if there is a connection between
  the NAT gateway and the machine.
makeNew(self, password, node_id=1)
Creates a brand new UbiVirtualMachine. That is, it creates the actual
disk image and configures it so it may be started by webdesc.
This is super complicated and can only be understood by reading the
following (pretty self explanatory) code.

reboot(self, node_id=1)
Reboots the VM

save(self)
Save the object to the database

shutdown(self, node_id=1)
Shutsdown the vm

start(self, node_id=1)
Starts the vm by restoring it from it's suspend file the filename of which
is found in self.checkpoint

stop(self, node_id=1)
Stop the vm and writes a suspend image to the file indicated by self.check

Static methods defined here:
Get(id)
Given an id retrieves the corresponding object from the database

GetByHostname(hostname)
Given a hostname this will fetch the UbiVirtualMachine which corresponds.

GetHostnames()
Return a set of all of the hostnames of user VMs

Data descriptors defined here:
__dict__
dictionary for instance variables (if defined)

__weakref__
list of weak references to the object (if defined)

Data and other attributes defined here:
off = 1
running = 3
stopped = 2
unbuilt = 0

class UbiVirtualMachineSession(__buillin__object)
The abstract ubi-Desk virtual machine.

HACKish --currently coded as ssh calls to xen-tools/xm. Should be recoded for libvirt/XenAPI.

Methods defined here:
__init__(self)
    Initializes a blank UbiVirtualMachineSession

save(self)
    Save the object to the database

Static methods defined here:
Get(id)
    Given an id retrieves the corresponding object from the database

Data descriptors defined here:
__dict__
    dictionary for instance variables (if defined)

__weakref__
    list of weak references to the object (if defined)

class VmCreationError(UbiObjectException)
    This exception is raised when a vm fails to be built correctly

Method resolution order:
    VmCreationError
    UbiObjectException
    exceptions.Exception
    exceptions.BaseException
    __builtin__.object

Methods defined here:
__init__(self, id)
__str__(self)

Data descriptors inherited from UbiObjectException:
__weakref__
    list of weak references to the object (if defined)

Data and other attributes inherited from exceptions.Exception:
__new__ = <built-in method __new__ of type object at 0x8141680>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

Methods inherited from exceptions.BaseException:
__delattr__(...)  
  x. __delattr__('name') == del x.name

__getattr__(...)  
  x. __getattr__('name') == x.name

__getitem__(...)  
  x. __getitem__(y) == x[y]

__getslice__(...)  
  x. __getslice__(i, j) == x[i:j]

  Use of negative indices is not supported.

__reduce__(...)  
__repr__(...)  
  x. __repr__() == repr(x)

__setattr__(...)  
  x. __setattr__('name', value) == x.name = value

__strstate__(...)  
Data descriptors inherited from exceptions.BaseException:
  __dict__
  args
  message
    exception message

Functions

runOnNode(cmd, user='root', port=6951, node_id=1)
  Runs the cmd on node.
  user is the username which it will use to ssh to the machine.
  port is the port on which to connect to via ssh.
  node_id is the UbiNode on which the command is to be run on.

strftime(...)  
  strftime(format[, tuple]) -> string

  Convert a time tuple to a string according to a format specification.
  See the library reference manual for formatting codes. When the time tuple is not present, current time as returned by localtime() is used.
Controller

## Controller Code

### Modules

- Daemonize
- md5
- string
- MySQLdb
- os
- threading
- Utilities
- re
- time

### Classes

```python
SimpleXMLRPCServer.SimpleXMLRPCServer(SocketServer.TCPServer,
SimpleXMLRPCServer.SimpleXMLRPCDispatcher)
Controller
__builtin__.object
UbIFarm
threading.Thread(threading._Verbose)
ConnectionChecker
```

class ConnectionChecker(threading.Thread):
    This **object** runs in the background and checks for virtual machines that the user is no longer connected to. If it finds one it shuts it down and records the bandwidth information. It then saves this and marks the VMSession as inactive.

Method resolution order:
- ConnectionChecker
- threading.Thread
- threading._Verbose
- __builtin__.object

Methods defined here:

**_init_**(self, name='Checker', interval=60)
    constructor, setting initial variables

**getRunningVMs**(self)
    Returns a list of active hostnames from xm list.

**join**(self, timeout=None)
    Stop the thread and wait for it to end.

**run**(self)
    main control loop
Methods inherited from `threading.Thread`:
__repr__(self)
getName(self)
isAlive(self)
isDaemon(self)
setDaemon(self, daemonic)
setName(self, name)
start(self)

Data descriptors inherited from `threading.Verbose`:
__dict__
dictionary for instance variables (if defined)

__weakref__
list of weak references to the object (if defined)

class Controller(`SimpleXMLRPCServer.SimpleXMLRPCServer`)
Will threading work; what are the implications?

