Acropolis App Mobility Fabric Guide

Acropolis 4.5

10-Dec-2015

Notice

Copyright

License

The provision of this software to you does not grant any licenses or other rights under any Microsoft patents with respect to anything other than the file server implementation portion of the binaries for this software, including no licenses or any other rights in any hardware or any devices or software that are used to communicate with or in connection with this software.

Conventions

Convention	Description
variable_value	The action depends on a value that is unique to your environment.
`ncli> command`	The commands are executed in the Nutanix nCLI.
`user@host$ command`	The commands are executed as a non-privileged user (such as nutanix) in the system shell.
`root@host# command`	The commands are executed as the root user in the vSphere or Acropolis host shell.
`> command`	The commands are executed in the Hyper-V host shell.
`output`	The information is displayed as output from a command or in a log file.

Default Cluster Credentials

Interface	Target	Username	Password
Nutanix web console	Nutanix Controller VM	admin	admin
vSphere Web Client	ESXi host	root	nutanix/4u

Interface	Target	Username	Password
vSphere client	ESXi host	root	nutanix/4u
SSH client or console	ESXi host	root	nutanix/4u
SSH client or console	AHV host	root	nutanix/4u
SSH client or console	Hyper-V host	Administrator	nutanix/4u
SSH client	Nutanix Controller VM	nutanix	nutanix/4u

Version

Last modified: December 10, 2015 (2015-12-10 16:37:14 GMT-8)

Contents

1

Virtualization Management

Nutanix nodes with the Acropolis hypervisor include a distributed VM management service responsible for storing VM configuration, making scheduling decisions, and exposing a management interface.

Snapshots

Snapshots are consistent for failures. They do not include the VM's current memory image, only the VM configuration and its disk contents. The snapshot is taken atomically across the VM configuration and disks to ensure consistency.

If multiple VMs are specified when creating a snapshot, all of their configurations and disks are placed into the same consistency group. Do not specify more than 8 VMs at a time.

If no snapshot name is provided, the snapshot is referred to as "*vm_name-timestamp*", where the timestamp is in ISO-8601 format (`YYYY-MM-DDTHH:MM:SS.mmmmmm`).

VM Disks

A disk drive may either be a regular disk drive or a CD-ROM drive.

By default, regular disk drives are configured on the SCSI bus, and CD-ROM drives are configured on the IDE bus. By default, a disk drive is placed on the first available bus slot.

Disks on the SCSI bus may optionally be configured for passthrough on platforms that support iSCSI. When in passthrough mode, SCSI commands are passed directly to DSF over iSCSI. When SCSI passthrough is disabled, the hypervisor provides a SCSI emulation layer and treats the underlying iSCSI target as a block device. By default, SCSI passthrough is enabled for SCSI devices on supported platforms.

If you do not specify a container when creating a virtual disk, it is placed in the container named "default". You do not need to create the default container.

Virtual Networks (Layer 2)

Each VM network interface is bound to a virtual network. Each virtual network is bound to a single VLAN; trunking VLANs to a virtual network is not supported. Networks are designated by the L2 type (`vlan`) and the VLAN number. For example, a network bound to VLAN 66 would be named `vlan.66`.

Each virtual network maps to virtual switch `br0`. The user is responsible for ensuring that the specified virtual switch exists on all hosts, and that the physical switch ports for the virtual switch uplinks are properly configured to receive VLAN-tagged traffic.

A VM NIC must be associated with a virtual network. It is not possible to change this association. To connect a VM to a different virtual network, it is necessary to create a new NIC. While a virtual network is in use by a VM, it cannot be modified or deleted.

Managed Networks (Layer 3)

A virtual network can have an IPv4 configuration, but it is not required. A virtual network with an IPv4 configuration is a *managed network*; one without an IPv4 configuration is an *unmanaged network*. A VLAN

can have at most one managed network defined. If a virtual network is managed, every NIC must be assigned an IPv4 address at creation time.

A managed network can optionally have one or more non-overlapping DHCP pools. Each pool must be entirely contained within the network's managed subnet.

If the managed network has a DHCP pool, the NIC automatically gets assigned an IPv4 address from one of the pools at creation time, provided at least one address is available. Addresses in the DHCP pool are not reserved. That is, you can manually specify an address belonging to the pool when creating a virtual adapter. If the network has no DHCP pool, you must specify the IPv4 address manually.

All DHCP traffic on the network is rerouted to an internal DHCP server, which allocates IPv4 addresses. DHCP traffic on the virtual network (that is, between the guest VMs and the Controller VM) does not reach the physical network, and vice versa.

A network must be configured as managed or unmanaged when it is created. It is not possible to convert one to the other.

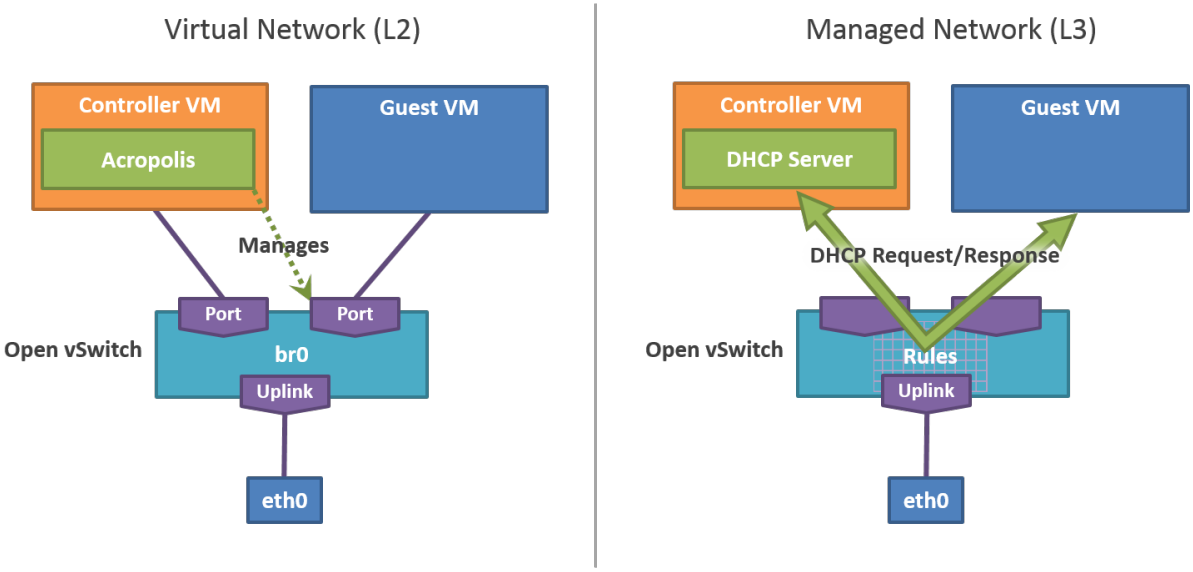

Figure: Acropolis Networking Architecture

Host Maintenance

When a host is in maintenance mode, it is marked as unschedulable so that no new VMs are instantiated on it. Subsequently, an attempt is made to evacuate VMs from the host.

If the evacuation attempt fails (for example, because there are insufficient resources available elsewhere in the cluster), the host remains in the "entering maintenance mode" state, where it is marked unschedulable, waiting for user remediation. You can shut down VMs on the host or move them to other nodes. Once the host has no more running VMs it is in maintenance mode.

When a host is in maintenance mode, VMs are moved from that host to a temporary host. After exiting maintenance mode, those VMs are automatically returned to the original host, eliminating the need to manually move them.

Limitations

Number of online VMs per host	128
Number of online VM virtual disks per host	256

Number of VMs per consistency group (with `snapshot.create`)	8
Number of VMs to edit concurrently (for example, with `vm.create/delete` and power operations)	64

Storage Overview

Acropolis uses iSCSI and NFS for storing VM files.

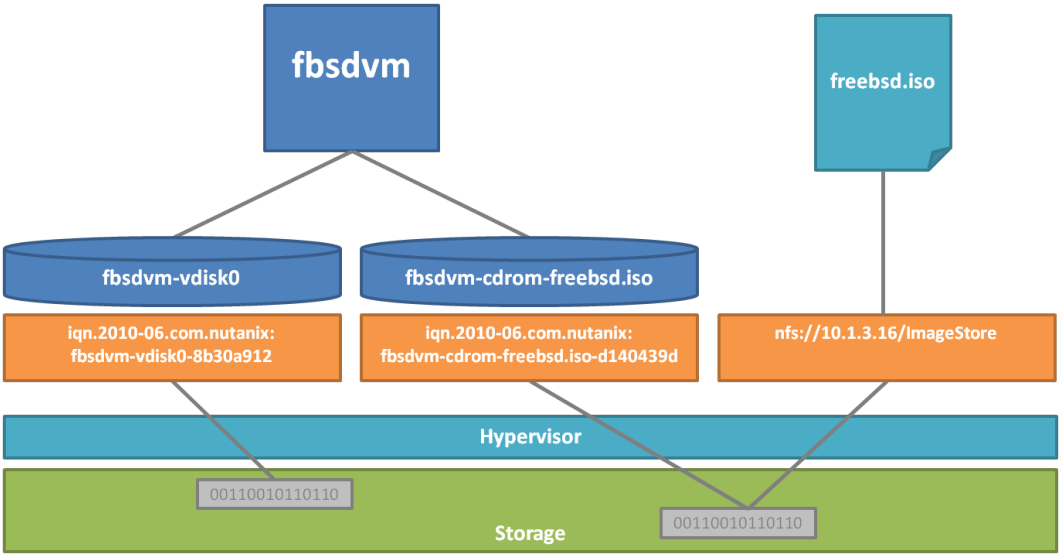

Figure: Acropolis Storage Example

iSCSI for VMs

Each disk which maps to a VM is defined as a separate iSCSI target. The Nutanix scripts work with `libvirtd` in the kernel to create the necessary iSCSI structures in Acropolis. These structures map to vDisks created in the Nutanix container specified by the administrator. If no container is specified, the script uses the default container name.

Storage High Availability with I/O Path Optimization

Unlike with Microsoft Hyper-V and VMware ESXi clusters, in which the entire traffic on a node is rerouted to a randomly selected healthy Controller VM when the local Controller VM becomes unavailable, in an Acropolis cluster, a rerouting decision is taken on a per-vDisk basis. When the local Controller VM becomes unavailable, iSCSI connections are individually redirected to a randomly selected healthy Controller VM, resulting in distribution of load across the cluster.

Instead of maintaining live, redundant connections to other Controller VMs, as is the case with the Device Mapper Multipath feature, the Acropolis hypervisor initiates an iSCSI connection to a healthy CVM only when the connection is required. When the local Controller VM becomes available, connections to other Controller VMs are terminated and the guest VMs reconnect to the local Controller VM.

NFS Datastores for Images

Nutanix containers can be accessed by the Acropolis host as NFS datastores. NFS datastores are used to manage images which may be used by multiple VMs, such as ISO files. When mapped to a VM, the script maps the file in the NFS datastore to the VM as a iSCSI device, just as it does for virtual disk files.

Images must be specified by absolute path, as if relative to the NFS server. For example, if a datastore named ImageStore exists with a subdirectory called linux, the path required to access this set of files would be `/ImageStore/linux`. Use the `nfs_ls` script to browse the datastore from the Controller VM:

```
nutanix@cvm$ nfs_ls --long --human_readable /ImageStore/linux
  -rw-rw-r-- 1 1000 1000 Dec  7  2012   1.6G CentOS-6.3-x86_64-LiveDVD.iso
  -rw-r--r-- 1 1000 1000 Jun 19 08:56 523.0M archlinux-2013.06.01-dual.iso
  -rw-rw-r-- 1 1000 1000 Jun  3 19:22 373.0M grml64-full_2013.02.iso
  -rw-rw-r-- 1 1000 1000 Nov 29  2012 694.3M ubuntu-12.04.1-amd64.iso
```

Virtualization Management Web Console Interface

Many of the virtualization management features can be managed from the Prism GUI.

In virtualization management-enabled clusters, you can do the following through the web console:

- Configure network connections
- Create virtual machines
- Manage virtual machines (launch console, start/shut down, take snapshots, migrate, clone, update, and delete)
- Monitor virtual machines
- Enable VM high availability

For more information about these features, see the *Web Console Guide*.

2

VM Management

Virtualization Management Workflows

Creating a Network

1. Define the network.

   → Virtual network

   ```
   <acropolis> net.create vlan.id
   ```

   → Managed network

   ```
   <acropolis> net.create vlan.id ip_config=default_gateway/prefix
   ```

   - Replace *id* with the number of the VLAN (`0` means no VLAN).
   - Replace *default_gateway* with the IP address of the gateway for the network and *prefix* with the network prefix (CIDR notation, for example, `10.1.1.1/24`).

2. (Optional) Configure the DHCP server to include DNS servers and search paths.

   ```
   <acropolis> net.update_dhcp_dns vlan.id servers=dns_servers domains=domains
   ```

   - Replace *dns_servers* with a comma-delimited list of DNS servers.
   - Replace *domains* with a comma-delimited list of domains.

3. (Optional) Define a pool of addresses for automatic assignment to virtual NICs.

   ```
   <acropolis> net.add_dhcp_pool vlan.id start=first_ip_addr end=last_ip_addr
   ```

   Replace *first_ip_addr* with the first IP address in the pool and *last_ip_addr* with the last.

   → If no address pool is defined, you must manually assign IP addresses to virtual NICs when you create the NIC with `vm.nic_create`.

   → If you want to exclude addresses from the defined range, you can do so with `net.add_to_ip_blacklist`.

   → If you want to define multiple pools for the network, run `net.add_dhcp_pool` multiple times.

Creating a VM

1. Define the number of virtual CPUs and memory for the VM.

   ```
   <acropolis> vm.create vm num_vcpus=vcpus memory=mem
   ```

   - Replace *vm* with the name of the VM to create.
   - Replace *vcpus* with the number of vCPUs for the VM.

- Replace *mem* with the amount of memory for the VM (with suffix **G** for size in GiB or **M** for size in MiB).

2. Attach a CD-ROM image, cloned from an DSF file.

```
<acropolis> vm.disk_create vm clone_from_adsf_file=file_path cdrom=1
```

Replace *file_path* with the path of the ISO file on DSF (for example, `/default/grml.iso`).

3. Create a virtual disk.

```
<acropolis> vm.disk_create vm create_size=disk_size
```

Replace *disk_size* with the size for the virtual disk (with suffix to indicate size in bytes: **s**=512, **k**=1000, **K**=1024, **m**=1e6, **M**=2^20, **g**=1e9, **G**=2^30, **t**=1e12, **T**=2^40).

4. Attach the VM to the network.

```
<acropolis> vm.nic_create vm network=vlan.id
```

Replace *id* with the number of the VLAN where the VM should reside.

If no DHCP pool is defined for the VLAN, add `ip=ip_addr` to set the IP address for the VM.

5. Start the VM.

```
<acropolis> vm.on vm
```

Snapshot Lifecycle

1. Create the snapshot.

```
<acropolis> vm.snapshot_create vm snapshot_name_list=vm_snap
```

- Replace *vm* with the name of the VM to create a snapshot from.
- Replace *vm_snap* with a name for the snapshot.

2. Clone from the snapshot.

```
<acropolis> vm.create new_vm clone_from_snapshot=vm_snap
```

Replace *new_vm* with the name of the VM to create from the snapshot.

3. Restore from the snapshot.

```
<acropolis> vm.restore vm vm_snap
```

4. Delete the snapshot.

```
<acropolis> snapshot.delete vm_snap
```

Volume Management

A volume group is a collection of logically related vDisks called volumes. Each volume group is identified by a UUID. Each disk of the volume group also has a UUID, and a name, and is supported by a file on DSF. Disks in a volume group are also provided with integer IDs to specify the ordering of disks. For external attachment through iSCSI, the iSCSI target name identifies the volume group, and the LUN number identifies the disk in the group.

Volume groups are managed independently of the VMs to which volumes must be explicitly attached or detached. A volume group may be configured for either exclusive or shared access.

Caution: You can attach a volume group to multiple VMs at same time. If a VM writes to a vDisk that belong to a shared volume group it may lead to data corruption. Do not use VM disks for write operations simultaneously from multiple VMs.

The volumes API exposes back-end DSF storage to guest operating system, physical hosts, and containers through iSCSI. iSCSI support allows any operating system to use the storage capabilities of DSF. In this deployment scenario, the operating system works directly with Nutanix storage bypassing any hypervisor.

