» vsphere_custom_attribute

The `vsphere_custom_attribute` data source can be used to reference custom attributes that are not managed by Terraform. Its attributes are exactly the same as the `vsphere_custom_attribute` resource, and, like importing, the data source takes a name to search on. The `id` and other attributes are then populated with the data found by the search.

NOTE: Custom attributes are unsupported on direct ESXi connections and require vCenter.

» Example Usage

```
data "vsphere_custom_attribute" "attribute" {
  name = "terraform-test-attribute"
}
```

» Argument Reference

- `name` - (Required) The name of the custom attribute.

» Attribute Reference

In addition to the `id` being exported, all of the fields that are available in the `vsphere_custom_attribute` resource are also populated. See that page for further details.

» vsphere_datacenter

The `vsphere_datacenter` data source can be used to discover the ID of a vSphere datacenter. This can then be used with resources or data sources that require a datacenter, such as the `vsphere_host` data source.

» Example Usage

```
data "vsphere_datacenter" "datacenter" {
  name = "dc1"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Optional) The name of the datacenter. This can be a name or path. Can be omitted if there is only one datacenter in your inventory.

NOTE: When used against ESXi, this data source *always* fetches the server's "default" datacenter, which is a special datacenter unrelated to the datacenters that exist in any vCenter server that might be managing this host. Hence, the `name` attribute is completely ignored.

» Attribute Reference

The only exported attribute is `id`, which is the managed object ID of this datacenter.

» vsphere_datastore

The `vsphere_datastore` data source can be used to discover the ID of a datastore in vSphere. This is useful to fetch the ID of a datastore that you want to use to create virtual machines in using the `vsphere_virtual_machine` resource.

» Example Usage

```
data "vsphere_datacenter" "datacenter" {
  name = "dc1"
}

data "vsphere_datastore" "datastore" {
  name          = "datastore1"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the datastore. This can be a name or path.
- `datacenter_id` - (Optional) The managed object reference ID of the datacenter the datastore is located in. This can be omitted if the search

path used in `name` is an absolute path. For default datacenters, use the id attribute from an empty `vsphere_datacenter` data source.

» Attribute Reference

Currently, the only exported attribute from this data source is `id`, which represents the ID of the datastore that was looked up.

» vsphere_distributed_virtual_switch

The `vsphere_distributed_virtual_switch` data source can be used to discover the ID and uplink data of a of a vSphere distributed virtual switch (DVS). This can then be used with resources or data sources that require a DVS, such as the `vsphere_distributed_port_group` resource, for which an example is shown below.

NOTE: This data source requires vCenter and is not available on direct ESXi connections.

» Example Usage

The following example locates a DVS that is named `terraform-test-dvs`, in the datacenter `dc1`. It then uses this DVS to set up a `vsphere_distributed_port_group` resource that uses the first uplink as a primary uplink and the second uplink as a secondary.

```
data "vsphere_datacenter" "datacenter" {
  name = "dc1"
}

data "vsphere_distributed_virtual_switch" "dvs" {
  name          = "terraform-test-dvs"
  datacenter_id = "${data.vsphere_datacenter.datacenter.id}"
}

resource "vsphere_distributed_port_group" "pg" {
  name                            = "terraform-test-pg"
  distributed_virtual_switch_uuid = "${data.vsphere_distributed_virtual_switch.dvs.id}"

  active_uplinks  = ["${data.vsphere_distributed_virtual_switch.dvs.uplinks[0]}"]
  standby_uplinks = ["${data.vsphere_distributed_virtual_switch.dvs.uplinks[1]}"]
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the distributed virtual switch. This can be a name or path.
- `datacenter_id` - (Optional) The managed object reference ID of the datacenter the DVS is located in. This can be omitted if the search path used in `name` is an absolute path. For default datacenters, use the id attribute from an empty `vsphere_datacenter` data source.

» Attribute Reference

The following attributes are exported:

- `id`: The UUID of the distributed virtual switch.
- `uplinks`: The list of the uplinks on this DVS, as per the `uplinks` argument to the `vsphere_distributed_virtual_switch` resource.

» vsphere_host

The `vsphere_host` data source can be used to discover the ID of a vSphere host. This can then be used with resources or data sources that require a host managed object reference ID.

» Example Usage

```
data "vsphere_datacenter" "datacenter" {
  name = "dc1"
}

data "vsphere_host" "host" {
  name          = "esxi1"
  datacenter_id = "${data.vsphere_datacenter.datacenter.id}"
}
```

» Argument Reference

The following arguments are supported:

- `datacenter_id` - (Required) The managed object reference ID of a datacenter.

- `name` - (Optional) The name of the host. This can be a name or path. Can be omitted if there is only one host in your inventory.

NOTE: When used against an ESXi host directly, this data source *always* fetches the server's host object ID, regardless of what is entered into `name`.

» Attribute Reference

The only exported attribute is `id`, which is the managed object ID of this host.

» vsphere_network

The `vsphere_network` data source can be used to discover the ID of a network in vSphere. This can be any network that can be used as the backing for a network interface for `vsphere_virtual_machine` or any other vSphere resource that requires a network. This includes standard (host-based) port groups, DVS port groups, or opaque networks such as those managed by NSX.

» Example Usage

```
data "vsphere_datacenter" "datacenter" {
  name = "dc1"
}

data "vsphere_network" "net" {
  name          = "terraform-test-net"
  datacenter_id = "${data.vsphere_datacenter.datacenter.id}"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the network. This can be a name or path.
- `datacenter_id` - (Optional) The managed object reference ID of the datacenter the network is located in. This can be omitted if the search path used in `name` is an absolute path. For default datacenters, use the id attribute from an empty `vsphere_datacenter` data source.

» Attribute Reference

The following attributes are exported:

- `id`: The managed object ID of the network in question.
- `type`: The managed object type for the discovered network. This will be one of `DistributedVirtualPortgroup` for DVS port groups, `Network` for standard (host-based) port groups, or `OpaqueNetwork` for networks managed externally by features such as NSX.

» vsphere_resource_pool

The `vsphere_resource_pool` data source can be used to discover the ID of a resource pool in vSphere. This is useful to fetch the ID of a resource pool that you want to use to create virtual machines in using the `vsphere_virtual_machine` resource.

» Example Usage

```
data "vsphere_datacenter" "datacenter" {
  name = "dc1"
}

data "vsphere_resource_pool" "pool" {
  name          = "resource-pool-1"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
```

» Specifying the root resource pool for a cluster or standalone host

All clusters and standalone hosts have a resource pool, even if one has not been explicitly created. This resource pool is referred to as the *root resource pool* and can be looked up by specifying the path as per the example below:

```
data "vsphere_resource_pool" "pool" {
  name          = "cluster1/Resources"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
```

For more information on the root resource pool, see Managing Resource Pools in the vSphere documentation.

» Argument Reference

The following arguments are supported:

- `name` - (Optional) The name of the resource pool. This can be a name or path. This is required when using vCenter.
- `datacenter_id` - (Optional) The managed object reference ID of the datacenter the resource pool is located in. This can be omitted if the search path used in `name` is an absolute path. For default datacenters, use the id attribute from an empty `vsphere_datacenter` data source.

Note when using with standalone ESXi: When using ESXi without vCenter, you don't have to specify either attribute to use this data source. An empty declaration will load the host's root resource pool.

» Attribute Reference

Currently, the only exported attribute from this data source is `id`, which represents the ID of the resource pool that was looked up.

» vsphere_tag

The `vsphere_tag` data source can be used to reference tags that are not managed by Terraform. Its attributes are exactly the same as the `vsphere_tag` resource, and, like importing, the data source takes a name and category to search on. The `id` and other attributes are then populated with the data found by the search.

NOTE: Tagging support is unsupported on direct ESXi connections and requires vCenter 6.0 or higher.

» Example Usage

```
data "vsphere_tag_category" "category" {
  name = "terraform-test-category"
}

data "vsphere_tag" "tag" {
  name        = "terraform-test-tag"
  category_id = "${data.vsphere_tag_category.category.id}"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the tag.

- `category_id` - (Required) The ID of the tag category the tag is located in.

» Attribute Reference

In addition to the `id` being exported, all of the fields that are available in the `vsphere_tag` resource are also populated. See that page for further details.

» vsphere_tag_category

The `vsphere_tag_category` data source can be used to reference tag categories that are not managed by Terraform. Its attributes are exactly the same as the `vsphere_tag_category` resource, and, like importing, the data source takes a name to search on. The `id` and other attributes are then populated with the data found by the search.

NOTE: Tagging support is unsupported on direct ESXi connections and requires vCenter 6.0 or higher.

» Example Usage

```
data "vsphere_tag_category" "category" {
  name = "terraform-test-category"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the tag category.

» Attribute Reference

In addition to the `id` being exported, all of the fields that are available in the `vsphere_tag_category` resource are also populated. See that page for further details.

» vsphere_virtual_machine

The `vsphere_virtual_machine` data source can be used to find the UUID of an existing virtual machine or template. Its most relevant purpose is for finding the UUID of a template to be used as the source for cloning into a new `vsphere_virtual_machine` resource. It also reads the guest ID so that can be supplied as well.

» Example Usage

```
data "vsphere_datacenter" "datacenter" {
  name = "dc1"
}

data "vsphere_virtual_machine" "template" {
  name          = "test-vm-template"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the virtual machine. This can be a name or path.
- `datacenter_id` - (Optional) The managed object reference ID of the datacenter the virtual machine is located in. This can be omitted if the search path used in `name` is an absolute path. For default datacenters, use the `id` attribute from an empty `vsphere_datacenter` data source.
- `scsi_controller_scan_count` - (Optional) The number of SCSI controllers to scan for disk attributes and controller types on. Default: `1`.

NOTE: For best results, ensure that all the disks on any templates you use with this data source reside on the primary controller, and leave this value at the default. See the `vsphere_virtual_machine` resource documentation for the significance of this setting, specifically the additional requirements and notes for cloning section.

» Attribute Reference

The following attributes are exported:

- `id` - The UUID of the virtual machine or template.
- `guest_id` - The guest ID of the virtual machine or template.

- `alternate_guest_name` - The alternate guest name of the virtual machine when guest_id is a non-specific operating system, like `otherGuest`.
- `scsi_type` - The common type of all SCSI controllers on this virtual machine. Will be one of `lsilogic` (LSI Logic Parallel), `lsilogic-sas` (LSI Logic SAS), `pvscsi` (VMware Paravirtual), `buslogic` (BusLogic), or `mixed` when there are multiple controller types. Only the first number of controllers defined by `scsi_controller_scan_count` are scanned.
- `disks` - Information about each of the disks on this virtual machine or template. These are sorted by bus and unit number so that they can be applied to a `vsphere_virtual_machine` resource in the order the resource expects while cloning. This is useful for discovering certain disk settings while performing a linked clone, as all settings that are output by this data source must be the same on the destination virtual machine as the source. Only the first number of controllers defined by `scsi_controller_scan_count` are scanned for disks. The sub-attributes are:
  - `size` - The size of the disk, in GIB.
  - `eagerly_scrub` - Set to `true` if the disk has been eager zeroed.
  - `thin_provisioned` - Set to `true` if the disk has been thin provisioned.
- `network_interface_types` - The network interface types for each network interface found on the virtual machine, in device bus order. Will be one of `e1000`, `e1000e`, `pcnet32`, `sriov`, `vmxnet2`, or `vmxnet3`.

NOTE: Keep in mind when using the results of `scsi_type` and `network_interface_types`, that the `vsphere_virtual_machine` resource only supports a subset of the types returned from this data source. See the resource docs for more details.

» vsphere_vmfs_disks

The `vsphere_vmfs_disks` data source can be used to discover the storage devices available on an ESXi host. This data source can be combined with the `vsphere_vmfs_datastore` resource to create VMFS datastores based off a set of discovered disks.

» Example Usage

```
data "vsphere_datacenter" "datacenter" {
  name = "dc1"
}

data "vsphere_host" "host" {
  name          = "esxi1"
  datacenter_id = "${data.vsphere_datacenter.datacenter.id}"
}
```

```
data "vsphere_vmfs_disks" "available" {
  host_system_id = "${data.vsphere_host.host.id}"
  rescan         = true
  filter         = "mpx.vmhba1:C0:T[12]:L0"
}
```

» Argument Reference

The following arguments are supported:

- `host_system_id` - (Required) The managed object ID of the host to look for disks on.
- `rescan` - (Optional) Whether or not to rescan storage adapters before searching for disks. This may lengthen the time it takes to perform the search. Default: `false`.
- `filter` - (Optional) A regular expression to filter the disks against. Only disks with canonical names that match will be included.

NOTE: Using a `filter` is recommended if there is any chance the host will have any specific storage devices added to it that may affect the order of the output `disks` attribute below, which is lexicographically sorted.

» Attribute Reference

- `disks` - A lexicographically sorted list of devices discovered by the operation, matching the supplied `filter`, if provided.

» vsphere_license

Provides a VMware vSphere license resource. This can be used to add and remove license keys.

» Example Usage

```
resource "vsphere_license" "licenseKey" {
  license_key = "452CQ-2EK54-K8742-00000-00000"

  labels {
    VpxClientLicenseLabel = "Hello World"
    Workflow = "Hello World"
  }
```

```
}
```

» Argument Reference

The following arguments are supported:

- `license_key` - (Required) The license key to add.
- `labels` - (Optional) A map of key/value pairs to be attached as labels (tags) to the license key.

» Attributes Reference

The following attributes are exported:

- `edition_key` - The product edition of the license key.
- `total` - Total number of units (example: CPUs) contained in the license.
- `used` - The number of units (example: CPUs) assigned to this license.
- `name` - The display name for the license.

» vsphere_custom_attribute

The `vsphere_custom_attribute` resource can be used to create and manage custom attributes, which allow users to associate user-specific meta-information with vSphere managed objects. Custom attribute values must be strings and are stored on the vCenter Server and not the managed object.

For more information about custom attributes, click here.

NOTE: Custom attributes are unsupported on direct ESXi connections and require vCenter.

» Example Usage

This example creates a custom attribute named `terraform-test-attribute`. The resulting custom attribute can be assigned to VMs only.

