Package 'AzureVM'

October 15, 2020

Title  Virtual Machines in 'Azure'
Version  2.2.2
Description  Functionality for working with virtual machines (VMs) in Microsoft's 'Azure' cloud: <https://azure.microsoft.com/en-us/services/virtual-machines/>. Includes facilities to deploy, startup, shutdown, and cleanly delete VMs and VM clusters. Deployment configurations can be highly customised, and can make use of existing resources as well as creating new ones. A selection of predefined configurations is provided to allow easy deployment of commonly used Linux and Windows images, including Data Science Virtual Machines. With a running VM, execute scripts and install optional extensions. Part of the 'AzureR' family of packages.

BugReports  https://github.com/Azure/AzureVM/issues
License  MIT + file LICENSE
VignetteBuilder  knitr
Depends  R (>= 3.3)
Imports  R6 (>= 2.4.1), AzureRMR (>= 2.3.0), jsonlite
Suggests  knitr, rmarkdown, testthat, parallel, AzureKeyVault, AzureVMmetadata
RoxygenNote  7.1.1
NeedsCompilation  no
Author  Hong Ooi [aut, cre], Microsoft [cph]
Maintainer  Hong Ooi <hongooi73@gmail.com>
Repository  CRAN
Date/Publication  2020-10-14 23:50:07 UTC

R topics documented:

  autoscaler_config .......................................................... 2
  az_vmss_resource .......................................................... 3
autoscaler_config

Autoscaler configuration

Usage

autoscaler_config(profiles = list(autoscaler_profile()), ...)

autoscaler_profile(name = "Profile", minsize = 1, maxsize = NA,
default = NA, scale_out = 0.75, scale_in = 0.25, interval = "PT1M",
window = "PT5M")

Arguments

profiles A list of autoscaling profiles, each obtained via a call to autoscaler_profile.
...
name For autoscaler_profile, a name for the profile.
minsize, maxsize, default For autoscaler_profile, the minimum, maximum and default number of instances.
scale_out, scale_in
For autoscaler_profile, the CPU usage (a fraction between 0 and 1) at which to scale out and in, respectively.

interval
For autoscaler_profile, The interval between samples, in ISO 8601 format. The default is 1 minute.

window
For autoscaler_profile, the window width over which to compute the percentage CPU. The default is 5 minutes.

See Also
create_vm_scaleset, vmss_config

Examples
autoscaler_config()
autoscaler_config(list(
    autoscaler_profile(minsize=2, maxsize=50, scale_out=0.9, scale_in=0.1)
))

Description
Class representing a virtual machine scaleset resource. In general, the methods in this class should not be called directly, nor should objects be directly instantiated from it. Use the az_vmss_template class for interacting with scalesets instead.

Format
An R6 object of class az_vmss_resource, inheriting from AzureRMR::az_resource.

Details
A single virtual machine scaleset in Azure is actually a collection of resources, including any and all of the following.

- Virtual network (Azure resource type Microsoft.Network/virtualNetworks)
- Load balancer (Azure resource type Microsoft.Network/loadBalancers)
- Public IP address (Azure resource type Microsoft.Network/publicIPAddresses)
- Autoscaler (Azure resource type Microsoft.Insights/autoscaleSettings)
- The scaleset itself (Azure resource type Microsoft.Compute/virtualMachineScaleSets)

By wrapping the deployment template used to create these resources, the az_vmss_template class allows managing them all as a single entity.
Methods

The following methods are available, in addition to those provided by the AzureRMR::az_template class.

- **sync_vmss_status**: Check the status of the scaleset.
- **list_instances()**: Return a list of az_vm_resource objects, one for each VM instance in the scaleset. Note that if the scaleset has a load balancer attached, the number of instances will vary depending on the load.
- **get_instance(id)**: Return a specific VM instance in the scaleset.
- **start(id=NULL, wait=FALSE)**: Start the scaleset. In this and the other methods listed here, id can be an optional character vector of instance IDs; if supplied, only carry out the operation for those instances.
- **restart(id=NULL, wait=FALSE)**: Restart the scaleset.
- **stop(deallocate=TRUE, id=NULL, wait=FALSE)**: Stop the scaleset.
- **get_public_ip_address()**: Get the public IP address of the scaleset (technically, of the load balancer). If the scaleset doesn't have a load balancer attached, returns NA.
- **get_vm_public_ip_addresses(id=NULL, nic=1, config=1)**: Get the public IP addresses for the instances in the scaleset. Returns NA for the instances that are stopped or not publicly accessible.
- **get_vm_private_ip_addresses(id=NULL, nic=1, config=1)**: Get the private IP addresses for the instances in the scaleset.
- **get_vnet(nic=1, config=1)**: Get the scaleset's virtual network resource.
- **get_nsg(nic=1, config=1)**: Get the scaleset's network security group resource.
- **run_deployed_command(command, parameters=NULL, script=NULL, id=NULL)**: Run a PowerShell command on the instances in the scaleset.
- **run_script(script, parameters=NULL, id=NULL)**: Run a script on the VM. For a Linux VM, this will be a shell script; for a Windows VM, a PowerShell script. Pass the script as a character vector.
- **reimage(id=NULL, datadisks=FALSE)**: Reimage the instances in the scaleset. If datadisks is TRUE, reimage any attached data disks as well.
- **redeploy(id=NULL)**: Redeploy the instances in the scaleset.
- **mapped_vm_operation(...)**: Carry out an arbitrary operation on the instances in the scaleset. See the do_operation method of the AzureRMR::az_resource class for more details.
- **add_extension(publisher, type, version, settings=list(), protected_settings=list(), key_vault_settings=list())**: Add an extension to the scaleset.
- **do_vmss_operation(...)**: Carry out an arbitrary operation on the scaleset resource (as opposed to the instances in the scaleset).

Instance operations

AzureVM has the ability to parallelise scaleset instance operations using a background process pool provided by AzureRMR. This can lead to significant speedups when working with scalesets.
with high instance counts. The pool is created automatically the first time that it is required, and remains persistent for the session. You can control the size of the process pool with the `azure_vm_minpoolsize` and `azure_vm_maxpoolsize` options, which have default values 2 and 10 respectively.

The `id` argument lets you specify a subset of instances on which to carry out an operation. This can be a character vector of instance IDs; a list of instance objects such as returned by `list_instances`; or a single instance object. The default (NULL) is to carry out the operation on all instances.

See Also

AzureRMR::az_resource, get_vm_scaleset_resource, az_vmss_template, AzureRMR::init_pool

VM scaleset API reference

---

### az_vmss_template

**Virtual machine scaleset (cluster) template class**

**Description**

Class representing a virtual machine scaleset deployment template. This class keeps track of all resources that are created as part of deploying a scaleset, and exposes methods for managing them.

**Format**

An R6 object of class `az_vmss_template`, inheriting from `AzureRMR::az_template`.

**Details**

A virtual machine scaleset in Azure is actually a collection of resources, including any and all of the following.

- Virtual network (Azure resource type `Microsoft.Network/virtualNetworks`)
- Load balancer (Azure resource type `Microsoft.Network/loadBalancers`)
- Public IP address (Azure resource type `Microsoft.Network/publicIPAddresses`)
- Autoscaler (Azure resource type `Microsoft.Insights/autoscaleSettings`)
- The scaleset itself (Azure resource type `Microsoft.Compute/virtualMachineScaleSets`)

By wrapping the deployment template used to create these resources, the `az_vmss_template` class allows managing them all as a single entity.

**Fields**

The following fields are exposed, in addition to those provided by the `AzureRMR::az_template` class.

- `dns_name`: The DNS name for the scaleset (technically, the name for its load balancer). Will be NULL if the scaleset is not publicly visible, or doesn’t have a load balancer attached.
- `identity`: The managed identity details for the scaleset. Will be NULL if the scaleset doesn’t have an identity assigned.
Methods

The following methods are available, in addition to those provided by the AzureRMR::az_template class.

