
ADLES Documentation

Release 1.4.0

Christopher Goes

Sep 05, 2019

Contents:

1	Installation	3
2	Getting started	5
3	Usage	7
4	API Documentation	9
4.1	Interfaces	9
4.2	Platforms	11
4.3	Scripts	27
4.4	Utility Functions and the Parser	27
5	Indices and tables	31
	Python Module Index	33
	Index	35

ADLES automates the deterministic creation of virtualized environments for use in Cybersecurity and Information Technology (IT) education.

CHAPTER 1

Installation

```
pip install adles
```


CHAPTER 2

Getting started

```
adles -h
adles --print-spec exercise
adles --print-spec infra
adles --list-examples
adles --print-example competition
```


CHAPTER 3

Usage

Creating an environment using ADLES:

- Read the exercise and infrastructure specifications and examples of them.
- Write an infrastructure specification for your platform. (Currently, VMware vSphere is the only platform supported)
- Write an exercise specification with the environment you want created.
- Check its syntax, run the mastering phase, make your changes, and then run the deployment phase.

```
# Validate spec
adles validate my-competition.yaml

# Create Master images
adles masters my-competition.yaml

# Deploy the exercise
adles deploy my-competition.yaml

# Cleanup the environment
adles cleanup my-competition.yaml
```


4.1 Interfaces

Interfaces for the various platforms ADLES supports.

4.1.1 Generic Interface

class `adles.interfaces.interface.Interface` (*infra, spec*)

Base class for all Interfaces.

cleanup_environment (*network_cleanup=False*)

Cleans up a deployed environment.

Parameters **network_cleanup** (*bool*) – If networks should be cleaned up

cleanup_masters (*network_cleanup=False*)

Cleans up master instances.

Parameters **network_cleanup** (*bool*) – If networks should be cleaned up

create_masters ()

Master creation phase.

deploy_environment ()

Environment deployment phase.

4.1.2 vSphere Interface

```
class adles.interfaces.vsphere_interface.VsphereInterface (infra,  
                                                         spec)
```

Generic interface for the VMware vSphere platform.

```
cleanup_environment (network_cleanup=False)
```

Cleans up a deployed environment.

Parameters **network_cleanup** (*bool*) – If networks should be cleaned up

```
cleanup_masters (network_cleanup=False)
```

Cleans up any master instances.

Parameters **network_cleanup** (*bool*) – If networks should be cleaned up

```
create_masters ()
```

Exercise Environment Master creation phase.

```
deploy_environment ()
```

Exercise Environment deployment phase

4.1.3 Docker Interface

```
class adles.interfaces.docker_interface.DockerInterface (infra,  
                                                         spec)
```

Generic interface for the Docker platform.

```
cleanup_environment (network_cleanup=False)
```

Cleans up a deployed environment.

Parameters **network_cleanup** (*bool*) – If networks should be cleaned up

```
cleanup_masters (network_cleanup=False)
```

Cleans up master instances.

Parameters **network_cleanup** (*bool*) – If networks should be cleaned up

```
create_masters ()
```

Master creation phase.

```
deploy_environment ()
```

Environment deployment phase.

4.1.4 Cloud Interface

```
class adles.interfaces.cloud_interface.CloudInterface (infra,
                                                    spec)
```

Generic interface for all cloud platforms.

```
cleanup_environment (network_cleanup=False)
```

Cleans up a deployed environment.

Parameters **network_cleanup** (*bool*) – If networks should be cleaned up

```
cleanup_masters (network_cleanup=False)
```

Cleans up master instances.

Parameters **network_cleanup** (*bool*) – If networks should be cleaned up

```
create_masters ()
```

Master creation phase.

```
deploy_environment ()
```

Environment deployment phase.

4.2 Platforms

Wrappers for the various platforms ADLES supports.

4.2.1 vSphere

The vSphere platform serves as a wrapper around the pyVmomi library.

Vsphere

Holds the state and provides methods to interact with the vCenter server or ESXi host.

```
class adles.vsphere.vsphere_class.Vsphere (username=None,      pass-
                                           word=None,          host-
                                           name=None,          datacen-
                                           ter=None, datastore=None,
                                           port=443, use_ssl=False)
```

Maintains connection, logging, and constants for a vSphere instance

```
create_folder (folder_name, create_in=None)
```

Creates a VM folder in the specified folder.

Parameters

- **folder_name** (*str*) – Name of folder to create
- **create_in** – Folder to create the new folder in

[default: root folder of datacenter] :type create_in: str or vim.Folder :return: The created folder :rtype: vim.Folder

find_by_ds_path (*path*)

Finds a VM by it's location on a Datastore.

Parameters **path** (*str*) – Path to the VM's .vmx file on the Datastore

Returns The VM found

Return type vim.VirtualMachine or **None**

find_by_hostname (*hostname*, *vm_search=True*)

Find a VM or Host using a Fully-Qualified Domain Name (FQDN).

Parameters

- **hostname** (*str*) – FQDN of the VM to find
- **vm_search** – Search for VMs if True, Hosts if False

Returns The VM or host found

Return type vim.VirtualMachine or vim.HostSystem or **None**

find_by_inv_path (*path*, *datacenter=None*)

Finds a vim.ManagedEntity (VM, host, folder, etc) in a inventory.

Parameters **path** (*str*) – Path to the entity. This must include the hidden

Vsphere folder for the type: vm | network | datastore | host Example: "vm/some-things/more-things/vm-name" :param str datacenter: Name of datacenter to search in [default: instance's datacenter] :return: The entity found :rtype: vim.ManagedEntity or None

find_by_ip (*ip*, *vm_search=True*)

Find a VM or Host using a IP address.