```
resource "vsphere_custom_attribute" "attribute" {
  name                = "terraform-test-attribute"
  managed_object_type = "VirtualMachine"
}
```

» Using Custom Attributes in a Supported Resource

Custom attributes can be set on vSphere resources in Terraform via the `custom_attributes` argument in any supported resource.

The following example builds on the above example by creating a `vsphere_virtual_machine` and assigning a value to created custom attribute on it.

```
resource "vsphere_custom_attribute" "attribute" {
  name                = "terraform-test-attribute"
  managed_object_type = "VirtualMachine"
}

resource "vpshere_virtual_machine" "web" {
  ...

  custom_attributes = "${map(vsphere_custom_attribute.attribute.id, "value")}"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the custom attribute.
- `managed_object_type` - (Optional) The object type that this attribute may be applied to. If not set, the custom attribute may be applied to any object type. For a full list, click here. Forces a new resource if changed.

» Managed Object Types

The following table will help you determine what value you need to enter for the managed object type you want the attribute to apply to.

Note that if you want a attribute to apply to all objects, leave the type unspecified.

Type

Value

Folders

`Folder`

Clusters

`ClusterComputeResource`

Datacenters

`Datacenter`

Datastores

`Datastore`

Datastore Clusters

`StoragePod`

DVS Portgroups

`DistributedVirtualPortgroup`

Distributed vSwitches

`DistributedVirtualSwitch`
`VmwareDistributedVirtualSwitch`

Hosts

`HostSystem`

Content Libraries

`com.vmware.content.Library`

Content Library Items

`com.vmware.content.library.Item`

Networks

`HostNetwork`
`Network`
`OpaqueNetwork`

Resource Pools

`ResourcePool`

vApps

`VirtualApp`

Virtual Machines

`VirtualMachine`

» Attribute Reference

This resource only exports the `id` attribute for the vSphere custom attribute.

» Importing

An existing custom attribute can be imported into this resource via its name, using the following command:

```
terraform import vsphere_custom_attribute.attribute terraform-test-attribute
```

» vsphere_datacenter

Provides a VMware vSphere datacenter resource. This can be used as the primary container of inventory objects such as hosts and virtual machines.

» Example Usages

Create datacenter on the root folder:

```
resource "vsphere_datacenter" "prod_datacenter" {
  name       = "my_prod_datacenter"
}
```

Create datacenter on a subfolder:

```
resource "vsphere_datacenter" "research_datacenter" {
  name       = "my_research_datacenter"
  folder     = "/research/"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the datacenter. This name needs to be unique within the folder. Forces a new resource if changed.
- `folder` - (Optional) The folder where the datacenter should be created. Forces a new resource if changed.
- `tags` - (Optional) The IDs of any tags to attach to this resource. See here for a reference on how to apply tags.

NOTE: Tagging support is unsupported on direct ESXi connections and requires vCenter 6.0 or higher.

- `custom_attributes` - (Optional) Map of custom attribute ids to value strings to set for datacenter resource. See here for a reference on how to set values for custom attributes.

NOTE: Custom attributes are unsupported on direct ESXi connections and require vCenter.

» Attribute Reference

The only attribute exported is `id`, which is the name of the datacenter.

» vsphere_folder

The `vsphere_folder` resource can be used to manage vSphere inventory folders. The resource supports creating folders of the 5 major types - datacenter folders, host and cluster folders, virtual machine folders, datastore folders, and network folders.

Paths are always relative to the specific type of folder you are creating. Subfolders are discovered by parsing the relative path specified in `name`, so `foo/bar` will create a folder named `bar` in the parent folder `foo`, as long as that folder exists.

» Example Usage

The basic example below creates a virtual machine folder named `terraform-test-folder` in the default datacenter's VM hierarchy.

```
data "vsphere_datacenter" "dc" {}

resource "vsphere_folder" "folder" {
  path          = "terraform-test-folder"
  type          = "vm"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
```

» Example with subfolders

The below example builds off of the above by first creating a folder named `terraform-test-parent`, and then locating `terraform-test-folder` in that folder. To ensure the parent is created first, we create an interpolation dependency off the parent's `name` attribute.

Note that if you change parents (for example, went from the above basic configuration to this one), your folder will be moved to be under the correct parent.

```
data "vsphere_datacenter" "dc" {}

resource "vsphere_folder" "parent" {
  path          = "terraform-test-parent"
  type          = "vm"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

resource "vsphere_folder" "folder" {
  path          = "${vsphere_folder.parent.name}/terraform-test-folder"
  type          = "vm"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
```

» Argument Reference

The following arguments are supported:

- `path` - (Required) The path of the folder to be created. This is relative to the root of the type of folder you are creating, and the supplied datacenter. For example, given a default datacenter of `default-dc`, a folder of type `vm` (denoting a virtual machine folder), and a supplied folder of `terraform-test-folder`, the resulting path would be `/default-dc/vm/terraform-test-folder`.

NOTE: `path` can be modified - the resulting behavior is dependent on what section of `path` you are modifying. If you are modifying the parent (so any part before the last `/`), your folder will be moved to that new parent. If modifying the name (the part after the last `/`), your folder will be renamed.

- `type` - (Required) The type of folder to create. Allowed options are `datacenter` for datacenter folders, `host` for host and cluster folders, `vm` for virtual machine folders, `datastore` for datastore folders, and `network` for network folders. Forces a new resource if changed.
- `datacenter_id` - The ID of the datacenter the folder will be created in. Required for all folder types except for datacenter folders. Forces a new resource if changed.
- `tags` - (Optional) The IDs of any tags to attach to this resource. See here for a reference on how to apply tags.

NOTE: Tagging support is unsupported on direct ESXi connections and requires vCenter 6.0 or higher.

- `custom_attributes` - (Optional) Map of custom attribute ids to attribute value strings to set for folder. See here for a reference on how to set values for custom attributes.

NOTE: Custom attributes are unsupported on direct ESXi connections and require vCenter.

» Attribute Reference

The only attribute that this resource exports is the `id`, which is set to the managed object ID of the folder.

» Importing

An existing folder can be imported into this resource via its full path, via the following command:

```
terraform import vsphere_folder.folder /default-dc/vm/terraform-test-folder
```

The above command would import the folder from our examples above, the VM folder named `terraform-test-folder` located in the datacenter named `default-dc`.

» vsphere_tag

The `vsphere_tag` resource can be used to create and manage tags, which allow you to attach metadata to objects in the vSphere inventory to make these objects more sortable and searchable.

For more information about tags, click here.

NOTE: Tagging support is unsupported on direct ESXi connections and requires vCenter 6.0 or higher.

» Example Usage

This example creates a tag named `terraform-test-tag`. This tag is assigned the `terraform-test-category` category, which was created by the `vsphere_tag_category` resource. The resulting tag can be assigned to VMs and datastores only, and can be the only value in the category that can be assigned, as per the restrictions defined by the category.

```
resource "vsphere_tag_category" "category" {
  name        = "terraform-test-category"
  cardinality = "SINGLE"
  description = "Managed by Terraform"

  associable_types = [
```

```
    "VirtualMachine",
    "Datastore",
  ]
}

resource "vsphere_tag" "tag" {
  name        = "terraform-test-tag"
  category_id = "${vsphere_tag_category.category.id}"
  description = "Managed by Terraform"
}
```

» Using Tags in a Supported Resource

Tags can be applied to vSphere resources in Terraform via the `tags` argument in any supported resource.

The following example builds on the above example by creating a `vsphere_virtual_machine` and applying the created tag to it:

```
resource "vsphere_tag_category" "category" {
  name        = "terraform-test-category"
  cardinality = "SINGLE"
  description = "Managed by Terraform"

  associable_types = [
    "VirtualMachine",
    "Datastore",
  ]
}

resource "vsphere_tag" "tag" {
  name        = "terraform-test-tag"
  category_id = "${vsphere_tag_category.category.id}"
  description = "Managed by Terraform"
}

resource "vsphere_virtual_machine" "web" {
  ...

  tags = ["${vsphere_tag.tag.id}"]
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The display name of the tag. The name must be unique within its category.
- `category_id` - (Required) The unique identifier of the parent category in which this tag will be created. Forces a new resource if changed.
- `description` - (Optional) A description for the tag.

» Attribute Reference

The only attribute that is exported for this resource is the `id`, which is the uniform resource name (URN) of this tag.

» Importing

An existing tag can be imported into this resource by supplying both the tag's category name and the name of the tag as a JSON string to `terraform import`, as per the example below:

```
terraform import vsphere_tag.tag \
  '{"category_name": "terraform-test-category", "tag_name": "terraform-test-tag"}'
```

» vsphere_tag_category

The `vsphere_tag_category` resource can be used to create and manage tag categories, which determine how tags are grouped together and applied to specific objects.

For more information about tags, click here. For more information about tag categories specifically, click here.

NOTE: Tagging support is unsupported on direct ESXi connections and requires vCenter 6.0 or higher.

» Example Usage

This example creates a tag category named `terraform-test-category`, with single cardinality (meaning that only one tag in this category can be assigned to an object at any given time). Tags in this category can only be assigned to VMs and datastores.

```
resource "vsphere_tag_category" "category" {
  name        = "terraform-test-category"
  description = "Managed by Terraform"
  cardinality = "SINGLE"
```

```
  associable_types = [
    "VirtualMachine",
    "Datastore",
  ]
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the category.
- `cardinality` - (Required) The number of tags that can be assigned from this category to a single object at once. Can be one of `SINGLE` (object can only be assigned one tag in this category), to `MULTIPLE` (object can be assigned multiple tags in this category). Forces a new resource if changed.
- `associable_types` - (Required) A list object types that this category is valid to be assigned to. For a full list, click here.
- `description` - (Optional) A description for the category.

NOTE: You can add associable types to a category, but you cannot remove them. Attempting to do so will result in an error.

» Associable Object Types

The following table will help you determine what values you need to enter for the associable type you want to associate with a tag category.

Note that if you want a tag to apply to all objects, the `All` alias exists - just remember that you will not be able to revert this later, and this category will permanently allow all objects.

Type

Value

Folders

`Folder`

Clusters

`ClusterComputeResource`

Datacenters

`Datacenter`

Datastores

`Datastore`

Datastore Clusters

`StoragePod`

DVS Portgroups

`DistributedVirtualPortgroup`

Distributed vSwitches

`DistributedVirtualSwitch`
`VmwareDistributedVirtualSwitch`

Hosts

`HostSystem`

Content Libraries

`com.vmware.content.Library`

Content Library Items

`com.vmware.content.library.Item`

Networks

`HostNetwork`
`Network`
`OpaqueNetwork`

Resource Pools

`ResourcePool`

vApps

`VirtualApp`

Virtual Machines

`VirtualMachine`

» Attribute Reference

The only attribute that is exported for this resource is the `id`, which is the uniform resource name (URN) of this tag category.

» Importing

An existing tag category can be imported into this resource via its name, using the following command:

```
terraform import vsphere_tag_category.category terraform-test-category
```

» vsphere_distributed_port_group

The `vsphere_distributed_port_group` resource can be used to manage vSphere distributed virtual port groups. These port groups are connected to distributed virtual switches, which can be managed by the `vsphere_distributed_virtual_switch` resource.

Distributed port groups can be used as networks for virtual machines, allowing VMs to use the networking supplied by a distributed virtual switch (DVS), with a set of policies that apply to that individual newtork, if desired.

For an overview on vSphere networking concepts, see this page. For more information on vSphere DVS portgroups, see this page.

NOTE: This resource requires vCenter and is not available on direct ESXi connections.

» Example Usage

The configuration below builds on the example given in the `vsphere_distributed_virtual_switch` resource by adding the `vsphere_distributed_port_group` resource, attaching itself to the DVS created here and assigning VLAN ID 1000.

```
variable "esxi_hosts" {
  default = [
    "esxi1",
    "esxi2",
    "esxi3",
  ]
}

variable "network_interfaces" {
  default = [
    "vmnic0",
    "vmnic1",
    "vmnic2",
    "vmnic3",
  ]
}

data "vsphere_datacenter" "dc" {
  name = "dc1"
}
```

```
data "vsphere_host" "host" {
  count         = "${length(var.esxi_hosts)}"
  name          = "${var.esxi_hosts[count.index]}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

resource "vsphere_distributed_virtual_switch" "dvs" {
  name          = "terraform-test-dvs"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"

  uplinks         = ["uplink1", "uplink2", "uplink3", "uplink4"]
  active_uplinks  = ["uplink1", "uplink2"]
  standby_uplinks = ["uplink3", "uplink4"]

  host {
    host_system_id = "${data.vsphere_host.host.0.id}"
    devices        = ["${var.network_interfaces}"]
  }

  host {
    host_system_id = "${data.vsphere_host.host.1.id}"
    devices        = ["${var.network_interfaces}"]
  }

  host {
    host_system_id = "${data.vsphere_host.host.2.id}"
    devices        = ["${var.network_interfaces}"]
  }
}

resource "vsphere_distributed_port_group" "pg" {
  name                            = "terraform-test-pg"
  distributed_virtual_switch_uuid = "${vsphere_distributed_virtual_switch.dvs.id}"

  vlan_id = 1000
}
```

» Overriding DVS policies

All of the default port policies available in the `vsphere_distributed_virtual_switch` resource can be overridden on the port group level by specifying new settings for them.

As an example, we also take this example from the `vsphere_distributed_virtual_switch`

resource where we manually specify our uplink count and uplink order. While the DVS has a default policy of using the first uplink as an active uplink and the second one as a standby, the overridden port group policy means that both uplinks will be used as active uplinks in this specific port group.

```
resource "vsphere_distributed_virtual_switch" "dvs" {
  name          = "terraform-test-dvs"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"

  uplinks         = ["tfup1", "tfup2"]
  active_uplinks  = ["tfup1"]
  standby_uplinks = ["tfup2"]
}

resource "vsphere_distributed_port_group" "pg" {
  name                            = "terraform-test-pg"
  distributed_virtual_switch_uuid = "${vsphere_distributed_virtual_switch.dvs.id}"

  vlan_id = 1000

  active_uplinks  = ["tfup1", "tfup2"]
  standby_uplinks = []
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the port group.
- `distributed_virtual_switch_uuid` - (Required) The ID of the DVS to add the port group to. Forces a new resource if changed.
- `type` - (Optional) The port group type. Can be one of `earlyBinding` (static binding) or `ephemeral`. Default: `earlyBinding`.
- `description` - (Optional) An optional description for the port group.
- `number_of_ports` - (Optional) The number of ports available on this port group. Cannot be decreased below the amount of used ports on the port group.
- `auto_expand` - (Optional) Allows the port group to create additional ports past the limit specified in `number_of_ports` if necessary. Default: `true`.