- **sync_vmss_status**: Check the status of the scaleset.
- **list_instances()**: Return a list of az_vm_resource objects, one for each VM instance in the scaleset. Note that if the scaleset has an autoscaler attached, the number of instances will vary depending on the load.
- **get_instance(id)**: Return a specific VM instance in the scaleset.
- **start(id=NULL, wait=FALSE)**: Start the scaleset. In this and the other methods listed here, id can be an optional character vector of instance IDs; if supplied, only carry out the operation for those instances.
- **restart(id=NULL, wait=FALSE)**: Restart the scaleset.
- **stop(deallocate=TRUE, id=NULL, wait=FALSE)**: Stop the scaleset.
- **get_public_ip_address()**: Get the public IP address of the scaleset (technically, of the load balancer). If the scaleset doesn’t have a load balancer attached, returns NA.
- **get_vm_public_ip_addresses(id=NULL, nic=1, config=1)**: Get the public IP addresses for the instances in the scaleset. Returns NA for the instances that are stopped or not publicly accessible.
- **get_vm_private_ip_addresses(id=NULL, nic=1, config=1)**: Get the private IP addresses for the instances in the scaleset.
- **get_public_ip_resource()**: Get the Azure resource for the load balancer’s public IP address.
- **get_vnet(nic=1, config=1)**: Get the scaleset’s virtual network resource.
- **get_nsg(nic=1, config=1)**: Get the scaleset’s network security group resource.
- **get_load_balancer()**: Get the scaleset’s load balancer resource.
- **get_autoscaler()**: Get the scaleset’s autoscaler resource.
- **run_deployed_command(command, parameters=NULL, script=NULL, id=NULL)**: Run a PowerShell command on the instances in the scaleset.
- **run_script(script, parameters=NULL, id=NULL)**: Run a script on the VM. For a Linux VM, this will be a shell script; for a Windows VM, a PowerShell script. Pass the script as a character vector.
- **reimage(id=NULL, datadisks=FALSE)**: Reimage the instances in the scaleset. If datadisks is TRUE, reimage any attached data disks as well.
- **redeploy(id=NULL)**: Redeploy the instances in the scaleset.
- **mapped_vm_operation(..., id=NULL)**: Carry out an arbitrary operation on the instances in the scaleset. See the do_operation method of the AzureRMR::az_resource class for more details.
- **add_extension(publisher, type, version, settings=list(), protected_settings=list(), key_vault_settings=list())**: Add an extension to the scaleset.
- **do_vmss_operation(...)** Carry out an arbitrary operation on the scaleset resource (as opposed to the instances in the scaleset).

Many of these methods are actually provided by the az_vmss_resource class, and propagated to the template as active bindings.
 Instance operations

AzureVM has the ability to parallelise scaleset instance operations using a background process pool provided by AzureRMR. This can lead to significant speedups when working with scalesets with high instance counts. The pool is created automatically the first time that it is required, and remains persistent for the session. You can control the size of the process pool with the `azure_vm_minpoolsize` and `azure_vm_maxpoolsize` options, which have default values 2 and 10 respectively.

The `id` argument lets you specify a subset of instances on which to carry out an operation. This can be a character vector of instance IDs; a list of instance objects such as returned by `list_instances`; or a single instance object. The default (NULL) is to carry out the operation on all instances.

See Also

`AzureRMR::az_template`, `create_vm_scaleset`, `get_vm_scaleset`, `delete_vm_scaleset`, `AzureRMR::init_pool`

VM scaleset API reference

Examples

```r
## Not run:
sub <- AzureRMR::get_azure_login()
  get_subscription("subscription_id")

vmss <- sub$get_vm_scaleset("myscaleset")

vmss$identity

vmss$get_public_ip_address()  # NA if the scaleset doesn't have a load balancer

vmss$start()
vmss$get_vm_private_ip_addresses()
vmss$get_vm_public_ip_addresses()  # NA if scaleset nodes are not publicly visible

instances <- vmss$list_instances()
first <- instances[1]
vmss$run_script("echo hello world! > /tmp/hello.txt", id=first)
vmss$stop(id=first)
vmss$reimage(id=first)

vmss$sync_vmss_status()

## End(Not run)
```
Description

Class representing a virtual machine resource. In general, the methods in this class should not be called directly, nor should objects be directly instantiated from it. Use the `az_vm_template` class for interacting with VMs instead.

Format

An R6 object of class `az_vm_resource`, inheriting from `AzureRMR::az_resource`.

Methods

The following methods are available, in addition to those provided by the `AzureRMR::az_resource` class:

- `start(wait=TRUE)`: Start the VM. By default, wait until the startup process is complete.
- `stop(deallocate=TRUE, wait=FALSE)`: Stop the VM. By default, deallocate it as well.
- `restart(wait=TRUE)`: Restart the VM.
- `run_deployed_command(command, parameters, script)`: Run a PowerShell command on the VM.
- `run_script(script, parameters)`: Run a script on the VM. For a Linux VM, this will be a shell script; for a Windows VM, a PowerShell script. Pass the script as a character vector.
- `sync_vm_status()`: Check the status of the VM.
- `resize(size, deallocate=FALSE, wait=FALSE)`: Resize the VM. Optionally stop and deallocate it first (may sometimes be necessary).
- `redeploy()`: Redeploy the VM.
- `reimage()`: Reimage the VM.
- `get_public_ip_address(nic=1, config=1)`: Get the public IP address of the VM. Returns NA if the VM is shut down, or is not publicly accessible.
- `get_private_ip_address(nic=1, config=1)`: Get the private IP address of the VM.
- `get_public_ip_resource(nic=1, config=1)`: Get the Azure resource for the VM's public IP address.
- `get_nic(nic=1)`: Get the VM's network interface resource.
- `get_vnet(nic=1, config=1)`: Get the VM's virtual network resource.
- `get_nsg(nic=1, config=1)`: Get the VM's network security group resource. Note that an NSG can be attached to either the VM's network interface or to its virtual network subnet; if there is an NSG attached to both, this method returns a list containing the two NSG resource objects.
- `get_disk(disk="os")`: Get a managed disk resource attached to the VM. The disk argument can be "os" for the OS disk, or a number indicating the LUN of a data disk. AzureVM only supports managed disks.
- `add_extension(publisher, type, version, settings=list(), protected_settings=list(), key_vault_settings=list())`: Add an extension to the VM.
- `do_vm_operation(...)`: Carry out an arbitrary operation on the VM resource. See the `do_operation` method of the `AzureRMR::az_resource` class for more details.
az_vm_template

Description

Class representing a virtual machine deployment template. This class keeps track of all resources that are created as part of deploying a VM, and exposes methods for managing them.

Format

An R6 object of class `az_vm_template`, inheriting from `AzureRMR::az_template`.

Details

A single virtual machine in Azure is actually a collection of resources, including any and all of the following.

- Network interface (Azure resource type `Microsoft.Network/networkInterfaces`)
- Virtual network (Azure resource type `Microsoft.Network/virtualNetworks`)
- Public IP address (Azure resource type `Microsoft.Network/publicIPAddresses`)
- The VM itself (Azure resource type `Microsoft.Compute/virtualMachines`)

By wrapping the deployment template used to create these resources, the `az_vm_template` class allows managing them all as a single entity.

Fields

The following fields are exposed, in addition to those provided by the `AzureRMR::az_template` class.

- `dns_name`: The DNS name for the VM. Will be NULL if the VM is not publicly visible, or doesn’t have a domain name assigned to its public IP address.
- `identity`: The managed identity details for the VM. Will be NULL if the VM doesn’t have an identity assigned.
Methods

The following methods are available, in addition to those provided by the `AzureRMR::az_template` class.

- `start(wait=TRUE)`: Start the VM. By default, wait until the startup process is complete.
- `stop(deallocate=TRUE,wait=FALSE)`: Stop the VM. By default, deallocate it as well.
- `restart(wait=TRUE)`: Restart the VM.
- `run_deployed_command(command,parameters,script)`: Run a PowerShell command on the VM.
- `run_script(script,parameters)`: Run a script on the VM. For a Linux VM, this will be a shell script; for a Windows VM, a PowerShell script. Pass the script as a character vector.
- `sync_vm_status()`: Check the status of the VM.
- `resize(size,deallocate=FALSE,wait=FALSE)`: Resize the VM. Optionally stop and deallocate it first (may sometimes be necessary).
- `redeploy()`: Redeploy the VM.
- `reimage()`: Reimage the VM.
- `get_public_ip_address(nic=1,config=1)`: Get the public IP address of the VM. Returns NA if the VM is stopped, or is not publicly accessible.
- `get_private_ip_address(nic=1,config=1)`: Get the private IP address of the VM.
- `get_public_ip_resource(nic=1,config=1)`: Get the Azure resource for the VM’s public IP address.
- `get_nic(nic=1)`: Get the VM’s network interface resource.
- `get_vnet(nic=1,config=1)`: Get the VM’s virtual network resource.
- `get_nsg(nic=1,config=1)`: Get the VM’s network security group resource. Note that an NSG can be attached to either the VM’s network interface or to its virtual network subnet; if there is an NSG attached to both, this method returns a list containing the two NSG resource objects.
- `get_disk(disk="os")`: Get a managed disk resource attached to the VM. The disk argument can be "os" for the OS disk, or a number indicating the LUN of a data disk. AzureVM only supports managed disks.
- `add_extension(publisher,type,version,settings=list(),protected_settings=list(),key_vault_settings=list())`: Add an extension to the VM.
- `do_vm_operation(...)`: Carries out an arbitrary operation on the VM resource. See the `do_operation` method of the `AzureRMR::az_resource` class for more details.