Note: Currently, you can configure iSCSI multipathing by utilizing the Windows MPIO feature.

Volumes API consists of the following entities:

Volume group
iSCSI target and group of disk devices.

Disks
Storage devices in the volume group (displayed as LUNs for the iSCSI target).

Attachment
Allowing a specified initiator IQN access to the volume group.

The following image shows an example of a VM running on Nutanix with its operating system hosted on the Nutanix storage, mounting the volumes directly.

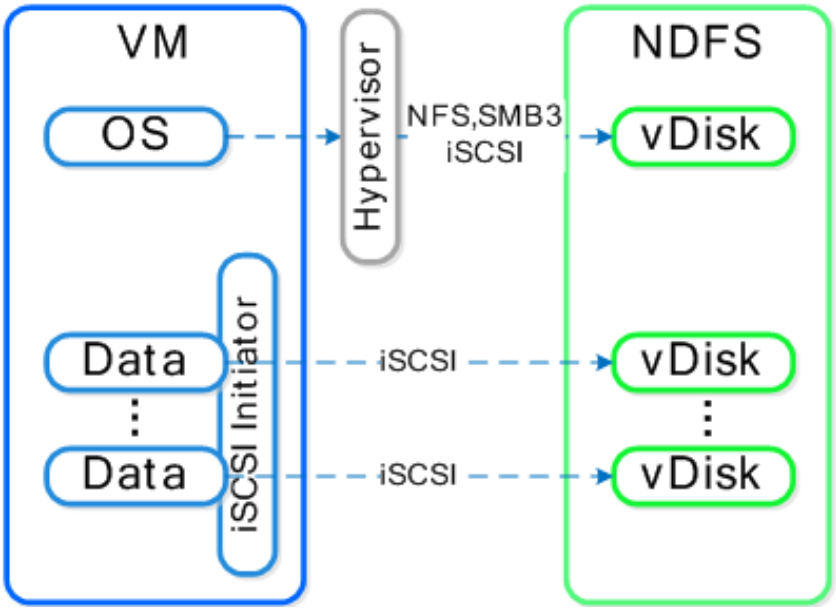

Figure: Regular Deployment Scenario

Configuring iSCSI multipathing

In Windows deployments, you can configure iSCSI multipathing with the Windows MPIO feature. It is recommended to use the failover only policy (default) to ensure vDisk ownership does not change.

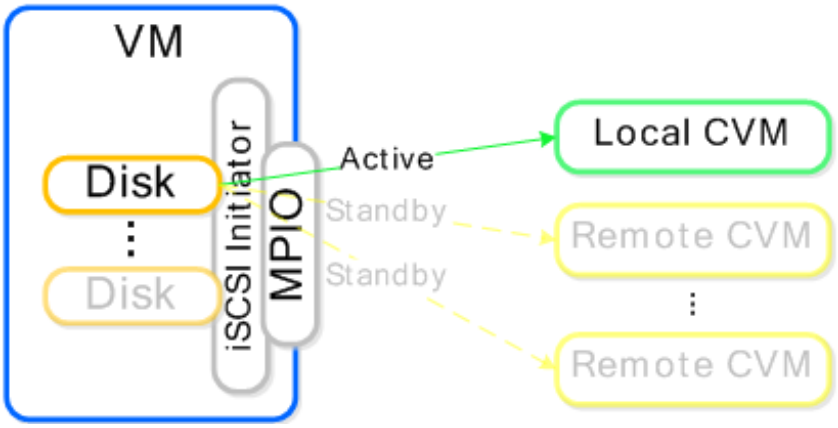

Figure: Configure iSCSI multipathing

If multiple disk devices are present, each disk can have an active path to the local Controller VM as displayed in the following image.

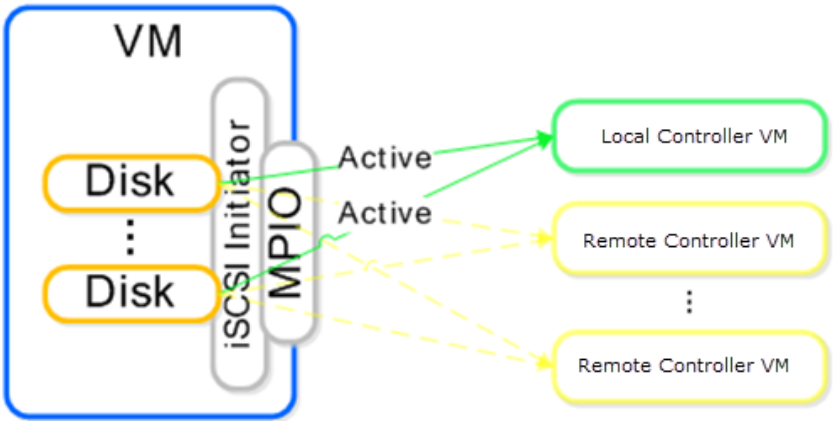

Figure: Active Path for Multiple Disk Devices

If the active Controller VM fails as displayed in the following image, another path becomes active and I/O resumes.

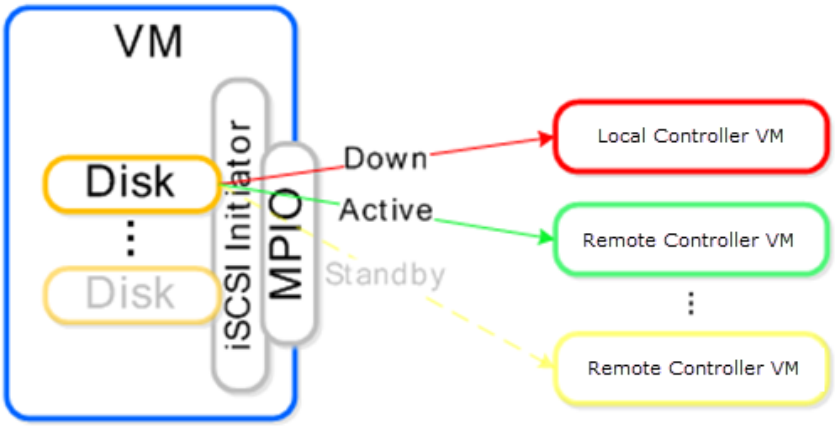

Figure: Failure Scenario

MPIO takes approximately 15 to 16 seconds to complete, which is within the Windows disk I/O timeout (default is 60 seconds). If RAID or Logical Volume Management (LVM) is desired the attached disk devices can be configured as a dynamic or a logical disk.

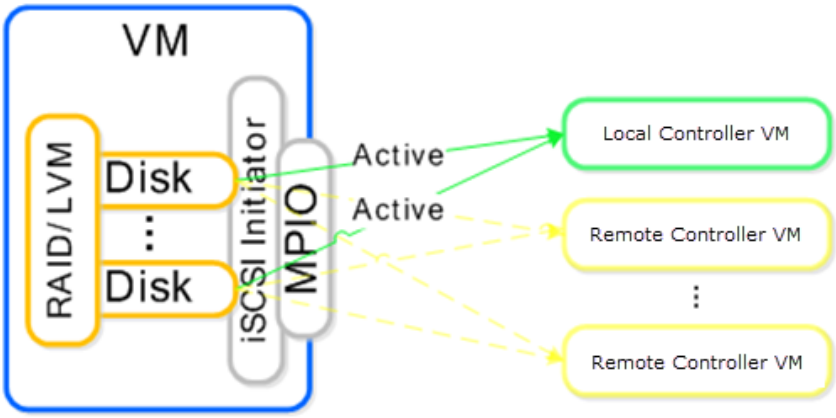

Figure: New Active Path

If the local Controller VM is heavily used, it is possible to have active paths to other Controller VMs. Having multiple active paths balances the I/O load across multiple Controller VMs; however, primary I/O has to traverse the network.

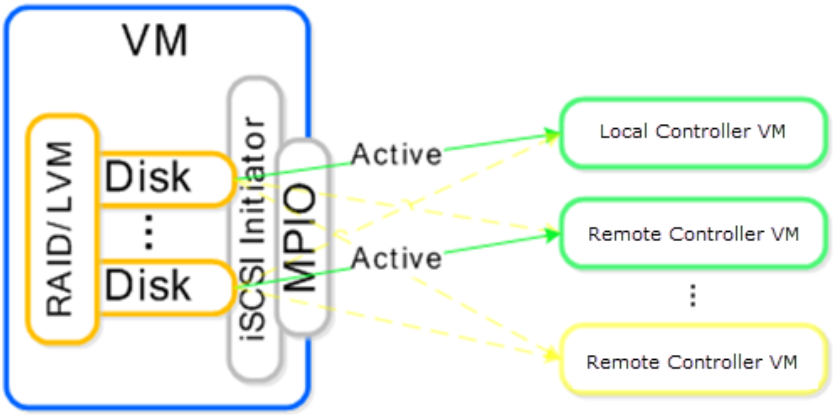

Figure: Multiple Active Paths

Configuring a Volume Group (aCLI)

To configure a volume group, do the following.

1. Create a volume group.

```
<acropolis> vg.create vg_name
```

> **Note:** The Nutanix native snapshots and disaster recovery is not supported for volume groups.

2. Add one or more disks to a volume group.

```
<acropolis> vg.disk_create vg_name container=container_name create_size=disk_size
```

3. Attach initiator IQN to volume group.

```
<acropolis> vg.attach_external vg_name_initiator_iqn
```

For detailed information volume group command-line interfaces, see *Acropolis Command-Line Interface* on page 30 .

Image Service

The image service feature enables you to import the images (ISOs, disk images, or any images which are supported in ESXi or Hyper-V format) directly into virtualization management.

The raw, vhd, vmdk, vdi, iso, qcow2 disk formats are supported.

You can create an image, delete an image, get metadata information of an existing image, list the images that you have created, and update an existing image.

You can use this feature to create disks for a VM from images (images that are stored in the image library or repository) and also an option to clone from an image.

You can import the images from the http or NFS source URL. You can use this feature to create disks for a VM from images (images that are stored in the image library or repository) and also an option to clone from an image. You must install virtIO drivers on the image prior to importing these images into the image library. For more information on how to create a VM from the imported image, see the Nutanix *Web Console Guide*.

Creating an image

```
<acropolis> image.create image_name keyword_arguments
```

If the image is created from a source_url then a container must also be provided. Otherwise, the container keyword argument is ignored and the image resides in the same container as the vmdisk. You also need to specify an image type. Image types can either be an ISO (kIsoImage) or a disk image (kDiskImage). Optionally, a checksum can also be specified if you are creating an image from a source_url in order to verify the correctness of the image.

For example, to create an image (testimage) from an image located at http://example.com/disk_image, you can use the following command.

```
<acropolis> image.create testimage source_url=http://example.com/image_iso container=default
 image_type=kIsoImage
```

For example, to create an image (testimage) from an image located at NFS server, you can use the following command.

```
<acropolis> image.create testimage source_url=nfs://nfs_server_path/path_to_image
```

To create an image (image_template) from a vmdisk 0b4fc60b-cc56-41c6-911e-67cc8406d096 (UUID of the VM).

```
<acropolis> image.create image_template clone_from_vmdisk=0b4fc60b-
cc56-41c6-911e-67cc8406d096 image_type=kDiskImage
```

For detailed information image service command-line interfaces, see *Acropolis Command-Line Interface* on page 30 .

VM Import

If you have legacy KVM VMs from a Nutanix solution that did not offer virtualization management, you must import the VMs using the `import_vm` utility from the Controller VM.

import_vm

Usage

```
nutanix@cvm$ import_vm vm [vm2 vm3 .. vmN]
```

Required Arguments

A space-separated list of VMs to import

Examples

Import two VMs

```
nutanix@cvm$ import_vm vm24 vm25
```

Import VMs from another host

```
nutanix@cvm$ import_vm --host 10.1.231.134 vm24 vm25
```

Use default network vlan.231

```
nutanix@cvm$ import_vm --default_network vlan.231 vm24 vm25
```

Optional Arguments

`--convert_virtio_disks`
: Convert Virtio disks attached to the VM to SCSI disks. (default: skip Virtio disks)

`--default_network`
: Add NICs to the network specified with this parameter if the utility cannot determine the appropriate network. (default: do not attach indeterminate NICs to any network)

`--host`
: Connect to a host other than the host where the Controller VM is running. (default: host of the Controller VM where the utility is run)

`--ignore_multiple_tags`
: Add NICs to the network specified with **--default_network** if they have multiple VLAN tags. (default: do not attach NICs with multiple VLAN tags to any network)

Supported Guest VM Types for AHV

The following shows the supported guest OS types for VMs on the AHV.

OS types with SCSI bus types

vDisk Maximum: 256

- Windows 7, 8
- Windows Server 2008, 2012
- RHEL 6.4, 6.5, 6.6, 7.0
- CentOS 6.4, 6.5, 6.6, 7.0
- Ubuntu 14.04
- FreeBSD 9.3, 10.0, 10.1

OS types with PCI bus types

vDisk Maximum: 6

- RHEL 5.10, 5.11, 6.3
- CentOS 5.10, 5.11, 6.3
- Ubuntu 12.04

Uploading Files to DSF for Microsoft Windows Users

If you are a Microsoft Windows user, you can securely upload files to DSF by using the following procedure.

1. Authenticate by using Prism username and password or, for advanced users, use the public key that is managed through the Prism cluster lockdown user interface.
2. Use WinSCP to connect to Controller VM through port 2222 and start browsing the DSF data store.

Note: The root directory displays containers and you cannot change it. You can only upload files to one of the containers and not directly to the root directory. To create or delete containers, you can use the prism user interface.

Windows VM Provisioning

Creating a Windows VM on AHV after Migration

The following describes how to create a Windows VM after you have migrated the VM to AHV from a non-Nutanix source. To install a Windows VM with Nutanix VirtIO, complete the following.

1. Log in to the Prism UI using your Nutanix credentials.
2. At the top left corner, click **Home** > **VM**.
   The *VM page* appears.
3. Click **+ Create VM** in the corner of the page.
   The *Create VM* dialog box appears.

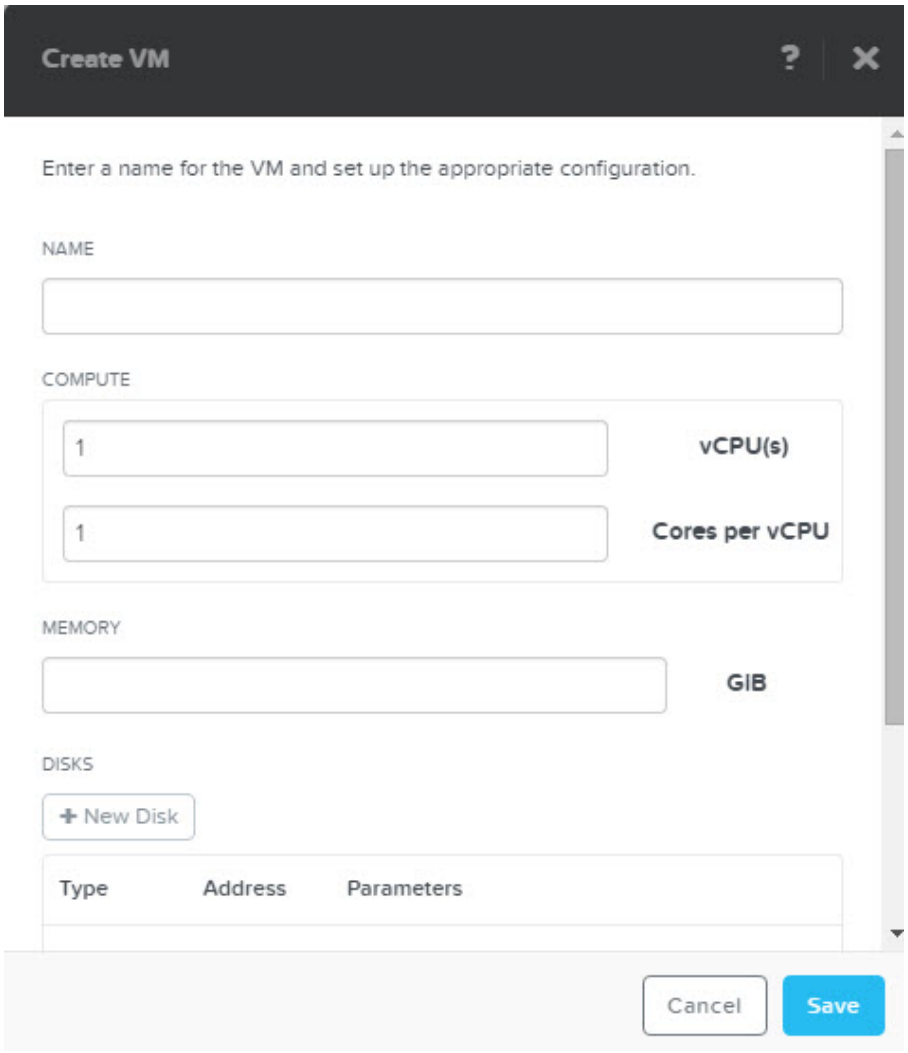

Figure: The Create VM dialog box

4. Create the VM by completing the indicated fields. Match the configuration of the previous VM.
   a. **NAME**: Enter a name for the VM.
   b. **vCPU(s)**: Enter the number of vCPUs
   c. **Cores per vCPU**: Enter the number of cores per vCPU.
   d. **MEMORY**: Enter the amount of memory for the VM (in GIB).