Method resolution order:
Controller
`SimpleXMLRPCServer.SimpleXMLRPCServer`
`SocketServer.TCPServer`
`SocketServer.BaseServer`
`SimpleXMLRPCServer.SimpleXMLRPCDispatcher`

Methods defined here:
serve_forever(self)

Methods inherited from `SimpleXMLRPCServer.SimpleXMLRPCServer`:
__init__(self, addr, requestHandler=<class `SimpleXMLRPCServer.SimpleXMLRPCRequestHandler` at 0x87c476be>, logRequests=True, allow_none=False, encoding=None)

Data and other attributes inherited from `SimpleXMLRPCServer.SimpleXMLRPCServer`:
allow_reuse_address = True

Methods inherited from `SocketServer.TCPServer`:
close_request(self, request)
Called to clean up an individual request.

fileno(self)
Return socket file number.

Interface required by select().

get_request(self)
Get the request and client address from the socket.

May be overridden.
server_activate(self)
    Called by constructor to activate the server.

    May be overridden.

server_bind(self)
    Called by constructor to bind the socket.

    May be overridden.

server_close(self)
    Called to clean-up the server.

    May be overridden.

Data and other attributes inherited from SocketServer.TCPServer:
address_family = 2
request_queue_size = 5
socket_type = 1

Methods inherited from SocketServer.BaseServer:
finish_request(self, request, client_address)
    Finish one request by instantiating RequestHandlerClass.

handle_error(self, request, client_address)
    Handle an error gracefully. May be overridden.

    The default is to print a traceback and continue.

handle_request(self)
    Handle one request, possibly blocking.

process_request(self, request, client_address)
    Call finish_request.

    Overridden by ForkingMixIn and ThreadingMixIn.

verify_request(self, request, client_address)
    Verify the request. May be overridden.

    Return True if we should proceed with this request.

Methods inherited from SimpleXMLRPCServer.SimpleXMLRPCDispatcher:
register_function(self, function, name=None)
    Registers a function to respond to XML-RPC requests.

    The optional name argument can be used to set a Unicode name
    for the function.
register_instance(self, instance, allow_dotted_names=False)
Registers an instance to respond to XML-RPC requests.

Only one instance can be installed at a time.

If the registered instance has a _dispatch method then that method will be called with the name of the XML-RPC method and its parameters as a tuple e.g. instance._dispatch('add',(2,3))

If the registered instance does not have a _dispatch method then the instance will be searched to find a matching method and, if found, will be called. Methods beginning with an '_' are considered private and will not be called by SimpleXMLRPCServer.

If a registered function matches a XML-RPC request, then it will be called instead of the registered instance.

If the optional allow_dotted_names argument is true and the instance does not have a _dispatch method, method names containing dots are supported and resolved, as long as none of the name segments start with an '_'.

*** SECURITY WARNING: ***

Enabling the allow_dotted_names options allows intruders to access your module's global variables and may allow intruders to execute arbitrary code on your machine. Only use this option on a secure, closed network.

register_introspection_functions(self)
Registers the XML-RPC introspection methods in the system namespace.

see http://xmlrpc.usefulinc.com/doc/reserved.html

register_multicall_functions(self)
Registers the XML-RPC multicall method in the system namespace.

see http://www.xmlrpc.com/discuss/msgReader$1208

system_listMethods(self)
system.listMethods() => ['add', 'subtract', 'multiple']

Returns a list of the methods supported by the server.
**system_methodHelp(self, method_name)**

```python
system.methodHelp('add') => "Adds two integers together"
```

Returns a string containing documentation for the specified method.

**system_methodSignature(self, method_name)**

```python
system.methodSignature('add') => [double, int, int]
```

Returns a list describing the signature of the method. In the above example, the add method takes two integers as arguments and returns a double result.

This server does NOT support system.methodSignature.

**system_multicall(self, call_list)**

```python
system.multicall([{"methodName": 'add', 'params': [2, 2], ...}]) => [[4],...]
```

Allows the caller to package multiple XML-RPC calls into a single request.

See [http://www.xmlrpc.com/discuss/msgReader$1208](http://www.xmlrpc.com/discuss/msgReader$1208)

---

**class UbiFarm(__builtins__.object)**

This class does most of the work in the node controller including creating VMs, starting, and stopping them, and adding users to the system.

Methods defined here:

**__init__(self)**

This initialized the **UbiFarm** by starting the connection checker in a separate thread.

**createVirtualMachine(self, username, password, hostname)**

Given a username, password, and hostname this will create a virtual machine by running xen-create-image with the correct parameters.

**shutdown(self)**

This shuts down the **UbiFarm** and kills the connection checker.

**startVirtualMachine(self, id)**

Given a virtual machine id this will start the virtual machine on the least loaded node.
If no virtual machine with the given id exists, it returns false.
Otherwise it returns a hash with the information on the virtual machine that has been started inside of it.

Data descriptors defined here:

**__dict__**

dictionary for instance variables (if defined)
Functions

**__weakref__**
list of weak references to the object (if defined)

**bandwidthMonitorAdd(ip, vmsession_id)**
This helper function takes a session id and adds a rule to iptables to keep track of the bandwidth used

**bandwidthMonitorDelete(ip, vmsession_id)**
This helper function takes a session id and deletes the corresponding iptables rule which keep track of the bandwidth used

**bandwidthMonitorQuery(vmsession_id)**
This helper function takes a session id and returns the a list of the form (up, down) which corresponds to the outbound and inbound bandwidth of this virtual machine session.