NOTE: Using `auto_expand` with a statically defined `number_of_ports` may lead to errors when the port count grows past the amount specified. If you specify `number_of_ports`, you may wish to set `auto_expand` to `false`.

- `port_name_format` - (Optional) An optional formatting policy for naming of the ports in this port group. See the `portNameFormat` attribute listed here for details on the format syntax.

- `network_resource_pool_key` - (Optional) The key of a network resource pool to associate with this port group. The default is `-1`, which implies no association.
- `custom_attributes` (Optional) Map of custom attribute ids to attribute value string to set for port group. See here for a reference on how to set values for custom attributes.

NOTE: Custom attributes are unsupported on direct ESXi connections and require vCenter.

» Policy options

In addition to the above options, you can configure any policy option that is available under the `vsphere_distributed_virtual_switch` policy options section. Any policy option that is not set is inherited from the DVS, its options propagating to the port group.

See the link for a full list of options that can be set.

» Port override options

The following options below control whether or not the policies set in the port group can be overridden on the individual port:

- `block_override_allowed` - (Optional) Allow the port shutdown policy to be overridden on an individual port.
- `live_port_moving_allowed` - (Optional) Allow a port in this port group to be moved to another port group while it is connected.
- `netflow_override_allowed` - (Optional) Allow the Netflow policy on this port group to be overridden on an individual port.
- `network_resource_pool_override_allowed` - (Optional) Allow the network resource pool set on this port group to be overridden on an individual port.
- `port_config_reset_at_disconnect` - (Optional) Reset a port's settings to the settings defined on this port group policy when the port disconnects.
- `security_policy_override_allowed` - (Optional) Allow the security policy settings defined in this port group policy to be overridden on an individual port.
- `shaping_override_allowed` - (Optional) Allow the traffic shaping options on this port group policy to be overridden on an individual port.
- `traffic_filter_override_allowed` - (Optional) Allow any traffic filters on this port group to be overridden on an individual port.
- `uplink_teaming_override_allowed` - (Optional) Allow the uplink teaming options on this port group to be overridden on an individual port.

- `vlan_override_allowed` - (Optional) Allow the VLAN settings on this port group to be overridden on an individual port.

» Attribute Reference

The following attributes are exported:

- `id`: The managed object reference ID of the created port group.
- `key`: The generated UUID of the portgroup.

NOTE: While `id` and `key` may look the same in state, they are documented differently in the vSphere API and come from different fields in the port group object. If you are asked to supply an managed object reference ID to another resource, be sure to use the `id` field.

- `config_version`: The current version of the port group configuration, incremented by subsequent updates to the port group.

» Importing

An existing port group can be imported into this resource via the path to the port group, via the following command:

```
terraform import vsphere_distributed_port_group.pg /dc1/network/pg
```

The above would import the port group named `pg` that is located in the `dc1` datacenter.

» vsphere_distributed_virtual_switch

The `vsphere_distributed_virtual_switch` resource can be used to manage VMware Distributed Virtual Switches.

An essential component of a distributed, scalable VMware datacenter, the vSphere Distributed Virtual Switch (DVS) provides centralized management and monitoring of the networking configuration of all the hosts that are associated with the switch. In addition to adding port groups (see the `vsphere_distributed_port_group` resource) that can be used as networks for virtual machines, a DVS can be configured to perform advanced high availability, traffic shaping, network monitoring, and more.

For an overview on vSphere networking concepts, see this page. For more information on vSphere DVS, see this page.

NOTE: This resource requires vCenter and is not available on direct ESXi connections.

» Example Usage

The following example below demonstrates a ”standard” example of configuring a vSphere DVS in a 3-node vSphere datacenter named `dc1`, across 4 NICs with two being used as active, and two being used as passive. Note that the NIC failover order propagates to any port groups configured on this DVS and can be overridden there.

```
variable "esxi_hosts" {
  default = [
    "esxi1",
    "esxi2",
    "esxi3",
  ]
}

variable "network_interfaces" {
  default = [
    "vmnic0",
    "vmnic1",
    "vmnic2",
    "vmnic3",
  ]
}

data "vsphere_datacenter" "dc" {
  name = "dc1"
}

data "vsphere_host" "host" {
  count         = "${length(var.esxi_hosts)}"
  name          = "${var.esxi_hosts[count.index]}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

resource "vsphere_distributed_virtual_switch" "dvs" {
  name          = "terraform-test-dvs"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"

  uplinks         = ["uplink1", "uplink2", "uplink3", "uplink4"]
  active_uplinks  = ["uplink1", "uplink2"]
  standby_uplinks = ["uplink3", "uplink4"]

  host {
    host_system_id = "${data.vsphere_host.host.0.id}"
    devices        = ["${var.network_interfaces}"]
```

```
  }

  host {
    host_system_id = "${data.vsphere_host.host.1.id}"
    devices        = ["${var.network_interfaces}"]
  }

  host {
    host_system_id = "${data.vsphere_host.host.2.id}"
    devices        = ["${var.network_interfaces}"]
  }
}
```

» Uplink name and count control

The following abridged example below demonstrates how you can manage the number of uplinks, and the name of the uplinks via the `uplinks` parameter.

Note that if you change the uplink naming and count after creating the DVS, you may need to explicitly specify `active_uplinks` and `standby_uplinks` as these values are saved to Terraform state after creation, regardless of being specified in config, and will drift if not modified, causing errors.

```
resource "vsphere_distributed_virtual_switch" "dvs" {
  name          = "terraform-test-dvs"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"

  uplinks         = ["tfup1", "tfup2"]
  active_uplinks  = ["tfup1"]
  standby_uplinks = ["tfup2"]
}
```

NOTE: The default uplink names when a DVS is created are `uplink1` through to `uplink4`, however this default is not guaranteed to be stable and you are encouraged to set your own.

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the distributed virtual switch.
- `datacenter_id` - (Required) The ID of the datacenter where the distributed virtual switch will be created. Forces a new resource if changed.
- `folder` - (Optional) The folder to create the DVS in. Forces a new resource if changed.
- `description` - (Optional) A detailed description for the DVS.

- `contact_name` - (Optional) The name of the person who is responsible for the DVS.
- `contact_detail` - (Optional) The detailed contact information for the person who is responsible for the DVS.
- `ipv4_address` - (Optional) An IPv4 address to identify the switch. This is mostly useful when used with the Netflow arguments found below.
- `lacp_api_version` - (Optional) The Link Aggregation Control Protocol group version to use with the switch. Possible values are `singleLag` and `multipleLag`.
- `link_discovery_operation` - (Optional) Whether to `advertise` or `listen` for link discovery traffic.
- `link_discovery_protocol` - (Optional) The discovery protocol type. Valid types are `cdp` and `lldp`.
- `max_mtu` - (Optional) The maximum transmission unit (MTU) for the virtual switch.
- `multicast_filtering_mode` - (Optional) The multicast filtering mode to use with the switch. Can be one of `legacyFiltering` or `snooping`.
- `version` - (Optional) - The version of the DVS to create. The default is to create the DVS at the latest version supported by the version of vSphere being used. A DVS can be upgraded to another version, but cannot be downgraded.
- `tags` - (Optional) The IDs of any tags to attach to this resource. See here for a reference on how to apply tags.

NOTE: Tagging support requires vCenter 6.0 or higher.

- `custom_attributes` - (Optional) Map of custom attribute ids to attribute value strings to set for virtual switch. See here for a reference on how to set values for custom attributes.

NOTE: Custom attributes are unsupported on direct ESXi connections and require vCenter.

» Uplink arguments

- `uplinks` - (Optional) A list of strings that uniquely identifies the names of the uplinks on the DVS across hosts. The number of items in this list controls the number of uplinks that exist on the DVS, in addition to the names. See here for an example on how to use this option.

» Host management arguments

- `host` - (Optional) Use the `host` sub-resource to declare a host specification. The options are:
  - `host_system_id` - (Required) The host system ID of the host to add to the DVS.

- `devices` - (Required) The list of NIC devices to map to uplinks on the DVS, added in order they are specified.

» Netflow arguments

The following options control settings that you can use to configure Netflow on the DVS:

- `netflow_active_flow_timeout` - (Optional) The number of seconds after which active flows are forced to be exported to the collector. Allowed range is `60` to `3600`. Default: `60`.
- `netflow_collector_ip_address` - (Optional) IP address for the Netflow collector, using IPv4 or IPv6. IPv6 is supported in vSphere Distributed Switch Version 6.0 or later. Must be set before Netflow can be enabled.
- `netflow_collector_port` - (Optional) Port for the Netflow collector. This must be set before Netflow can be enabled.
- `netflow_idle_flow_timeout` - (Optional) The number of seconds after which idle flows are forced to be exported to the collector. Allowed range is `10` to `600`. Default: `15`.
- `netflow_internal_flows_only` - (Optional) Whether to limit analysis to traffic that has both source and destination served by the same host. Default: `false`.
- `netflow_observation_domain_id` - (Optional) The observation domain ID for the Netflow collector.
- `netflow_sampling_rate` - (Optional) The ratio of total number of packets to the number of packets analyzed. The default is `0`, which indicates that the switch should analyze all packets. The maximum value is `1000`, which indicates an analysis rate of 0.001%.

» Network I/O control arguments

The following arguments manage network I/O control. Network I/O control (also known as network resource control) can be used to set up advanced traffic shaping for the DVS, allowing control of various classes of traffic in a fashion similar to how resource pools work for virtual machines. Configuration of network I/O control is also a requirement for the use of network resource pools, if their use is so desired.

» General network I/O control arguments

- `network_resource_control_enabled` - (Optional) Set to `true` to enable network I/O control. Default: `false`.
- `network_resource_control_version` - (Optional) The version of network I/O control to use. Can be one of `version2` or `version3`. Default: `version2`.

» Network I/O control traffic classes

There are currently 9 traffic classes that can be used for network I/O control - they are below.

Each of these classes has 4 options that can be tuned that are discussed in the next section.

Type

Class Name

Fault Tolerance (FT) Traffic

`faulttolerance`

vSphere Replication (VR) Traffic

`hbr`

iSCSI Traffic

`iscsi`

Management Traffic

`management`

NFS Traffic

`nfs`

vSphere Data Protection

`vdp`

Virtual Machine Traffic

`virtualmachine`

vMotion Traffic

`vmotion`

VSAN Traffic

`vsan`

» Traffic class resource options

There are 4 traffic resource options for each class, prefixed with the name of the traffic classes seen above.

For example, to set the traffic class resource options for virtual machine traffic, see the example below:

```
resource "vsphere_distributed_virtual_switch" "dvs" {
  ...
  virtualmachine_share_level      = "custom"
  virtualmachine_share_count      = 150
  virtualmachine_maximum_mbit     = 200
  virtualmachine_reservation_mbit = 20
}
```

The options are:

- `share_level` - (Optional) A pre-defined share level that can be assigned to this resource class. Can be one of `low`, `normal`, `high`, or `custom`.
- `share_count` - (Optional) The number of shares for a custom level. This is ignored if `share_level` is not `custom`.
- `maximum_mbit` - (Optional) The maximum amount of bandwidth allowed for this traffic class in Mbits/sec.
- `reservation_mbit` - (Optional) The guaranteed amount of bandwidth for this traffic class in Mbits/sec.

» Default port group policy arguments

The following arguments are shared with the `vsphere_distributed_port_group` resource. Setting them here defines a default policy here that will be inherited by other port groups on this switch that do not have these values otherwise overridden. Not defining these options in a DVS will infer defaults that can be seen in the Terraform state after the initial apply.

Of particular note to a DVS are the HA policy options, which is where the `active_uplinks` and `standby_uplinks` options are controlled, allowing the ability to create a NIC failover policy that applies to the entire DVS and all portgroups within it that don't override the policy.

» VLAN options

The following options control the VLAN behaviour of the port groups the port policy applies to. One one of these 3 options may be set:

- `vlan` - (Optional) The member VLAN for the ports this policy applies to. A value of `0` means no VLAN.
- `vlan_range` - (Optional) Used to denote VLAN trunking. Use the `min_vlan` and `max_vlan` sub-arguments to define the tagged VLAN range. Multiple `vlan_range` definitions are allowed, but they must not overlap. Example below:

```
resource "vsphere_distributed_virtual_switch" "dvs" {
  ...
  vlan_range {
```

```
    min_vlan = 1
    max_vlan = 1000
  }
  vlan_range {
    min_vlan = 2000
    max_vlan = 4094
  }
}
```

- `port_private_secondary_vlan_id` - (Optional) Used to define a secondary VLAN ID when using private VLANs.

» HA policy options

The following options control HA policy for ports that this policy applies to:

- `active_uplinks` - (Optional) A list of active uplinks to be used in load balancing. These uplinks need to match the definitions in the `uplinks` DVS argument. See here for more details.
- `standby_uplinks` - (Optional) A list of standby uplinks to be used in failover. These uplinks need to match the definitions in the `uplinks` DVS argument. See here for more details.
- `check_beacon` - (Optional) Enables beacon probing as an additional measure to detect NIC failure.

NOTE: VMware recommends using a minimum of 3 NICs when using beacon probing.

- `failback` - (Optional) If `true`, the teaming policy will re-activate failed uplinks higher in precedence when they come back up.
- `notify_switches` - (Optional) If `true`, the teaming policy will notify the broadcast network of an uplink failover, triggering cache updates.
- `teaming_policy` - (Optional) The uplink teaming policy. Can be one of `loadbalance_ip`, `loadbalance_srcmac`, `loadbalance_srcid`, or `failover_explicit`.

» LACP options

The following options allow the use of LACP for NIC teaming for ports that this policy applies to.

NOTE: These options are ignored for non-uplink port groups and hence are only useful at the DVS level.