Many of these methods are actually provided by the `az_vm_resource` class, and propagated to the template as active bindings.

See Also

`AzureRMR::az_template`, `create_vm`, `get_vm`, `delete_vm`

VM API reference
## Not run:

```r
sub <- AzureRMR::get_azure_login()
  get_subscription("subscription_id")

vm <- sub$get_vm("myvm")
vm$identity
vm$start()
vm$get_private_ip_address()
vm$get_public_ip_address()
vm$run_script("echo hello world! > /tmp/hello.txt")
vm$stop()
vm$get_private_ip_address()
vm$get_public_ip_address()  # NA, assuming VM has a dynamic IP address
vm$resize("Standard_DS13_v2")
vm$sync_vm_status()
```

## End(Not run)

---

**build_template_definition.vm_config**

*Build template definition and parameters*

### Description

Build template definition and parameters

### Usage

```r
## S3 method for class 'vm_config'
build_template_definition(config, ...)

## S3 method for class 'vmss_config'
build_template_definition(config, ...)

## S3 method for class 'vm_config'
build_template_parameters(config, name, login_user, size, ...)

## S3 method for class 'vmss_config'
build_template_parameters(config, name, login_user, size, instances, ...)
```
Arguments

config
An object of class `vm_config` or `vmss_config` representing a virtual machine or scaleset deployment.

name
The VM or scaleset name. Will also be used for the domain name label, if a public IP address is included in the deployment.

login_user
An object of class `user_config` representing the login details for the admin user account on the VM.

size
The VM (instance) size.

instances
For `vmss_config`, the number of (initial) instances in the VM scaleset.

Details

These are methods for the generics defined in the AzureRMR package.

Value

Objects of class `json`, which are JSON character strings representing the deployment template and its parameters.

See Also

`create_vm, vm_config, vmss_config`

Examples

```r
cm <- ubuntu_18.04()
build_template_definition(vm)
build_template_parameters(vm, "myubuntuvm",
    user_config("username", "/.ssh/id_rsa.pub"), "Standard_DS3_v2")
```

---

**centos_7.5**

---

**Description**

VM configuration functions
Usage

```
centos_7.5(keylogin = TRUE, managed_identity = TRUE,
            datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)
```

```
centos_7.6(keylogin = TRUE, managed_identity = TRUE,
            datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)
```

```
centos_8.1(keylogin = TRUE, managed_identity = TRUE,
            datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)
```

```
derbian_8_backports(keylogin = TRUE, managed_identity = TRUE,
                     datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)
```

```
derbian_9_backports(keylogin = TRUE, managed_identity = TRUE,
                     datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)
```

```
derbian_10_backports(keylogin = TRUE, managed_identity = TRUE,
                     datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)
```

```
derbian_10_backports_gen2(keylogin = TRUE, managed_identity = TRUE,
                          datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)
```

```
ubuntu_dsvm(keylogin = TRUE, managed_identity = TRUE,
            datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh,
                                                        nsg_rule_allow_jupyter, nsg_rule_allow_rstudio)), ...)
```

```
ubuntu_dsvm_gen2(keylogin = TRUE, managed_identity = TRUE,
                 datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh,
                                                            nsg_rule_allow_jupyter, nsg_rule_allow_rstudio)), ...)
```

```
windows_dsvm(keylogin = FALSE, managed_identity = TRUE,
             datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_rdp)), ...)
```

```
rhel_7.6(keylogin = TRUE, managed_identity = TRUE,
          datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)
```

```
rhel_8(keylogin = TRUE, managed_identity = TRUE, datadisks = numeric(0),
        nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)
```

```
rhel_8.1(keylogin = TRUE, managed_identity = TRUE,
          datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)
```

```
rhel_8.1_gen2(keylogin = TRUE, managed_identity = TRUE,
               datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)
```

```
rhel_8.2(keylogin = TRUE, managed_identity = TRUE,
          datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)
```


```r
rhel_8.2_gen2(keylogin = TRUE, managed_identity = TRUE,
             datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)

ubuntu_16.04(keylogin = TRUE, managed_identity = TRUE,
              datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)

ubuntu_18.04(keylogin = TRUE, managed_identity = TRUE,
              datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)

ubuntu_20.04(keylogin = TRUE, managed_identity = TRUE,
              datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)

ubuntu_20.04_gen2(keylogin = TRUE, managed_identity = TRUE,
                   datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_ssh)), ...)

windows_2016(keylogin = FALSE, managed_identity = TRUE,
              datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_rdp)), ...)

windows_2019(keylogin = FALSE, managed_identity = TRUE,
              datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_rdp)), ...)

windows_2019_gen2(keylogin = FALSE, managed_identity = TRUE,
                   datadisks = numeric(0), nsg = nsg_config(list(nsg_rule_allow_rdp)), ...)

vm_config(image, keylogin, managed_identity = TRUE,
          os_disk_type = c("Premium_LRS", "StandardSSD_LRS", "Standard_LRS"),
          datadisks = numeric(0), nsg = nsg_config(), ip = ip_config(),
          vnet = vnet_config(), nic = nic_config(), other_resources = list(),
          variables = list(), ...)
```

**Arguments**

- **keylogin**: Whether to use an SSH public key to login (TRUE) or a password (FALSE). Note that Windows does not support SSH key logins.

- **managed_identity**: Whether to provide a managed system identity for the VM.

- **datadisks**: The data disks to attach to the VM. Specify this as either a vector of numeric disk sizes in GB, or a list of datadisk_config objects for more control over the specification.

- **nsg**: The network security group for the VM. Can be a call to nsg_config to create a new NSG; an AzureRMR resource object or resource ID to reuse an existing NSG; or NULL to not use an NSG (not recommended).

... For the specific VM configurations, other customisation arguments to be passed to vm_config. For vm_config, named arguments that will be folded into the VM resource definition in the template. In particular use properties=list(*) to set additional properties for the VM, beyond those set by the various configuration functions.
For vm_config, the VM image to deploy. This should be an object of class image_config, created by the function of the same name.

The type of primary disk for the VM. Can be "Premium_LRS" (the default), "StandardSSD_LRS", or "Standard_LRS". Of these, "Standard_LRS" uses hard disks and the others use SSDs as the underlying hardware. Change this to "StandardSSD_LRS" or "Standard_LRS" if the VM size doesn’t support premium storage.

The public IP address for the VM. Can be a call to ip_config to create a new IP address; an AzureRMR resource object or resource ID to reuse an existing address resource; or NULL if the VM should not be accessible from outside its subnet.

The virtual network for the VM. Can be a call to vnet_config to create a new virtual network, or an AzureRMR resource object or resource ID to reuse an existing virtual network. Note that by default, AzureVM will associate the NSG with the virtual network/subnet, not with the VM’s network interface.

The network interface for the VM. Can be a call to nic_config to create a new interface, or an AzureRMR resource object or resource ID to reuse an existing interface.

An optional list of other resources to include in the deployment.

An optional named list of variables to add to the template.

Details

These functions are for specifying the details of a new virtual machine deployment: the VM image and related options, along with the Azure resources that the VM may need. These include the datadisks, network security group, public IP address (if the VM is to be accessible from outside its subnet), virtual network, and network interface. vm_config is the base configuration function, and the others call it to create VMs with specific operating systems and other image details.

- ubuntu_dsvm: Data Science Virtual Machine, based on Ubuntu 18.04
- windows_dsvm: Data Science Virtual Machine, based on Windows Server 2019
- ubuntu_16.04, ubuntu_18.04, ubuntu_20.04, ubuntu_20.04_gen2: Ubuntu LTS
- rhel_7.6, rhel_8, rhel_8.1, rhel_8.1_gen2, rhel_8.2, "rhel_8.2_gen2: Red Hat Enterprise Linux
- centos_7.5, centos_7.6, centos_8.1: CentOS
-debian_8_backports, debian_9_backports, debian_10_backports, debian_10_backports_gen2: Debian with backports

The VM configurations with gen2 in the name are generation 2 VMs, which feature several technical improvements over the earlier generation 1. Consider using these for greater efficiency, however note that gen2 VMs are only available for select images and do not support all possible VM sizes. Each resource can be specified in a number of ways:

- To create a new resource as part of the deployment, call the corresponding *_config function.
• To use an existing resource, supply either an AzureRM::az_resource object (recommended)
or a string containing the resource ID.
• If the resource is not needed, specify it as NULL.
• For the other_resources argument, supply a list of resources, each of which should be a listof resource fields (name, type, properties, sku, etc).