**finis()**

**forward(counter, toPort=22, inInterface='eth0')**
Create a chain to send forwardPort of the gateway to port 22 of the VM

**initNetworking()**
This is just NAT code and should be run once somewhere. This code creates the IPtables rule to allow the host that the controller is running on to act as a NAT gateway.

**main()**

**strftime(...)**

```python
strftime(format[, tuple]) -> string
```
Convert a time tuple to a string according to a format specification. See the library reference manual for formatting codes. When the time tuple is not present, current time as returned by localtime() is used.
Daemonize (version 0.2)
Disk And Execution MONitor (Daemon)

Configurable daemon behaviors:

1.) The current working directory set to the "/" directory.
2.) The current file creation mode mask set to 0.
3.) Close all open files (1024).
4.) Redirect standard I/O streams to "/dev/null".

A failed call to fork() now raises an exception.

References:
1) Advanced Programming in the Unix Environment: W. Richard Stevens
2) Unix Programming Frequently Asked Questions:
   http://www.erlenstar.demon.co.uk/unix/faq_toc.html

Modules

| os  | sys |

Functions

createDaemon()

Detach a process from the controlling terminal and run it in the background as a daemon.

Data

MAXFD = 1024
REDIRECT_TO = '/dev/null'
UMASK = 0
WORKDIR = '/'
__author__ = 'Chad J. Schroeder'
__copyright__ = 'Copyright (C) 2005 Chad J. Schroeder'
__revision__ = '$Id$'
__version__ = '0.2'

Author

Chad J. Schroeder
Exceptions

exceptions.Exception (exceptions.BaseException)
ConnectError

class ConnectError (exceptions.Exception)

Method resolution order:
    ConnectError
    exceptions.Exception
    exceptions.BaseException
    __builtin__.object

Data descriptors defined here:
    __weakref__
        list of weak references to the object (if defined)

Methods inherited from exceptions.Exception:
    __init__(...)
        x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Data and other attributes inherited from exceptions.Exception:
    __new__ = <built-in method __new__ of type object at 0x8141680>
        T.__new__(S, ...) -> a new object with type S, a subtype of T

Methods inherited from exceptions.BaseException:
    __delattr__(...)
        x.__delattr__('name') ==> del x.name

    __getattribute__(...)
        x.__getattribute__('name') ==> x.name

    __getitem__(...)
        x.__getitem__(y) ==> x[y]

    __getslice__(...)
        x.__getslice__(i, j) ==> x[i:j]

    Use of negative indices is not supported.
__reduce__(...)  
__repr__(...)  
x. __repr__() <=> repr(x)  

__setattr__(...)  
x. __setattr__('name', value) <=> x.name = value  

__setstate__(...)  
__str__(...)  
x. __str__() <=> str(x)  

Data descriptors inherited from exceptions.BaseException:  
__dict__  
args  
message  
    exception message  

Functions

appendToFile(fileName, s)  
dbConnect()  
log(*args)  
runLocal(cmd, ret=False)  
runRemote(cmd, user='root', port=6951, server='10.0.0.10')  

Data

BASE_IP = '10.0.1.1'  
BASE_MAC = '00:16:3e:00:00:00'  
BASE_PORT = 20001  
COMMAND = 'tar'  
CONFIG = '/etc/xen'  
CONTROLLER_IP = '10.0.1.1'  
CONTROLLER_PORT = 34564  
DAEMONIZE = False  
DEBUG = True  
DELAY = 60  
DIR = '/mnt/global'  
DIST = 'gutsy'  
DOMAIN = '.ubi-desk.com'  
FORCE = '--force'  
INITRD = '/boot/initrd.img-2.6.22-14-xen'  
KERNEL = '/boot/vmlinuz-2.6.22-14-xen'  
LOGFILE = '/var/www/controller.log'  
MEMORY = 256  
MIRROR = 'http://ubuntu.media.mit.edu/ubuntu/'  
NETMASK = '255.0.0.0'  
NETWORK = '10.0.0.0/8'  
NODE = 1  
NUMBER = 200  
NXKEY = '-----BEGIN DSA PRIVATE
KEY-----\n-----END DSA PRIVATE

\n'
OFFSET = 10
PIPE = -1
PRIVILEGED = '10.0.0.10'
PUBLIC_IP = '129.170.43.226'
RELAY = '/mnt/global/relay/folders'
ROLE = 'passwd.role.py'
SIZE = 10
SOURCE = '/mnt/global/domains/gutsy32-template/template.tar'
SSH_PORT = 6951
SSL_OPTS = '-i /home/nsillik/webdesc-controller/id_dsa'
STOPPED = '/mnt/global/stopped'
SUPER = 'root'
TOKEN = '###'
TRACK = '/proc/net/ip_conntrack'
VALID = set([10, 11, 12, 13, 14, 15, ...])
db = 'relay'
dbhost = 'localhost'
dbpass = 'relaydbpass'
dbuser = 'relaydbuser'
passwordKey = 'fe38a93484b2997a236788783209c0f0'