- `lacp_enabled` - (Optional) Enables LACP for the ports that this policy applies to.
- `lacp_mode` - (Optional) The LACP mode. Can be one of `active` or `passive`.

» Security options

The following options control security settings for the ports that this policy applies to:

- `allow_forged_transmits` - (Optional) Controls whether or not a virtual network adapter is allowed to send network traffic with a different MAC address than that of its own.
- `allow_mac_changes` - (Optional) Controls whether or not the Media Access Control (MAC) address can be changed.
- `allow_promiscuous` - (Optional) Enable promiscuous mode on the network. This flag indicates whether or not all traffic is seen on a given port.

» Traffic shaping options

The following options control traffic shaping settings for the ports that this policy applies to:

- `ingress_shaping_enabled` - (Optional) `true` if the traffic shaper is enabled on the port for ingress traffic.
- `ingress_shaping_average_bandwidth` - (Optional) The average bandwidth in bits per second if ingress traffic shaping is enabled on the port.
- `ingress_shaping_peak_bandwidth` - (Optional) The peak bandwidth during bursts in bits per second if ingress traffic shaping is enabled on the port.
- `ingress_shaping_burst_size` - (Optional) The maximum burst size allowed in bytes if ingress traffic shaping is enabled on the port.
- `egress_shaping_enabled` - (Optional) `true` if the traffic shaper is enabled on the port for egress traffic.
- `egress_shaping_average_bandwidth` - (Optional) The average bandwidth in bits per second if egress traffic shaping is enabled on the port.
- `egress_shaping_peak_bandwidth` - (Optional) The peak bandwidth during bursts in bits per second if egress traffic shaping is enabled on the port.
- `egress_shaping_burst_size` - (Optional) The maximum burst size allowed in bytes if egress traffic shaping is enabled on the port.

» Miscellaneous options

The following are some general options that also affect ports that this policy applies to:

- `block_all_ports` - (Optional) Shuts down all ports in the port groups that this policy applies to, effectively blocking all network access to connected virtual devices.

- `netflow_enabled` - (Optional) Enables Netflow on all ports that this policy applies to.
- `tx_uplink` - (Optional) Forward all traffic transmitted by ports for which this policy applies to its DVS uplinks.
- `directpath_gen2_allowed` - (Optional) Allow VMDirectPath Gen2 for the ports for which this policy applies to.

» Attribute Reference

The following attributes are exported:

- `id`: The UUID of the created DVS.
- `config_version`: The current version of the DVS configuration, incremented by subsequent updates to the DVS.

» Importing

An existing DVS can be imported into this resource via the path to the DVS, via the following command:

```
terraform import vsphere_distributed_virtual_switch.dvs /dc1/network/dvs
```

The above would import the DVS named `dvs` that is located in the `dc1` datacenter.

» vsphere_host_port_group

The `vsphere_host_port_group` resource can be used to manage vSphere standard port groups on an ESXi host. These port groups are connected to standard virtual switches, which can be managed by the `vsphere_host_virtual_switch` resource.

For an overview on vSphere networking concepts, see this page.

» Example Usages

Create a virtual switch and bind a port group to it:

```
data "vsphere_datacenter" "datacenter" {
  name = "dc1"
}

data "vsphere_host" "esxi_host" {
  name          = "esxi1"
```

```
  datacenter_id = "${data.vsphere_datacenter.datacenter.id}"
}

resource "vsphere_host_virtual_switch" "switch" {
  name           = "vSwitchTerraformTest"
  host_system_id = "${data.vsphere_host.esxi_host.id}"

  network_adapters = ["vmnic0", "vmnic1"]

  active_nics  = ["vmnic0"]
  standby_nics = ["vmnic1"]
}

resource "vsphere_host_port_group" "pg" {
  name                = "PGTerraformTest"
  host_system_id      = "${data.vsphere_host.esxi_host.id}"
  virtual_switch_name = "${vsphere_host_virtual_switch.switch.name}"
}
```

Create a port group with VLAN set and some overrides:

This example sets the trunk mode VLAN (`4095`, which passes through all tags) and sets `allow_promiscuous` to ensure that all traffic is seen on the port. The latter setting overrides the implicit default of `false` set on the virtual switch.

```
data "vsphere_datacenter" "datacenter" {
  name = "dc1"
}

data "vsphere_host" "esxi_host" {
  name          = "esxi1"
  datacenter_id = "${data.vsphere_datacenter.datacenter.id}"
}

resource "vsphere_host_virtual_switch" "switch" {
  name           = "vSwitchTerraformTest"
  host_system_id = "${data.vsphere_host.esxi_host.id}"

  network_adapters = ["vmnic0", "vmnic1"]

  active_nics  = ["vmnic0"]
  standby_nics = ["vmnic1"]
}

resource "vsphere_host_port_group" "pg" {
  name                = "PGTerraformTest"
  host_system_id      = "${data.vsphere_host.esxi_host.id}"
```

```
  virtual_switch_name = "${vsphere_host_virtual_switch.switch.name}"

  vlan_id = 4095

  allow_promiscuous = true
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the port group. Forces a new resource if changed.
- `host_system_id` - (Required) The managed object ID of the host to set the port group up on. Forces a new resource if changed.
- `virtual_switch_name` - (Required) The name of the virtual switch to bind this port group to. Forces a new resource if changed.
- `vlan_id` - (Optional) The VLAN ID/trunk mode for this port group. An ID of `0` denotes no tagging, an ID of `1-4094` tags with the specific ID, and an ID of `4095` enables trunk mode, allowing the guest to manage its own tagging. Default: `0`.

» Policy Options

In addition to the above options, you can configure any policy option that is available under the `vsphere_host_virtual_switch` policy options section. Any policy option that is not set is **inherited** from the virtual switch, its options propagating to the port group.

See the link for a full list of options that can be set.

» Attribute Reference

The following attributes are exported:

- `id` - An ID unique to Terraform for this port group. The convention is a prefix, the host system ID, and the port group name. An example would be `tf-HostPortGroup:host-10:PGTerraformTest`.
- `computed_policy` - A map with a full set of the policy options computed from defaults and overrides, explaining the effective policy for this port group.
- `key` - The key for this port group as returned from the vSphere API.
- `ports` - A list of ports that currently exist and are used on this port group.

» vsphere_host_virtual_switch

The `vsphere_host_virtual_switch` resource can be used to manage vSphere standard switches on an ESXi host. These switches can be used as a backing for standard port groups, which can be managed by the `vsphere_host_port_group` resource.

For an overview on vSphere networking concepts, see this page.

» Example Usages

Create a virtual switch with one active and one standby NIC:

```
data "vsphere_datacenter" "datacenter" {
  name = "dc1"
}

data "vsphere_host" "host" {
  name          = "esxi1"
  datacenter_id = "${data.vsphere_datacenter.datacenter.id}"
}

resource "vsphere_host_virtual_switch" "switch" {
  name           = "vSwitchTerraformTest"
  host_system_id = "${data.vsphere_host.host.id}"

  network_adapters = ["vmnic0", "vmnic1"]

  active_nics  = ["vmnic0"]
  standby_nics = ["vmnic1"]
}
```

Create a virtual switch with extra networking policy options:

```
data "vsphere_datacenter" "datacenter" {
  name = "dc1"
}

data "vsphere_host" "host" {
  name          = "esxi1"
  datacenter_id = "${data.vsphere_datacenter.datacenter.id}"
}

resource "vsphere_host_virtual_switch" "switch" {
  name           = "vSwitchTerraformTest"
  host_system_id = "${data.vsphere_host.host.id}"
```

```
  network_adapters = ["vmnic0", "vmnic1"]

  active_nics    = ["vmnic0"]
  standby_nics   = ["vmnic1"]
  teaming_policy = "failover_explicit"

  allow_promiscuous      = false
  allow_forged_transmits = false
  allow_mac_changes      = false

  shaping_enabled           = true
  shaping_average_bandwidth = 50000000
  shaping_peak_bandwidth    = 100000000
  shaping_burst_size        = 1000000000
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the virtual switch. Forces a new resource if changed.
- `host_system_id` - (Required) The managed object ID of the host to set the virtual switch up on. Forces a new resource if changed.
- `mtu` - (Optional) The maximum transmission unit (MTU) for the virtual switch. Default: `1500`.
- `number_of_ports` - (Optional) The number of ports to create with this virtual switch. Default: `128`.

NOTE: Changing the port count requires a reboot of the host. Terraform will not restart the host for you.

» Bridge Options

The following arguments are related to how the virtual switch binds to physical NICs:

- `network_adapters` - (Required) The network interfaces to bind to the bridge.
- `beacon_interval` - (Optional) The interval, in seconds, that a NIC beacon packet is sent out. This can be used with `check_beacon` to offer link failure capability beyond link status only. Default: `1`.
- `link_discovery_operation` - (Optional) Whether to `advertise` or `listen` for link discovery traffic. Default: `listen`.

- `link_discovery_protocol` - (Optional) The discovery protocol type. Valid types are `cpd` and `lldp`. Default: `cdp`.

» Policy Options

The following options relate to how network traffic is handled on this virtual switch. It also controls the NIC failover order. This subset of options is shared with the `vsphere_host_port_group` resource, in which options can be omitted to ensure options are inherited from the switch configuration here.

» NIC Teaming Options

NOTE on NIC failover order: An adapter can be in `active_nics`, `standby_nics`, or neither to flag it as unused. However, virtual switch creation or update operations will fail if a NIC is present in both settings, or if the NIC is not a valid NIC in `network_adapters`.

NOTE: VMware recommends using a minimum of 3 NICs when using beacon probing (configured with `check_beacon`).

- `active_nics` - (Required) The list of active network adapters used for load balancing.
- `standby_nics` - (Required) The list of standby network adapters used for failover.
- `check_beacon` - (Optional) Enable beacon probing - this requires that the `beacon_interval` option has been set in the bridge options. If this is set to `false`, only link status is used to check for failed NICs. Default: `false`.
- `teaming_policy` - (Optional) The network adapter teaming policy. Can be one of `loadbalance_ip`, `loadbalance_srcmac`, `loadbalance_srcid`, or `failover_explicit`. Default: `loadbalance_srcid`.
- `notify_switches` - (Optional) If set to `true`, the teaming policy will notify the broadcast network of a NIC failover, triggering cache updates. Default: `true`.
- `failback` - (Optional) If set to `true`, the teaming policy will re-activate failed interfaces higher in precedence when they come back up. Default: `true`.

» Security Policy Options

- `allow_promiscuous` - (Optional) Enable promiscuous mode on the network. This flag indicates whether or not all traffic is seen on a given port. Default: `false`.
- `allow_forged_transmits` - (Optional) Controls whether or not the virtual network adapter is allowed to send network traffic with a different MAC address than that of its own. Default: `true`.

- `allow_mac_changes` - (Optional) Controls whether or not the Media Access Control (MAC) address can be changed. Default: `true`.

» Traffic Shaping Options

- `shaping_enabled` - (Optional) Set to `true` to enable the traffic shaper for ports managed by this virtual switch. Default: `false`.
- `shaping_average_bandwidth` - (Optional) The average bandwidth in bits per second if traffic shaping is enabled. Default: `0`
- `shaping_peak_bandwidth` - (Optional) The peak bandwidth during bursts in bits per second if traffic shaping is enabled. Default: `0`
- `shaping_burst_size` - (Optional) The maximum burst size allowed in bytes if shaping is enabled. Default: `0`

» Attribute Reference

The only exported attribute, other than the attributes above, is the `id` of the resource. This is set to an ID value unique to Terraform - the convention is a prefix, the host system ID, and the virtual switch name. An example would be `tf-HostVirtualSwitch:host-10:vSwitchTerraformTest`.

» vsphere_file

The `vsphere_file` resource can be used to upload files (such as virtual disk files) from the host machine that Terraform is running on to a target datastore. The resource can also be used to copy files between datastores, or from one location to another on the same datastore.

Updates to destination parameters such as `datacenter`, `datastore`, or `destination_file` will move the managed file a new destination based on the values of the new settings. If any source parameter is changed, such as `source_datastore`, `source_datacenter` or `source_file`), the resource will be re-created. Depending on if destination parameters are being changed as well, this may result in the destination file either being overwritten or deleted at the old location.

» Example Usages

» Uploading a file

```
resource "vsphere_file" "ubuntu_disk_upload" {
  datacenter       = "my_datacenter"
  datastore        = "local"
```

```
  source_file      = "/home/ubuntu/my_disks/custom_ubuntu.vmdk"
  destination_file = "/my_path/disks/custom_ubuntu.vmdk"
}
```

» Copying a file

```
resource "vsphere_file" "ubuntu_disk_copy" {
  source_datacenter = "my_datacenter"
  datacenter        = "my_datacenter"
  source_datastore  = "local"
  datastore         = "local"
  source_file       = "/my_path/disks/custom_ubuntu.vmdk"
  destination_file  = "/my_path/custom_ubuntu_id.vmdk"
}
```

» Argument Reference

If `source_datacenter` and `source_datastore` are not provided, the file resource will upload the file from the host that Terraform is running on. If either `source_datacenter` or `source_datastore` are provided, the resource will copy from within specified locations in vSphere.

The following arguments are supported:

- `source_file` - (Required) The path to the file being uploaded from the Terraform host to vSphere or copied within vSphere. Forces a new resource if changed.
- `destination_file` - (Required) The path to where the file should be uploaded or copied to on vSphere.
- `source_datacenter` - (Optional) The name of a datacenter in which the file will be copied from. Forces a new resource if changed.
- `datacenter` - (Optional) The name of a datacenter in which the file will be uploaded to.
- `source_datastore` - (Optional) The name of the datastore in which file will be copied from. Forces a new resource if changed.
- `datastore` - (Required) The name of the datastore in which to upload the file to.
- `create_directories` - (Optional) Create directories in `destination_file` path parameter if any missing for copy operation.

NOTE: Any directory created as part of the operation when `create_directories` is enabled will not be deleted when the resource is destroyed.

» vsphere_nas_datastore

The `vsphere_nas_datastore` resource can be used to create and manage NAS datastores on an ESXi host or a set of hosts. The resource supports mounting NFS v3 and v4.1 shares to be used as datastores.