Parameters

- **ip** (*str*) – IP address string as returned by VMware Tools ipAddress
- **vm_search** – Search for VMs if True, Hosts if False

Returns The VM or host found

Return type vim.VirtualMachine or vim.HostSystem or **None**

find_by_uuid (*uuid*, *instance_uuid=True*)

Find a VM in the datacenter with the given Instance or BIOS UUID.

Parameters

- **uuid** (*str*) – UUID to search for (Instance or BIOS for VMs)
- **instance_uuid** (*bool*) – If True, search by VM Instance UUID,

otherwise search by BIOS UUID :return: The VM found :rtype: vim.VirtualMachine or None

get_all_vms ()

Finds and returns all VMs registered in the Datacenter.

Returns All VMs in the Datacenter defined for the class

Return type *list*(vim.VirtualMachine)

get_cluster (cluster_name=None)

Finds and returns the named Cluster.

Parameters **cluster_name** (*str*) – Name of the cluster

[default: first cluster found in datacenter] :return: The cluster found :rtype: vim.ClusterComputeResource or None

get_clusters ()

Get all the clusters associated with the vCenter server.

Returns All clusters associated with the vCenter server

Return type *list*(vim.ClusterComputeResource)

get_datastore (datastore_name=None)

Finds and returns the named Datastore.

Parameters **datastore_name** (*str*) – Name of the datastore

[default: first datastore in datacenter] :return: The datastore found :rtype: vim.Datastore or None

get_entity_permissions (entity, inherited=True)

Gets permissions defined on or effective on a managed entity.

Parameters

- **entity** (*vim.ManagedEntity*) – The entity to get permissions for
- **inherited** (*bool*) – Include propagating permissions

defined in parent :return: The permissions for the entity :rtype: vim.AuthorizationManager.Permission or None

get_folder (folder_name=None)

Finds and returns the named Folder.

Parameters **folder_name** (*str*) – Name of folder [default: Datacenter vmFolder]

Returns The folder found

Return type vim.Folder

get_host (*host_name=None*)

Finds and returns the named Host System.

Parameters **host_name** (*str*) – Name of the host

[default: first host found in datacenter] :return: The host found :rtype: vim.HostSystem or None

get_info ()

Retrieves and formats basic information about the vSphere instance.

Returns formatted server information

Return type str

get_item (*vimtype, name=None, container=None, recursive=True*)

Get a item of specified name and type. Intended to be simple version of :meth: get_obj

Parameters

- **vimtype** (*vimtype*) – Type of item
- **name** (*str*) – Name of item
- **container** – Container to search in

[default: vCenter server content root folder] :param bool recursive: Recursively search for the item :return: The item found :rtype: vimtype or None

get_network (*network_name, distributed=False*)

Finds and returns the named Network.

Parameters

- **network_name** (*str*) – Name or path of the Network
- **distributed** (*bool*) – If the Network is a Distributed PortGroup

Returns The network found

Return type vim.Network or vim.dvs.DistributedVirtualPortgroup or None

get_obj (*container, vimtypes, name, recursive=True*)

Finds and returns named vim object of specified type.

Parameters

- **container** – Container to search in
- **vimtypes** (*list*) – vimtype objects to look for

- **name** (*str*) – Name of the object
- **recursive** (*bool*) – Recursively search for the item

Returns Object found with the specified name

Return type vimtype or `None`

get_objs (*container*, *vimtypes*, *recursive=True*)

Get all the vim objects associated with a given type.

Parameters

- **container** – Container to search in
- **vimtypes** (*list*) – Objects to search for
- **recursive** (*bool*) – Recursively search for the item

Returns All vimtype objects found

Return type `list(vimtype)` or `None`

get_pool (*pool_name=None*)

Finds and returns the named vim.ResourcePool.

Parameters **pool_name** (*str*) – Name of the resource pool

[default: first pool found in datacenter] :return: The resource pool found :rtype: vim.ResourcePool or None

get_role_permissions (*role_id*)

Gets all permissions that use a particular role.

Parameters **role_id** (*int*) – ID of the role

Returns The role permissions

Return type vim.AuthorizationManager.Permission or `None`

get_users (*search=""*, *domain=""*, *exact=False*, *belong_to_group=None*,
have_user=None, *find_users=True*, *find_groups=False*)

Returns a list of the users and groups defined for the server

Note: You must hold the Authorization.ModifyPermissions

privilege to invoke this method.

Parameters **search** (*str*) – Case insensitive substring used to filter results

[default: all users] :param str domain: Domain to be searched [default: local machine] :param bool exact: Search should match user/group name exactly :param str belong_to_group: Only find users/groups that directly belong to this group :param str

have_user: Only find groups that directly contain this user :param bool find_users: Include users in results :param bool find_groups: Include groups in results :return: The users and groups defined for the server :rtype: list(vim.UserSearchResult) or None

get_vm (*vm_name*)

Finds and returns the named VM.

Parameters **vm_name** (*str*) – Name of the VM

Returns The VM found

Return type vim.VirtualMachine or **None**

map_items (*vimtypes, func, name=None, container=None, recursive=True*)

Apply a function to item(s) in a container.

Parameters

- **vimtypes** (*list*) – List of vimtype objects to look for
- **func** – Function to apply
- **name** (*str*) – Name of item to apply to
- **container** – Container to search in [default: content.rootFolder]
- **recursive** (*bool*) – Whether to recursively descend

Returns List of values returned from the function call(s)

Return type *list*

set_entity_permissions (*entity, permission*)

Defines or updates rule(s) for the given user or group on the entity.

Parameters

- **entity** (*vim.ManagedEntity*) – The entity on which to set permissions
- **permission** (*vim.AuthorizationManager.Permission*) – The permission to set

set_motd (*message*)

Sets vCenter server Message of the Day (MOTD).