A VM configuration defines the following template variables by default, depending on its resources.
If a particular resource is created, the corresponding *Name, *Id and *Ref variables will be available. If a resource is referred to but not created, the *Name* and *Id variables will be available. Other variables can be defined via the variables argument.

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>[resourceGroup().location]</td>
<td>Region to deploy resources</td>
</tr>
<tr>
<td>vmId</td>
<td>[resourceId('Microsoft.Compute/virtualMachines', parameters('vmName'))]</td>
<td>VM resource ID</td>
</tr>
<tr>
<td>vmRef</td>
<td>[concat('Microsoft.Compute/virtualMachines/', parameters('vmName'))]</td>
<td>VM template reference</td>
</tr>
<tr>
<td>nsgName</td>
<td>[concat(parameters('vmName'), '-nsg')]</td>
<td>Network security group name</td>
</tr>
<tr>
<td>nsgId</td>
<td>[resourceId('Microsoft.Network/networkSecurityGroups', variables('nsgName'))]</td>
<td>NSG resource ID</td>
</tr>
<tr>
<td>nsgRef</td>
<td>[concat('Microsoft.Network/networkSecurityGroups/', variables('nsgName'))]</td>
<td>NSG template reference</td>
</tr>
<tr>
<td>ipName</td>
<td>[concat(parameters('vmName'), '-ip')]</td>
<td>Public IP address resource name</td>
</tr>
<tr>
<td>ipId</td>
<td>[resourceId('Microsoft.Network/publicIPAddresses', variables('ipName'))]</td>
<td>IP resource ID</td>
</tr>
<tr>
<td>ipRef</td>
<td>[concat('Microsoft.Network/publicIPAddresses/', variables('ipName'))]</td>
<td>IP template reference</td>
</tr>
<tr>
<td>vnetName</td>
<td>[concat(parameters('vmName'), '-vnet')]</td>
<td>Virtual network resource name</td>
</tr>
<tr>
<td>vnetId</td>
<td>[resourceId('Microsoft.Network/virtualNetworks', variables('vnetName'))]</td>
<td>Vnet resource ID</td>
</tr>
<tr>
<td>vnetRef</td>
<td>[concat('Microsoft.Network/virtualNetworks/', variables('vnetName'))]</td>
<td>Vnet template reference</td>
</tr>
<tr>
<td>subnet</td>
<td>subnet</td>
<td>Subnet name. Only defined if a Vnet was created or supplied as an az_resource object.</td>
</tr>
<tr>
<td>subnetId</td>
<td>[concat(variables('vnetId'), '/subnets/', variables('subnet'))]</td>
<td>Subnet resource ID. Only defined if a Vnet was created or supplied as an az_resource object.</td>
</tr>
<tr>
<td>nicName</td>
<td>[concat(parameters('vmName'), '-nic')]</td>
<td>Network interface resource name</td>
</tr>
<tr>
<td>nicId</td>
<td>[resourceId('Microsoft.Network/networkInterfaces', variables('nicName'))]</td>
<td>NIC resource ID</td>
</tr>
<tr>
<td>nicRef</td>
<td>[concat('Microsoft.Network/networkInterfaces/', variables('nicName'))]</td>
<td>NIC template reference</td>
</tr>
</tbody>
</table>

Thus, for example, if you are creating a VM named "myvm" along with all its associated resources,
the NSG is named "myvm-nsg", the public IP address is "myvm-ip", the virtual network is "myvm-vnet", and the network interface is "myvm-nic".

Value

An object of S3 class vm_config, that can be used by the create_vm method.

See Also

image_config, user_config, datadisk_config for options relating to the VM resource itself
nsg_config, ip_config, vnet_config, nic_config for other resource configs
build_template for template builder methods
vmss_config for configuring a virtual machine scaleset
create_vm
Examples

# basic Linux (Ubuntu) and Windows configs
ubuntu_18.04()
windows_2019()

# Windows DSVM with 500GB data disk, no public IP address
windows_dsvm(datadisks=500, ip=NULL)

# RHEL VM exposing ports 80 (HTTP) and 443 (HTTPS)
rhel_8(nsg=nsg_config(nsg_rule_allow_http, nsg_rule_allow_https))

# exposing no ports externally, spot (low) priority
rhel_8(nsg=nsg_config(list()), properties=list(priority="spot"))

# deploying an extra resource: storage account
ubuntu_18.04(
    variables=list(storName=[concat(parameters('vmName'), 'stor')])),
    other_resources=list(
        list(
            type="Microsoft.Storage/storageAccounts",
            name=[variables('storName')],
            apiVersion="2018-07-01",
            location=[variables('location')],
            properties=list(supportsHttpsTrafficOnly=TRUE),
            sku=list(name="Standard_LRS"),
            kind="Storage"
        )
    )
)

## custom VM configuration: Windows 10 Pro 1903 with data disks
## this assumes you have a valid Win10 desktop license
user <- user_config("myname", password="Use-strong-passwords!")
image <- image_config(
    publisher="MicrosoftWindowsDesktop",
    offer="Windows-10",
    sku="19h1-pro"
)

datadisks <- list(
    datadisk_config(250, type="Premium_LRS"),
    datadisk_config(1000, type="Standard_LRS")
)

nsg <- nsg_config(
    list(nsg_rule_allow_rdp)
)

vm_config(
    image=image,
    keylogin=FALSE,
    datadisks=datadisks,
    nsg=nsg,
    properties=list(licenseType="Windows_Client")
)
## Not run:

# reusing existing resources: placing multiple VMs in one vnet/subnet
rg <- AzureRMR::get_azure_login()
  get_subscription("sub_id")
  get_resource_group("rgname")

vnet <- rg$get_resource(type="Microsoft.Network/virtualNetworks", name="myvnet")

# by default, the NSG is associated with the subnet, so we don't need a new NSG either
vmconfig1 <- ubuntu_18.04(vnet=vnet, nsg=NULL)
vmconfig2 <- debian_9_backports(vnet=vnet, nsg=NULL)
vmconfig3 <- windows_2019(vnet=vnet, nsg=NULL)

## End(Not run)

---

**centos_7.5_ss**

Virtual machine scaleset configuration functions

**Description**

Virtual machine scaleset configuration functions

**Usage**

```r
centos_7.5_ss(datadisks = numeric(0),
  nsg = nsg_config(list(nsg_rule_allow_ssh)),
  load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
  list(lb_probe_ssh)), ...)
```

```r
centos_7.6_ss(datadisks = numeric(0),
  nsg = nsg_config(list(nsg_rule_allow_ssh)),
  load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
  list(lb_probe_ssh)), ...)
```

```r
centos_8.1_ss(datadisks = numeric(0),
  nsg = nsg_config(list(nsg_rule_allow_ssh)),
  load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
  list(lb_probe_ssh)), ...)
```

```r
debian_8_backports_ss(datadisks = numeric(0),
  nsg = nsg_config(list(nsg_rule_allow_ssh)),
  load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
  list(lb_probe_ssh)), ...)
```
debian_9_backports_ss(datadisks = numeric(0),
    nsg = nsg_config(list(nsg_rule_allow_ssh)),
    load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
        list(lb_probe_ssh)), ...)

debian_10_backports_ss(datadisks = numeric(0),
    nsg = nsg_config(list(nsg_rule_allow_ssh)),
    load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
        list(lb_probe_ssh)), ...)

debian_10_backports_gen2_ss(datadisks = numeric(0),
    nsg = nsg_config(list(nsg_rule_allow_ssh)),
    load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
        list(lb_probe_ssh)), ...)

ubuntu_dsvm_ss(datadisks = numeric(0),
    nsg = nsg_config(list(nsg_rule_allow_ssh, nsg_rule_allow_jupyter,
        nsg_rule_allow_rstudio)), load_balancer = lb_config(rules =
        list(lb_rule_ssh, lb_rule_jupyter, lb_rule_rstudio), probes =
        list(lb_probe_ssh, lb_probe_jupyter, lb_probe_rstudio)), ...)

ubuntu_dsvm_gen2_ss(datadisks = numeric(0),
    nsg = nsg_config(list(nsg_rule_allow_ssh, nsg_rule_allow_jupyter,
        nsg_rule_allow_rstudio)), load_balancer = lb_config(rules =
        list(lb_rule_ssh, lb_rule_jupyter, lb_rule_rstudio), probes =
        list(lb_probe_ssh, lb_probe_jupyter, lb_probe_rstudio)), ...)