NOTE: Unlike `vsphere_vmfs_datastore`, a NAS datastore is only mounted on the hosts you choose to mount it on. To mount on multiple hosts, you must specify each host that you want to add in the `host_system_ids` argument.

» Example Usage

The following example would set up a NFS v3 share on 3 hosts connected through vCenter in the same datacenter - `esxi1`, `esxi2`, and `esxi3`. The remote host is named `nfs` and has `/export/terraform-test` exported.

```
variable "hosts" {
  default = [
    "esxi1",
    "esxi2",
    "esxi3",
  ]
}

data "vsphere_datacenter" "datacenter" {}

data "vsphere_host" "esxi_hosts" {
  count         = "${length(var.hosts)}"
  name          = "${var.hosts[count.index]}"
  datacenter_id = "${data.vsphere_datacenter.datacenter.id}"
}

resource "vsphere_nas_datastore" "datastore" {
  name            = "terraform-test"
  host_system_ids = ["${data.vsphere_host.esxi_hosts.*.id}"]

  type         = "NFS"
  remote_hosts = ["nfs"]
  remote_path  = "/export/terraform-test"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the datastore. Forces a new resource if changed.
- `host_system_ids` - (Required) The managed object IDs of the hosts to mount the datastore on.
- `type` - (Optional) The type of NAS volume. Can be one of `NFS` (to denote v3) or `NFS41` (to denote NFS v4.1). Default: `NFS`. Forces a new resource if changed.
- `remote_hosts` - (Required) The hostnames or IP addresses of the remote server or servers. Only one element should be present for NFS v3 but multiple can be present for NFS v4.1. Forces a new resource if changed.
- `remote_path` - (Required) The remote path of the mount point. Forces a new resource if changed.
- `access_mode` - (Optional) Access mode for the mount point. Can be one of `readOnly` or `readWrite`. Note that `readWrite` does not necessarily mean that the datastore will be read-write depending on the permissions of the actual share. Default: `readWrite`. Forces a new resource if changed.
- `security_type` - (Optional) The security type to use when using NFS v4.1. Can be one of `AUTH_SYS`, `SEC_KRB5`, or `SEC_KRB5I`. Forces a new resource if changed.
- `folder` - (Optional) The relative path to a folder to put this datastore in. This is a path relative to the datacenter you are deploying the datastore to. Example: for the `dc1` datacenter, and a provided `folder` of `foo/bar`, Terraform will place a datastore named `terraform-test` in a datastore folder located at `/dc1/datastore/foo/bar`, with the final inventory path being `/dc1/datastore/foo/bar/terraform-test`.
- `tags` - (Optional) The IDs of any tags to attach to this resource. See here for a reference on how to apply tags.

NOTE: Tagging support is unsupported on direct ESXi connections and requires vCenter 6.0 or higher.

- `custom_attributes` - (Optional) Map of custom attribute ids to attribute value strings to set on datasource resource. See here for a reference on how to set values for custom attributes.

NOTE: Custom attributes are unsupported on direct ESXi connections and require vCenter.

» Attribute Reference

The following attributes are exported:

- `id` - The managed object reference ID of the datastore.
- `accessible` - The connectivity status of the datastore. If this is `false`, some other computed attributes may be out of date.
- `capacity` - Maximum capacity of the datastore, in megabytes.
- `free_space` - Available space of this datastore, in megabytes.

- `maintenance_mode` - The current maintenance mode state of the datastore.
- `multiple_host_access` - If `true`, more than one host in the datacenter has been configured with access to the datastore.
- `uncommitted_space` - Total additional storage space, in megabytes, potentially used by all virtual machines on this datastore.
- `url` - The unique locator for the datastore.
- `protocol_endpoint` - Indicates that this NAS volume is a protocol endpoint. This field is only populated if the host supports virtual datastores.

» Importing

An existing NAS datastore can be imported into this resource via its managed object ID, via the following command:

```
terraform import vsphere_nas_datastore.datastore datastore-123
```

You need a tool like `govc` that can display managed object IDs.

In the case of govc, you can locate a managed object ID from an inventory path by doing the following:

```
$ govc ls -i /dc/datastore/terraform-test
Datastore:datastore-123
```

» vsphere_vmfs_datastore

The `vsphere_vmfs_datastore` resource can be used to create and manage VMFS datastores on an ESXi host or a set of hosts. The resource supports using any SCSI device that can generally be used in a datastore, such as local disks, or disks presented to a host or multiple hosts over Fibre Channel or iSCSI. Devices can be specified manually, or discovered using the `vsphere_vmfs_disks` data source.

» Auto-Mounting of Datastores Within vCenter

Note that the current behaviour of this resource will auto-mount any created datastores to any other host within vCenter that has access to the same disk.

Example: You want to create a datastore with a iSCSI LUN that is visible on 3 hosts in a single vSphere cluster (`esxi1`, `esxi2` and `esxi3`). When you create the datastore on `esxi1`, the datastore will be automatically mounted on `esxi2` and `esxi3`, without the need to configure the resource on either of those two hosts.

Future versions of this resource may allow you to control the hosts that a datastore is mounted to, but currently, this automatic behaviour cannot be changed, so keep this in mind when writing your configurations and deploying your disks.

» Increasing Datastore Size

To increase the size of a datastore, you must add additional disks to the `disks` attribute. Expanding the size of a datastore by increasing the size of an already provisioned disk is currently not supported (but may be in future versions of this resource).

NOTE: You cannot decrease the size of a datastore. If the resource detects disks removed from the configuration, Terraform will give an error. To reduce the size of the datastore, the resource needs to be re-created - run `terraform taint` to taint the resource so it can be re-created.

» Example Usage

» Addition of local disks on a single host

The following example uses the default datacenter and default host to add a datastore with local disks to a single ESXi server.

NOTE: There are some situations where datastore creation will not work when working through vCenter (usually when trying to create a datastore on a single host with local disks). If you experience trouble creating the datastore you need through vCenter, break the datstore off into a different configuration and deploy it using the ESXi server as the provider endpoint, using a similar configuration to what is below.

```
data "vsphere_datacenter" "datacenter" {}

data "vsphere_host" "esxi_host" {
  datacenter_id = "${data.vsphere_datacenter.datacenter.id}"
}

resource "vsphere_vmfs_datastore" "datastore" {
  name           = "terraform-test"
  host_system_id = "${data.vsphere_host.esxi_host.id}"

  disks = [
    "mpx.vmhba1:C0:T1:L0",
    "mpx.vmhba1:C0:T2:L0",
    "mpx.vmhba1:C0:T2:L0",
  ]
```

```
}
```

» Auto-detection of disks via `vsphere_vmfs_disks`

The following example makes use of the `vsphere_vmfs_disks` data source to auto-detect exported iSCSI LUNS matching a certain NAA vendor ID (in this case, LUNs exported from a NetApp). These discovered disks are then loaded into `vsphere_vmfs_datastore`. The datastore is also placed in the `datastore-folder` folder afterwards.

```
data "vsphere_datacenter" "datacenter" {
  name = "dc1"
}

data "vsphere_host" "esxi_host" {
  name          = "esxi1"
  datacenter_id = "${data.vsphere_datacenter.datacenter.id}"
}

data "vsphere_vmfs_disks" "available" {
  host_system_id = "${data.vsphere_host.esxi_host.id}"
  rescan         = true
  filter         = "naa.60a98000"
}

resource "vsphere_vmfs_datastore" "datastore" {
  name           = "terraform-test"
  host_system_id = "${data.vsphere_host.esxi_host.id}"
  folder         = "datastore-folder"

  disks = ["${data.vsphere_vmfs_disks.available.disks}"]
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the datastore. Forces a new resource if changed.
- `host_system_id` - (Required) The managed object ID of the host to set the datastore up on. Note that this is not necessarily the only host that the datastore will be set up on - see here for more info. Forces a new resource if changed.
- `disks` - (Required) The disks to use with the datastore.

- `folder` - (Optional) The relative path to a folder to put this datastore in. This is a path relative to the datacenter you are deploying the datastore to. Example: for the `dc1` datacenter, and a provided `folder` of `foo/bar`, Terraform will place a datastore named `terraform-test` in a datastore folder located at `/dc1/datastore/foo/bar`, with the final inventory path being `/dc1/datastore/foo/bar/terraform-test`.
- `tags` - (Optional) The IDs of any tags to attach to this resource. See here for a reference on how to apply tags.

NOTE: Tagging support is unsupported on direct ESXi connections and requires vCenter 6.0 or higher.

- `custom_attributes` (Optional) Map of custom attribute ids to attribute value string to set on datastore resource. See here for a reference on how to set values for custom attributes.

NOTE: Custom attributes are unsupported on direct ESXi connections and require vCenter.

» Attribute Reference

The following attributes are exported:

- `id` - The managed object reference ID of the datastore.
- `accessible` - The connectivity status of the datastore. If this is `false`, some other computed attributes may be out of date.
- `capacity` - Maximum capacity of the datastore, in megabytes.
- `free_space` - Available space of this datastore, in megabytes.
- `maintenance_mode` - The current maintenance mode state of the datastore.
- `multiple_host_access` - If `true`, more than one host in the datacenter has been configured with access to the datastore.
- `uncommitted_space` - Total additional storage space, in megabytes, potentially used by all virtual machines on this datastore.
- `url` - The unique locator for the datastore.

» Importing

An existing VMFS datastore can be imported into this resource via its managed object ID, via the command below. You also need the host system ID.

```
terraform import vsphere_vmfs_datastore.datastore datastore-123:host-10
```

You need a tool like `govc` that can display managed object IDs.

In the case of govc, you can locate a managed object ID from an inventory path by doing the following:

```
$ govc ls -i /dc/datastore/terraform-test
Datastore:datastore-123
```

To locate host IDs, it might be a good idea to supply the `-l` flag as well so that you can line up the names with the IDs:

```
$ govc ls -l -i /dc/host/cluster1
ResourcePool:resgroup-10 /dc/host/cluster1/Resources
HostSystem:host-10 /dc/host/cluster1/esxi1
HostSystem:host-11 /dc/host/cluster1/esxi2
HostSystem:host-12 /dc/host/cluster1/esxi3
```

» vsphere_virtual_disk

The `vsphere_virtual_disk` resource can be used to create virtual disks outside of any given `vsphere_virtual_machine` resource. These disks can be attached to a virtual machine by creating a disk sub-resource with the `attach` parameter.

» Example Usage

```
resource "vsphere_virtual_disk" "myDisk" {
  size         = 2
  vmdk_path    = "myDisk.vmdk"
  datacenter   = "Datacenter"
  datastore    = "local"
  type         = "thin"
}
```

» Argument Reference

The following arguments are supported:

NOTE: All fields in the `vsphere_virtual_disk` resource are currently immutable and force a new resource if changed.

- `vmdk_path` - (Required) The path, including filename, of the virtual disk to be created. This needs to end in `.vmdk`.
- `datastore` - (Required) The name of the datastore in which to create the disk.
- `size` - (Required) Size of the disk (in GB).
- `datacenter` - (Optional) The name of the datacenter in which to create the disk. Can be omitted when when ESXi or if there is only one datacenter in your infrastructure.

- `type` - (Optional) The type of disk to create. Can be one of `eagerZeroedThick`, `lazy`, or `thin`. Default: `eagerZeroedThick`. For information on what each kind of disk provisioning policy means, click here.
- `adapter_type` - (Optional) The adapter type for this virtual disk. Can be one of `ide`, `lsiLogic`, or `busLogic`. Default: `lsiLogic`.

NOTE: `adapter_type` is **deprecated**: it does not dictate the type of controller that the virtual disk will be attached to on the virtual machine. Please see the `scsi_type` parameter in the `vsphere_virtual_machine` resource for information on how to control disk controller types. This parameter will be removed in future versions of the vSphere provider.

» vsphere_virtual_machine

The `vsphere_virtual_machine` resource can be used to manage the complex lifecycle of a virtual machine. It supports management of disk, network interface, and CDROM devices, creation from scratch or cloning from template, and migration through both host and storage vMotion.

For more details on working with virtual machines in vSphere, see this page.

» About Working with Virtual Machines in Terraform

A high degree of control and flexibility is afforded to a vSphere user when it comes to how to configure, deploy, and manage virtual machines - much more control than given in a traditional cloud provider. As such, Terraform has to make some decisions on how to manage the virtual machines it creates and manages. This section documents things you need to know about your virtual machine configuration that you should consider when setting up virtual machines, creating templates to clone from, or migrating from previous versions of this resource.

» Disks

The `vsphere_virtual_machine` resource currently only supports standard VMDK-backed virtual disks - it does not support other special kinds of disk devices like RDM disks.

Disks are managed by an arbitrary label supplied to the `label` attribute of a `disk` sub-resource. This is separate from the automatic naming that vSphere picks for you when creating a virtual machine. Control over a virtual disk's

name is not supported unless you are attaching an external disk with the `attach` attribute.

Virtual disks can be SCSI disks only. The SCSI controllers managed by Terraform can vary, depending on the value supplied to `scsi_controller_count`. This also dictates the controllers that are checked when looking for disks during a cloning process. By default, this value is `1`, meaning that you can have up to 15 disks configured on a virtual machine. These are all configured with the controller type defined by the `scsi_type` setting. If you are cloning from a template, devices will be added or re-configured as necessary.

When cloning from a template, you must specify disks of either the same or greater size than the disks in the source template when creating a traditional clone, or exactly the same size when cloning from snapshot (also known as a linked clone). For more details, see the section on creating a virtual machine from a template.

A maximum of 60 virtual disks can be configured when the `scsi_controller_count` setting is configured to its maximum of `4` controllers. See the disk options section for more details.

» Customization and network waiters

Terraform waits during various parts of a virtual machine deployment to ensure that it is in a correct expected state before proceeding. These happen when a VM is created, or also when it's updated, depending on the waiter.