5. Create a disk from the disk image by clicking the **+ New Disk** button and completing the indicated fields.
   a. **TYPE**: DISK
   b. **OPERATION**: CLONE FROM IMAGE
   c. **BUS TYPE**: SCSI
   d. **CLONG FROM IMAGE SERVICE**: Click the drop-down menu and choose the image you created previously.

e. Click **Add** to add the disk driver.

6. (Optional) Add a network interface card (NIC) by clicking the **+New NIC** button and fill out the indicated fields.

   a. **VLAN ID**: Choose the VLAN ID according to network requirements and enter the IP address if required.

   b. Click **Add**.

7. Once you have filled in the indicated fields, click **Save**.

Installing Windows on a VM

Before you begin: Create a Windows VM by following *Creating a Windows VM on AHV after Migration* on page 17.

To install a Windows VM, do the following.

Note: Nutanix VirtIO cannot be used to install Windows 7 or Windows Server 2008 R2.

1. Log in to the Prism UI using your Nutanix credentials.
2. At the top left corner, click **Home** > **VM**.
   The *VM page* appears.
3. Select the Windows VM.
4. In the center of the VM page, click the **Power On** button.
5. Click the **Launch Console** button.
   The Windows console opens in a new window.
6. Select the desired language, time and currency format, and keyboard information.
7. Click **Next** > **Install Now**.
   The Windows Setup windows displays the operating systems to install.
8. Select the Windows OS you want to install.
9. Click **Next** and accept the license terms.
10. Click **Next** > **Custom: Install Windows only (advanced)** > **Load Driver** > **OK** > **Browse**.
    The browse folder opens.
11. Choose the Nutanix VirtIO CD drive and the Windows version by choosing the Nutanix VirtIO drive and then the Windows OS folder. Click **OK**.

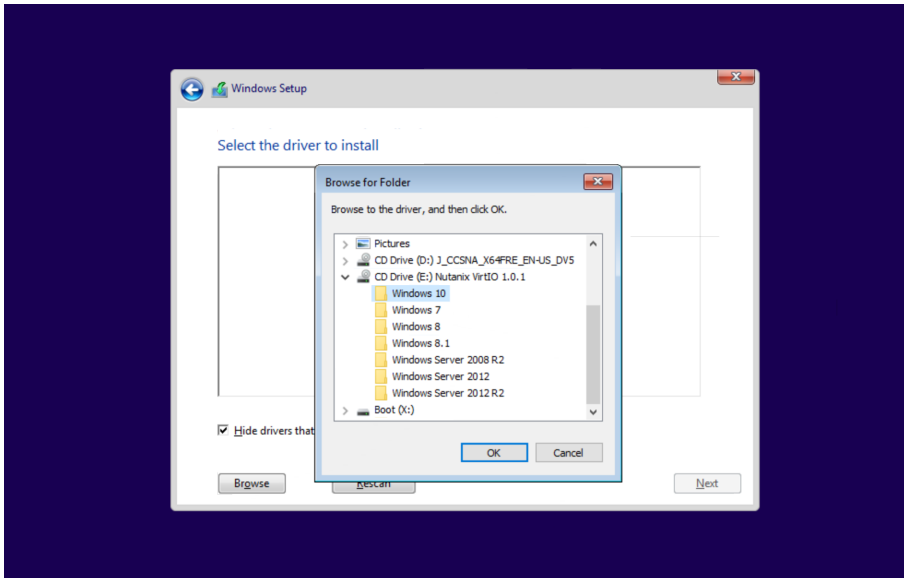

Figure: Select the Nutanix VirtIO drivers for your OS

The *Select the driver to install* window appears.

12. Select all listed drivers and click **Next**.
13. Select the allocated disk space for the VM and click **Next**.
    Windows shows the installation progress which can take several minutes.
14. Fill in your user name and password information and click **Finish**.
    Installation can take several minutes.
    Once you complete the login information, Windows Setup completes installation.

3

Nutanix VirtIO for Windows

Nutanix VirtIO is a collection of drivers for paravirtual devices that enhance stability and performance of VMs on Acropolis Hypervisor (AHV).

Nutanix VirtIO is available in two formats:

- An ISO used when installing Windows in a VM on AHV.
- An installer used to update VirtIO for Windows.

VirtIO Requirements

The following are requirements for Nutanix VirtIO for Windows.

- Operating system:

  Note:

  Nutanix VirtIO only supports 64-bit operating systems.

  For Windows 7 and Windows 2008 R2 operating systems, you have to install the SHA-2 code signing support patch before installing the Nutanix VM Mobility installer. For more information, see *https://technet.microsoft.com/en-us/library/security/3033929*

  - Microsoft Windows Server Version: Windows 2008 R2 or later versions.
  - Microsoft Windows Client Version: Windows 7 or later versions.

Installing Nutanix VirtIO for Windows

This topic describes how to download the Nutanix VirtIO and Nutanix VirtIO Microsoft Installer (MSI). The MSI installs and upgrades the Nutanix VirtIO drivers.

Before you begin: Be sure you have the VirtIO requirements, see *VirtIO Requirements* on page 21.

To download the Nutanix VirtIO, perform the following.

1. Go to the Nutanix Support Portal and click **Downloads** > **Tools & Firmware**.
   The *Tools & Firmware* page appears.
2. Click and download the Nutanix VirtIO package.
   → You can choose the ISO, if you are creating a new Windows VM. The installer is available on the ISO if your VM does not have internet access.
   → You can choose the MSI, if you are updating drivers in a Windows VM.
3. Upload the ISO to the cluster as described in Configuring Images in the Web Console Guide.
4. Run the Nutanix VirtIO MSI by opening the download.
5. Read and accept the Nutanix VirtIO License Agreement. Click **Install**.

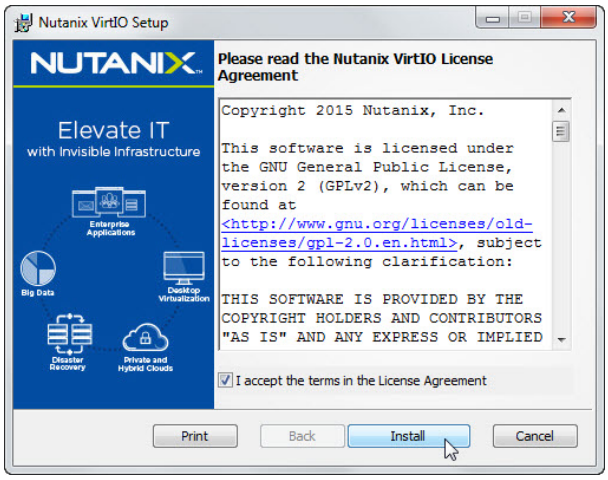

Figure: Nutanix VirtIO Windows Setup Wizard

The Nutanix VirtIO Setup Wizard shows a status bar and completes installation.

Creating a new Windows VM on AHV with Nutanix VirtIO

Before you begin:

- Upload your Windows Installer ISO to your cluster as described in *Configuring Images* in the *Web Console Guide*.
- Upload the Nutanix VirtIO ISO to your cluster as described in *Configuring Images* in the *Web Console Guide*.

The following task describes how to create a new Windows VM in AHV with the Nutanix VirtIO drivers. To install a new Windows VM with Nutanix VirtIO, do the following.

1. Log in to the Prism UI using your Nutanix credentials.
2. At the top left corner, click **Home** > **VM**.
   The *VM page* appears.
3. Click **+ Create VM** in the corner of the page.
   The *Create VM* dialog box appears.

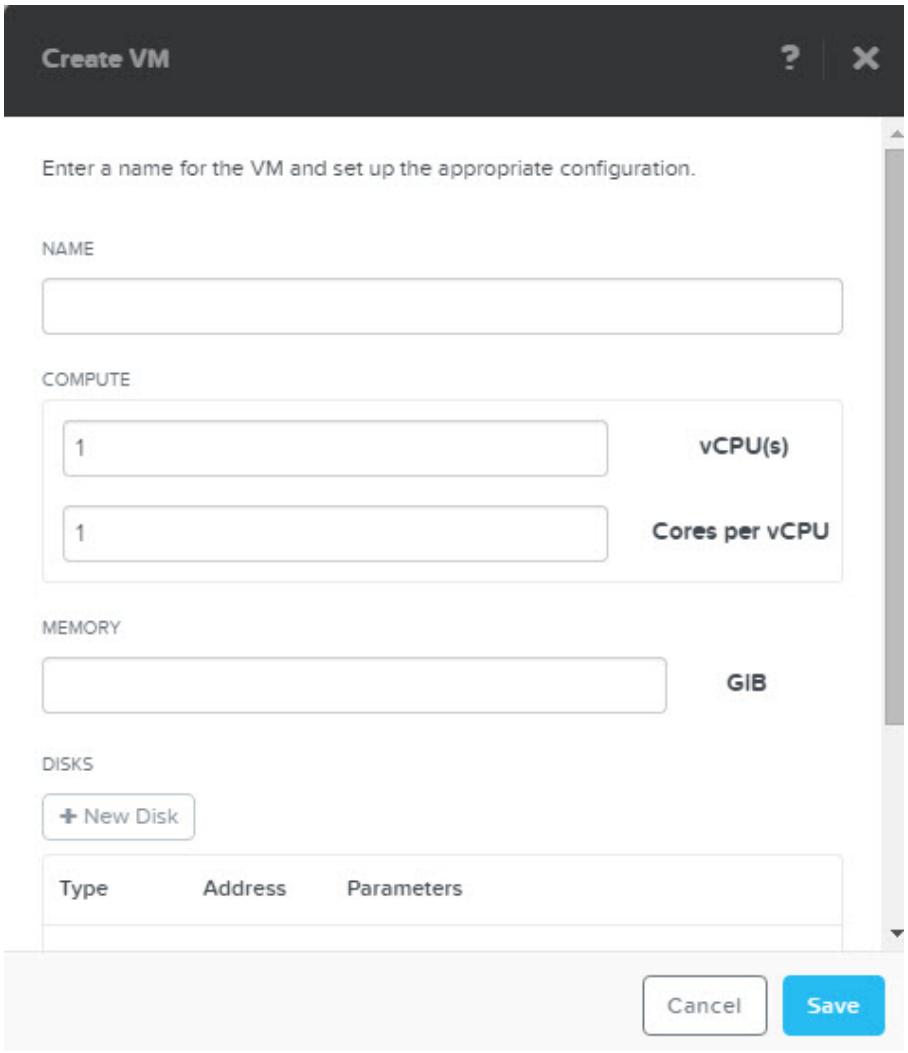

Figure: The Create VM dialog box

4. Complete the indicated fields.
   a. **NAME**: Enter a name for the VM.
   b. **vCPU(s)**: Enter the number of vCPUs
   c. **Cores per vCPU**: Enter the number of cores per vCPU.
   d. **MEMORY**: Enter the amount of memory for the VM (in GIB).

5. Add a Windows CDROM to the VM.
   a. Click the pencil icon next to the CDROM that is already present and fill out the indicated fields. The current CDROM opens in a new window.
   b. **OPERATION**: CLONE FROM IMAGE SERVICE
   c. **BUS TYPE**: IDE
   d. **IMAGE**: Select the Windows OS Install ISO.
   e. Click **Update**.

6. Add the Nutanix VirtIO ISO by clicking on the **+ New Disk** button and complete the indicated fields.
   a. **TYPE**: CDROM
   b. **OPERATION**:CLONE FROM IMAGE SERVICE
   c. **BUS TYPE**: IDE
   d. **IMAGE**: Select the Nutanix VirtIO ISO.
   e. Click **Add**.
7. Add a new disk for the hard drive.
   a. **TYPE**: DISK
   b. **OPERATION**: ALLOCATE ON CONTAINER
   c. **BUS TYPE**: SCSI
   d. **CONTAINER**: Select the appropriate container.
   e. **SIZE**: Enter the number for the size of the hard drive (in GiB).
   f. Click **Add** to add the disk driver.
8. (Optional) Add a network interface card (NIC) by clicking the **+New NIC** button and completing the indicated fields.
   a. **VLAN ID**: Choose the VLAN ID according to network requirements and enter the IP address if required.
   b. Click **Add**.
9. Once you complete the indicated fields, click **Save**.

What to do next: Install Windows by following *Installing Windows on a VM* on page 19.

Installing Windows on a VM

Before you begin: Create a Windows VM by following *Creating a Windows VM on AHV after Migration* on page 17.

To install a Windows VM, do the following.

Note: Nutanix VirtIO cannot be used to install Windows 7 or Windows Server 2008 R2.

1. Log in to the Prism UI using your Nutanix credentials.
2. At the top left corner, click **Home** > **VM**.
   The *VM page* appears.
3. Select the Windows VM.
4. In the center of the VM page, click the **Power On** button.

5. Click the **Launch Console** button.
   The Windows console opens in a new window.
6. Select the desired language, time and currency format, and keyboard information.
7. Click **Next** > **Install Now**.
   The Windows Setup windows displays the operating systems to install.
8. Select the Windows OS you want to install.
9. Click **Next** and accept the license terms.
10. Click **Next** > **Custom: Install Windows only (advanced)** > **Load Driver** > **OK** > **Browse**.
    The browse folder opens.
11. Choose the Nutanix VirtIO CD drive and the Windows version by choosing the Nutanix VirtIO drive and then the Windows OS folder. Click **OK**.

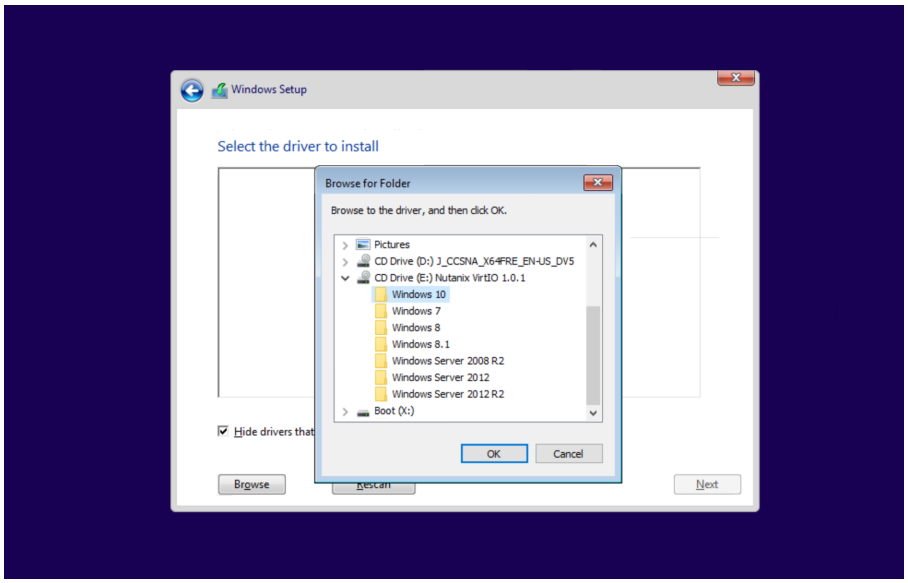

Figure: Select the Nutanix VirtIO drivers for your OS

   The *Select the driver to install* window appears.

12. Select all listed drivers and click **Next**.
13. Select the allocated disk space for the VM and click **Next**.
    Windows shows the installation progress which can take several minutes.
14. Fill in your user name and password information and click **Finish**.
    Installation can take several minutes.
    Once you complete the login information, Windows Setup completes installation.