Parameters **message** (*str*) – Message to set

VM

Represents a Virtual Machine.

class `adles.vsphere.vm.VM`(*vm=None, name=None, folder=None, re-source_pool=None, datastore=None, host=None*)
Represents a VMware vSphere Virtual Machine instance.

Warning: You must call `create()` if a `vim.VirtualMachine` object is not used to initialize the instance.

add_nic(*network, summary='default-summary', model='e1000'*)

Add a NIC in the portgroup to the VM. :param `vim.Network network`: Network to attach NIC to :param `str summary`: Human-readable device info [default: `default-summary`] :param `str model`: Model of virtual network adapter. Options: (`e1000` | `e1000e` | `vmxnet` | `vmxnet2` | `vmxnet3` | `pcnet32` | `sriov`) `e1000` will work on Windows Server 2003+, and `e1000e` is supported on Windows Server 2012+. VMXNET adapters require VMware Tools to be installed, and provide enhanced performance. [Read this for more details](#):

attach_iso(*iso_path, datastore=None, boot=True*)

Attaches an ISO image to a VM. :param `str iso_path`: Path in the Datastore of the ISO image to attach :param `vim.Datastore datastore`: Datastore where the ISO resides [defaults to the VM's datastore] :param `bool boot`: Set VM to boot from the attached ISO

change_hdd_mode(*mode, disk_number=1, disk_prefix='Hard disk '*)

Change the mode on a virtual HDD. :param `str mode`: New disk mode :param `int disk_number`: Disk number :param `str disk_prefix`: Disk label prefix :return: If the disk mode change operation was successful :rtype: `bool`

change_state(*state, attempt_guest=True*)

Generic power state change that uses guest OS operations if available. :param `str state`: State to change to (`on` | `off` | `reset` | `suspend`) :param `bool attempt_guest`: Attempt to use guest operations :return: If state change succeeded :rtype: `bool`

convert_template()

Converts a Virtual Machine to a Template.

convert_vm()

Converts a Template to a Virtual Machine.

create(*template=None, cpus=None, cores=None, memory=None, max_consoles=None, version=None, firmware='efi', datastore_path=None*)

Creates a Virtual Machine. :param `vim.VirtualMachine template`: Template VM to clone :param `int cpus`: Number of processors :param `int cores`: Number of processor cores :param `int memory`: Amount of RAM in MB :param `int max_consoles`: Maximum number of active console connections :param `int version`: Hardware version of the VM [default: highest host supports] :param `str firmware`: Firmware to emulate for

the VM (efi | bios) :param str datastore_path: Path to existing VM files on datastore
:return: If the creation was successful :rtype: bool

create_snapshot (*name*, *description=""*, *memory=False*, *quiesce=True*)

Creates a snapshot of the VM. :param str name: Name of the snapshot :param str description: Text description of the snapshot :param bool memory: Memory dump of the VM is included in the snapshot :param bool quiesce: Quiesce VM disks (Requires VMware Tools)

destroy ()

Destroys the VM.

edit_nic (*nic_id*, *network=None*, *summary=None*)

Edits a vNIC based on it's number. :param int nic_id: Number of network adapter on VM :param network: Network to assign the vNIC to :type network: vim.Network :param str summary: Human-readable device description :return: If the edit operation was successful :rtype: bool

edit_resources (*cpus=None*, *cores=None*, *memory=None*,
max_consoles=None)

Edits the resource limits for the VM. :param int cpus: Number of CPUs :param int cores: Number of CPU cores :param int memory: Amount of RAM in MB :param int max_consoles: Maximum number of simultaneous Mouse-KeyBoard-Screen (MKS) console connections

execute_program (*process_manager*, *program_path*, *username=None*, *password=None*, *program_args=""*)

Executes a commandline program in the VM. This requires VMware Tools to be installed on the VM. :param vim.vm.guest.ProcessManager process_manager: vSphere process manager object :param str program_path: Path to the program inside the VM :param str username: User on VM to execute program using [default: current ADLES user] :param str password: Plaintext password for the User [default: prompt user] :param str program_args: Commandline arguments for the program :return: Program Process ID (PID) if it was executed successfully, -1 if not :rtype: int

get_all_snapshots ()

Retrieves a list of all snapshots of the VM. :return: Nested List of vim.Snapshot objects :rtype: list(vim.Snapshot) or None

get_all_snapshots_info ()

Enumerates the full snapshot tree of the VM and makes it human-readable. :return: The human-readable snapshot tree info :rtype: str

get_datastore_folder ()

Gets the name of the VM's folder on the datastore. :return: The name of the datastore folder with the VM's files :rtype: str

get_hdd_by_name (*name*)

Gets a Virtual HDD from the VM. :param name: Name of the virtual HDD :return: The

HDD device :rtype: vim.vm.device.VirtualDisk or None

get_info (*detailed=False, uuids=False, snapshot=False, vnics=False*)

Get human-readable information for a VM. :param bool detailed: Add detailed information, e.g maximum memory used :param bool uuids: Whether to get UUID information :param bool snapshot: Shows the current snapshot, if any :param bool vnics: Add information about vNICs on the VM :return: The VM's information :rtype: str

get_nic_by_id (*nic_id*)

Get a vNIC by integer ID. :param int nic_id: ID of the vNIC :return: The vNIC found :rtype: vim.vm.device.VirtualEthernetCard or None

get_nic_by_name (*name*)

Gets a Virtual Network Interface Card (vNIC) from a VM. :param str name: Name of the vNIC :return: The vNIC found :rtype: vim.vm.device.VirtualEthernetCard or None

get_nic_by_network (*network*)

Finds a vNIC by it's network backing. :param vim.Network network: Network of the vNIC to match :return: Name of the vNIC :rtype: str or None

get_nics ()