Two waiters of note are:

- **The customization waiter:** This waiter watches events in vSphere to monitor when customization on a virtual machine completes during VM creation. Depending on your vSphere or VM configuration it may be necessary to change the timeout or turn it off. This can be controlled by the `timeout` setting in the customization settings block.
- **The network waiter:** This waiter waits for a *routeable* interface to show up on a guest virtual machine close to the end of both VM creation and update. This waiter is necessary to ensure that correct IP information gets reported to the guest virtual machine, mainly to facilitate the availability of a valid, routeable default IP address for any provisioners. This option can be managed or turned off via the `wait_for_guest_net_timeout` top-level setting.

» Migrating from a previous version of this resource

NOTE: This section only applies to versions of this resource available in versions v0.4.2 of this provider or earlier.

The path for migrating to the current version of this resource is very similar to the import path, with the exception that the `terraform import` command does not need to be run. See that section for details on what is required before you run `terraform plan` on a state that requires migration.

A successful import usually only results in a diff where configured disks transition their `keep_on_remove` settings from `true` to `false`. This operation does not perform any virtual machine operations and is safe to run while the virtual machine is running.

» Example Usage

» Creating a virtual machine from scratch

The following block contains all that is necessary to create a new virtual machine, with a single disk and network interface.

The resource makes use of the following data sources to do its job: `vsphere_datacenter` to locate the datacenter, `vsphere_datastore` to locate the default datastore to put the virtual machine in, `vsphere_resource_pool` to locate a resource pool located in a cluster or standalone host, and `vsphere_network` to locate a network.

```
data "vsphere_datacenter" "dc" {
  name = "dc1"
}

data "vsphere_datastore" "datastore" {
  name          = "datastore1"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_resource_pool" "pool" {
  name          = "cluster1/Resources"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_network" "network" {
  name          = "public"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

resource "vsphere_virtual_machine" "vm" {
  name             = "terraform-test"
  resource_pool_id = "${data.vsphere_resource_pool.pool.id}"
  datastore_id     = "${data.vsphere_datastore.datastore.id}"
```

```
  num_cpus = 2
  memory   = 1024
  guest_id = "other3xLinux64Guest"

  network_interface {
    network_id = "${data.vsphere_network.network.id}"
  }

  disk {
    label = "disk0"
    size  = 20
  }
}
```

» Cloning and customization example

Building on the above example, the below configuration creates a VM by cloning it from a template, fetched via the `vsphere_virtual_machine` data source. This allows us to locate the UUID of the template we want to clone, along with settings for network interface type, SCSI bus type (especially important on Windows machines), and disk attributes.

```
data "vsphere_datacenter" "dc" {
  name = "dc1"
}

data "vsphere_datastore" "datastore" {
  name          = "datastore1"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_resource_pool" "pool" {
  name          = "cluster1/Resources"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_network" "network" {
  name          = "public"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_virtual_machine" "template" {
  name          = "ubuntu-16.04"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
```

```
}

resource "vsphere_virtual_machine" "vm" {
  name             = "terraform-test"
  resource_pool_id = "${data.vsphere_resource_pool.pool.id}"
  datastore_id     = "${data.vsphere_datastore.datastore.id}"

  num_cpus = 2
  memory   = 1024
  guest_id = "${data.vsphere_virtual_machine.template.guest_id}"

  scsi_type = "${data.vsphere_virtual_machine.template.scsi_type}"

  network_interface {
    network_id   = "${data.vsphere_network.network.id}"
    adapter_type = "${data.vsphere_virtual_machine.template.network_interface_types[0]}"
  }

  disk {
    label            = "disk0"
    size             = "${data.vsphere_virtual_machine.template.disks.0.size}"
    eagerly_scrub    = "${data.vsphere_virtual_machine.template.disks.0.eagerly_scrub}"
    thin_provisioned = "${data.vsphere_virtual_machine.template.disks.0.thin_provisioned}"
  }

  clone {
    template_uuid = "${data.vsphere_virtual_machine.template.id}"

    customize {
      linux_options {
        host_name = "terraform-test"
        domain    = "test.internal"
      }

      network_interface {
        ipv4_address = "10.0.0.10"
        ipv4_netmask = 24
      }

      ipv4_gateway = "10.0.0.1"
    }
  }
}
```

» Cloning from an OVF/OVA-created template with vApp properties

This alternate example details how to clone a VM from a template that came from an OVF/OVA file. This leverages the resource's vApp properties capabilities to set appropriate keys that control various configuration settings on the virtual machine or virtual appliance. In this scenario, using `customize` is not recommended as the functionality has tendency to overlap.

NOTE: Neither the `vsphere_virtual_machine` resource nor the vSphere provider supports importing of OVA or OVF files as this is a workflow that is fundamentally not the domain of Terraform. The supported path for deployment in Terraform is to first import the virtual machine into a template that has not been powered on, and then clone from that template. This can be accomplished with Packer, govc's `import.ovf` and `import.ova` subcommands, or ovftool.

```
data "vsphere_datacenter" "dc" {
  name = "dc1"
}

data "vsphere_datastore" "datastore" {
  name          = "datastore1"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_resource_pool" "pool" {
  name          = "cluster1/Resources"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_network" "network" {
  name          = "public"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_virtual_machine" "tempate_from_ovf" {
  name          = "template_from_ovf"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

resource "vsphere_virtual_machine" "vm" {
  name             = "terraform-test"
  resource_pool_id = "${data.vsphere_resource_pool.pool.id}"
  datastore_id     = "${data.vsphere_datastore.datastore.id}"

  num_cpus = 2
```

```
  memory   = 1024
  guest_id = "${data.vsphere_virtual_machine.template.guest_id}"

  scsi_type = "${data.vsphere_virtual_machine.template.scsi_type}"

  network_interface {
    network_id   = "${data.vsphere_network.network.id}"
    adapter_type = "${data.vsphere_virtual_machine.template.network_interface_types[0]}"
  }

  disk {
    name             = "disk0"
    size             = "${data.vsphere_virtual_machine.template.disks.0.size}"
    eagerly_scrub    = "${data.vsphere_virtual_machine.template.disks.0.eagerly_scrub}"
    thin_provisioned = "${data.vsphere_virtual_machine.template.disks.0.thin_provisioned}"
  }

  clone {
    template_uuid = "${data.vsphere_virtual_machine.template_from_ovf.id}"
  }

  vapp {
    properties {
      "guestinfo.hostname"                       = "terraform-test.foobar.local"
      "guestinfo.interface.0.name"               = "ens192"
      "guestinfo.interface.0.ip.0.address"       = "10.0.0.100/24"
      "guestinfo.interface.0.route.0.gateway"    = "10.0.0.1"
      "guestinfo.interface.0.route.0.destination" = "0.0.0.0/0"
      "guestinfo.dns.server.0"                   = "10.0.0.10"
    }
  }
}
```

» Argument Reference

The following arguments are supported:

» General options

The following options are general virtual machine and Terraform workflow options:

- `name` - (Required) The name of the virtual machine.

- `resource_pool_id` - (Required) The managed object reference ID of the resource pool to put this virtual machine in. See the section on virtual machine migration for details on changing this value.

NOTE: All clusters and standalone hosts have a resource pool, even if one has not been explicitly created. For more information, see the section on specifying the root resource pool for a cluster or standalone host in the `vsphere_resource_pool` data source documentation. This resource does not take a cluster or standalone host resource directly.

- `datastore_id` - (Required) The managed object reference ID of the virtual machine's datastore. The virtual machine configuration is placed here, along with any virtual disks that are created where a datastore is not explicitly specified. See the section on virtual machine migration for details on changing this value.
- `folder` - (Optional) The path to the folder to put this virtual machine in, relative to the datacenter that the resource pool is in.
- `host_system_id` - (Optional) An optional managed object reference ID of a host to put this virtual machine on. See the section on virtual machine migration for details on changing this value. If a `host_system_id` is not supplied, vSphere will select a host in the resource pool to place the virtual machine, according to any defaults or DRS policies in place.
- `disk` - (Required) A specification for a virtual disk device on this virtual machine. See disk options below.
- `network_interface` - (Required) A specification for a virtual NIC on this virtual machine. See network interface options below.
- `cdrom` - (Optional) A specification for a CDROM device on this virtual machine. See CDROM options below.
- `clone` - (Optional) When specified, the VM will be created as a clone of a specified template. Optional customization options can be submitted as well. See creating a virtual machine from a template for more details.
- `vapp` - (Optional) Optional vApp configuration. The only sub-key available is `properties`, which is a key/value map of properties for virtual machines imported from OVF or OVA files. See Using vApp properties to supply OVF/OVA configuration for more details.
- `guest_id` - (Optional) The guest ID for the operating system type. For a full list of possible values, see here. Default: `other-64`.
- `alternate_guest_name` - (Optional) The guest name for the operating system when `guest_id` is `other` or `other-64`.
- `annotation` - (Optional) A user-provided description of the virtual machine. The default is no annotation.

- `firmware` - (Optional) The firmware interface to use on the virtual machine. Can be one of `bios` or `EFI`. Default: `bios`.
- `extra_config` - (Optional) Extra configuration data for this virtual machine. Can be used to supply advanced parameters not normally in configuration, such as data for cloud-config (under the guestinfo namespace).

NOTE: Do not use `extra_config` when working with a template imported from OVF or OVA as more than likely your settings will be ignored. Use the `vapp` sub-resource's `properties` section as outlined in Using vApp properties to supply OVF/OVA configuration.

- `scsi_type` - (Optional) The type of SCSI bus this virtual machine will have. Can be one of lsilogic (LSI Logic Parallel), lsilogic-sas (LSI Logic SAS) or pvscsi (VMware Paravirtual). Defualt: `pvscsi`.
- `tags` - (Optional) The IDs of any tags to attach to this resource. See here for a reference on how to apply tags.

NOTE: Tagging support is unsupported on direct ESXi connections and requires vCenter 6.0 or higher.

- `custom_attributes` - (Optional) Map of custom attribute ids to attribute value strings to set for virtual machine. See here for a reference on how to set values for custom attributes.

NOTE: Custom attributes are unsupported on direct ESXi connections and require vCenter.

» CPU and memory options

The following options control CPU and memory settings on the virtual machine:

- `num_cpus` - (Optional) The number of virtual processors to assign to this virtual machine. Default: `1`.
- `num_cores_per_socket` - (Optional) The number of cores to distribute among the CPUs in this virtual machine. If specified, the value supplied to `num_cpus` must be evenly divisible by this value. Default: `1`.
- `cpu_hot_add_enabled` - (Optional) Allow CPUs to be added to this virtual machine while it is running.
- `cpu_hot_remove_enabled` - (Optional) Allow CPUs to be removed to this virtual machine while it is running.
- `memory` - (Optional) The size of the virtual machine's memory, in MB. Default: `1024` (1 GB).
- `memory_hot_add_enabled` - (Optional) Allow memory to be added to this virtual machine while it is running.

NOTE: Certain CPU and memory hot-plug options are not available on every operating system. Check the VMware Guest OS Compatibility Guide first to see what settings your guest operating system is eligible for. In addition, at least

one `terraform apply` must be executed before being able to take advantage of CPU and memory hot-plug settings, so if you want the support, enable it as soon as possible.

» Boot options

The following options control boot settings on the virtual machine:

- `boot_delay` - (Optional) The number of milliseconds to wait before starting the boot sequence. The default is no delay.
- `efi_secure_boot_enabled` - (Optional) When the `firmware` type is set to is `efi`, this enables EFI secure boot. Default: `false`.

NOTE: EFI secure boot is only available on vSphere 6.5 and higher.

- `boot_retry_delay` - (Optional) The number of milliseconds to wait before retrying the boot sequence. This only valid if `boot_retry_enabled` is true. Default: `10000` (10 seconds).
- `boot_retry_enabled` - (Optional) If set to true, a virtual machine that fails to boot will try again after the delay defined in `boot_retry_delay`. Default: `false`.

» VMware Tools options

The following options control VMware tools options on the virtual machine:

- `sync_time_with_host` - (Optional) Enable guest clock synchronization with the host. Requires VMware tools to be installed. Default: `false`.
- `run_tools_scripts_after_power_on` - (Optional) Enable the execution of post-power-on scripts when VMware tools is installed. Default: `true`.
- `run_tools_scripts_after_resume` - (Optional) Enable the execution of post-resume scripts when VMware tools is installed. Default: `true`.
- `run_tools_scripts_before_guest_reboot` - (Optional) Enable the execution of pre-reboot scripts when VMware tools is installed. Default: `false`.
- `run_tools_scripts_before_guest_shutdown` - (Optional) Enable the execution of pre-shutdown scripts when VMware tools is installed. Default: `true`.
- `run_tools_scripts_before_guest_standby` - (Optional) Enable the execution of pre-standby scripts when VMware tools is installed. Default: `true`.

» Resource allocation options

The following options allow control over CPU and memory allocation on the virtual machine. Note that the resource pool that this VM is in may affect

these options.

- `cpu_limit` - (Optional) The maximum amount of CPU (in MHz) that this virtual machine can consume, regardless of available resources. The default is no limit.
- `cpu_reservation` - (Optional) The amount of CPU (in MHz) that this virtual machine is guaranteed. The default is no reservation.
- `cpu_share_level` - (Optional) The allocation level for CPU resources. Can be one of `high`, `low`, `normal`, or `custom`. Default: `custom`.
- `cpu_share_count` - (Optional) The number of CPU shares allocated to the virtual machine when the `cpu_share_level` is `custom`.
- `memory_limit` - (Optional) The maximum amount of memory (in MB) that this virtual machine can consume, regardless of available resources. The default is no limit.
- `memory_reservation` - (Optional) The amount of memory (in MB) that this virtual machine is guaranteed. The default is no reservation.
- `memory_share_level` - (Optional) The allocation level for memory resources. Can be one of `high`, `low`, `normal`, or `custom`. Default: `custom`.
- `memory_share_count` - (Optional) The number of memory shares allocated to the virtual machine when the `memory_share_level` is `custom`.

» Advanced options

The following options control advanced operation of the virtual machine, or control various parts of Terraform workflow, and should not need to be modified during basic operation of the resource. Only change these options if they are explicitly required, or if you are having trouble with Terraform's default behavior.