Migrating VMs from a Non-Nutanix Source to AHV

After you install Nutanix VM Mobility installer on the VMs, you can migrate the VMs from a non-Nutanix source to Acropolis hypervisor (AHV) by performing the following procedure.

1. Install Nutanix VM Mobility. See the Web Console Guide.
2. Migrate the VM disks to Acropolis Distributed Storage Fabric (ADSF). See *Migrating VMs disks to Acropolis Distributed Storage Fabric* on page 26.
3. Convert the VM disks to AHV. See *Converting the Migrated Disks to AHV Format* on page 26.
4. Create a new Windows VM on AHV so that the AHV VM wraps around the Windows disk. See *Creating a Windows VM on AHV after Migration* on page 17.

Migrating VMs disks to Acropolis Distributed Storage Fabric

In order to migrate VMs, you need to configure the host to be able to mount the Acropolis container as a temporary NFS Datastore or SMB share.

Before you begin: Install Nutanix VM Mobility.

1. Add the source hypervisor host IP address to the target AHV cluster Filesystem Whitelist. For information, see Configuring a Whitelist in the Web Console Guide.
2. Use Storage vMotion for the VM disks to Nutanix AHV container datastore.

   For example, you can enter the file path `nfs://127.0.0.1/container_name/vm_name/vm_name.vmdk`.

   Replace *container_name* with the name of the container where the image is placed and replace *vm_name* with the name of the VM where the image is placed.
3. When Storage vMotion is complete, power off the source VM.

Converting the Migrated Disks to AHV Format

Use the Acropolis Image Service to import the virtual disk file copied to the Acropolis container as an Image.

To create an image from the ESXi virtual disk file on the target AHV Controller VM, do the following:

1. In the Prism UI, click **Settings** > **Image Configuration**.
2. In the *Image Configuration Dialog Box*, click **Upload the Image**.
   The *Create Image* dialog box appears.
3. In the *Create Image* dialog box, complete the indicated fields.

   a. For **IMAGE SOURCE**, point to the .vdmk file migrated by Storage vMotion.

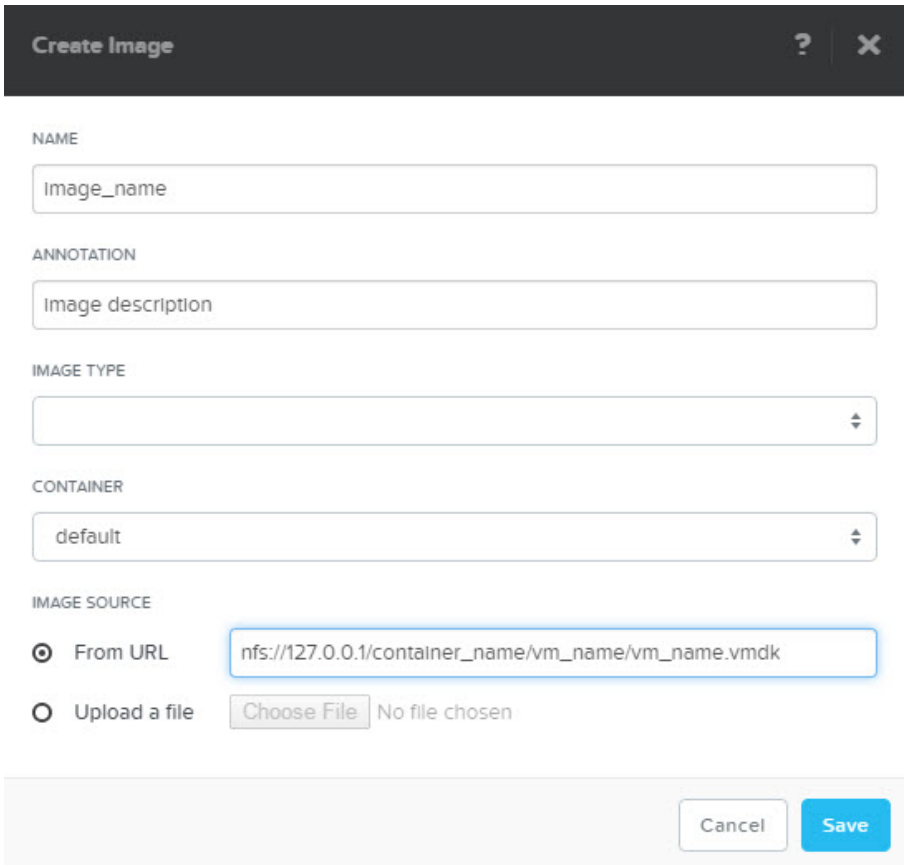

Figure: Completed fields in the Create Image dialog box

For example, in the *Image Configuration* UI, you can enter `nfs://127.0.0.1/container_name/vm_name/vm_name.vmdk`.

Replace *container_name* with the name of the container where the image is placed and replace *vm_name* with the name of the VM where the image is placed.

Creating a Windows VM on AHV after Migration

The following describes how to create a Windows VM after you have migrated the VM to AHV from a non-Nutanix source. To install a Windows VM with Nutanix VirtIO, complete the following.

1. Log in to the Prism UI using your Nutanix credentials.
2. At the top left corner, click **Home** > **VM**.
   The *VM page* appears.
3. Click **+ Create VM** in the corner of the page.
   The *Create VM* dialog box appears.

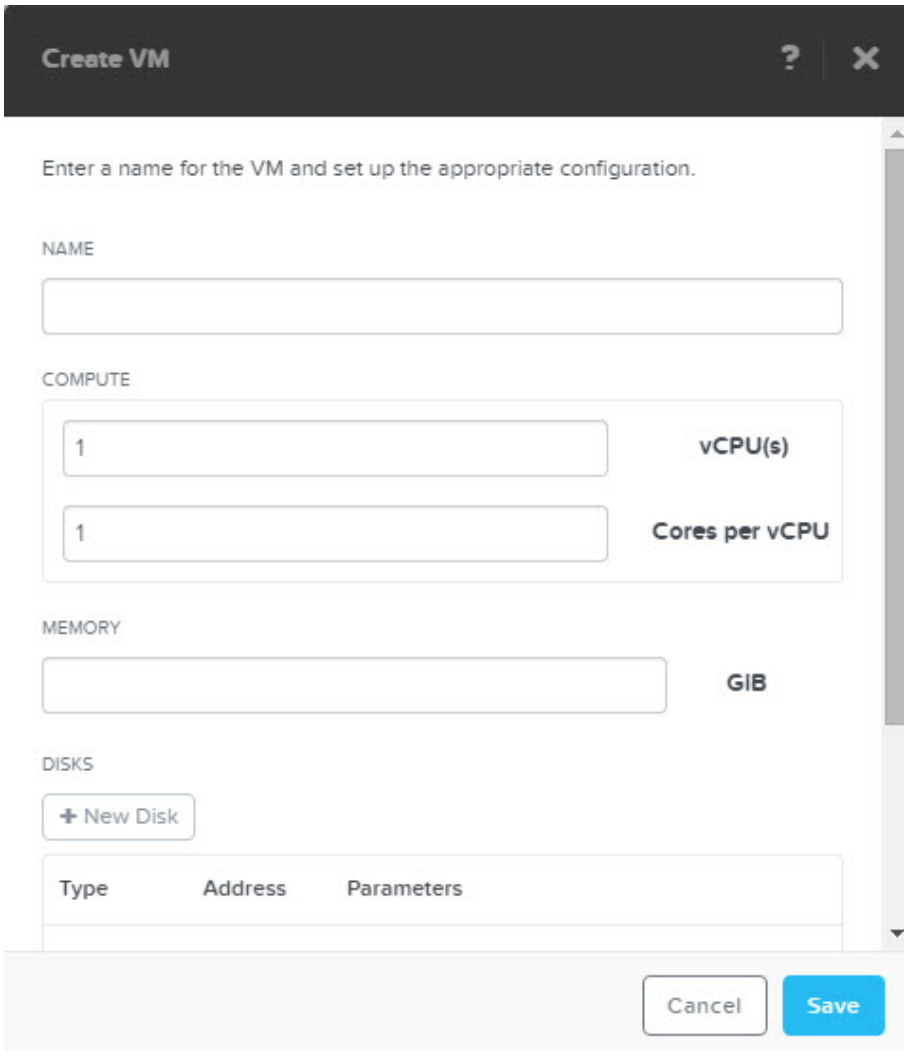

Figure: The Create VM dialog box

4. Create the VM by completing the indicated fields. Match the configuration of the previous VM.
   a. **NAME**: Enter a name for the VM.
   b. **vCPU(s)**: Enter the number of vCPUs
   c. **Cores per vCPU**: Enter the number of cores per vCPU.
   d. **MEMORY**: Enter the amount of memory for the VM (in GIB).

5. Create a disk from the disk image by clicking the **+ New Disk** button and completing the indicated fields.
   a. **TYPE**: DISK
   b. **OPERATION**: CLONE FROM IMAGE
   c. **BUS TYPE**: SCSI
   d. **CLONG FROM IMAGE SERVICE**: Click the drop-down menu and choose the image you created previously.

e. Click **Add** to add the disk driver.

6. (Optional) Add a network interface card (NIC) by clicking the **+New NIC** button and fill out the indicated fields.

   a. **VLAN ID**: Choose the VLAN ID according to network requirements and enter the IP address if required.

   b. Click **Add**.

7. Once you have filled in the indicated fields, click **Save**.

Troubleshooting VM Migration

The following information identifies Nutanix VM Mobility troubleshooting.

Creating disks online after exporting to AHV

In Windows, the SAN policy determines whether a newly discovered disk is brought online or remains offline, and whether it becomes read/write, or remains read-only. When a disk is offline, the disk layout can be read, but no volume devices surface through plug and play. Hence, no file system can be mounted on the disk. When a disk is online, one or more volume devices are installed for the disk. When you export a VM with multiple SCSI disks to AHV, the non-boot disks do not get attached (but are visible) by Windows. You need to create these disks online after exporting to AHV. You can bring the disks online by using one of the following three methods.

1. Go to **Start** > **Control Panel** > **System and Security** > **Administrative Tools** > **Computer Management** > **Storage** > **Disk Management** and right-click the offline disk and select **Online**.
2. Run the following PowerShell command.

   ```
   > Get-Disk | Where-Object IsOffline –Eq $True | Set-Disk –IsOffline $False
   ```

3. Use the `diskpart` command-line utility.

   a. Open the Windows command prompt and run `diskpart.exe` command.

   b. List the disks to confirm their status, by running this command:

   ```
   DISKPART> list disk
   ```

   An output similar to the following should be displayed.

   ```
   Disk ###  Status      Size     Free     Dyn  Gpt
            --------  ----------  -------  -------  ---  ---
            Disk 0    Online        34 GB    12 MB
            Disk 1    Offline     1024 MB      0 B
   ```

   c. Bring a disk online.

   ```
   DISKPART> select disk disk number
   DISKPART> ATTRIBUTES DISK CLEAR READONLY
   DISKPART> Online
   ```

   The output displays: `DiskPart successfully onlined the selected disk.`

4

Acropolis Command-Line Interface

Acropolis provides a command-line interface for managing hosts, networks, snapshots, and VMs.

Accessing the Acropolis CLI

To access the Acropolis CLI, log on to a Controller VM in the cluster with SSH and type `acli` at the shell prompt.

To exit the Acropolis CLI and return to the shell, type `exit` at the `<acropolis>` prompt.

CLI Reference Conventions

This command-line interface reference uses the following conventions.

- Parameters in italic are unique to your environment.

  value

- Parameters in square brackets are optional.

  [*value*]

- Parameters in curly brackets must be one of a limited set of values.

  { *value1* | *value2* }

  One example is boolean parameters: `{ true | false }`

- The `keyword` is a literal string required by the command, and the *value* is the unique value for your environment.

  keyword=*value*

core

Operations

- Exits the CLI : `core.exit`
- Provides help text for the named object : `core.help`

Exits the CLI

```
<acropolis> core.exit
```

Required arguments

None

Provides help text for the named object

```
<acropolis> core.help [ name  ]
```

Required arguments

None

Optional arguments

`name`

Command or namespace to describe

Type: command or namespace name

ha

Operations

- Get current HA configuration : `ha.get`
- Enable, disable or modify VM availability configuration : `ha.update`

Get current HA configuration

```
<acropolis> ha.get
```

Required arguments

None

Enable, disable or modify VM availability configuration

```
<acropolis> ha.update [  enable_failover="{ true | false }" ][  evacuation_mode="{ live
| cold | power_off }" ][  num_host_failures_to_tolerate="number" ][
reservation_type="reservation_type" ][  wait="{ true | false }" ]
```

Required arguments

None

Optional arguments

`enable_failover`

Enable VM restart on host failure.

Type: boolean

Default: true

`evacuation_mode`

Evacuation mode

Type: VM evacuation option

Default: 3

`num_host_failures_to_tolerate`

Number of host failures to tolerate.

Type: int

reservation_type
Reservation type

Type: HA reservation type

wait
If True, wait for the host evacuation attempt to finish

Type: boolean

Default: true

host

Operations

- Puts a host into maintenance mode : `host.enter_maintenance_mode`
- Takes a host out of maintenance mode : `host.exit_maintenance_mode`
- Retrieves scheduler information about a Host : `host.get`
- Gets the host's current maintenance mode state : `host.get_maintenance_mode_state`
- Lists hosts in the cluster : `host.list`
- Lists VMs currently running on the host : `host.list_vms`

Puts a host into maintenance mode

This command initiates a transition into maintenance mode. The host will be marked as unschedulable, so that no new VMs are instantiated on it. Subsequently, an attempt is made to evacuate VMs from the host. If the evacuation attempt fails (e.g., because there are insufficient resources available elsewhere in the cluster), the host will remain in the "entering maintenance mode" state, where it is marked unschedulable, waiting for user remediation. The user may safely run this command again, and may do so with different options (e.g., by specifying mode=power_off to power off the remaining VMs on the host). A request to enter maintenance mode may be aborted at any time using the host.exit_maintenance_mode command. The user should use the host.get_maintenance_mode_state command to determine the host's current maintenance mode state.

```
<acropolis> host.enter_maintenance_mode host [  mode="{ live | cold | power_off }" ][
use_ha_reservations="{ true | false }" ][  wait="{ true | false }" ]
```

Required arguments

host
Host identifier

Type: host

Optional arguments

mode
Evacuation mode ('live', 'cold', 'power_off')

Type: string

Default: live

use_ha_reservations
Use HA reservations for migrating VMs.