Returns a list of all Virtual Network Interface Cards (vNICs) on the VM. :return: All vNICs on the VM :rtype: list(vim.vm.device.VirtualEthernetCard) or list

get_snapshot (*snapshot=None*)

Retrieves the named snapshot from the VM. :param str snapshot: Name of the snapshot [default: current snapshot] :return: The snapshot found :rtype: vim.Snapshot or None

get_snapshot_info (*name=None*)

Human-readable info on a snapshot. :param str name: Name of the snapshot to get [defaults to the current snapshot] :return: Info on the snapshot found :rtype: str

get_vim_vm ()

Get the vim.VirtualMachine instance of the VM. :return: The vim instance of the VM :rtype: vim.VirtualMachine

has_tools ()

Checks if VMware Tools is installed and working. :return: If tools are installed and working :rtype: bool

is_template ()

Checks if VM is a template. :return: If the VM is a template :rtype: bool

is_windows ()

Checks if a VM's guest OS is Windows. :return: If guest OS is Windows :rtype: bool

mount_tools ()

Mount the installer for VMware Tools.

powered_on ()

Determines if a VM is powered on. :return: If VM is powered on :rtype: bool

relocate (*host=None, datastore=None*)

Relocates the VM to a new host and/or datastore :param vim.Host host: :param vim.Datastore datastore:

remove_all_snapshots (*consolidate_disks=True*)

Removes all snapshots associated with the VM. :param bool consolidate_disks: Virtual disks of the deleted snapshot will be merged with other disks if possible

remove_device (*device_spec*)

Removes a device from the VM. :param device_spec: The specification of the device to remove :type device_spec: vim.vm.device.VirtualDeviceSpec

remove_hdd (*disk_number*)

Removes a numbered Virtual Hard Disk from the VM. :param int disk_number: Number of the HDD to remove :return: If the HDD was successfully removed :rtype: bool

remove_nic (*nic_number*)

Deletes a vNIC based on it's number. :param int nic_number: Number of the vNIC to delete :return: If removal succeeded :rtype: bool

remove_snapshot (*snapshot, remove_children=True, consolidate_disks=True*)

Removes the named snapshot from the VM. :param str snapshot: Name of the snapshot to remove :param bool remove_children: Removal of the entire snapshot subtree :param bool consolidate_disks: Virtual disks of deleted snapshot will be merged with other disks if possible

rename (*name*)

Renames the VM. :param str name: New name for the VM

resize_hdd (*size, disk_number=1, disk_prefix='Hard disk '*)

Resize a virtual HDD on the VM. :param int size: New disk size in KB :param int disk_number: Disk number :param disk_prefix: Disk label prefix :return: If the resize was successful :rtype: bool

revert_to_current_snapshot ()

Reverts the VM to the most recent snapshot.

revert_to_snapshot (*snapshot*)

Reverts VM to the named snapshot. :param str snapshot: Name of the snapshot to revert to

screenshot ()

Takes a screenshot of a VM. :return: Path to datastore location of the screenshot :rtype: str

set_note (*note*)

Sets the note on the VM. :param str note: String to set the note to

snapshot_disk_usage ()

Determines the total disk usage of a VM's snapshots. :return: Human-readable disk usage of the snapshots :rtype: str

upgrade (*version*)

Upgrades the hardware version of the VM. :param int version: Version of hardware to upgrade VM to [defaults to the latest version the VM's host supports]

`adles.vsphere.vm.is_vnic` (*device*)

Checks if the device is a VirtualEthernetCard. :param device: The device to check :return: If the device is a vNIC :rtype: bool

Host

Represents an ESXi host.

class `adles.vsphere.host.Host` (*host*)

Represents an ESXi host in a VMware vSphere environment.

create_portgroup (*name, vswitch_name, vlan=0, promiscuous=False*)

Creates a portgroup.

Parameters

- **name** (*str*) – Name of portgroup to create
- **vswitch_name** (*str*) – Name of vSwitch to create the port group on
- **vlan** (*int*) – VLAN ID of the port group
- **promiscuous** (*bool*) – Put portgroup in promiscuous mode

create_vswitch (*name, num_ports=512*)

Creates a vSwitch.

Parameters

- **name** (*str*) – Name of the vSwitch to create
- **num_ports** (*int*) – Number of ports the vSwitch should have

delete_network (*name, network_type*)

Deletes the named network from the host.

Parameters

- **name** (*str*) – Name of the vSwitch to delete
- **network_type** (*str*) – Type of the network to remove

(“vswitch” | “portgroup”)

enter_maintenance_mode (*timeout=0, spec=None*)

Enter maintenance mode.

Parameters

- **timeout** (*int*) – Seconds to wait
- **spec** (*vim.HostMaintenanceSpec*) – Actions to be taken upon entering maintenance mode

exit_maintenance_mode (*timeout=0*)

Exit maintenance mode.

Parameters **timeout** (*int*) – Seconds to wait

get_info ()

Get information about the host.

Returns Formatted host information

Return type *str*

get_net_item (*object_type, name*)

Retrieves a network object of the specified type and name from a host.

Parameters **object_type** (*str*) – Type of object to get:

(portgroup | vswitch | proxyswitch | vnic | pnic) :param *str* name: Name of network object [default: first object found] :return: The network object :rtype: vim.Network or vim.VirtualSwitch or vim.VirtualEthernetCard or None .. todo:: determine what possible return types there are

get_net_obj (*object_type, name, refresh=False*)

Retrieves a network object of the specified type and name from a host.

Parameters **object_type** (*str*) – Type of object to get:

(portgroup | vswitch | proxyswitch | vnic | pnic) :param name: Name of network object :param bool refresh: Refresh the host's network system information :return: The network object :rtype: vim.Network or vim.VirtualSwitch or vim.VirtualEthernetCard or None

Todo: determine what possible return types there are

get_net_objs (*object_type, refresh=False*)

Retrieves all network objects of the specified type from the host.