windows_dsvm_ss(datadisks = numeric(0),
    nsg = nsg_config(list(nsg_rule_allow_rdp)),
    load_balancer = lb_config(rules = list(lb_rule_rdp), probes =
        list(lb Probe_rdp)), options = scaleset_options(keylogin = FALSE), ...)

rhel_7.6_ss(datadisks = numeric(0),
    nsg = nsg_config(list(nsg_rule_allow_ssh)),
    load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
        list(lb Probe_ssh)), ...)

rhel_8_ss(datadisks = numeric(0),
    nsg = nsg_config(list(nsg_rule_allow_ssh)),
    load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
        list(lb Probe_ssh)), ...)

rhel_8.1_ss(datadisks = numeric(0),
    nsg = nsg_config(list(nsg_rule_allow_ssh)),
    load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
        list(lb Probe_ssh)), ...
rhel_8.1_gen2_ss(datadisks = numeric(0),
  nsg = nsg_config(list(nsg_rule_allow_ssh)),
  load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
  list(lb_probe_ssh)), ...)

rhel_8.2_gen2_ss(datadisks = numeric(0),
  nsg = nsg_config(list(nsg_rule_allow_ssh)),
  load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
  list(lb_probe_ssh)), ...)

ubuntu_16.04_ss(datadisks = numeric(0),
  nsg = nsg_config(list(nsg_rule_allow_ssh)),
  load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
  list(lb_probe_ssh)), ...)

ubuntu_18.04_ss(datadisks = numeric(0),
  nsg = nsg_config(list(nsg_rule_allow_ssh)),
  load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
  list(lb_probe_ssh)), ...)

ubuntu_20.04_ss(datadisks = numeric(0),
  nsg = nsg_config(list(nsg_rule_allow_ssh)),
  load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
  list(lb_probe_ssh)), ...)

ubuntu_20.04_gen2_ss(datadisks = numeric(0),
  nsg = nsg_config(list(nsg_rule_allow_ssh)),
  load_balancer = lb_config(rules = list(lb_rule_ssh), probes =
  list(lb_probe_ssh)), ...)

windows_2016_ss(datadisks = numeric(0),
  nsg = nsg_config(list(nsg_rule_allow_rdp)),
  load_balancer = lb_config(rules = list(lb_rule_rdp), probes =
  list(lb_probe_rdp)), options = scaleset_options(keylogin = FALSE), ...)

windows_2019_ss(datadisks = numeric(0),
  nsg = nsg_config(list(nsg_rule_allow_rdp)),
  load_balancer = lb_config(rules = list(lb_rule_rdp), probes =
  list(lb_probe_rdp)), options = scaleset_options(keylogin = FALSE), ...)

windows_2019_gen2_ss(datadisks = numeric(0),
  nsg = nsg_config(list(nsg_rule_allow_rdp)),
  load_balancer = lb_config(rules = list(lb_rule_rdp), probes =
list(lb_probe_rdp)), options = scaleset_options(keylogin = FALSE), ...)

vmss_config(image, options = scaleset_options(), datadisks = numeric(0),
nsg = nsg_config(), vnet = vnet_config(), load_balancer = lb_config(),
load_balancer_address = ip_config(), autoscaler = autoscaler_config(),
other_resources = list(), variables = list(), ...)

Arguments

datadisks The data disks to attach to the VM. Specify this as either a vector of numeric
disk sizes in GB, or a list of datadisk_config objects for more control over the
specification.

nsg The network security group for the scaleset. Can be a call to nsg_config to
create a new NSG; an AzureRMR resource object or resource ID to reuse an
existing NSG; or NULL to not use an NSG (not recommended).

load_balancer The load balancer for the scaleset. Can be a call to lb_config to create a new
load balancer; an AzureRMR resource object or resource ID to reuse an existing
load balancer; or NULL if load balancing is not required.

... For the specific VM configurations, other customisation arguments to be passed
to vm_config. For vmss_config, named arguments that will be folded into the
scaleset resource definition in the template.

options Scaleset options, as obtained via a call to scaleset_options.

image For vmss_config, the VM image to deploy. This should be an object of class
image_config, created by the function of the same name.

vnet The virtual network for the scaleset. Can be a call to vnet_config to create a new
virtual network, or an AzureRMR resource object or resource ID to reuse an
existing virtual network. Note that by default, AzureVM will associate the
NSG with the virtual network/subnet, not with the VM's network interface.

load_balancer_address The public IP address for the load balancer. Can be a call to ip_config to create a
new IP address, or an AzureRMR resource object or resource ID to reuse an
existing address resource. Ignored if load_balancer is NULL.

autoscaler The autoscaler for the scaleset. Can be a call to autoscaler_config to create a new autoscaler; an
AzureRMR resource object or resource ID to reuse an existing autoscaler; or NULL if autoscaling is not required.

other_resources An optional list of other resources to include in the deployment.

variables An optional named list of variables to add to the template.

Details

These functions are for specifying the details of a new virtual machine scaleset deployment: the
base VM image and related options, along with the Azure resources that the scaleset may need.
These include the network security group, virtual network, load balancer and associated public IP
address, and autoscaler.

Each resource can be specified in a number of ways:
- To create a new resource as part of the deployment, call the corresponding *_config function.
- To use an existing resource, supply either an AzureRMR::az_resource object (recommended) or a string containing the resource ID.
- If the resource is not needed, specify it as NULL.
- For the other_resources argument, supply a list of resources, each of which should be a list of resource fields (name, type, properties, sku, etc).

The vmss_config function is the base configuration function, and the others call it to create VM scalesets with specific operating systems and other image details.

- ubuntu_dsvm_ss: Data Science Virtual Machine, based on Ubuntu 18.04
- windows_dsvm_ss: Data Science Virtual Machine, based on Windows Server 2019
- ubuntu_16.04_ss, ubuntu_18.04_ss, ubuntu_20.04_ss, ubuntu_20.04_gen2_ss: Ubuntu LTS
- rhel_7.6_ss, rhel_8_ss, rhel_8.1_ss, rhel_8.1_gen2_ss, rhel_8.2_ss, rhel_8.2_gen2_ss: Red Hat Enterprise Linux
- centos_7.5_ss, centos_7.6_ss, centos_8.1_ss: CentOS
- debian_8_backports_ss, debian_9_backports_ss, debian_10_backports_ss, debian_10_backports_gen2_ss: Debian with backports

The VM scaleset configurations with gen2 in the name use generation 2 VMs, which feature several technical improvements over the earlier generation 1. Consider using these for greater efficiency, however note that gen2 VMs are only available for select images and do not support all possible VM sizes.

A VM scaleset configuration defines the following template variables by default, depending on its resources. If a particular resource is created, the corresponding *Name, *Id and *Ref variables will be available. If a resource is referred to but not created, the *Name* and *Id variables will be available. Other variables can be defined via the variables argument.

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>[resourceGroup().location]</td>
<td>Region to deploy VM scaleset resources.</td>
</tr>
<tr>
<td>vmId</td>
<td>[resourceId('Microsoft.Compute/virtualMachines', parameters('vmName'))]</td>
<td>Scaleset template variable.</td>
</tr>
<tr>
<td>vmRef</td>
<td>[concat('Microsoft.Compute/virtualMachines/', parameters('vmName'))]</td>
<td>Network security group resource name.</td>
</tr>
<tr>
<td>nsgName</td>
<td>[concat(parameters('vmName'), '-nsg')]</td>
<td>NSG resource ID.</td>
</tr>
<tr>
<td>nsgRef</td>
<td>[concat('Microsoft.Network/networkSecurityGroups/', variables('nsgName'))]</td>
<td>Virtual network resource name.</td>
</tr>
<tr>
<td>vnetName</td>
<td>[concat(parameters('vmName'), '-vnet')]</td>
<td>Vnet resource ID.</td>
</tr>
<tr>
<td>vnetId</td>
<td>[resourceId('Microsoft.Network/virtualNetworks', variables('vnetName'))]</td>
<td>Vnet template reference.</td>
</tr>
<tr>
<td>vnetRef</td>
<td>[concat('Microsoft.Network/virtualNetworks/', variables('vnetName'))]</td>
<td>Subnet name.</td>
</tr>
<tr>
<td>subnet</td>
<td>subnet</td>
<td>Subnet resource name.</td>
</tr>
<tr>
<td>subnetId</td>
<td>[concat(variables('vnetId'), '/subnets/', variables('subnet'))]</td>
<td>Load balancer resource name.</td>
</tr>
<tr>
<td>lbName</td>
<td>[concat(parameters('vmName'), '-lb')]</td>
<td>Load balancer frontend name.</td>
</tr>
<tr>
<td>lbId</td>
<td>[resourceId('Microsoft.Network/loadBalancers', variables('lbName'))]</td>
<td>Load balancer backend name.</td>
</tr>
<tr>
<td>lbRef</td>
<td>[concat('Microsoft.Network/loadBalancers/', variables('lbName'))]</td>
<td>Load balancer frontend name.</td>
</tr>
<tr>
<td>lbFrontendName</td>
<td>frontend</td>
<td>Load balancer backend name.</td>
</tr>
<tr>
<td>lbBackendName</td>
<td>backend</td>
<td>Load balancer frontend name.</td>
</tr>
</tbody>
</table>
Thus, for example, if you are creating a VM scaleset named "myvmss" along with all its associated resources, the NSG is named "myvmss-nsg", the virtual network is "myvmss-vnet", the load balancer is "myvmss-lb", the public IP address is "myvmss-ip", and the autoscaler is "myvm-as".