Type: boolean

Default: true

wait
If True, wait for the host evacuation attempt to finish

Type: boolean

Default: true

Takes a host out of maintenance mode

This command may be used to abort a prior attempt to enter maintenance mode, even if the attempt is ongoing. If the host is no longer in maintenance mode, this command has no effect. The host may not be removed from maintenance mode synchronously. Use the host.get_maintenance_mode_state command to check the host's current maintenance mode state.

```
<acropolis> host.exit_maintenance_mode host
```

Required arguments

host
Host identifier

Type: host

Retrieves scheduler information about a Host

```
<acropolis> host.get host_list
```

Required arguments

host_list
Host identifier

Type: list of hosts

Gets the host's current maintenance mode state

```
<acropolis> host.get_maintenance_mode_state host
```

Required arguments

host
Host UUID

Type: host

Lists hosts in the cluster

```
<acropolis> host.list
```

Required arguments

None

Lists VMs currently running on the host

```
<acropolis> host.list_vms host
```

Required arguments

host
Host UUID

Type: host

`image`

Operations

- Create an image : `image.create`
- Delete an image(s) : `image.delete`
- Retrieves information about an image : `image.get`
- List all Images : `image.list`
- Update an image : `image.update`

Create an image

We support two different modes of creation. A URL to a disk image can be provided with the source_url keyword argument or an existing vmdisk can be provided with the clone_from_vmdisk keyword argument. If the image is created from a source_url then a container must also be provided. Otherwise the container keyword argument is ignored and the image will reside in the same container as the vmdisk. In addition to a creation mode, an image type must also be provided. Image types can either be an ISO (kIsoImage) or a disk image (kDiskImage). Optionally, a checksum may also be specified if we are creating an image from a source_url in order to verify the correctness of the image.

```
<acropolis> image.create name [  annotation="annotation" ][  clone_from_vmdisk="uuid"
][  compute_checksum="{ true | false }" ][  container="container_name" ][
image_type="image_type" ][  sha1_checksum="sha1" ][  sha256_checksum="sha256" ][
source_url="source_url" ][  wait="{ true | false }" ]
```

Required arguments

name
Image name

Type: string

Optional arguments

`annotation`
Image description

Type: string

`clone_from_vmdisk`
UUID of the source vmdisk

Type: VM disk

`compute_checksum`
If True, we will compute the checksum of the image

Type: boolean

Default: false

`container`
Destination container

Type: container

`image_type`
Image type

Type: image type

sha1_checksum
SHA-1 checksum

Type: hex checksum

sha256_checksum
SHA-256 checksum

Type: hex checksum

source_url
URL location of the source image

Type: image URL

wait
If True, we will wait for the image creation to complete

Type: boolean

Default: true

Examples

1. Create an image named 'foo' from an image located at http://test.com/disk_image.

```
<acropolis> image.create foo source_url=http://test.com/image_iso container=default
 image_type=kIsoImage
```

2. Create an image named 'bar' from a vmdisk 0b4fc60b-cc56-41c6-911e-67cc8406d096.

```
<acropolis> image.create bar clone_from_vmdisk=0b4fc60b-cc56-41c6-911e-67cc8406d096
 image_type=kDiskImage
```

Delete an image(s)

```
<acropolis> image.delete image_list
```

Required arguments

image_list
Image identifiers

Type: list of images

Retrieves information about an image

```
<acropolis> image.get image_list
```

Required arguments

image_list
Image identifiers

Type: list of images

List all Images

```
<acropolis> image.list
```

Required arguments

None

Update an image

```
<acropolis> image.update image [  annotation="annotation" ][  clone_from_vmdisk="uuid"
][  compute_checksum="{ true | false }" ][  container="container_name"
][  image_type="image_type" ][  name="name" ][  sha1_checksum="sha1" ][
sha256_checksum="sha256" ][  source_url="source_url" ][  wait="{ true | false }" ]
```

Required arguments

image
Image identifier

Type: image

Optional arguments

`annotation`
Image description

Type: string

`clone_from_vmdisk`
UUID of the source vmdisk

Type: VM disk

`compute_checksum`
If True, we will compute the checksum of the image

Type: boolean

Default: false

`container`
Destination container

Type: container

`image_type`
Image type

Type: image type

`name`
Image name

Type: string

`sha1_checksum`
SHA-1 checksum

Type: hex checksum

`sha256_checksum`
SHA-256 checksum

Type: hex checksum

`source_url`
URL location of the source image

Type: image URL

`wait`
If True, we will wait for the image creation to complete

Type: boolean

Default: true

Examples

1. Update the name of an image named 'foo'.

```
<acropolis> image.update foo name=bar
```

net

Operations

- Add a DHCP pool to a managed network : `net.add_dhcp_pool`
- Blacklists IP addresses for a managed network : `net.add_to_ip_blacklist`
- Clear the DHCP DNS configuration for a managed network : `net.clear_dhcp_dns`
- Clear the DHCP TFTP configuration for a managed network : `net.clear_dhcp_tftp`
- Creates a new virtual network for VMs : `net.create`
- Deletes a network : `net.delete`
- Delete a DHCP pool from a managed network : `net.delete_dhcp_pool`
- Removes IP addresses from a managed network's blacklist : `net.delete_from_ip_blacklist`
- Retrieves information about a network : `net.get`
- Lists all networks : `net.list`
- List blacklisted IPs for a managed network : `net.list_ip_blacklist`
- Lists VMs configured on the network : `net.list_vms`
- Updates network metadata : `net.update`
- Configure the DHCP DNS configuration for a managed network : `net.update_dhcp_dns`
- Configure the DHCP TFTP configuration for a managed network : `net.update_dhcp_tftp`

Add a DHCP pool to a managed network

A managed network may have zero or more non-overlapping DHCP pools. Each pool must be entirely contained within the network's managed subnet. In the absence of a DHCP pool, the user must specify an IPv4 address when creating a virtual network adapter (see vm.nic_create). If the managed network has a DHCP pool, the user need not provide an address; the NIC will automatically be assigned an IPv4 address from one of the pools at creation time, provided at least one address is available. Addresses in the DHCP pool are not reserved. That is, a user may manually specify an address belonging to the pool when creating a virtual adapter.

```
<acropolis> net.add_dhcp_pool network [  end="ip_addr" ][  start="ip_addr" ]
```

Required arguments

network
Network identifier

Type: network

Optional arguments

end
Last IPv4 address

Type: IPv4 address

start
First IPv4 address

Type: IPv4 address

Examples

1. Auto-assign addresses from the inclusive range 192.168.1.16 - 192.168.1.32.

```
<acropolis> net.add_dhcp_pool vlan.16 start=192.168.1.16 end=192.168.1.32
```

Blacklists IP addresses for a managed network

A blacklisted IP address can not be assigned to a VM network adapter. This property may be useful for avoiding conflicts between VMs and other hosts on the physical network.

```
<acropolis> net.add_to_ip_blacklist network [  ip_list="ip_addr_list" ]
```

Required arguments

network
Network identifier

Type: network

Optional arguments

ip_list
Comma-delimited list of IP addresses

Type: list of IPv4 addresses

Clear the DHCP DNS configuration for a managed network

```
<acropolis> net.clear_dhcp_dns network
```

Required arguments

network
Network identifier

Type: network

Examples

1. Clear DNS servers and search domains.

```
<acropolis> net.clear_dhcp_dns vlan.123
```

Clear the DHCP TFTP configuration for a managed network

```
<acropolis> net.clear_dhcp_tftp network
```

Required arguments

network
Network identifier

Type: network

Examples

1. Clear TFTP server name and boot filename.

```
<acropolis> net.clear_dhcp_tftp vlan.123
```

Creates a new virtual network for VMs

Each VM network interface is bound to a virtual network (see vm.nic_create). While a virtual network is in use by a VM, it cannot be modified or deleted. Currently, the only supported L2 type is VLAN. Each

virtual network is bound to a single VLAN, and trunking VLANs to a virtual network is not supported. A virtual network on VLAN 66 would be named "vlan.66". Each virtual network maps to a hypervisor-specific default vswitch. On KVM, this is "br0". To use a different vswitch (e.g., with different uplinks), you can append the vswitch name to the network identifier. For example, "vlan.66.br1". The user is responsible for ensuring that the specified vswitch exists on all hosts, and that the physical switch ports for the vswitch uplinks are properly configured to receive VLAN-tagged traffic. On hypervisors where it is supported, a virtual network may have an IPv4 configuration. Such a network is a "managed" network. A network without an IPv4 configuration is an "unmanaged" network. A network must be configured as "managed" or "unmanaged" at creation time. It is not possible to convert one to the other. A particular L2 (i.e., a particular VLAN) may have at most one managed network defined at a time. To create a managed network, the user specifies the "ip_config" keyword. This consists of an IPv4 default gateway address and subnet in CIDR notation. The user may optionally specify a DHCP server address, to avoid conflict with other services on the network. By default, the last available host address in the subnet is used. Optionally mtu can be specified as part of network create if mtu other than default (1500) is required. Every virtual NIC on a managed network must be assigned an IPv4 address at NIC creation time. All DHCP traffic on the network will be rerouted to an internal DHCP server, who hands out configured IPv4 addresses. DHCP traffic on the physical network will not reach the virtual network, and vice versa. For more about managed networks, see the following commands: network.add_dhcp_pool network.add_to_ip_blacklist network.clear_dhcp_dns network.clear_dhcp_tftp network.delete_dhcp_pool network.delete_from_ip_blacklist network.list_ip_blacklist network.update_dhcp_dns network.update_dhcp_tftp

```
<acropolis> net.create name [  annotation="annotation" ][  dhcp_address="dhcp_addr"
][  ip_config="default_gateway/prefix" ][  mtu="mtu" ][  vlan="vlan" ][
vswitch_name="vswitch_name" ]
```

Required arguments

name
Network name

Type: string

Optional arguments

`annotation`
Annotation string

Type: string

`dhcp_address`
DHCP server address (for managed networks)

Type: IPv4 address

`ip_config`
IP configuration in CIDR notation ("default_gateway/prefix")

Type: string

`mtu`
MTU setting

Type: int

`vlan`
VLAN ID

Type: int

`vswitch_name`
Vswitch name

Type: string

Examples

1. Create an unmanaged network on VLAN 66.

```
<acropolis> net.create mynet vlan=66
```

2. Create an unmanaged network on VLAN 66 with MTU 9000.

```
<acropolis> net.create mynet vlan=66 mtu=9000
```

3. Create a managed network on VLAN 99, bound to vswitch br1. The managed IPv4 range is 10.1.1.0 - 10.1.1.255, the default gateway is 10.1.1.1, and the DHCP server is 10.1.1.254

```
<acropolis> net.create mynet vlan=99 vswitch_name=br1 ip_config=10.1.1.1/24
```

4. Create an untagged managed network. The managed IPv4 range is 192.168.0.0 - 192.168.3.255, and the default gateway is 192.168.5.254. In this example, the DHCP server will be automatically configured as 192.168.5.253 to avoid collision with the default gateway.

```
<acropolis> net.create mynet vlan=0 ip_config=192.168.5.254/22
```

Deletes a network

Note that a network may not be deleted while VMs are still attached to it. To determine which VMs are on a network, use network.list_vms.

```
<acropolis> net.delete network
```

Required arguments

network
Network identifier

Type: network

Delete a DHCP pool from a managed network

See network.add_dhcp_pool for more information.

```
<acropolis> net.delete_dhcp_pool network [  start="ip_addr" ]
```

Required arguments

network
Network identifier

Type: network

Optional arguments

start
First IPv4 address

Type: IPv4 address

Removes IP addresses from a managed network's blacklist

```
<acropolis> net.delete_from_ip_blacklist network [  ip_list="ip_addr_list" ]
```

Required arguments

network
Network identifier

Type: network

Optional arguments

ip_list
Comma-delimited list of IP addresses

Type: list of IPv4 addresses

Retrieves information about a network

```
<acropolis> net.get network_list
```

Required arguments

network_list
Network identifier

Type: list of networks

Lists all networks

```
<acropolis> net.list
```

Required arguments

None

List blacklisted IPs for a managed network

```
<acropolis> net.list_ip_blacklist network
```

Required arguments

network
Network identifier

Type: network

Lists VMs configured on the network

```
<acropolis> net.list_vms network
```

Required arguments

network
Network identifier

Type: network

Updates network metadata

```
<acropolis> net.update network [  annotation="annotation" ][  name="name" ]
```

Required arguments

network
Network identifier

Type: network

Optional arguments

annotation
Annotation string

Type: string

name
Network name

Type: string

Configure the DHCP DNS configuration for a managed network

This command is used to configure the DNS information that the DHCP server includes in its responses to clients on the virtual network. In particular, it is used to configure a list of DNS server IP addresses, and domain search paths. The DHCP server's DNS configuration may be modified while VMs are connected to the network. However, the DHCP server hands out infinite leases, so clients will need to manually renew to pick up the new settings.

```
<acropolis> net.update_dhcp_dns network [  domains="search_domains" ][
servers="dns_servers" ]
```

Required arguments

network
Network identifier

Type: network

Optional arguments

domains
Comma-delimited list of search domains

Type: list of DNS domains

servers
Comma-delimited list of DNS server IP addresses

Type: list of IPv4 addresses

Examples

1. Configure DNS servers and search domains.

```
<acropolis> net.update_dhcp_dns vlan.123 servers=10.1.1.1,10.1.1.2
 domains=eng.nutanix.com,corp.nutanix.com
```

Configure the DHCP TFTP configuration for a managed network

This command is used to configure the TFTP information that the DHCP server includes in its responses to clients on the virtual network. In particular, it is used to configure the TFTP server name (option 66) and boot file name (option 67). The DHCP server's TFTP configuration may be modified while VMs are connected to the network. However, the TFTP server hands out infinite leases, so clients will need to manually renew to pick up the new settings.

```
<acropolis> net.update_dhcp_tftp network [  bootfile_name="bootfile_name" ][
server_name="server_name" ]
```

Required arguments

network
Network identifier

Type: network

Optional arguments

bootfile_name
Boot file name

Type: string

server_name
TFTP server name

Type: string

Examples

1. Configure TFTP server and bootfile.

```
<acropolis> net.update_dhcp_tftp vlan.123 server_name=10.1.1.1 bootfile_name=ARDBP32.BIN
```

snapshot

Operations

- Deletes one or more snapshots : `snapshot.delete`
- Retrieves information about a snapshot : `snapshot.get`
- Lists all snapshots : `snapshot.list`

Deletes one or more snapshots

```
<acropolis> snapshot.delete snapshot_list
```

Required arguments

snapshot_list
Comma-delimited list of snapshot identifiers

Type: list of snapshots

Retrieves information about a snapshot

```
<acropolis> snapshot.get snapshot_list
```

Required arguments

snapshot_list
Snapshot identifier

Type: list of snapshots

Lists all snapshots

```
<acropolis> snapshot.list
```

Required arguments

None

task

Operations

- Retrieves information about list of tasks based on identifiers specified : `task.get`
- Lists tasks based on specified filters : `task.list`

- Poll for task completion : `task.poll`

Retrieves information about list of tasks based on identifiers specified

```
<acropolis> task.get task_list [  resolve_entity_names="{ true | false }" ]
```

Required arguments

task_list
Task identifier

Type: list of tasks

Optional arguments

`resolve_entity_names`
Make a best effort to resolve entity names

Type: boolean

Default: true

Examples

1. Get details of list of tasks based on identifiers

```
<acropolis> task.get 783d9fed-131e-406d-ae4b-e8ca2726cc02,90a13330-
b2cb-4995-82cf-06e2efb51d3d
```

Lists tasks based on specified filters

If no filters are specified, then only tasks that are currently in progress are returned. Any filters specified will be used with an AND condition. However, within each filter that consitutes a list, an OR condition will be used to match the values in the list with tasks. For eg. if entity_type_list filter with kVM and kNode is specified along with an operation_type_filter with kCreateVM and KDeleteVM, tasks are first filtered based on whether they are associated with a kVM OR a kNode. This filtered list of tasks is then further filtered based whether they are associated with a kCreateVM OR kDeleteVM operation. The age_hours filter is only valid when the argument to include completed tasks is set to True.

```
<acropolis> task.list [  age_hours="hours" ][  entity_list="entity_list" ][
entity_type_list="entity_type_list" ][  include_completed="{ true | false }" ][
limit="num" ][  operation_type_list="operation_type_list" ]
```

Required arguments

None

Optional arguments

`age_hours`
Cutoff time in hours

Type: int

`entity_list`
Entity identifiers

Type: list of entities

`entity_type_list`
Entity types

Type: list of entities

`include_completed`
Include completed tasks

Type: boolean

`limit`
Number of tasks to return

Type: int

`operation_type_list`
Operation types

Type: list of operation types

Examples

1. List tasks without any filters.

```
<acropolis> task.list
```

2. List tasks including those that have completed.

```
<acropolis> task.list include_completed=True
```

3. List tasks that are associated with specific entity types, eg. vm, node

```
<acropolis> task.list entity_type_list=kVM,kNode
```

4. List tasks that are associated with a specific operation type.

```
<acropolis> task.list operation_type_list=kVmCreate
```

5. List tasks with that have completed within the last 5 hours

```
<acropolis> task.list age_hours=5 include_completed=True
```

Poll for task completion

If any of the specified tasks finish, then the poll returns. The response will specify if the request timed out without any tasks completing or if they did, the exact of tasks that completed. Invalid task uuids will also be specified in the response.

```
<acropolis> task.poll task_list [  resolve_entity_names="{ true | false }" ][
timeout="secs" ]
```

Required arguments

task_list
Task identifier

Type: list of tasks

Optional arguments

`resolve_entity_names`
Make a best effort to resolve entity names

Type: boolean

Default: true

`timeout`
Poll timeout in seconds

Type: int

Examples

1. Poll list of tasks for completion.

```
<acropolis> task.poll 783d9fed-131e-406d-ae4b-e8ca2726cc02,90a13330-
b2cb-4995-82cf-06e2efb51d3d
```

Operations

- Allow volume group to be accessed from an external intiator : `vg.attach_external`
- Attach a VG to the specified VM : `vg.attach_to_vm`
- Creates one or more VGs : `vg.create`
- Deletes one or more VGs and its backing disks : `vg.delete`
- Stop allowing volume group to be accessed from an external intiator : `vg.detach_external`
- Detach a VG from the specified VM : `vg.detach_from_vm`
- Add a new disk to a VG : `vg.disk_create`
- Remove a disk from a VG : `vg.disk_delete`
- Retrieves information about a VG : `vg.get`
- Lists all VGs : `vg.list`
- Updates the specified VGs : `vg.update`

Allow volume group to be accessed from an external intiator

```
<acropolis> vg.attach_external vg initiator_name
```

Required arguments

vg
VG identifier

Type: volume group

initiator_name
Name of external initiator

Type: iSCSI IQN

Attach a VG to the specified VM

```
<acropolis> vg.attach_to_vm vg vm [  index="index" ]
```

Required arguments

vg
VG identifier

Type: volume group

vm
VM identifier

Type: VM

Optional arguments

index
Device index on the scsi bus

Type: int

Creates one or more VGs

```
<acropolis> vg.create name_list [  annotation="annotation" ][
iscsi_target_name_list="iscsi_target_name_list" ][  shared="{ true | false }" ]
```

Required arguments

name_list
Comma-delimited list of VG names

Type: list of strings

Optional arguments

`annotation`
Annotation string

Type: string

`iscsi_target_name_list`
Comma-delimited list of iscsi target names for each of the VGs

Type: list of strings

`shared`
Allow VG to be attached to multiple VMs simultaneously?