Parameters **object_type** (*str*) – Type of object to get:

(portgroup | vswitch | proxyswitch | vnic | pnic) :param bool refresh: Refresh the host's network system information :return: list of the network objects :rtype: list(vimtype) or None

reboot (*force=False*)

Reboots the host.

Parameters **force** (*bool*) – Force a reboot even if the host

is not in maintenance mode

shutdown (*force=False*)

Shuts down the host.

Parameters **force** (*bool*) – Force a reboot even if the host

is not in maintenance mode

Utility functions

exception `adles.vsphere.vsphere_utils.VsphereException`

`adles.vsphere.vsphere_utils.get_datastore_info(ds_obj)`

Gets a human-readable summary of a Datastore.

Parameters **ds_obj** (*vim.Datastore*) – The datastore to get information on

Returns The datastore's information

Return type *str*

`adles.vsphere.vsphere_utils.is_folder(obj)`

Checks if object is a `vim.Folder`.

Parameters **obj** – The object to check

Returns If the object is a folder

Return type *bool*

`adles.vsphere.vsphere_utils.is_vm(obj)`

Checks if object is a `vim.VirtualMachine`.

Parameters **obj** – The object to check

Returns If the object is a VM

Return type *bool*

`adles.vsphere.vsphere_utils.make_vsphere(filename=None)`

Creates a vSphere object using either a JSON file or by prompting the user.

Parameters **filename** (*str*) – Name of JSON file with connection info

Returns vSphere object

Return type *Vsphere*

`adles.vsphere.vsphere_utils.resolve_path(server, thing, prompt="")`

This is a hacked together script utility to get folders or VMs.

Parameters

- **server** (*Vsphere*) – Vsphere instance
- **thing** (*str*) – String name of thing to get (folder | vm)
- **prompt** (*str*) – Message to display

Returns (thing, thing name)

Return type *tuple*(*vimtype*, *str*)

```
adles.vsphere.vsphere_utils.wait_for_task (task, timeout=60.0,  
                                              pause_timeout=True)
```

Waits for a single vim.Task to finish and returns its result.

Parameters

- **task** (*vim.Task*) – The task to wait for
- **timeout** (*float*) – Number of seconds to wait before terminating task
- **pause_timeout** (*bool*) – Pause timeout counter while task

is queued on server :return: Task result information (task.info.result) :rtype: str or None

```
adles.vsphere.folder_utils.cleanup (folder, vm_prefix="",  
                                       folder_prefix="", recursive=False,  
                                       destroy_folders=False, de-  
                                       stroy_self=False)
```

Cleans a folder by selectively destroying any VMs and folders it contains.

Parameters

- **folder** (*vim.Folder*) – Folder to cleanup
- **vm_prefix** (*str*) – Only destroy VMs with names starting with the prefix
- **folder_prefix** (*str*) – Only destroy or search in folders with names

starting with the prefix :param bool recursive: Recursively descend into any sub-folders
:param bool destroy_folders: Destroy folders in addition to VMs :param bool destroy_self:
Destroy the folder specified

```
adles.vsphere.folder_utils.create_folder (folder, folder_name)
```

Creates a VM folder in the specified folder.

Parameters

- **folder** (*vim.Folder*) – Folder to create the folder in
- **folder_name** (*str*) – Name of folder to create

Returns The created folder

Return type *vim.Folder* or *None*

```
adles.vsphere.folder_utils.enumerate_folder(folder, recursive=True,  
                                              power_status=False)
```

Enumerates a folder structure and returns the result.

as a python object with the same structure :param folder: Folder to enumerate :type folder: vim.Folder :param bool recursive: Whether to recurse into any sub-folders :param bool power_status: Display the power state of the VMs in the folder :return: The nested python object with the enumerated folder structure :rtype: list(list, str)

```
adles.vsphere.folder_utils.find_in_folder(folder, name, recur-  
                                           sive=False, vimtype=None)
```

Finds and returns an specific object in a folder.

Parameters

- **folder** (*vim.Folder*) – Folder to search in
- **name** (*str*) – Name of the object to find
- **recursive** (*bool*) – Recurse into sub-folders
- **vimtype** – Type of object to search for

Returns The object found

Return type vimtype or *None*

```
adles.vsphere.folder_utils.format_structure(structure, indent=4,  
                                           _depth=0)
```

Converts a nested structure of folders into a formatted string.

Parameters

- **structure** (*tuple(list(str), str)*) – structure to format
- **indent** (*int*) – Number of spaces to indent each level of nesting
- **_depth** (*int*) – Current depth (USED INTERNALLY BY FUNCTION)

Returns Formatted string of the folder structure

Return type *str*

```
adles.vsphere.folder_utils.get_in_folder(folder, name, recur-  
                                           sive=False, vimtype=None)
```

Retrieves an item from a datacenter folder.

Parameters

- **folder** (*vim.Folder*) – Folder to search in
- **name** (*str*) – Name of object to find
- **recursive** (*bool*) – Recurse into sub-folders

- **vimtype** – Type of object to search for

Returns The object found

Return type vimtype or `None`

`adles.vsphere.folder_utils.move_into(folder, entity_list)`

Moves a list of managed entities into the named folder.

Parameters

- **folder** (`vim.Folder`) – Folder to move entities into
- **entity_list** (`list(vim.ManagedEntity)`) – Entities to move into the folder

`adles.vsphere.folder_utils.rename(folder, name)`

Renames a folder.