Value

An object of S3 class `vmss_config`, that can be used by the `create_vm_scaleset` method.

See Also

`scaleset_options` for options relating to the scaleset resource itself

`nsg_config`, `ip_config`, `vnet_config`, `lb_config`, `autoscaler_config` for other resource configs

`build_template` for template builder methods

`vm_config` for configuring an individual virtual machine

`create_vm_scaleset`

Examples

# basic Linux (Ubuntu) and Windows configs
ubuntu_18.04_ss()
windows_2019_ss()

# Windows DSVM scaleset, no load balancer and autoscaler
windows_dsvm_ss(load_balancer=NULL, autoscaler=NULL)

# RHEL VM exposing ports 80 (HTTP) and 443 (HTTPS)
rhel_8_ss(nsg=nsg_config(nsg_rule_allow_http, nsg_rule_allow_https))

# exposing no ports externally
rhel_8_ss(nsg=nsg_config(list()))

# low-priority (spot) VMs, large scaleset (>100 instances allowed), no managed identity
rhel_8_ss(options=scaleset_options(priority="spot", large_scaleset=TRUE, managed_identity=FALSE))

## Not run:
# Create a new virtual machine or scaleset of virtual machines

**Description**

Method for the `AzureRMR::az_subscription` and `AzureRMR::az_resource_group` classes.

**Usage**

```r
## R6 method for class 'az_resource_group'
create_vm(name, login_user, size = "Standard_DS3_v2", config = "ubuntu_dsvm",
          managed_identity = TRUE, datadisks = numeric(0), ...
          template, parameters, mode = "Incremental", wait = TRUE)

## R6 method for class 'az_subscription'
create_vm(name, ..., resource_group = name, location)

## R6 method for class 'az_resource_group'
create_vm_scaleset(name, login_user, instances, size = "Standard_DS1_v2",
                   config = "ubuntu_dsvm_ss", ...
                   template, parameters, mode = "Incremental", wait = TRUE)

## R6 method for class 'az_subscription'
create_vm_scaleset(name, ..., resource_group = name, location)
```

**Arguments**

- **name**: The name of the VM or scaleset.
- **location**: For the subscription methods, the location for the VM or scaleset. Use the `list_locations()` method of the `AzureRMR::az_subscription` class to see what locations are available.
- **resource_group**: For the subscription methods, the resource group in which to place the VM or scaleset. Defaults to a new resource group with the same name as the VM.
- **login_user**: The details for the admin login account. An object of class `user_config`, obtained by a call to the `user_config` function.
- **size**: The VM (instance) size. Use the `list_vm_sizes` method to see what sizes are available.
create_vm

- **config**: The VM or scaleset configuration. See 'Details' below for how to specify this. The default is to use an Ubuntu Data Science Virtual Machine.
- **managed_identity**: For create_vm, whether the VM should have a managed identity attached.
- **datadisks**: Any data disks to attach to the VM or scaleset. See 'Details' below.
- **instances**: For create_vm_scaleset, the initial number of instances in the scaleset.
- **...**: For the subscription methods, any of the other arguments listed here, which will be passed to the resource group method. For the resource group method, additional arguments to pass to the VM/scaleset configuration functions `vm_config` and `vmss_config`. See the examples below.
- **template,parameters**: The template definition and parameters to deploy. By default, these are constructed from the values of the other arguments, but you can supply your own template and/or parameters as well.
- **wait**: Whether to wait until the deployment is complete.
- **mode**: The template deployment mode. If "Complete", any existing resources in the resource group will be deleted.

**Details**

These methods deploy a template to create a new virtual machine or scaleset.

The config argument can be specified in the following ways:

- As the name of a supplied VM or scaleset configuration, like "ubuntu_dsvm" or "ubuntu_dsvm_ss". AzureVM comes with a number of supplied configurations to deploy commonly used images, which can be seen at `vm_config` and `vmss_config`. Any arguments in ... will be passed to the configuration, allowing you to customise the deployment.
- As a call to the `vm_config` or `vmss_config` functions, to deploy a custom VM image.
- As an object of class `vm_config` or `vmss_config`.

The data disks for the VM can be specified as either a vector of numeric disk sizes in GB, or as a list of `datadisk_config` objects, created via calls to the `datadisk_config` function. Currently, AzureVM only supports creating data disks at deployment time for single VMs, not scalesets.

You can also supply your own template definition and parameters for deployment, via the template and parameters arguments. See `AzureRMR::az_template` for information how to create templates.

The `AzureRMR::az_subscription` methods will by default create the VM in `exclusive` mode, meaning a new resource group is created solely to hold the VM or scaleset. This simplifies managing the VM considerably; in particular deleting the resource group will also automatically delete all the deployed resources.

**Value**

For `create_vm`, an object of class `az_vm_template` representing the created VM. For `create_vm_scaleset`, an object of class `az_vmss_template` representing the scaleset.
See Also

- `az_vm_template`, `az_vmss_template`
- `vm_config`, `vmss_config`, `user_config`, `datadisk_config`
- `AzureRMR::az_subscription`, `AzureRMR::az_resource_group`, Data Science Virtual Machine

Examples

```r
# Not run:

sub <- AzureRMR::get_azure_login()
  get_subscription("subscription_id")

# default Ubuntu 18.04 VM:
# SSH key login, Standard_DS3_v2, publicly accessible via SSH
sub$create_vm("myubuntuvm", user_config("myname", "~/.ssh/id_rsa.pub"),
  location="australiaeast")

# Windows Server 2019, with a 500GB datadisk attached, not publicly accessible
sub$create_vm("mywinvm", user_config("myname", password="Use-strong-passwords!"),
  size="Standard_DS4_v2", config="windows_2019", datadisks=500, ip=NULL,
  location="australiaeast")

# Ubuntu D SVM, GPU-enabled
sub$create_vm("mydsvm", user_config("myname", "~/.ssh/id_rsa.pub"), size="Standard_NC12",
  config="ubuntu_dsvm_ss",
  location="australiaeast")

## custom VM configuration: Windows 10 Pro 1903 with data disks
## this assumes you have a valid Win10 desktop license
user <- user_config("myname", password="Use-strong-passwords!

image <- image_config(
  publisher="MicrosoftWindowsDesktop",
  offer="Windows-10",
  sku="19h1-pro"
)

datadisks <- list(
  datadisk_config(250, type="Premium_LRS"),
  datadisk_config(1000, type="Standard_LRS")
)

nsg <- nsg_config(
  list(nsg_rule_allow_rdp)
)

config <- vm_config(
  image=image,
  keylogin=FALSE,
  datadisks=datadisks,
  nsg=nsg,
  properties=list(licenseType="Windows_Client")
)

sub$create_vm("mywin10vm", user, size="Standard_DS2_v2", config=config,
  location="australiaeast")
```
# default Ubuntu scaleset:
# load balancer and autoscaler enabled, Standard_DS1_v2
sub$create_vm_scaleset("mydsvmss", user_config("myname", "/.ssh/id_rsa.pub"),
    instances=5,
    location="australiaeast")

# Ubuntu DSVM scaleset with public GPU-enabled instances, no load balancer or autoscaler
sub$create_vm_scaleset("mydsvmss", user_config("myname", "/.ssh/id_rsa.pub"),
    instances=5, size="Standard_NC12", config="ubuntu_dsvm_ss",
    options=scaleset_options(public=TRUE),
    load_balancer=NULL, autoscaler=NULL,
    location="australiaeast")

# RHEL scaleset, allow http/https access
sub$create_vm_scaleset("myrhelss", user_config("myname", "/.ssh/id_rsa.pub"),
    instances=5, config="rhel_8_ss",
    nsg=nsg_config(list(nsg_rule_allow_http, nsg_rule_allow_https)),
    location="australiaeast")

# Large Debian scaleset, using low-priority (spot) VMs
# need to set the instance size to something that supports low-pri
sub$create_vm_scaleset("mydebss", user_config("myname", "/.ssh/id_rsa.pub"),
    instances=50, size="Standard_DS3_v2", config="debian_9_backports_ss",
    options=scaleset_options(priority="spot", large_scaleset=TRUE),
    location="australiaeast")

## VM and scaleset in the same resource group and virtual network
# first, create the resgroup
rg <- sub$create_resource_group("rgname", "australiaeast")

# create the master
rg$create_vm("mastervm", user_config("myname", "/.ssh/id_rsa.pub"))

# get the vnet resource
vnet <- rg$get_resource(type="Microsoft.Network/virtualNetworks", name="mastervm-vnet")

# create the scaleset
rg$create_vm_scaleset("slavess", user_config("myname", "/.ssh/id_rsa.pub"),
    instances=5, vnet=vnet, nsg=NULL, load_balancer=NULL, autoscaler=NULL)

## End(Not run)

---

**defunct**

---

### Description

Defunct methods
delete_vm

Usage

get_vm_cluster(...)  
create_vm_cluster(...)  
delete_vm_cluster(...)