Type: boolean

Deletes one or more VGs and its backing disks

```
<acropolis> vg.delete vg_list
```

Required arguments

vg_list
Comma-delimited VG identifiers

Type: list of volume groups

Stop allowing volume group to be accessed from an external intiator

```
<acropolis> vg.detach_external vg initiator_name
```

Required arguments

vg
VG identifier

Type: volume group

initiator_name
Name of external initiator

Type: VG external initiator name

Detach a VG from the specified VM

```
<acropolis> vg.detach_from_vm vg vm
```

Required arguments

vg
VG identifier

Type: volume group

vm
VM identifier

Type: VM

Add a new disk to a VG

Exactly one of the following options is required: clone_from_adsf_file, clone_from_vmdisk, create_size. Disk sizes must be specified with a multiplicative suffix. The size will be rounded up to the nearest sector size. The following suffixes are valid: c=1, s=512, k=1000, K=1024, m=1e6, M=2^20, g=1e9, G=2^30, t=1e12, T=2^40. If the disk image is cloned from an existing vmdisk or ADSF file, the user may specify a minimum size for the resulting clone. This can be used to expand a disk image at clone time.

```
<acropolis> vg.disk_create vg [  clone_from_adsf_file="file_path" ][
clone_from_vmdisk="vmdisk" ][  clone_min_size="size" ][  container="container" ][
create_size="size" ][  index="index" ]
```

Required arguments

vg
VG identifier

Type: volume group

Optional arguments

`clone_from_adsf_file`
Path to an ADSF file

Type: ADSF path

`clone_from_vmdisk`
A vmdisk UUID

Type: VM disk

`clone_min_size`
Minimum size of the resulting clone (only applies to cloned disks)

Type: size with cskKmMgGtT suffix

`container`
Container (only applies to newly-created disks)

Type: container

`create_size`
Size of new disk

Type: size with cskKmMgGtT suffix

`index`
Device index on bus

Type: int

Examples

1. Create a blank 5GiB disk on ctr, and add it to my_vg at index 3

```
<acropolis> vg.disk_create my_vg create_size=5G container=ctr index=3
```

2. Clone a disk from the ADSF file /ctr/plan9.iso, add it to first open slot

```
<acropolis> vg.disk_create my_vg clone_from_adsf_file=/ctr/plan9.iso
```

3. Clone a disk from the existing vmdisk, and add it to the first open slot

```
<acropolis> vg.disk_create my_vg clone_from_vmdisk=0b4fc60b-cc56-41c6-911e-67cc8406d096
```

Remove a disk from a VG

```
<acropolis> vg.disk_delete vg index
```

Required arguments

vg
VG identifier

Type: volume group

index
Disk index

Type: VG disk index

Retrieves information about a VG

```
<acropolis> vg.get vg_list [  include_vmdisk_sizes="{ true | false }" ]
```

Required arguments

vg_list
VG identifier

Type: list of volume groups

Optional arguments

`include_vmdisk_sizes`
Fetch disk sizes (in bytes)

Type: boolean

Default: true

Lists all VGs

```
<acropolis> vg.list
```

Required arguments

None

Updates the specified VGs

```
<acropolis> vg.update vg_list [  annotation="annotation" ][
iscsi_target_name_list="iscsi_target_name_list" ][  name="name" ][  shared="{ true |
false }" ]
```

Required arguments

vg_list
Comma-delimited list of VG identifiers

Type: list of volume groups

Optional arguments

`annotation`
Annotation string

Type: string

`iscsi_target_name_list`
Comma-delimited list of iscsi target names for each of the VGs

Type: list of strings

`name`
VG name

Type: string

`shared`
Allow VG to be attached to multiple VMs simultaneously?

Type: boolean

vm

Operations

- Aborts an in-progress migration : `vm.abort_migrate`
- Clones a VM : `vm.clone`
- Creates one or more VMs : `vm.create`
- Deletes one or more VMs : `vm.delete`
- Attaches a new disk drive to a VM : `vm.disk_create`
- Detaches a disk drive from a VM : `vm.disk_delete`
- Gets details about the disks attached to a VM : `vm.disk_get`
- Lists the disks attached to a VM : `vm.disk_list`
- Updates the backing for the specified disk drive : `vm.disk_update`
- Force VM into the powered off state : `vm.force_off`
- Retrieves information about a VM : `vm.get`
- Lists all VMs : `vm.list`
- Live migrates a VM to another host : `vm.migrate`
- Attaches a network adapter to a VM : `vm.nic_create`
- Detaches a NIC from a VM : `vm.nic_delete`
- Gets details about the NICs attached to a VM : `vm.nic_get`
- Lists the NICs attached to a VM : `vm.nic_list`
- Powers off the specified VMs : `vm.off`
- Powers on the specified VMs : `vm.on`
- Pauses the specified VMs : `vm.pause`
- Power cycles the specified VMs : `vm.power_cycle`
- Initiates a reboot by issuing an ACPI event : `vm.reboot`
- Resets the specified VMs : `vm.reset`
- Restores a VM to a snapshotted state : `vm.restore`
- Resumes the specified VMs : `vm.resume`
- Resumes all paused VMs : `vm.resume_all`
- Initiates a shutdown by issuing an ACPI event : `vm.shutdown`
- Creates one or more snapshots in a single consistency group : `vm.snapshot_create`
- Prints the graph representation of the snapshot history for a VM : `vm.snapshot_get_tree`
- Gets a list of all snapshots associated with a VM : `vm.snapshot_list`
- Updates the specified VMs : `vm.update`
- Updates a VM's boot device : `vm.update_boot_device`

Aborts an in-progress migration

```
<acropolis> vm.abort_migrate vm
```

Required arguments

vm
VM identifier

Type: VM

Clones a VM

One of the 'clone_from_*' arguments must be provided. The resulting VMs will be cloned from the specified source. When the source has exactly one NIC on a managed network, the caller may optionally provide an initial IP address. The first clone will get the first IP address, and subsequent clones will be assigned subsequent IP addresses in sequence. If memory size or CPU-related parameters are specified, they override the values allotted to the source VM/snapshot. Memory size must be specified with a multiplicative suffix. The following suffixes are valid: M=2^20, G=2^30.

```
<acropolis> vm.clone name_list [  clone_from_snapshot="snapshot_id" ][
clone_from_vm="vm_id" ][  clone_ip_address="ip_addr" ][  memory="memory" ][
num_cores_per_vcpu="num" ][  num_vcpus="num" ]
```

Required arguments

name_list
Comma-delimited list of VM names

Type: list of strings with expansion wildcards

Optional arguments

`clone_from_snapshot`
Snapshot from which to clone

Type: snapshot

`clone_from_vm`
VM from which to clone

Type: VM

`clone_ip_address`
First IP address to assign to clones

Type: string

`memory`
Memory size

Type: size with MG suffix

`num_cores_per_vcpu`
Number of cores per vCPU

Type: int

`num_vcpus`
Number of vCPUs

Type: int

Creates one or more VMs

Memory size must be specified with a multiplicative suffix. The following suffixes are valid: M=2^20, G=2^30.

```
<acropolis> vm.create name_list [  ha_priority="num" ][  memory="memory" ][
num_cores_per_vcpu="num" ][  num_vcpus="num" ]
```

Required arguments

name_list
Comma-delimited list of VM names

Type: string

Optional arguments

ha_priority
Numeric priority for HA restart. Negative value indicates no restart.

Type: int

memory
Memory size

Type: size with MG suffix

Default: 2G

num_cores_per_vcpu
Number of cores per vCPU

Type: int

num_vcpus
Number of vCPUs

Type: int

Default: 1

Deletes one or more VMs

If the VM is powered on, it will be powered off and then deleted.

```
<acropolis> vm.delete vm_list [  delete_snapshots="{ true | false }" ]
```

Required arguments

vm_list
Comma-delimited VM identifiers

Type: list of VMs

Optional arguments

delete_snapshots
Delete snapshots?

Type: boolean

Default: false

Attaches a new disk drive to a VM

Exactly one of the following options is required: clone_from_adsf_file, clone_from_vmdisk, create_size, empty. A disk drives may either be a regular disk drive, or a CD-ROM drive. Only CD-ROM drives may be empty. Disk sizes must be specified with a multiplicative suffix. The size will be rounded up to the nearest sector size. The following suffixes are valid: c=1, s=512, k=1000, K=1024, m=1e6, M=2^20, g=1e9, G=2^30, t=1e12, T=2^40. By default, regular disk drives are configured on the SCSI bus, and CD-ROM drives are configured on the IDE bus. The user may override this behavior with the "bus" keyword. By default, a disk drive is placed on the first available bus slot. The user may override this behavior with the "index" keyword. Disks on the SCSI bus may optionally be configured for passthrough on platforms that support iSCSI. When in passthrough mode, SCSI commands are passed directly to NDFS via iSCSI. When SCSI passthrough is disalbed, the hypervisor provides a SCSI emulation layer, and treats the

underlying iSCSI target as a block device. By default, SCSI passthrough is enabled for SCSI devices on supported platforms. If the disk image is cloned from an existing vmdisk or ADSF file, the user may specify a minimum size for the resulting clone. This can be used to expand a disk image at clone time. If the VM is running, the disk is hot-added to the VM. Note that certain buses, like IDE, are not hot-pluggable.

```
<acropolis> vm.disk_create vm [  bus="device_bus" ][  cdrom="{ true | false }"
][  clone_from_adsf_file="file_path" ][  clone_from_image="image" ][
clone_from_vmdisk="vmdisk_id" ][  clone_min_size="GiB" ][  container="container" ][
create_size="GiB" ][  empty="{ true | false }" ][  index="index" ][  scsi_passthru="{ true
| false }" ]
```

Required arguments

vm
VM identifier

Type: VM

Optional arguments

`bus`
Device bus

Type: string

`cdrom`
Indicates if the disk is a CDROM drive

Type: boolean

`clone_from_adsf_file`
Path to an ADSF file

Type: ADSF path

`clone_from_image`
An image name/UUID

Type: image

`clone_from_vmdisk`
A vmdisk UUID

Type: VM disk

`clone_min_size`
Minimum size of the resulting clone (only applies to cloned disks)

Type: size with cskKmMgGtT suffix

`container`
Container (only applies to newly-created disks)

Type: container

`create_size`
Size of new disk

Type: size with cskKmMgGtT suffix

`empty`
Whether the disk is empty (only applies to CDROMs)

Type: boolean

`index`
Device index on bus

Type: int

scsi_passthru
Passthrough disk?

Type: boolean

Examples

1. Create a blank 5GiB disk on ctr, and attach it as SCSI:3.

```
<acropolis> vm.disk_create my_vm create_size=5G container=ctr bus=scsi index=3
```

2. Clone a disk from the ADSF file /ctr/plan9.iso, and use it as the backing image for a newly-created CD-ROM drive on the first available IDE slot.

```
<acropolis> vm.disk_create my_vm clone_from_adsf_file=/ctr/plan9.iso cdrom=1
```

3. Clone a disk from the existing vmdisk, and attach it to the first available SCSI slot.

```
<acropolis> vm.disk_create my_vm clone_from_vmdisk=0b4fc60b-cc56-41c6-911e-67cc8406d096
```

4. Create a disk from an Acropolis image and attach it to the first SCSI slot.

```
<acropolis> vm.disk_create my_vm clone_from_image=my_image
```

5. Create a new empty CD-ROM drive, and attach it to the first available IDE slot.

```
<acropolis> vm.disk_create my_vm empty=1 cdrom=1
```

Detaches a disk drive from a VM

If the VM is running, the disk is hot-removed from the VM. Note that certain buses, like IDE, are not hot-pluggable.

```
<acropolis> vm.disk_delete vm disk_addr
```

Required arguments

vm
VM identifier

Type: VM

disk_addr
Disk address ("bus.index")

Type: VM disk

Gets details about the disks attached to a VM

```
<acropolis> vm.disk_get vm [  disk_addr="disk_addr" ][  include_vmdisk_sizes="{ true | false }" ]
```

Required arguments

vm
VM identifier

Type: VM

Optional arguments

disk_addr
Disk address ("bus.index")

Type: VM disk

include_vmdisk_sizes
Fetch vmdisk sizes (in bytes)

Type: boolean

Default: true

Lists the disks attached to a VM

```
<acropolis> vm.disk_list vm
```

Required arguments

vm
VM identifier

Type: VM

Updates the backing for the specified disk drive

Exactly one of the following options is required: clone_from_adsf_file, clone_from_vmdisk, empty, create_size. Disk sizes must be specified with a multiplicative suffix. The size will be rounded up to the nearest sector size. The following suffixes are valid: c=1, s=512, k=1000, K=1024, m=1e6, M=2^20, g=1e9, G=2^30, t=1e12, T=2^40. The existing disk image will be deleted and replaced by the new image (which may be a clone of the existing image).

```
<acropolis> vm.disk_update vm disk_addr [  clone_from_adsf_file="file_path" ][
clone_from_image="image" ][  clone_from_vmdisk="vmdisk_id" ][  clone_min_size="GiB" ][
container="container" ][  create_size="GiB" ][  empty="{ true | false }" ]
```

Required arguments

vm
VM identifier

Type: VM

disk_addr
Disk address ("bus.index")

Type: VM disk

Optional arguments

`clone_from_adsf_file`
Path to an ADSF file

Type: ADSF path

`clone_from_image`
An image name/UUID

Type: image

`clone_from_vmdisk`
A vmdisk UUID

Type: VM disk

`clone_min_size`
Minimum size of the resulting clone (only applies to cloned disks)

Type: size with cskKmMgGtT suffix

`container`
Container (only applies to newly-created disks)

Type: container

create_size
Size of new disk

Type: size with cskKmMgGtT suffix

empty
Whether the disk is empty (only applies to CDROMs)

Type: boolean

Examples

1. Replace the disk at SCSI:0 with blank 5GiB disk on ctr.

```
<acropolis> vm.disk_update my_vm scsi.0 create_size=5G container=ctr
```

2. Replace the disk at IDE:0 with a clone of /ctr/plan9.iso. Note that if IDE:0 is a CD-ROM drive, it remains such.

```
<acropolis> vm.disk_update my_vm ide.0 clone_from_adsf_file=/ctr/plan9.iso
```

3. Replace the disk at SCSI:0 with a clone of the existing vmdisk.

```
<acropolis> vm.disk_update my_vm scsi.0 clone_from_vmdisk=0b4fc60b-
cc56-41c6-911e-67cc8406d096
```

4. Eject the image from the CD-ROM drive at IDE:0.

```
<acropolis> vm.disk_update my_vm ide.0 empty=1
```

Force VM into the powered off state

If a VM's host becomes disconnected from the cluster, and is not expected to return, this command may be used to force the VM back to the powered off state. Use this command with extreme caution; if the VM is actually still running on the host after a force off, a subsequent attempt to power on the VM elsewhere may succeed. The two instances may experience IP conflicts, or corrupt the VM's virtual disks. Therefore, the user should take adequate precautions to ensure that the old instance is really gone.

```
<acropolis> vm.force_off vm
```

Required arguments

vm
VM identifier

Type: VM

Retrieves information about a VM

```
<acropolis> vm.get vm_list [  include_address_assignments="{ true | false }" ][
include_vmdisk_sizes="{ true | false }" ]
```

Required arguments

vm_list
VM identifier

Type: list of VMs

Optional arguments

include_address_assignments
Fetch configured IP addresses

Type: boolean

Default: true

include_vmdisk_sizes
Fetch vmdisk sizes (in bytes)

Type: boolean

Default: true

Lists all VMs

```
<acropolis> vm.list
```

Required arguments

None

Live migrates a VM to another host

If no host is specified, the scheduler will pick the one with the most available CPU and memory that can support the VM. Note that no such host may be available. The user may abort an in-progress migration with the vm.abort_migrate command. If multiple VMs are specified, it is recommended to also provide the bandwidth_mbps parameter. This limit is applied to each of the migrations individually.

```
<acropolis> vm.migrate vm_list [  bandwidth_mbps="mbps" ][  host="host" ][  live="{ true | false }" ]
```

Required arguments

vm_list
Comma-delimited list of VM identifiers

Type: list of VMs

Optional arguments

bandwidth_mbps
Maximum bandwidth in MiB/s

Type: int

Default: 0

host
Destination host

Type: host

live
Live migration or suspended migration?