Parameters

- **folder** (`vim.Folder`) – Folder to rename
- **name** (`str`) – New name for the folder

`adles.vsphere.folder_utils.retrieve_items(folder, vm_prefix="",
folder_prefix="", recursive=False)`

Retrieves VMs and folders from a folder structure.

Parameters **folder** – Folder to begin search in

(Note: it is NOT returned in list of folders) :type folder: `vim.Folder` :param str vm_prefix: VM prefix to search for :param str folder_prefix: Folder prefix to search for :param bool recursive: Recursively descend into sub-folders

Warning: This will recurse regardless of folder prefix!

Returns The VMs and folders found in the folder

Return type `tuple(list(vim.VirtualMachine), list(vim.Folder))`

`adles.vsphere.folder_utils.traverse_path(folder, path,
lookup_root=None, generate=False)`

Traverses a folder path to find a object with a specific name.

Parameters

- **folder** (`vim.Folder`) – Folder to search in
- **path** (`str`) – Path in POSIX format

(Templates/Windows/ to get the 'Windows' folder) :param lookup_root: If root of path is not found in folder, lookup using this Vsphere object :type lookup_root: Vsphere or None :param bool generate: Parts of the path that do not exist are created. :return: Object at the end of the path :rtype: vimtype or None

```
adles.vsphere.network_utils.create_portgroup(name, host,
                                              vswitch_name, vlan=0,
                                              promiscuous=False)
```

Creates a portgroup on a ESXi host.

Parameters

- **name** – Name of portgroup to create
- **host** – vim.HostSystem on which to create the port group
- **vswitch_name** – Name of vSwitch on which to create the port group
- **vlan** – VLAN ID of the port group
- **promiscuous** – Put portgroup in promiscuous mode

4.3 Scripts

There are a number of scripts for each platform that perform simple tasks without having to write specifications.

4.3.1 vSphere Scripts

4.4 Utility Functions and the Parser

4.4.1 Groups

class `adles.group.Group` (*name, group, instance=None*)
Manages a group of users that has been loaded from a specification

`adles.group.get_ad_groups` (*groups*)
Extracts Active Directory-type groups from a dict of groups.

Parameters `groups` (*dict*) – Dict of groups and lists of groups

Returns List of AD groups extracted

Return type `list(Group)`

4.4.2 Parser

`adles.parser.check_syntax(specfile_path: str, spec_type: str = 'exercise') → Optional[dict]`

Checks the syntax of a specification file.

Parameters

- **specfile_path** – Path to the YAML specification file
- **spec_type** – Type of specification file

(exercise | package | infra) :return: The specification

`adles.parser.parse_yaml(filename: str) → Optional[dict]`

Parses a YAML file and returns a nested dictionary containing its contents.

Parameters **filename** – Name of YAML file to parse

Returns Parsed file contents

`adles.parser.parse_yaml_file(file: _io.TextIOWrapper) → Optional[dict]`

Parses a YAML file and returns a nested dictionary containing its contents.

Parameters **file** – Handle of file to parse

Returns Parsed file contents

`adles.parser.verify_exercise_syntax(spec: dict) → Tuple[int, int]`

Verifies the syntax of an environment specification.

Parameters **spec** – Dictionary of environment specification

Returns Number of errors, Number of warnings

`adles.parser.verify_infra_syntax(infra: dict) → Tuple[int, int]`

Verifies the syntax of an infrastructure specification.

Parameters **infra** – infrastructure

Returns Number of errors, Number of warnings

`adles.parser.verify_package_syntax(package: dict) → Tuple[int, int]`

Verifies the syntax of a package specification.

Parameters **package** – Dictionary representation of the package specification

Returns Number of errors, Number of warnings

4.4.3 Utils

`class adles.utils.TqdmHandler(level=0)`

emit (*record*)

Emit a record.

If a formatter is specified, it is used to format the record. The record is then written to the stream with a trailing newline. If exception information is present, it is formatted using `traceback.print_exception` and appended to the stream. If the stream has an ‘encoding’ attribute, it is used to determine how to do the output to the stream.

`adles.utils.get_vlan()` → int

Generates a globally unique VLAN tags.

Returns VLAN tag

`adles.utils.handle_keyboard_interrupt` (*func: Callable*) → Callable

Function decorator to handle keyboard interrupts in a consistent manner.

`adles.utils.pad` (*value: int, length: int = 2*) → str

Adds leading and trailing zeros to value (“pads” the value).

```
>>> pad(5)
05
>>> pad(9, 3)
009
```

Parameters

- **value** – integer value to pad
- **length** – Length to pad to

Returns string of padded value

`adles.utils.read_json` (*filename: str*) → Optional[dict]

Reads input from a JSON file and returns the contents.

Parameters **filename** – Path to JSON file to read

Returns Contents of the JSON file

`adles.utils.setup_logging` (*filename: str, colors: bool = True, console_verbose: bool = False, server: Tuple[str, int] = None, show_progress: bool = True*)

Configures the logging interface used by everything for output.

Parameters

- **filename** – Name of file that logs should be saved to
- **colors** – Color the terminal output
- **console_verbose** – Print DEBUG logs to terminal
- **server** – SysLog server to forward logs to

- **show_progress** – Show live status as operations progress

`adles.utils.sizeof_fmt (num: float) → str`
Generates the human-readable version of a file size.

```
>>> sizeof_fmt(512)
512bytes
>>> sizeof_fmt(2048)
2KB
```

Parameters **num** – Robot-readable file size in bytes

Returns Human-readable file size

`adles.utils.split_path (path: str) → Tuple[List[str], str]`
Splits a filepath.

```
>>> split_path('/path/To/A/file')
(['path', 'To', 'A'], 'file')
```

Parameters **path** – Path to split

Returns Path, basename

`adles.utils.time_execution (func: Callable) → Callable`
Function decorator to time the execution of a function and log to debug.