- `enable_disk_uuid` - (Optional) Expose the UUIDs of attached virtual disks to the virtual machine, allowing access to them in the guest. Default: `false`.
- `hv_mode` - (Optional) The (non-nested) hardware virtualization setting for this virtual machine. Can be one of `hvAuto`, `hvOn`, or `hvOff`. Default: `hvAuto`.
- `ept_rvi_mode` - (Optional) The EPT/RVI (hardware memory virtualization) setting for this virtual machine. Can be one of `automatic`, `on`, or `off`. Default: `automatic`.
- `nested_hv_enabled` - (Optional) Enable nested hardware virtualization on this virtual machine, facilitating nested virtualization in the guest. Default: `false`.
- `enable_logging` - (Optional) Enable logging of virtual machine events to a log file stored in the virtual machine directory. Default: `false`.
- `cpu_performance_counters_enabled` - (Optional) Enable CPU performance counters on this virtual machine. Default: `false`.

- `swap_placement_policy` - (Optional) The swap file placement policy for this virtual machine. Can be one of `inherit`, `hostLocal`, or `vmDirectory`. Default: `inherit`.
- `wait_for_guest_net_timeout` - (Optional) The amount of time, in minutes, to wait for a routeable IP address on this virtual machine. A value less than 1 disables the waiter. Defualt: 5 minutes.
- `shutdown_wait_timeout` - (Optional) The amount of time, in minutes, to wait for a graceful guest shutdown when making necessary updates to the virtual machine. If `force_power_off` is set to true, the VM will be force powered-off after this timeout, otherwise an error is returned. Default: 3 minutes.
- `migrate_wait_timeout` - (Optional) The amount of time, in minutes, to wait for a virtual machine migration to complete before failing. Default: 10 minutes. Also see the section on virtual machine migration.
- `force_power_off` - (Optional) If a guest shutdown failed or timed out while updating or destroying (see `shutdown_wait_timeout`), force the power-off of the virtual machine. Default: `true`.
- `scsi_controller_count` - (Optional) The number of SCSI controllers that Terraform manages on this virtual machine. This directly affects the amount of disks you can add to the virtual machine and the maximum disk unit number. Note that lowering this value does not remove controllers. Default: `1`.

NOTE: `scsi_controller_count` should only be modified when you will need more than 15 disks on a single virtual machine, or in rare cases that require a dedicated controller for certain disks. HashiCorp does not support exploiting this value to add out-of-band devices.

» Disk options

Virtual disks are managed by adding an instance of the `disk` sub-resource.

At the very least, there must be `name` and `size` attributes. `unit_number` is required for any disk other than the first, and there must be at least one resource with the implicit number of 0.

An abridged multi-disk example is below:

```
resource "vsphere_virtual_machine" "vm" {
  ...

  disk {
    label = "disk0"
    size  = "10"
  }

  disk {
```

```
    label       = "disk1"
    size        = "100"
    unit_number = 1
  }

  ...
}
```

The options are:

- `label` - (Required) A label for the disk. Forces a new disk if changed.

NOTE: It's recommended that you set the disk label to a format matching `diskN`, where `N` is the number of the disk, starting from disk number 0. This will ensure that your configuration is compatible when importing a virtual machine. For more information, see the section on importing.

NOTE: Do not choose a label that starts with `orphaned_disk_` (example: `orphaned_disk_0`), as this prefix is reserved for disks that Terraform does not recognize, such as disks that are attached externally. Terraform will issue an error if you try to label a disk with this prefix.

- `name` - (Optional) An alias for both `label` and `path`, the latter when using `attach`. Required if not using `label`.

NOTE: This parameter has been deprecated and will be removed in future versions of the vSphere provider. You cannot use `name` on a disk that has previously had a `label`, and using this argument is not recommend for new configurations.

NOTE: In previous versions of the vSphere provider this argument controlled file names for non-attached disks - this behavior has now been removed, and the only time this controls path is when attaching a disk externally with `attach` when the `path` field is not specified.

- `size` - (Required) The size of the disk, in GiB.
- `unit_number` - (Optional) The disk number on the SCSI bus. The maximum value for this setting is the value of `scsi_controller_count` times 15, minus 1 (so `14`, `29`, `44`, and `59`, for 1-4 controllers respectively). The default is `0`, for which one disk must be set to. Duplicate unit numbers are not allowed.
- `datastore_id` - (Optional) A managed object reference ID to the datastore for this virtual disk. The default is to use the datastore of the virtual machine. See the section on virtual machine migration for details on changing this value.
- `attach` - (Optional) Attach an external disk instead of creating a new one. Implies and conflicts with `keep_on_remove`. If set, you cannot set `size`, `eagerly_scrub`, or `thin_provisioned`. Must set `path` if used.

- `path` - (Optional) When using `attach`, this parameter controls the path of a virtual disk to attach externally. Otherwise, it is a computed attribute that contains the virtual disk's current filename.
- `keep_on_remove` - (Optional) Keep this disk when removing the sub-resource or destroying the virtual machine. Default: `false`.
- `disk_mode` - (Optional) The mode of this this virtual disk for purposes of writes and snapshotting. Can be one of `append`, `independent_nonpersistent`, `independent_persistent`, `nonpersistent`, `persistent`, or `undoable`. Default: `persistent`. For an explanation of options, click here.
- `eagerly_scrub` - (Optional) If set to `true`, the disk space is zeroed out on VM creation. This will delay the creation of the disk or virtual machine. See the section on picking a disk type. Default: `false`.
- `thin_provisioned` - (Optional) If `true`, this disk is thin provisioned, with space for the file being allocated on an as-needed basis. See the section on picking a disk type. Default: `true`.
- `disk_sharing` - (Optional) The sharing mode of this virtual disk. Can be one of `sharingMultiWriter` or `sharingNone`. Default: `sharingNone`.

NOTE: Disk sharing is only available on vSphere 6.0 and higher.

- `write_through` - (Optional) If `true`, writes for this disk are sent directly to the filesystem immediately instead of being buffered. Default: `false`.
- `io_limit` - (Optional) The upper limit of IOPS that this disk can use. The default is no limit.
- `io_reservation` - (Optional) The I/O reservation (guarantee) that this disk has, in IOPS. The default is no reservation.
- `io_share_level` - (Optional) The share allocation level for this disk. Can be one of `low`, `normal`, `high`, or `custom`. Default: `normal`.
- `io_share_count` - (Optional) The share count for this disk when the share level is `custom`.

» Computed disk attributes

- `uuid` - The UUID of the virtual disk's VMDK file. This is used to track the virtual disk on the virtual machine.

» Picking a disk type

The `eagerly_scrub` and `thin_provisioned` options control the space allocation type of a virtual disk. These show up in the vSphere console as a unified enumeration of options, the equivalents of which are explained below. The defaults in the sub-resource are the equivalent of thin provisioning.

- **Thick provisioned lazy zeroed:** Both `eagerly_scrub` and `thin_provisioned` should be set to `false`.
- **Thick provisioned eager zeroed:** `eagerly_scrub` should be set to true, and `thin_provisioned` should be set to `false`.
- **Thin provisioned:** `eagerly_scrub` should be set to `false`, and `thin_provisioned` should be set to `true`.

For the technical details of each virtual disk provisioning policy, click here.

NOTE: Not all disk types are available on some types of datastores. Attempting to set options inappropriate for a datastore that a disk is deployed to will result in a successful initial apply, but vSphere will silently correct the options, and subsequent plans will fail with an appropriate error message until the settings are corrected.

NOTE: The disk type cannot be changed once set.

» Network interface options

Network interfaces are managed by adding an instance of the `network_interface` sub-resource.

Interfaces are assigned to devices in the specific order they are declared. This has different implications for different operating systems.

Given the following example:

```
resource "vsphere_virtual_machine" "vm" {
  ...

  network_interface {
    network_id   = "${data.vsphere_network.public.id}"
  }

  network_interface {
    network_id   = "${data.vsphere_network.private.id}"
  }
}
```

The first interface with the `public` network assigned to it would show up in order before the interface assigned to `private`. On some Linux systems, this might mean that the first interface would show up as `eth0` and the second would show up as `eth1`.

The options are:

- `network_id` - (Required) The managed object reference ID of the network to connect this interface to.

- `adapter_type` - (Optional) The network interface type. Can be one of `e1000`, `e1000e`, or `vmxnet3`. Default: `vmxnet3`.
- `use_static_mac` - (Optional) If true, the `mac_address` field is treated as a static MAC address and set accordingly. Setting this to `true` requires `mac_address` to be set. Default: `false`.
- `mac_address` - (Optional) The MAC address of this network interface. Can only be manually set if `use_static_mac` is true, otherwise this is a computed value that gives the current MAC address of this interface.
- `bandwidth_limit` - (Optional) The upper bandwidth limit of this network interface, in Mbits/sec. The default is no limit.
- `bandwidth_reservation` - (Optional) The bandwidth reservation of this network interface, in Mbits/sec. The default is no reservation.
- `bandwidth_share_level` - (Optional) The bandwidth share allocation level for this interface. Can be one of `low`, `normal`, `high`, or `custom`. Default: `normal`.
- `bandwidth_share_count` - (Optional) The share count for this network interface when the share level is `custom`.

» CDROM options

A single virtual CDROM device can be created and attached to the virtual machine. The resource only supports attaching a CDROM from a datastore ISO.

An example is below:

```
resource "vsphere_virtual_machine" "vm" {
  ...

  cdrom {
    datastore_id = "${data.vsphere_datastore.iso_datastore.id}"
    path         = "ISOs/os-livecd.iso"
  }
}
```

The options are:

- `datastore_id` - (Required) The datastore ID that the ISO is located in.
- `path` - (Required) The path to the ISO file.

» Virtual device computed options

Virtual device resources (`disk`, `network_interface`, and `cdrom`) all export the following attributes. These options help locate the sub-resource on future Terraform runs. The options are:

- `key` - The ID of the device within the virtual machine.

- `device_address` - An address internal to Terraform that helps locate the device when `key` is unavailable. This follows a convention of `CONTROLLER_TYPE:BUS_NUMBER:UNIT_NUMBER`. Example: `scsi:0:1` means device unit 1 on SCSI bus 0.

» Creating a Virtual Machine from a Template

The `clone` sub-resource can be used to create a new virtual machine from an existing virtual machine or template. The resource supports both making a complete copy of a virtual machine, or cloning from a snapshot (otherwise known as a linked clone).

For see the cloning and customization example for a usage synopsis.

NOTE: Changing any option in `clone` after creation forces a new resource.

The options available in the `clone` sub-resource are:

- `template_uuid` - (Required) The UUID of the source virtual machine or template.
- `linked_clone` - (Optional) Clone this virtual machine from a snapshot. Templates must have a single snapshot only in order to be eligible. Default: `false`.
- `timeout` - (Optional) The timeout, in minutes, to wait for the virtual machine clone to complete. Default: 30 minutes.
- `customize` - (Optional) The customization spec for this clone. This allows the user to configure the virtual machine post-clone. For more details, see virtual machine customization.

» Virtual machine customization

As part of the `clone` operation, a virtual machine can be customized to configure host, network, or licensing settings.

To perform virtual machine customization as a part of the clone process, specify the `customize` sub-resource within the `clone` sub-resource with the respective customization options. See the cloning and customization example for a usage synopsis.

The settings for `customize` are as follows:

» Customization timeout settings

- `timeout` - (Optional) The time, in minutes that Terraform waits for customization to complete before failing. The default is 10 minutes, and setting the value to 0 or a negative value disables the waiter altogether.

» Network interface settings

The following settings should be in a `network_interface` block in the `customize` sub-resource. These settings configure network interfaces on a per-interface basis and are matched up to `network_interface` sub-resources in the main block in the order they are declared.

Given the following example:

```
resource "vsphere_virtual_machine" "vm" {
  ...

  network_interface {
    network_id   = "${data.vsphere_network.public.id}"
  }

  network_interface {
    network_id   = "${data.vsphere_network.private.id}"
  }

  clone {
    ...

    customize {
      ...

      network_interface {
        ipv4_address = "10.0.0.10"
        ipv4_netmask = 24
      }

      network_interface {
        ipv4_address = "172.16.0.10"
        ipv4_netmask = 24
      }

      ipv4_gateway = "10.0.0.1"
    }
  }
}
```

The first set of `network_interface` data would be assigned to the `public` interface, and the second to the `private` interface.

To use DHCP, declare an empty `network_interface` block for each interface being configured. So the above example would look like:

```
resource "vsphere_virtual_machine" "vm" {
  ...
```

```
  network_interface {
    network_id   = "${data.vsphere_network.public.id}"
  }

  network_interface {
    network_id   = "${data.vsphere_network.private.id}"
  }

  clone {
    ...

    customize {
      ...

      network_interface {}

      network_interface {}
    }
  }
}
```

The options are:

- `dns_server_list` - (Optional) Network interface-specific DNS server settings for Windows operating systems. Ignored on Linux and possibly other opearating systems - for those systems, please see the global DNS settings section.
- `dns_domain` - (Optional) Network interface-specific DNS search domain for Windows operating systems. Ignored on Linux and possibly other opearating systems - for those systems, please see the global DNS settings section.
- `ipv4_address` - (Optional) The IPv4 address assigned to this network adapter. If left blank or not included, DHCP is used.
- `ipv4_netmask` The IPv4 subnet mask, in bits (example: `24` for 255.255.255.0).
- `ipv6_address` - (Optional) The IPv6 address assigned to this network adapter. If left blank or not included, auto-configuration is used.
- `ipv6_netmask` - (Optional) The IPv6 subnet mask, in bits (example: `32`).

NOTE: The minimum setting for IPv4 in a customization specification is DHCP. If you are setting up an IPv6-exclusive network without DHCP, you might need to set `wait_for_guest_net_timeout` to a high enough value to cover the DHCP timeout of your virtual machine, or turn it off altogether by supplying a zero or negative value. Keep in mind that turning off `wait_for_guest_net_timeout` will more than likely mean that IP addresses will not be reported to any provisioners you may have configured on the

resource.

» Global routing settings

VM customization under the `vsphere_virtual_machine` resource does not take a per-interface gateway setting, but rather default routes are configured on a global basis. For an example, see the network interface settings section.

The settings here must match the IP/mask of at least one `network_interface` supplied to customization.