Type: boolean

Default: true

Attaches a network adapter to a VM

A VM NIC must be associated with a virtual network. It is not possible to change this association. To connect a VM to a different virtual network, it is necessary to create a new NIC. If the virtual network is managed (see network.create), the NIC must be assigned an IPv4 address at creation time. If the network has no DHCP pool, the user must specify the IPv4 address manually. If the VM is running, the NIC is hot-added to the VM.

```
<acropolis> vm.nic_create vm [  ip="ip_addr" ][  mac="mac_addr" ][  model="model" ][ network="network" ]
```

Required arguments

vm
VM identifier

Type: VM

Optional arguments

ip
IPv4 address

Type: IPv4 address

mac
MAC address

Type: MAC address

model
Virtual hardware model

Type: string

network
Network identifier

Type: network

Detaches a NIC from a VM

If the VM is running, the NIC is hot-removed from the VM. If the NIC to be removed is specified as the boot device in the boot configuration, the boot device configuration will be cleared as a side effect of removing the NIC.

```
<acropolis> vm.nic_delete vm mac_addr
```

Required arguments

vm
VM identifier

Type: VM

mac_addr
NIC MAC address

Type: NIC address

Gets details about the NICs attached to a VM

```
<acropolis> vm.nic_get vm [  include_address_assignments="{ true | false }" ][
mac_addr="mac_addr" ]
```

Required arguments

vm
VM identifier

Type: VM

Optional arguments

include_address_assignments
Fetch configured IP addresses

Type: boolean

Default: true

mac_addr
NIC MAC address

Type: NIC address

Lists the NICs attached to a VM

```
<acropolis> vm.nic_list vm
```

Required arguments

vm
VM identifier

Type: VM

Powers off the specified VMs

```
<acropolis> vm.off vm_list
```

Required arguments

vm_list
Comma-delimited list of VM identifiers

Type: list of VMs

Powers on the specified VMs

If no host is specified, the scheduler will pick the one with the most available CPU and memory that can support the VM. Note that no such host may be available.

```
<acropolis> vm.on vm_list [  host="host" ]
```

Required arguments

vm_list
Comma-delimited list of VM identifiers

Type: list of VMs

Optional arguments

host
Host on which to power on the VM

Type: host

Pauses the specified VMs

```
<acropolis> vm.pause vm_list
```

Required arguments

vm_list
Comma-delimited list of VM identifiers

Type: list of VMs

Power cycles the specified VMs

```
<acropolis> vm.power_cycle vm_list [  change_host="{ true | false }" ][  host="host" ]
```

Required arguments

vm_list
Comma-delimited list of VM identifiers

Type: list of VMs

Optional arguments

`change_host`
Whether to power on on a different host

Type: boolean

Default: false

`host`
Host on which to power on the VM

Type: host

Initiates a reboot by issuing an ACPI event

```
<acropolis> vm.reboot vm_list
```

Required arguments

vm_list
Comma-delimited list of VM identifiers

Type: list of VMs

Resets the specified VMs

```
<acropolis> vm.reset vm_list
```

Required arguments

vm_list
Comma-delimited list of VM identifiers

Type: list of VMs

Restores a VM to a snapshotted state

If the VM is currently running, it will be powered off. Since VM snapshots do not include the VM memory image, the VM will remain powered off after the restore is complete. A VM snapshot may no longer be compatible with the current virtual network configuration. In this case, the user may choose not to restore the VM's network adpaters using the "restore_network_config" keyword argument.

```
<acropolis> vm.restore vm snapshot_id [  restore_network_config="{ true | false }" ]
```

Required arguments

vm
VM identifier

Type: VM

snapshot
Snapshot identifier

Type: snapshot

Optional arguments

restore_network_config
Whether to restore the VM's networking configuration

Type: boolean

Default: true

Resumes the specified VMs

```
<acropolis> vm.resume vm_list
```

Required arguments

vm_list
Comma-delimited list of VM identifiers

Type: list of VMs

Resumes all paused VMs

```
<acropolis> vm.resume_all
```

Required arguments

None

Initiates a shutdown by issuing an ACPI event

```
<acropolis> vm.shutdown vm_list
```

Required arguments

vm_list
Comma-delimited list of VM identifiers

Type: list of VMs

Creates one or more snapshots in a single consistency group

If multiple VMs are specified, all of their configurations and disks will fall into the same consistency group. Since this operation requires the coordination of multiple resources, it should not be abused by specifying more than several VMs at a time. Snapshots are crash-consistent. They do not include the VM's current memory image, only the VM configuration and its disk contents. The snapshot is taken atomically across all of a VM's configuration and disks to ensure consistency. If no snapshot name is provided, the snapshot will be referred to as "<vm_name>-<timestamp>", where the timestamp is in ISO 8601 format (YYYY-MM-DDTHH:MM:SS.mmmmmm).

```
<acropolis> vm.snapshot_create vm_list [  snapshot_name_list="snapshot_name_list" ]
```

Required arguments

vm_list
Comma-delimited list of VM identifiers

Type: list of VMs

Optional arguments

snapshot_name_list
Comma-delimited list of names for each snapshot

Type: list of strings

Examples

1. Create a snapshot named 'dev-vm-gold' from a VM named 'dev-vm'.

```
<acropolis> vm.snapshot_create dev-vm snapshot_name_list=dev-vm-gold
```

2. Create a consistent snapshot across several VMs, using the default naming scheme.

```
<acropolis> vm.snapshot_create vm1,vm2,vm3
```

Prints the graph representation of the snapshot history for a VM

```
<acropolis> vm.snapshot_get_tree vm
```

Required arguments

vm
VM identifier

Type: VM

Gets a list of all snapshots associated with a VM

```
<acropolis> vm.snapshot_list vm
```

Required arguments

vm
VM identifier

Type: VM

Updates the specified VMs

Note that some attributes may not be modifiable while the VM is running. For instance, the KVM hypervisor does not support CPU or memory hot-plug. Memory size must be specified with a multiplicative suffix. The following suffixes are valid: M=2^20, G=2^30. The hwclock_timezone attribute specifies the VM's hardware clock timezone. Most operating systems assume the system clock is UTC, but some (like Windows) expect the local timezone. Changes to the clock timezone only take effect after a full VM power cycle. This command can be used to reclaim memory from guests using a balloon driver. It is not currently possible to return ballooned memory to a guest. The attempt to reclaim memory may fail if the balloon driver is not installed, or is unable to allocate the requested amount of memory from the guest OS. In this case, the VM's memory reservation will be restored to its value from before the balloon attempt.

```
<acropolis> vm.update vm_list [  annotation="string" ][  ha_priority="num"
][  hwclock_timezone="timezone" ][  memory="memory" ][  name="name" ][
num_cores_per_vcpu="num" ][  num_vcpus="num" ]
```

Required arguments

vm_list
Comma-delimited list of VM identifiers

Type: list of VMs

Optional arguments

annotation
Annotation string

Type: string

ha_priority
Numeric priority for HA restart. Negative value indicates no restart.

Type: int

hwclock_timezone
Hardware clock timezone

Type: timezone

memory
Memory size

Type: size with MG suffix

name
VM name

Type: string

num_cores_per_vcpu
Number of cores per vCPU

Type: int

num_vcpus
Number of vCPUs

Type: int

Updates a VM's boot device

```
<acropolis> vm.update_boot_device vm [  disk_addr="disk_addr" ][  mac_addr="mac_addr" ]
```

Required arguments

vm
VM identifier

Type: VM

Optional arguments

disk_addr
Disk bus address

Type: VM disk

mac_addr
NIC MAC address

Type: NIC address

5

Virtualization Management REST API Reference

Ha

Get current HA configuration

`GET /ha/`

Details

path	/ha/
method	GET
nickname	getHaConfig
type	get.dto.acropolis.HaConfigDTO

get.dto.acropolis.HaConfigDTO

Property	Type	Format
numHostFailuresToTolerate	integer	int64
logicalTimestamp	integer	int64
reservedHostUuids	array	
failoverInProgressHostUuids	array	
reservationType	enums.acropolis.HaReservationType	
failoverEnabled	boolean	
haState	dto.acropolis.HaConfigDTO$HaState	

Enable, disable or modify HA configuration

`PUT /ha/`

Details

path	`/ha/`
method	PUT
nickname	updateHaConfig
type	update.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
	HA configuration	**paramType**	body
		required	true
		type	update.dto.acropolis.HaUpdateDT

Hosts

Put a host in maintenance mode

```
POST /hosts/{hostid}/enter_maintenance_mode
```

Details

path	`/hosts/{hostid}/enter_maintenance_mode`
method	POST
nickname	enterMaintenanceMode
type	create.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
hostid	UUID of host to put in maintenance mode	**paramType**	path
		required	true
		type	string
	Maintenance mode options	**paramType**	body
		required	true
		type	create.dto.acropolis.MaintenanceI

Pull a host out of maintenance mode or abort a prior attempt

```
POST /hosts/{hostid}/exit_maintenance_mode
```

Details

path	`/hosts/{hostid}/exit_maintenance_mode`
method	POST
nickname	exitMaintenanceMode
type	create.dto.PrimitiveDTO<java.lang.Boolean>

Parameters

Parameter	Description	Details	
hostid	UUID of host to pull out of maintenance mode	**paramType**	path
		required	true
		type	string
	Logical timestamp associated with host object	**paramType**	body
		required	false
		type	create.dto.acropolis.RequestValue$LogicalTimestampDTO

create.dto.PrimitiveDTO<java.lang.Boolean>

Property	Type	Format
value	boolean	

Images

Get the list of Disk Images

```
GET /images/
```

Details

path	`/images/`
method	GET
nickname	getImages

type	get.base.EntityCollection<get.dto.acropolis.ImageInfoDTO>

Parameters

Parameter	Description	Details	
includeVmDiskSizes	Include VmDisk sizes	**paramType**	query
		required	false
		type	boolean

get.base.EntityCollection<get.dto.acropolis.ImageInfoDTO>

Property	Type	Format
metadata	get.base.Metadata	
entities	array	

Create a Disk Image

```
POST /images/
```

Details

path	`/images/`
method	POST
nickname	createImage
type	create.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
	Image Creation Specs	**paramType**	body
		required	true
		type	create.dto.acropolis.ImageSpecD

Delete a Disk Image

```
DELETE /images/{imageId}/
```

Details

path	`/images/{imageId}/`
method	DELETE
nickname	deleteImage
type	delete.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
imageId	Id of the Image	**paramType**	path
		required	true
		type	string
logicaltimestamp	Logical timestamp for synchronized delete	**paramType**	query
		required	false
		type	integer

delete.dto.acropolis.ReturnValueDTO$TaskIdDTO

Property	Type	Format
taskUuid	string	

Update a Disk Image

```
PUT /images/{imageId}/
```

Details

path	`/images/{imageId}/`
method	PUT
nickname	updateImage
type	update.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
imageId	Id of the Disk Image	**paramType**	path

Parameter	Description	Details	
		required	true
		type	string
	Disk Image Update Info	**paramType**	body
		required	true
		type	update.dto.acropolis.ImageSpecD

Get details of a specific Image based on the given Id

```
GET /images/{imageId}/
```

Details

path	`/images/{imageId}/`
method	GET
nickname	getImage
type	get.dto.acropolis.ImageInfoDTO

Parameters

Parameter	Description	Details	
imageId	Id of the Image	**paramType**	path
		required	true
		type	string
includeVmDiskSizes	Include VmDisk size	**paramType**	query
		required	false
		type	boolean

get.dto.acropolis.ImageInfoDTO

Property	Type	Format
annotation	string	
imageState	dto.acropolis.ImageInfoDTO$ImageState	
createdTimeInUsecs	integer	int64
uuid	string	
vmDiskSize	integer	int64

Property	Type	Format
deleted	boolean	
logicalTimestamp	integer	int64
vmDiskId	string	
name	string	
checksumType	dto.acropolis.ImageInfoDTO $ChecksumType	
checksum	string	
containerId	integer	int64
imageType	dto.acropolis.ImageInfoDTO$ImageType	
updatedTimeInUsecs	integer	int64

Networks

Get list of networks

```
GET /networks/
```

Details

path	/networks/
method	GET
nickname	getNetworks
type	get.base.EntityCollection<get.dto.acropolis.NetworkConfigDTO>

get.base.EntityCollection<get.dto.acropolis.NetworkConfigDTO>

Property	Type	Format
metadata	get.base.Metadata	
entities	array	

Create a network

```
POST /networks/
```

Details

path	`/networks/`
method	POST
nickname	createNetwork
type	create.dto.acropolis.ReturnValueDTO$NetworkIdDTO

Parameters

Parameter	Description	Details	
	Network config info	**paramType**	body
		required	true
		type	create.dto.acropolis.NetworkConf

Get info of a network

`GET /networks/{networkid}`

Details

path	`/networks/{networkid}`
method	GET
nickname	getNetwork
type	get.dto.acropolis.NetworkConfigDTO

Parameters

Parameter	Description	Details	
networkid	Id of the network	**paramType**	path
		required	true
		type	string

get.dto.acropolis.NetworkConfigDTO

Property	Type	Format
ipConfig	get.dto.acropolis.IpConfigDTO	
annotation	string	

Property	Type	Format
vswitchName	string	
logicalTimestamp	integer	int64
vlanId	integer	int32
name	string	
uuid	string	

Update a network

```
PUT /networks/{networkid}
```

Details

path	`/networks/{networkid}`
method	PUT
nickname	updateNetwork
type	update.dto.PrimitiveDTO<java.lang.Boolean>

Parameters

Parameter	Description	Details	
networkid	Id of the network	**paramType**	path
		required	true
		type	string
	Updated network spec	**paramType**	body
		required	true
		type	update.dto.acropolis.NetworkCon

update.dto.PrimitiveDTO<java.lang.Boolean>

Property	Type	Format
value	boolean	

Delete a network

```
DELETE /networks/{networkid}
```

Details

path	`/networks/{networkid}`
method	DELETE
nickname	deleteNetwork
type	delete.dto.PrimitiveDTO<java.lang.Boolean>

Parameters

Parameter	Description	Details	
networkid	Id of the network	**paramType**	path
		required	true
		type	string
logicaltimestamp	Logical timestamp for synchronized delete	**paramType**	query
		required	false
		type	integer

delete.dto.PrimitiveDTO<java.lang.Boolean>

Property	Type	Format
value	boolean	

Get IP addresses assigned in the specified network

```
GET /networks/{networkid}/addresses
```

Details

path	`/networks/{networkid}/addresses`
method	GET
nickname	getNetworkAddressTable
type	get.base.EntityCollection<get.dto.acropolis.AddressAssignmentDTO>

Parameters

Parameter	Description	Details	
networkid	Id of the network	**paramType**	path

Parameter	Description	Details	
		required	true
		type	string

get.base.EntityCollection<get.dto.acropolis.AddressAssignmentDTO>

Property	Type	Format
metadata	get.base.Metadata	
entities	array	

Remove an IP address from the managed network blacklist

```
DELETE /networks/{networkid}/blacklist_IP/{ipaddress}
```

Details

path	`/networks/{networkid}/blacklist_IP/{ipaddress}`
method	DELETE
nickname	unreserveIP
type	delete.dto.PrimitiveDTO<java.lang.Boolean>