Parameters **func** – The function to time execution of

Returns The decorated function

CHAPTER 5

Indices and tables

- `genindex`
- `modindex`
- `search`

a

- `adles.group`, [27](#)
- `adles.parser`, [28](#)
- `adles.scripts.script_base`, [27](#)
- `adles.utils`, [28](#)
- `adles.vsphere`, [11](#)
- `adles.vsphere.folder_utils`, [24](#)
- `adles.vsphere.host`, [21](#)
- `adles.vsphere.network_utils`, [27](#)
- `adles.vsphere.vm`, [16](#)
- `adles.vsphere.vsphere_scripts`, [27](#)
- `adles.vsphere.vsphere_utils`, [23](#)

A

`add_nic()` (*adles.vsphere.vm.VM method*), 17
`adles.group` (*module*), 27
`adles.parser` (*module*), 28
`adles.scripts.script_base` (*module*), 27
`adles.utils` (*module*), 28
`adles.vsphere` (*module*), 11
`adles.vsphere.folder_utils` (*module*), 24
`adles.vsphere.host` (*module*), 21
`adles.vsphere.network_utils` (*module*), 27
`adles.vsphere.vm` (*module*), 16
`adles.vsphere.vsphere_scripts` (*module*), 27
`adles.vsphere.vsphere_utils` (*module*), 23
`attach_iso()` (*adles.vsphere.vm.VM method*), 17

C

`change_hdd_mode()` (*adles.vsphere.vm.VM method*), 17
`change_state()` (*adles.vsphere.vm.VM method*), 17
`check_syntax()` (*in module adles.parser*), 28
`cleanup()` (*in module adles.vsphere.folder_utils*), 24
`cleanup_environment()`

(*adles.interfaces.cloud_interface.CloudInterface method*), 11
`cleanup_environment()` (*adles.interfaces.docker_interface.DockerInterface method*), 10
`cleanup_environment()` (*adles.interfaces.interface.Interface method*), 9
`cleanup_environment()` (*adles.interfaces.vsphere_interface.VsphereInterface method*), 10
`cleanup_masters()` (*adles.interfaces.cloud_interface.CloudInterface method*), 11
`cleanup_masters()` (*adles.interfaces.docker_interface.DockerInterface method*), 10
`cleanup_masters()` (*adles.interfaces.interface.Interface method*), 9
`cleanup_masters()` (*adles.interfaces.vsphere_interface.VsphereInterface method*), 10
`CloudInterface` (*class in adles.interfaces.cloud_interface*), 11
`convert_template()` (*adles.vsphere.vm.VM method*), 17
`convert_vm()` (*adles.vsphere.vm.VM method*), 17
`create()` (*adles.vsphere.vm.VM method*), 17
`create_folder()`

`(adles.vsphere.vsphere_class.Vsphere method), 11`

`create_folder()` (in module `adles.vsphere.folder_utils`), 24

`create_masters()` (`adles.interfaces.cloud_interface.CloudInterface` method), 11

`create_masters()` (`adles.interfaces.docker_interface.DockerInterface` method), 10

`create_masters()` (`adles.interfaces.interface.Interface` method), 9

`create_masters()` (`adles.interfaces.vsphere_interface.VsphereInterface` method), 10

`create_portgroup()` (`adles.vsphere.host.Host` method), 21

`create_portgroup()` (in module `adles.vsphere.network_utils`), 27

`create_snapshot()` (`adles.vsphere.vm.VM` method), 18

`create_vswitch()` (`adles.vsphere.host.Host` method), 21

D

`delete_network()` (`adles.vsphere.host.Host` method), 21

`deploy_environment()` (`adles.interfaces.cloud_interface.CloudInterface` method), 11

`deploy_environment()` (`adles.interfaces.docker_interface.DockerInterface` method), 10

`deploy_environment()` (`adles.interfaces.interface.Interface` method), 9

`deploy_environment()` (`adles.interfaces.vsphere_interface.VsphereInterface` method), 10