These methods for the az_subscription and az_resource_group classes are defunct in AzureVM 2.0. To work with virtual machine clusters, call the get_vm_scaleset, create_vm_scaleset and delete_vm_scaleset methods instead.

Description

Method for the AzureRMR::az_subscription and AzureRMR::az_resource_group classes.

Usage

## R6 method for class 'az_resource_group'
delete_vm(name, confirm = TRUE, free_resources = TRUE)

## R6 method for class 'az_subscription'
delete_vm(name, confirm = TRUE, free_resources = TRUE,  
resource_group = name)

## R6 method for class 'az_resource_group'
delete_vm_scaleset(name, confirm = TRUE, free_resources = TRUE)

## R6 method for class 'az_subscription'
delete_vm_scaleset(name, confirm = TRUE, free_resources = TRUE,  
resource_group = name)

Arguments

- name: The name of the VM or scaleset.
- confirm: Whether to confirm the delete.
- free_resources: If this was a deployed template, whether to free all resources created during the deployment process.
- resource_group: For the AzureRMR::az_subscription method, the resource group containing the VM or scaleset.

Details

For the subscription methods, deleting the VM or scaleset will also delete its resource group.

See Also

create_vm, az_vm_template, az_vm_resource, AzureRMR::az_subscription, AzureRMR::az_resource_group
### Examples

```r
## Not run:

sub <- AzureRMR::get_azure_login()
get_subscription("subscription_id")

sub$delete_vm("myvm")
sub$delete_vm_scaleset("myscaleset")

## End(Not run)
```

---

**get_vm**

Get existing virtual machine(s)

---

### Description

Method for the `AzureRMR::az_subscription` and `AzureRMR::az_resource_group` classes.

### Usage

```r
## R6 method for class 'az_subscription'
get_vm(name, resource_group = name)

## R6 method for class 'az_resource_group'
get_vm(name)

## R6 method for class 'az_subscription'
get_vm_scaleset(name, resource_group = name)

## R6 method for class 'az_resource_group'
get_vm_scaleset(name)

## R6 method for class 'az_resource_group'
get_vm_resource(name)
get_vm_scaleset_resource(name)
```

### Arguments

- `name`: The name of the VM or scaleset.
- `resource_group`: For the `az_subscription` methods, the resource group in which `get_vm()` and `get_vm_scaleset()` will look for the VM or scaleset. Defaults to the VM name.
Value

For `get_vm()`, an object representing the VM deployment. This will include other resources besides the VM itself, such as the network interface, virtual network, etc.

For `get_vm_scaleset()`, an object representing the scaleset deployment. Similarly to `get_vm()`, this includes other resources besides the scaleset.

For `get_vm_resource()` and `get_vm_scaleset_resource()`, the VM or scaleset resource itself.

See Also

`az_vm_template`, `az_vm_resource`, `az_vmss_template`, `az_vmss_resource` for the methods available for working with VMs and VM scalesets.

`AzureRMR::az_subscription`, `AzureRMR::az_resource_group`

Examples

```r
## Not run:
sub <- AzureRMR::get_azure_login()
  get_subscription("subscription_id")

sub$get_vm("myvirtualmachine")
sub$get_vm_scaleset("myscaleset")

rg <- sub$get_resource_group("rgname")
rg$get_vm("myothervirtualmachine")
rg$get_vm_scaleset("myotherscaleset")

## End(Not run)
```

---

**ip_config**

*Public IP address configuration*

Description

Public IP address configuration

Usage

```
ip_config(type = NULL, dynamic = NULL, ipv6 = FALSE,
  domain_name = "[parameters('vmName')]", ...)```
is_vm

Arguments

- **type**: The SKU of the IP address resource: "basic" or "standard". If NULL (the default), this will be determined based on the VM's configuration.
- **dynamic**: Whether the IP address should be dynamically or statically allocated. Note that the standard SKU only supports standard allocation. If NULL (the default) this will be determined based on the VM's configuration.
- **ipv6**: Whether to create an IPv6 address. The default is IPv4.
- **domain_name**: The domain name label to associate with the address.
- **...**: Other named arguments that will be treated as resource properties.

See Also

create_vm, vm_config, vmss_config

Examples

```r
ip_config()
ip_config(type="basic", dynamic=TRUE)

# if you don't want a domain name associated with the IP address
ip_config(domain_name=NULL)
```

---

is_vm

Is an object an Azure VM

Description

Is an object an Azure VM

Usage

- `is_vm(object)`
- `is_vm_template(object)`
- `is_vm_resource(object)`
- `is_vm_scaleset(object)`
- `is_vm_scaleset_template(object)`
- `is_vm_scaleset_resource(object)`

Arguments

- **object**: an R object.
Value

is_vm and is_vm_template return TRUE for an object representing a virtual machine deployment (which will include other resources besides the VM itself).

is_vm_resource returns TRUE for an object representing the specific VM resource.

is_vm_scaleset and is_vm_scaleset_template return TRUE for an object representing a VM scaleset deployment.

is_vm_scaleset_resource returns TRUE for an object representing the specific VM scaleset resource.

See Also

create_vm, create_vm_scaleset, az_vm_template, az_vm_resource, az_vmss_template, az_vmss_resource

lb_config

Load balancer configuration

Description

Load balancer configuration

Usage

lb_config(type = NULL, rules = list(), probes = list(), ...)

lb_probe(name, port, interval = 5, fail_on = 2, protocol = "Tcp")

lb_rule(name, frontend_port, backend_port = frontend_port,
          protocol = "Tcp", timeout = 5, floating_ip = FALSE, probe_name)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>The SKU of the load balancer resource: &quot;basic&quot; or &quot;standard&quot;. If NULL (the default), this will be determined based on the VM scaleset’s configuration. Note that the load balancer SKU must be the same as that of its public IP address.</td>
</tr>
<tr>
<td>rules</td>
<td>A list of load balancer rules, each obtained via a call to lb_rule.</td>
</tr>
<tr>
<td>probes</td>
<td>A list of health checking probes, each obtained via a call to lb_probe. There must be a probe corresponding to each rule.</td>
</tr>
<tr>
<td>...</td>
<td>Other named arguments that will be treated as resource properties.</td>
</tr>
<tr>
<td>name</td>
<td>For lb_rule, a name for the load balancing rule.</td>
</tr>
<tr>
<td>port</td>
<td>For lb_probe, the port to probe.</td>
</tr>
<tr>
<td>interval</td>
<td>For lb_probe, the time interval between probes in seconds.</td>
</tr>
<tr>
<td>fail_on</td>
<td>For lb_probe, the probe health check will fail after this many non-responses.</td>
</tr>
<tr>
<td>protocol</td>
<td>For lb_probe and lb_rule, the protocol: either &quot;Tcp&quot; or &quot;Ip&quot;.</td>
</tr>
</tbody>
</table>
lb_rule_ssh

frontend_port, backend_port
For lb_rule, the ports for this rule.

timeout
The timeout interval for the rule. The default is 5 minutes.

floating_ip
Whether to use floating IP addresses (direct server return). Only needed for specific scenarios, and when the frontend and backend ports don’t match.

probe_name
The name of the corresponding health check probe.

See Also

create_vm_scaleset, vmss_config, lb_rules for some predefined load balancing rules and probes

Examples

```
lb_config()
lb_config(type="basic")
lb_config(
    rules=list(lb_rule_ssh, lb_rule_rdp),
    probes=list(lb_probe_ssh, lb_probe_rdp)
)
```

Description

Load balancing rules

Usage