The options are:

- `ipv4_gateway` - (Optional) The IPv4 default gateway when using `network_interface` customization on the virtual machine.
- `ipv6_gateway` - (Optional) The IPv6 default gateway when using `network_interface` customization on the virtual machine.

» Global DNS settings

The following settings configure DNS globally, generally for Linux systems. For Windows systems, this is done per-interface, see network interface settings.

- `dns_server_list` - The list of DNS servers to configure on a virtual machine.
- `dns_suffix_list` - A list of DNS search domains to add to the DNS configuration on the virtual machine.

» Linux customization options

The settings in the `linux_options` sub-resource pertain to Linux guest OS customization. If you are customizing a Linux operating system, this section must be included.

Example:

```
resource "vsphere_virtual_machine" "vm" {
  ...

  clone {
    ...

    customize {
      ...

      linux_options {
        host_name = "terraform-test"
        domain    = "test.internal"
```

```
    }
  }
 }
}
```

The options are:

- `host_name` - (Required) The host name for this machine. This, along with `domain`, make up the FQDN of this virtual machine.
- `domain` - (Required) The domain name for this machine. This, along with `host_name`, make up the FQDN of this virtual machine.
- `hw_clock_utc` - (Optional) Tells the operating system that the hardware clock is set to UTC. Default: `true`.
- `time_zone` - (Optional) Sets the time zone. For a list of possible combinations, click here. The default is UTC.

» Windows customization options

The settings in the `windows_options` sub-resource pertain to Windows guest OS customization. If you are customizing a Windows operating system, this section must be included.

Example:

```
resource "vsphere_virtual_machine" "vm" {
  ...

  clone {
    ...

    customize {
      ...

      windows_options {
        computer_name  = "terraform-test"
        workgroup      = "test"
        admin_password = "VMw4re"
      }
    }
  }
}
```

The options are:

- `computer_name` - (Required) The computer name of this virtual machine.
- `admin_password` - (Optional) The administrator password for this virtual machine.

NOTE: `admin_password` is a sensitive field in Terraform and will not be output on-screen, but is stored in state and sent to the VM in plain text - keep this in mind when provisioning your infrastructure.

- `workgroup` - (Optional) The workgroup name for this virtual machine. One of this or `join_domain` must be included.
- `join_domain` - (Optional) The domain to join for this virtual machine. One of this or `workgroup` must be included.
- `domain_admin_user` - (Optional) The user of the domain administrator used to join this virtual machine to the domain. Required if you are setting `join_domain`.
- `domain_admin_password` - (Optional) The password of the domain administrator used to join this virtual machine to the domain. Required if you are setting `join_domain`.

NOTE: `domain_admin_password` is a sensitive field in Terraform and will not be output on-screen, but is stored in state and sent to the VM in plain text - keep this in mind when provisioning your infrastructure.

- `full_name` - (Optional) The full name of the user of this virtual machine. This populates the "user" field in the general Windows system information. Default: `Administrator`.
- `organization_name` - (Optional) The organization name this virtual machine is being installed for. This populates the "organization" field in the general Windows system information. Default: `Managed by Terraform`.
- `product_key` - (Optional) The product key for this virtual machine. The default is no key.
- `run_once_command_list` - (Optional) A list of commands to run at first user logon, after guest customization.
- `auto_logon` - (Optional) Specifies whether or not the VM automatically logs on as Administrator. Default: `false`.
- `auto_logon_count` - (Optional) Specifies how many times the VM should auto-logon the Administrator account when `auto_logon` is true. This should be set accordingly to ensure that all of your commands that run in `run_once_command_list` can log in to run. Default: `1`.
- `time_zone` - (Optional) The new time zone for the virtual machine. This is a numeric, sysprep-dictated, timezone code. For a list of codes, click here. The default is `85` (GMT/UTC).

» Supplying your own SysPrep file

Alternative to the `windows_options` supplied above, you can instead supply your own `sysprep.inf` file contents via the `windows_sysprep_text` option. This allows full control of the customization process out-of-band of vSphere. Example below:

```
resource "vsphere_virtual_machine" "vm" {
```

```
  ...

  clone {
    ...

    customize {
      ...

      windows_sysprep_text = "${file("${path.module}/sysprep.inf")}"
    }
  }
}
```

Note this option is mutually exclusive to `windows_options` - one must not be included if the other is specified.

» Using vApp properties to supply OVF/OVA configuration

Alternative to the settings in `customize`, one can use the settings in the `properties` section of the `vapp` sub-resource to supply configuration parameters to a virtual machine cloned from a template that came from an imported OVF or OVA file.

NOTE: The only supported usage path for vApp properties is for existing user-configurable keys. These generally come from an existing template that was created from an imported OVF or OVA file. You cannot set values for vApp properties on virtual machines created from scratch, virtual machines lacking a vApp configuration, or on property keys that do not exist.

The configuration looks similar to the one below:

```
resource "vsphere_virtual_machine" "vm" {
  ...

  clone {
    template_uuid = "${data.vsphere_virtual_machine.template_from_ovf.id}"
  }

  vapp {
    properties {
      "guestinfo.hostname"                        = "${var.vm_name}.foobar.local"
      "guestinfo.interface.0.name"                = "ens192"
      "guestinfo.interface.0.ip.0.address"        = "10.0.0.100/24"
      "guestinfo.interface.0.route.0.gateway"     = "10.0.0.1"
      "guestinfo.interface.0.route.0.destination" = "0.0.0.0/0"
      "guestinfo.dns.server.0"                    = "10.0.0.10"
    }
```

```
  }
}
```

» Additional requirements and notes for cloning

Note that when cloning from a template, there are additional requirements in both the resource configuration and source template:

- All disks on the virtual machine must be SCSI disks.
- You must specify at least the same number of `disk` sub-resources as there are disks that exist in the template. These sub-resources are ordered and lined up by the `unit_number` attribute. Additional disks can be added past this.
- The `size` of a virtual disk must be at least the same size as its counterpart disk in the template.
- When using `linked_clone`, the `size`, `thin_provisioned`, and `eagerly_scrub` settings for each disk must be an exact match to the individual disk's counterpart in the source template.
- The `scsi_controller_count` setting should be configured as necessary to cover all of the disks on the template. For best results, only configure this setting for the amount of controllers you will need to cover your disk quantity and bandwidth needs, and configure your template accordingly. For most workloads, this setting should be kept at its default of `1`, and all disks in the template should reside on the single, primary controller.
- Some operating systems (such as Windows) do not respond well to a change in disk controller type, so when using such OSes, take care to ensure that `scsi_type` is set to an exact match of the template's controller set. For maximum compatibility, make sure the SCSI controllers on the source template are all the same type.

To ease the gathering of some of these options, you can use the `vsphere_virtual_machine` data source, which will give you disk attributes, network interface types, SCSI bus types, and also the guest ID of the source template. See the cloning and customization example for usage details.

» Virtual Machine Migration

The `vsphere_virtual_machine` resource supports live migration (otherwise known as vMotion) both on the host and storage level. One can migrate the entire VM to another host, cluster, resource pool, or datastore, and migrate or pin a single disk to a specific datastore.

» Host, cluster, and resource pool migration

To migrate the virtual machine to another host or resource pool, change the `host_system_id` or `resource_pool_id` to the manged object IDs of the new host or resource pool accordingly. To change the virtual machine's cluster or standalone host, select a resource pool within the specific target.

The same rules apply for migration as they do for VM creation - any host specified needs to be a part of the resource pool supplied. Also keep in mind the implications of moving the virtual machine to a resource pool in another cluster or standalone host, namely ensuring that all hosts in the cluster (or the single standalone host) have access to the datastore that the virtual machine is in.

» Storage migration

Storage migration can be done on two levels:

- Global datastore migration can be handled by changing the global `datastore_id` attribute. This triggers a storage migration for all disks that do not have an explicit `datastore_id` specified.
- An individual `disk` sub-resource can be migrated by manually specifying the `datastore_id` in its sub-resource. This also pins it to the specific datastore that is specified - if at a later time the VM and any unpinned disks migrate to another host, the disk will stay on the specified datastore.

An example of datastore pinning is below. As long as the datastore in the `pinned_datastore` data source does not change, any change to the standard `vm_datastore` data source will not affect the data disk - the disk will stay where it is.

```
resource "vsphere_virtual_machine" "vm" {
  ...

  datastore_id     = "${data.vsphere_datastore.vm_datastore.id}"

  disk {
    label = "disk0"
    size  = 10
  }

  disk {
    datastore_id = "${data.vsphere_datastore.pinned_datastore.id}"
    label        = "disk1"
    size         = 100
    unit_number  = 1
  }
```

```
  ...
}
```

» Storage migration restrictions

Note that you cannot migrate external disks added with the `attach` parameter. As these disks have usually been created and assigned to a datastore outside of the scope of the `vsphere_virtual_machine` resource in question, such as by using the `vsphere_virtual_disk` resource, management of such disks would render their configuration unstable.

» Attribute Reference

The following attributes are exported on the base level of this resource:

- `id` - The UUID of the virtual machine.
- `reboot_required` - Value internal to Terraform used to determine if a configuration set change requires a reboot. This value is only useful during an update process and gets reset on refresh.
- `vmware_tools_status` - The state of VMware tools in the guest. This will determine the proper course of action for some device operations.
- `vmx_path` - The path of the virtual machine's configuration file in the VM's datastore.
- `imported` - This is flagged if the virtual machine has been imported, or the state has been migrated from a previous version of the resource, and blocks the `clone` configuration option from being set. See the section on importing below.
- `change_version` - A unique identifier for a given version of the last configuration applied, such the timestamp of the last update to the configuration.
- `uuid` - The UUID of the virtual machine. Also exposed as the `id` of the resource.
- `default_ip_address` - The IP address selected by Terraform to be used with any provisioners configured on this resource. Whenever possible, this is the first IPv4 address that is reachable through the default gateway configured on the machine, then the first reachable IPv6 address, and then the first general discovered address if neither exist. If VMware tools is not running on the virtual machine, or if the VM is powered off, this value will be blank.
- `guest_ip_addresses` - The current list of IP addresses on this machine, including the value of `default_ip_address`. If VMware tools is not running on the virtual machine, or if the VM is powered off, this list will be empty.
- `moid`: The managed object reference ID of the created virtual machine.

» Importing

An existing virtual machine can be imported into this resource via supplying the full path to the virtual machine. An example is below:

```
terraform import vsphere_virtual_machine.vm /dc1/vm/srv1
```

The above would import the virtual machine named `srv1` that is located in the `dc1` datacenter.

» Additional requirements and notes for importing

Many of the same requirements for cloning apply to importing, although since importing writes directly to state, a lot of these rules cannot be enforced at import time, so every effort should be made to ensure the correctness of the configuration before the import.

In addition to these rules, the following extra rules apply to importing:

- Disks need to have their `label` argument assigned in a convention matching `diskN`, starting with disk number 0, based on each disk's order on the SCSI bus. As an example, a disk on SCSI controller 0 with a unit number of 0 would be labeled `disk0`, a disk on the same controller with a unit number of 1 would be `disk1`, but the next disk, which is on SCSI controller 1 with a unit number of 0, still becomes `disk2`.
- Disks always get imported with `keep_on_remove` enabled until the first `terraform apply` runs, which will remove the setting for known disks. This is an extra safeguard against naming or accounting mistakes in the disk configuration.
- You cannot use the `clone` sub-resource on any imported VM. If you need to clone a new virtual machine or want a working configuration with `clone` features, you will need to create a new resource and destroy the old one.
- The `scsi_controller_count` for the resource is set to the number of contiguous SCSI controllers found, starting with the SCSI controller at bus number 0. If no SCSI controllers are found, the VM is not eligible for import. To ensure maximum compatibility, make sure your virtual machine has the exact number of SCSI controllers it needs, and set `scsi_controller_count` accordingly.

After importing, you should run `terraform plan`. Unless you have changed anything else in configuration that would be causing other attributes to change, the only difference should be the transition of `keep_on_remove` of known disks from `true` to `false`. The operation only updates Terraform state when applied, and is safe to run when the virtual machine is running. If more settings are being modified, you may need to plan maintenance accordingly for any necessary reconfiguration of the virtual machine.

» vsphere_virtual_machine_snapshot

The `vsphere_virtual_machine_snapshot` resource can be used to manage snapshots for a virtual machine.

For more information on managing snapshots and how they work in VMware, see here.

NOTE: A snapshot in VMware differs from traditional disk snapshots, and can contain the actual running state of the virtual machine, data for all disks that have not been set to be independent from the snapshot (including ones that have been attached via the attach parameter to the `vsphere_virtual_machine disk` sub-resource), and even the configuration of the virtual machine at the time of the snapshot. Virtual machine, disk activity, and configuration changes post-snapshot are not included in the original state. Use this resource with care! Neither VMware nor HashiCorp recommends retaining snapshots for a extended period of time and does NOT recommend using them as as backup feature. For more information on the limitation of virtual machine snapshots, see here.

» Example Usage

```
resource "vsphere_virtual_machine_snapshot" "demo1" {
  virtual_machine_uuid = "9aac5551-a351-4158-8c5c-15a71e8ec5c9"
  snapshot_name        = "Snapshot Name"
  description          = "This is Demo Snapshot"
  memory               = "true"
  quiesce              = "true"
  remove_children      = "false"
  consolidate          = "true"
}
```

» Argument Reference

The following arguments are supported:

NOTE: All attributes in the `vsphere_virtual_machine_snapshot` resource are immutable and force a new resource if changed.

- `virtual_machine_uuid` - (Required) The virtual machine UUID.
- `snapshot_name` - (Required) The name of the snapshot.
- `description` - (Required) A description for the snapshot.
- `memory` - (Required) If set to `true`, a dump of the internal state of the virtual machine is included in the snapshot.
- `quiesce` - (Required) If set to `true`, and the virtual machine is powered on when the snapshot is taken, VMware Tools is used to quiesce the file system in the virtual machine.

- `remove_children` - (Optional) If set to `true`, the entire snapshot subtree is removed when this resource is destroyed.
- `consolidate` - (Optional) If set to `true`, the delta disks involved in this snapshot will be consolidated into the parent when this resource is destroyed.

» Attribute Reference

The only attribute this resource exports is the resource `id`, which is set to the managed object reference ID of the snapshot.