`destroy()` (`adles.vsphere.vm.VM` method), 18

`DockerInterface` (class in `adles.interfaces.docker_interface`), 10

E

`edit_nic()` (`adles.vsphere.vm.VM` method), 18

`edit_resources()` (`adles.vsphere.vm.VM` method), 18

`emit()` (`adles.utils.TqdmHandler` method), 28

`enter_maintenance_mode()` (`adles.vsphere.host.Host` method), 21

`enumerate_folder()` (in module `adles.vsphere.folder_utils`), 24

`execute_program()` (`adles.vsphere.vm.VM` method), 18

`exit_maintenance_mode()` (`adles.vsphere.host.Host` method), 22

F

`find_by_ds_path()` (`adles.vsphere.vsphere_class.Vsphere` method), 12

`find_by_hostname()` (`adles.vsphere.vsphere_class.Vsphere` method), 12

`find_by_inv_path()` (`adles.vsphere.vsphere_class.Vsphere` method), 12

`find_by_ip()` (`adles.vsphere.vsphere_class.Vsphere` method), 12

`find_by_uuid()` (`adles.vsphere.vsphere_class.Vsphere` method), 12

`find_in_folder()` (in module `adles.vsphere.folder_utils`), 25

`format_structure()` (in module `adles.vsphere.folder_utils`), 25

G

`get_ad_groups()` (in module `adles.group`), 27

[get_all_snapshots\(\)](#) ([adles.vsphere.vm.VM method](#)), 18
[get_all_snapshots_info\(\)](#) ([adles.vsphere.vm.VM method](#)), 18
[get_all_vms\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 13
[get_cluster\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 13
[get_clusters\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 13
[get_datastore\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 13
[get_datastore_folder\(\)](#) ([adles.vsphere.vm.VM method](#)), 18
[get_datastore_info\(\)](#) (in [module adles.vsphere.vsphere_utils](#)), 23
[get_entity_permissions\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 13
[get_folder\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 13
[get_hdd_by_name\(\)](#) ([adles.vsphere.vm.VM method](#)), 18
[get_host\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 14
[get_in_folder\(\)](#) (in [module adles.vsphere.folder_utils](#)), 25
[get_info\(\)](#) ([adles.vsphere.host.Host method](#)), 22
[get_info\(\)](#) ([adles.vsphere.vm.VM method](#)), 19
[get_info\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 14
[get_item\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 14
[get_net_item\(\)](#) ([adles.vsphere.host.Host method](#)), 22
[get_net_obj\(\)](#) ([adles.vsphere.host.Host method](#)), 22
[get_net_objs\(\)](#) ([adles.vsphere.host.Host method](#)), 22
[get_network\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 14
[get_nic_by_id\(\)](#) ([adles.vsphere.vm.VM method](#)), 19
[get_nic_by_name\(\)](#) ([adles.vsphere.vm.VM method](#)), 19
[get_nic_by_network\(\)](#) ([adles.vsphere.vm.VM method](#)), 19
[get_nics\(\)](#) ([adles.vsphere.vm.VM method](#)), 19
[get_obj\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 14
[get_objs\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 15
[get_pool\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 15
[get_role_permissions\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 15
[get_snapshot\(\)](#) ([adles.vsphere.vm.VM method](#)), 19
[get_snapshot_info\(\)](#) ([adles.vsphere.vm.VM method](#)), 19
[get_users\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 15
[get_vim_vm\(\)](#) ([adles.vsphere.vm.VM method](#)), 19
[get_vlan\(\)](#) (in [module adles.utils](#)), 29
[get_vm\(\)](#) ([adles.vsphere.vsphere_class.Vsphere method](#)), 16
[Group](#) (class in [adles.group](#)), 27

H

[handle_keyboard_interrupt\(\)](#) (in [module adles.utils](#)), 29
[has_tools\(\)](#) ([adles.vsphere.vm.VM method](#)), 19
[Host](#) (class in [adles.vsphere.host](#)), 21

I

[Interface](#) (class in [adles.interfaces.interface](#)), 9

`is_folder()` (in module `adles.vsphere.vsphere_utils`), 23
`is_template()` (`adles.vsphere.vm.VM method`), 19
`is_vm()` (in module `adles.vsphere.vsphere_utils`), 23
`is_vnic()` (in module `adles.vsphere.vm`), 21
`is_windows()` (`adles.vsphere.vm.VM method`), 19

M

`make_vsphere()` (in module `adles.vsphere.vsphere_utils`), 23
`map_items()` (`adles.vsphere.vsphere_class.Vsphere method`), 16
`mount_tools()` (`adles.vsphere.vm.VM method`), 19
`move_into()` (in module `adles.vsphere.folder_utils`), 26

P

`pad()` (in module `adles.utils`), 29
`parse_yaml()` (in module `adles.parser`), 28
`parse_yaml_file()` (in module `adles.parser`), 28
`powered_on()` (`adles.vsphere.vm.VM method`), 19

R

`read_json()` (in module `adles.utils`), 29
`reboot()` (`adles.vsphere.host.Host method`), 22
`relocate()` (`adles.vsphere.vm.VM method`), 19
`remove_all_snapshots()` (`adles.vsphere.vm.VM method`), 20
`remove_device()` (`adles.vsphere.vm.VM method`), 20
`remove_hdd()` (`adles.vsphere.vm.VM method`), 20
`remove_nic()` (`adles.vsphere.vm.VM method`), 20
`remove_snapshot()` (`adles.vsphere.vm.VM method`), 20

`rename()` (`adles.vsphere.vm.VM method`), 20
`rename()` (in module `adles.vsphere.folder_utils`), 26
`resize_hdd()` (`adles.vsphere.vm.VM method`), 20
`resolve_path()` (in module `adles.vsphere.vsphere_utils`), 23
`retrieve_items()` (in module `adles.vsphere.folder_utils`), 26
`revert_to_current_snapshot()` (`adles.vsphere.vm.VM method`), 20
`revert_to_snapshot()` (`adles.vsphere.vm.VM method`), 20

S

`screenshot()` (`adles.vsphere.vm.VM method`), 20
`set_entity_permissions()` (`adles.vsphere.vsphere_class.Vsphere method`), 16
`set_motd()` (`adles.vsphere.vsphere_class.Vsphere method`), 16
`set_note()` (`adles.vsphere.vm.VM method`), 20
`setup_logging()` (in module `adles.utils`), 29
`shutdown()` (`adles.vsphere.host.Host method`), 23
`sizeof_fmt()` (in module `adles.utils`), 30
`snapshot_disk_usage()` (`adles.vsphere.vm.VM method`), 20
`split_path()` (in module `adles.utils`), 30

T

`time_execution()` (in module `adles.utils`), 30
`TqdmHandler` (class in `adles.utils`), 28
`traverse_path()` (in module `adles.vsphere.folder_utils`), 26

U

`upgrade()` (`adles.vsphere.vm.VM method`), 21

V

`verify_exercise_syntax()` (*in module*
adles.parser), [28](#)

`verify_infra_syntax()` (*in module*
adles.parser), [28](#)

`verify_package_syntax()` (*in module*
adles.parser), [28](#)

`VM` (*class in adles.vsphere.vm*), [16](#)

`Vsphere` (*class in*
adles.vsphere.vsphere_class), [11](#)

`VsphereException`, [23](#)

`VsphereInterface` (*class in*
adles.interfaces.vsphere_interface),
[10](#)

W

`wait_for_task()` (*in module*
adles.vsphere.vsphere_utils), [24](#)