Parameters

Parameter	Description	Details	
networkid	Id of the network	**paramType**	path
		required	true
		type	string
ipaddress	IP address	**paramType**	path
		required	true
		type	string

delete.dto.PrimitiveDTO<java.lang.Boolean>

Property	Type	Format
value	boolean	

Blacklist an IP address from managed network

```
POST /networks/{networkid}/blacklist_ip_addresses
```

Details

path	`/networks/{networkid}/blacklist_ip_addresses`
method	POST
nickname	reserveIP
type	create.dto.PrimitiveDTO<java.lang.Boolean>

Parameters

Parameter	Description	Details	
networkid	Id of the network	**paramType**	path
		required	true
		type	string
	IP addresses to reserve. Comma seperated list of IP addresses.	**paramType**	body
		required	true
		type	create.dto.PrimitiveDTO<java.lang

create.dto.PrimitiveDTO<java.lang.Boolean>

Property	Type	Format
value	boolean	

Snapshots

Get a list of snapshots in a cluster

```
GET /snapshots/
```

Details

path	`/snapshots/`
method	GET
nickname	getSnapshots

type	get.base.EntityCollection<get.dto.acropolis.SnapshotInfoDTO>

get.base.EntityCollection<get.dto.acropolis.SnapshotInfoDTO>

Property	Type	Format
metadata	get.base.Metadata	
entities	array	

Create Virtual Machine snapshots

`POST /snapshots/`

Details

path	`/snapshots/`
method	POST
nickname	createSnapshot
type	create.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
	Create a snapshot	**paramType**	body
		required	true
		type	create.dto.acropolis.SnapshotCre

Clone a Snapshot

`POST /snapshots/{snapshotid}/clone`

Details

path	`/snapshots/{snapshotid}/clone`
method	POST
nickname	cloneSnapshot
type	create.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
snapshotid	Id of the Snapshot	**paramType**	path
		required	true
		type	string
	Clone a Snapshot	**paramType**	body
		required	false
		type	create.dto.acropolis.SnapshotClo

Delete a snapshot

```
DELETE /snapshots/{uuid}
```

Details

path	`/snapshots/{uuid}`
method	DELETE
nickname	deleteSnapshot
type	delete.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
uuid	Id of the Snapshot	**paramType**	path
		required	true
		type	string
logicalTimestamp	Logical timestamp	**paramType**	query
		required	false
		type	integer

delete.dto.acropolis.ReturnValueDTO$TaskIdDTO

Property	Type	Format
taskUuid	string	

Get details of a specified snapshot

```
GET /snapshots/{uuid}
```

Details

path	`/snapshots/{uuid}`
method	GET
nickname	getSnapshot
type	get.dto.acropolis.SnapshotInfoDTO

Parameters

Parameter	Description	Details	
uuid	Id of the Snapshot	**paramType**	path
		required	true
		type	string

get.dto.acropolis.SnapshotInfoDTO

Property	Type	Format
deleted	boolean	
logicalTimestamp	integer	int64
groupUuid	string	
vmUuid	string	
vmCreateSpecification	get.dto.acropolis.VMCreateDTO	
snapshotName	string	
createdTime	integer	int64
uuid	string	

Tasks

Get a list of tasks

```
GET /tasks/
```

Details

path	`/tasks/`
method	GET
nickname	getTasks
type	get.base.EntityCollection<get.dto.acropolis.tasks.TaskDTO>

Parameters

Parameter	Description	Details	
entityTypes	Comma separated Entity types	**paramType**	query
		required	false
		type	string
entityUuids	Comma separated Entity types	**paramType**	query
		required	false
		type	string
operationTypeList	Comma separated Operation types	**paramType**	query
		required	false
		type	string
includeCompleted	Include Completed Tasks	**paramType**	query
		required	false
		type	boolean
epochCutOffTime	Tasks greater than cut off epoch time in microseconds will be returned. This is applicable only when include completed is set to True.	**paramType**	query
		required	false
		type	integer
count	Maximum number of tasks	**paramType**	query
		required	false
		type	integer
includeEntityNames	Include entity names	**paramType**	query
		required	false
		type	boolean

get.base.EntityCollection<get.dto.acropolis.tasks.TaskDTO>

Property	Type	Format
metadata	get.base.Metadata	
entities	array	

Get details of the specified task

```
GET /tasks/{taskid}
```

Details

path	`/tasks/{taskid}`
method	GET
nickname	getTask
type	get.dto.acropolis.tasks.TaskDTO

Parameters

Parameter	Description	Details	
taskid	Id of the task	**paramType**	path
		required	true
		type	string
includeEntityNames	Include entity names	**paramType**	query
		required	false
		type	boolean

get.dto.acropolis.tasks.TaskDTO

Property	Type	Format
metaResponse	get.dto.acropolis.tasks.MetaResponseDTO	
completeTime	integer	int64
message	string	
uuid	string	
createTime	integer	int64
percentageComplete	integer	int64
entityList	array	

Property	Type	Format
parentTaskUuid	string	
progressStatus	dto.acropolis.tasks.TaskDTO$Status	
startTime	integer	int64
lastUpdatedTime	integer	int64
operationType	dto.acropolis.tasks.TaskDTO $OperationType	
subtaskUuidList	array	
metaRequest	get.dto.acropolis.tasks.MetaRequestDTO	

Poll a task

```
GET /tasks/{taskid}/poll
```

Details

path	`/tasks/{taskid}/poll`
method	GET
nickname	pollTask
type	get.dto.acropolis.tasks.TaskPollResultDTO

Parameters

Parameter	Description	Details	
taskid	Id of the task	**paramType**	path
		required	true
		type	string
timeoutseconds	Timeout seconds	**paramType**	query
		required	false
		type	integer
includeEntityNames	Include entity names	**paramType**	query
		required	false
		type	boolean

get.dto.acropolis.tasks.TaskPollResultDTO

Property	Type	Format
timedOut	boolean	
isUnrecognized	boolean	
taskInfo	get.dto.acropolis.tasks.TaskDTO	

Vdisks

Get a list of vdisks in the cluster

```
GET /vdisks/
```

Details

path	`/vdisks/`
method	GET
nickname	getVdisks
type	get.base.EntityCollection<get.dto.NdfsFileDTO>

Parameters

Parameter	Description	Details	
path	Path of ndfs file	**paramType**	query
		required	true
		type	string

get.base.EntityCollection<get.dto.NdfsFileDTO>

Property	Type	Format
metadata	get.base.Metadata	
entities	array	

Vms

Get a list of KVM managed Virtual Machines

```
GET /vms/
```

Details

path	`/vms/`
method	GET
nickname	getVMs
type	get.base.EntityCollection<get.dto.acropolis.VMInfoDTO>

Parameters

Parameter	Description	Details	
includeVMDiskSizes	Whether to include Virtual Machine disk sizes in bytes	**paramType**	query
		required	false
		type	boolean
includeAddressAssignments	Whether to include network address assignments	**paramType**	query
		required	false
		type	boolean

get.base.EntityCollection<get.dto.acropolis.VMInfoDTO>

Property	Type	Format
metadata	get.base.Metadata	
entities	array	

Create a Virtual Machine

```
POST /vms/
```

Details

path	`/vms/`
method	POST
nickname	createVM
type	create.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
	Create a Virtual Machine	**paramType**	body
		required	true
		type	create.dto.acropolis.VMCreateDT

Get details of a KVM managed Virtual Machine

```
GET /vms/{vmid}
```

Details

path	`/vms/{vmid}`
method	GET
nickname	getVM
type	get.dto.acropolis.VMInfoDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
includeVMDiskSizes	Whether to include Virtual Machine disk sizes in bytes	**paramType**	query
		required	false
		type	boolean
includeAddressAssignments	Whether to include address assignments	**paramType**	query
		required	false
		type	boolean

get.dto.acropolis.VMInfoDTO

Property	Type	Format
logicalTimestamp	integer	int64
hostUuid	string	
state	dto.acropolis.VMInfoDTO$VMState	

Property	Type	Format
uuid	string	
config	get.dto.acropolis.VMConfigDTO	

Delete a Virtual Machine

```
DELETE /vms/{vmid}/
```

Details

path	`/vms/{vmid}/`
method	DELETE
nickname	deleteVM
type	delete.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string

delete.dto.acropolis.ReturnValueDTO$TaskIdDTO

Property	Type	Format
taskUuid	string	

Update a Virtual Machine

```
PUT /vms/{vmid}/
```

Details

path	`/vms/{vmid}/`
method	PUT
nickname	updateVM
type	update.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
	Virtual Machine Update Info	**paramType**	body
		required	true
		type	update.dto.acropolis.VMUpdateD

Clone a Virtual Machine

```
POST /vms/{vmid}/clone
```

Details

path	`/vms/{vmid}/clone`
method	POST
nickname	cloneVM
type	create.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
	Clone a Virtual Machine	**paramType**	body
		required	false
		type	create.dto.acropolis.VMCloneDTO

Get list of disks in a Virtual Machine

```
GET /vms/{vmid}/disks/
```

Details

path	`/vms/{vmid}/disks/`

method	GET
nickname	getDisks
type	get.base.EntityCollection<get.dto.acropolis.VMDiskConfigDTO>

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
busType	Bus Type of the disk(IDE/SCSI)	**paramType**	query
		required	false
		type	string
deviceIndex	Device Index	**paramType**	query
		required	false
		type	string
includeDiskSizes	Include disk sizes in bytes	**paramType**	query
		required	false
		type	boolean

get.base.EntityCollection<get.dto.acropolis.VMDiskConfigDTO>

Property	Type	Format
metadata	get.base.Metadata	
entities	array	

Create a disk in a Virtual Machine

```
POST /vms/{vmid}/disks/
```

Details

path	`/vms/{vmid}/disks/`
method	POST
nickname	createDisk
type	create.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
	Config of the disks to be created	**paramType**	body
		required	true
		type	create.dto.acropolis.VMDiskCreat

Update info of a disk in a Virtual Machine

```
PUT /vms/{vmid}/disks/{diskaddress}
```

Details

path	`/vms/{vmid}/disks/{diskaddress}`
method	PUT
nickname	updateDisk
type	update.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
diskaddress	Address of the Disk	**paramType**	path
		required	true
		type	string
	Identifier of disks	**paramType**	body
		required	true
		type	update.dto.acropolis.VMDiskUpda

Delete a disk from a Virtual Machine

```
DELETE /vms/{vmid}/disks/{diskaddress}
```

Details

path	`/vms/{vmid}/disks/{diskaddress}`
method	DELETE
nickname	deleteDisk
type	delete.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
diskaddress	Address of the Disk	**paramType**	path
		required	true
		type	string
vmLogicalTimestamp	Virtual Machine Logical timestamp	**paramType**	query
		required	false
		type	integer

delete.dto.acropolis.ReturnValueDTO$TaskIdDTO

Property	Type	Format
taskUuid	string	

Get info of a disk in a Virtual Machine

```
GET /vms/{vmid}/disks/{diskid}
```

Details

path	`/vms/{vmid}/disks/{diskid}`
method	GET
nickname	getDisk
type	get.dto.acropolis.VMDiskConfigDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
diskid	Id of the disk	**paramType**	path
		required	true
		type	string
includeDiskSizes	Include disk sizes in bytes	**paramType**	query
		required	false
		type	boolean

get.dto.acropolis.VMDiskConfigDTO

Property	Type	Format
vmDiskSize	integer	int64
isEmpty	boolean	
vmDiskUuid	string	
id	string	
addr	get.dto.acropolis.VMDiskAddressDTO	
containerId	integer	int64
isCdrom	boolean	
isSCSIPassthrough	boolean	

Migrate a Virtual Machine

`POST /vms/{vmid}/migrate`

Details

path	`/vms/{vmid}/migrate`
method	POST
nickname	migrateVM
type	create.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
	Migrate Virtual Machine	**paramType**	body
		required	true
		type	create.dto.acropolis.VMMigrateDT

Abort migrate of a Virtual Machine

```
POST /vms/{vmid}/migrate_abort
```

Details

path	`/vms/{vmid}/migrate_abort`
method	POST
nickname	migrateVMAbort
type	create.dto.PrimitiveDTO<java.lang.Boolean>

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
	Virtual Machine Logical timestamp	**paramType**	body
		required	true
		type	create.dto.acropolis.RequestValue$LogicalTimestampDTO

create.dto.PrimitiveDTO<java.lang.Boolean>

Property	Type	Format
value	boolean	

Add a NIC to a Virtual Machine

`POST /vms/{vmid}/nics/`

Details

path	`/vms/{vmid}/nics/`
method	POST
nickname	createNic
type	create.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
	NIC Config Info	**paramType**	body
		required	true
		type	create.dto.acropolis.VMNicCreate

Get list of NICs in a Virtual Machine

`GET /vms/{vmid}/nics/`

Details

path	`/vms/{vmid}/nics/`
method	GET
nickname	getNics
type	get.base.EntityCollection<get.dto.acropolis.VMNicSpecDTO>

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string

Parameter	Description	Details	
includeAddressAssignments	Include address assignments	**paramType**	query
		required	false
		type	boolean

get.base.EntityCollection<get.dto.acropolis.VMNicSpecDTO>

Property	Type	Format
metadata	get.base.Metadata	
entities	array	

Details of a NIC in a Virtual Machine

```
GET /vms/{vmid}/nics/{nicid}
```

Details

path	`/vms/{vmid}/nics/{nicid}`
method	GET
nickname	getNic
type	get.dto.acropolis.VMNicSpecDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
nicid	Virtual Machine NIC identifier	**paramType**	path
		required	true
		type	string
includeAddressAssignments	Include address assignments	**paramType**	query
		required	false
		type	boolean

get.dto.acropolis.VMNicSpecDTO

Property	Type	Format
macAddress	string	
model	string	
requestedIpAddress	string	
networkUuid	string	

Delete a NIC from a Virtual Machine

```
DELETE /vms/{vmid}/nics/{nicid}
```

Details

path	`/vms/{vmid}/nics/{nicid}`
method	DELETE
nickname	deleteNic
type	delete.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
nicid	NIC identifier	**paramType**	path
		required	true
		type	string
vmLogicalTimestamp	Virtual Machine Logical timestamp	**paramType**	query
		required	false
		type	integer

delete.dto.acropolis.ReturnValueDTO$TaskIdDTO

Property	Type	Format
taskUuid	string	

Power off a Virtual Machine

```
POST /vms/{vmid}/power_op/off
```

Details

path	`/vms/{vmid}/power_op/off`
method	POST
nickname	powerOff
type	create.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
vmid	Id of the VM	**paramType**	path
		required	true
		type	string
	Logical timestamp of the VM	**paramType**	body
		required	false
		type	create.dto.acropolis.RequestValue$LogicalTimestampDTO

Power on a Virtual Machine

```
POST /vms/{vmid}/power_op/on
```

Details

path	`/vms/{vmid}/power_op/on`
method	POST
nickname	powerOn
type	create.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
vmid	Id of the VM	**paramType**	path
		required	true

Parameter	Description	Details	
		type	string
	Logical timestamp of the VM and host UUID	**paramType**	body
		required	false
		type	create.dto.acropolis.RequestValue$PowerOnDTO

Restore a Virtual Machine to a snapshotted state

```
POST /vms/{vmid}/restore
```

Details

path	`/vms/{vmid}/restore`
method	POST
nickname	restoreVM
type	create.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
	Restore Virtual Machine	**paramType**	body
		required	true
		type	create.dto.acropolis.VMRestoreD

Set power state of a Virtual Machine

```
POST /vms/{vmid}/set_power_state/
```

Details

path	`/vms/{vmid}/set_power_state/`
method	POST
nickname	setPowerState
type	create.dto.acropolis.ReturnValueDTO$TaskIdDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
	Virtual Machine Power State Info	**paramType**	body
		required	true
		type	create.dto.acropolis.VMPowerSta

Get a hierarchy of snapshots for a Virtual Machine

```
GET /vms/{vmid}/snapshots
```

Details

path	`/vms/{vmid}/snapshots`
method	GET
nickname	getVMSnapshots
type	get.dto.acropolis.SnapshotTreeInfoDTO

Parameters

Parameter	Description	Details	
vmid	Id of the Virtual Machine	**paramType**	path
		required	true
		type	string
includeSnapshots	Whether to include snapshot info	**paramType**	query
		required	false
		type	boolean

get.dto.acropolis.SnapshotTreeInfoDTO

Property	Type	Format
linkList	array	
logicalTimestamp	integer	int64
vmUuid	string	

Property	Type	Format
parentSnapshotUuid	string	