```
lb_rule_ssh
lb_rule_http
lb_rule_https
lb_rule_rdp
lb_rule_jupyter
lb_rule_rstudio
lb_rule_mssql
lb_rule_mssql_browser
lb_probe_ssh
```
lb_probe_http
lb_probe_https
lb_probe_rdp
lb_probe_jupyter
lb_probe_rstudio
lb_probe_mssql
lb_probe_mssql_browser

Format
Objects of class lb_rule and lb_probe.
An object of class lb_rule of length 2.
An object of class lb_rule of length 2.
An object of class lb_rule of length 2.
An object of class lb_rule of length 2.
An object of class lb_rule of length 2.
An object of class lb_rule of length 2.
An object of class lb_rule of length 2.
An object of class lb_probe of length 2.
An object of class lb_probe of length 2.
An object of class lb_probe of length 2.
An object of class lb_probe of length 2.
An object of class lb_probe of length 2.
An object of class lb_probe of length 2.
An object of class lb_probe of length 2.
An object of class lb_probe of length 2.

Details
Some predefined load balancing objects, for commonly used ports. Each load balancing rule comes with its own health probe.

- HTTP: TCP port 80
- HTTPS: TCP port 443
- JupyterHub: TCP port 8000
- RDP: TCP port 3389
- RStudio Server: TCP port 8787
list_vm_sizes

- SSH: TCP port 22
- SQL Server: TCP port 1433
- SQL Server browser service: TCP port 1434

See Also

lb_config

list_vm_sizes  List available VM sizes

Description

Method for the AzureRMR::az_subscription and AzureRMR::az_resource_group classes.

Usage

```r
## R6 method for class 'az_subscription'
list_vm_sizes(location, name_only = FALSE)

## R6 method for class 'az_resource_group'
list_vm_sizes(name_only = FALSE)
```

Arguments

- **location**: For the subscription class method, the location/region for which to obtain available VM sizes.
- **name_only**: Whether to return only a vector of names, or all information on each VM size.

Value

If name_only is TRUE, a character vector of names, suitable for passing to create_vm. If FALSE, a data frame containing the following information for each VM size: the name, number of cores, OS disk size, resource disk size, memory, and maximum data disks.

See Also

create_vm

Examples

```r
## Not run:
sub <- AzureRMR::get_azure_login$get_subscription("subscription_id")
sub$list_vm_sizes("australiaeast")
```
# same output as above
rg <- sub$create_resource_group("rgname", location="australiaeast")
rg$list_vm_sizes()

## End(Not run)

---

### nic_config

#### Network Interface Configuration

**Description**

Network interface configuration

**Usage**

```r
nic_config(nic_ip = list(nic_ip_config()), ...)

nic_ip_config(name = "ipconfig", private_alloc = "dynamic",
               subnet = "[variables('subnetId')]",
               public_address = "[variables('ipId')]", ...)
```

**Arguments**

- **nic_ip**
  - For `nic_config`, a list of IP configuration objects, each obtained via a call to `nic_ip_config`.
  - Other named arguments that will be treated as resource properties.
- **name**
  - For `nic_ip_config`, the name of the IP configuration.
- **private_alloc**
  - For `nic_ip_config`, the allocation method for a private IP address. Can be "dynamic" or "static".
- **subnet**
  - For `nic_ip_config`, the subnet to associate with this private IP address.
- **public_address**
  - For `nic_ip_config`, the public IP address. Defaults to the public IP address created or used as part of this VM deployment. Ignored if the deployment does not include a public address.

**See Also**

- `create_vm`, `vm_config`

**Examples**

```r
nic_config()
```
Description

Network security group configuration

Usage

```r
nsg_config(rules = list(), ...)
```

```r
nsg_rule(name, dest_port = "*", dest_addr = "*", dest_asgs = NULL,
          source_port = "*", source_addr = "*", source_asgs = NULL,
          access = "allow", direction = "inbound", protocol = "Tcp",
          priority = NULL)
```

Arguments

- `rules` for `nsg_config`, a list of security rule objects, each obtained via a call to `nsg_rule`.
- `...` Other named arguments that will be treated as resource properties.
- `name` For `nsg_rule`, a name for the rule.
- `dest_port, dest_addr, dest_asgs` For `nsg_rule`, the destination port, address range, and application security groups for a rule.
- `source_port, source_addr, source_asgs` For `nsg_rule`, the source port, address range, and application security groups for a rule.
- `access` For `nsg_rule`, the action to take: "allow" or "deny".
- `direction` For `nsg_rule`, the direction of traffic: "inbound" or "outbound".
- `protocol` For `nsg_rule`, the network protocol: either "Tcp" or "Udp".
- `priority` For `nsg_rule`, the rule priority. If NULL, this will be set automatically by AzureVM.

See Also

`create_vm, vm_config, vmss_config, nsg_rules` for some predefined security rules

Examples

```r
nsg_config()
nsg_config(list(nsg_rule_allow_ssh)) # for Linux
nsg_config(list(nsg_rule_allow_rdp)) # for Windows
nsg_config(list(nsg_rule_allow_http, nsg_rule_allow_https))
```

# a custom rule
```
```
nsg_rule_allow_ssh

nsg_rule_allow_ssh
nsg_rule_allow_http
nsg_rule_allow_https
nsg_rule_allow_rdp
nsg_rule_allow_jupyter
nsg_rule_allow_rstudio
nsg_rule_allow_mssql
nsg_rule_allow_mssql_browser

Description

Network security rules

Usage

nsg_rule_allow_ssh
nsg_rule_allow_http
nsg_rule_allow_https
nsg_rule_allow_rdp
nsg_rule_allow_jupyter
nsg_rule_allow_rstudio
nsg_rule_allow_mssql
nsg_rule_allow_mssql_browser

Format

Objects of class nsg_rule.
An object of class nsg_rule of length 2.
An object of class nsg_rule of length 2.
An object of class nsg_rule of length 2.
An object of class nsg_rule of length 2.
An object of class nsg_rule of length 2.
An object of class nsg_rule of length 2.
An object of class nsg_rule of length 2.
Details

Some predefined network security rule objects, to unblock commonly used ports.

- HTTP: TCP port 80
- HTTPS: TCP port 443
- JupyterHub: TCP port 8000
- RDP: TCP port 3389
- RStudio Server: TCP port 8787
- SSH: TCP port 22
- SQL Server: TCP port 1433
- SQL Server browser service: TCP port 1434

See Also

nsg_config

scaleset_options

Virtual machine scaleset options

Description

Virtual machine scaleset options

Usage

scaleset_options(keylogin = TRUE, managed_identity = TRUE,
public = FALSE, priority = c("regular", "spot"),
delete_on_evict = FALSE, network_accel = FALSE, large_scaleset = FALSE,
overprovision = TRUE, upgrade_policy = list(mode = "manual"),
os_disk_type = c("Premium_LRS", "StandardSSD_LRS", "Standard_LRS"))

Arguments

keylogin Whether to use an SSH public key to login (TRUE) or a password (FALSE). Note that Windows does not support SSH key logins.
managed_identity Whether to provide a managed system identity for the VM.
public Whether the instances (nodes) of the scaleset should be visible from the public internet.
priority The priority of the VM scaleset, either regular or spot. Spot VMs are considerably cheaper but subject to eviction if other, higher-priority workloads require compute resources.
delete_on_evict If spot-priority VMs are being used, whether evicting (shutting down) a VM should delete it, as opposed to just deallocating it.
network_accel  Whether to enable accelerated networking. This option is only available for certain VM sizes.
large_scaleset  Whether to enable scaleset sizes > 100 instances.
overprovision  Whether to overprovision the scaleset on creation.
upgrade_policy  A list, giving the VM upgrade policy for the scaleset.
os_disk_type  The type of primary disk for the VM. Change this to "StandardSSD_LRS" or "Standard_LRS" if the VM size doesn’t support premium storage.

---

**user_config**  
Resource configuration functions for a virtual machine deployment

**Description**
Resource configuration functions for a virtual machine deployment

**Usage**

```
user_config(username, sshkey = NULL, password = NULL)

datadisk_config(size, name = "datadisk", create = "empty",
    type = c("StandardSSD_LRS", "Premium_LRS", "Standard_LRS", "UltraSSD_LRS"),
    write_accelerator = FALSE)

image_config(publisher = NULL, offer = NULL, sku = NULL,
    version = "latest", id = NULL)
```

**Arguments**

- **username**  For user_config, the name for the admin user account.
- **sshkey**  For user_config, the SSH public key. This can be supplied in a number of ways: as a string with the key itself; the name of the public key file; or an AzureRMR::az_resource object pointing to an SSH public key resource (of type "Microsoft.Compute/sshPublicKeys"). See the examples below.
- **password**  For user_config, the admin password. Supply either sshkey or password, but not both; also, note that Windows does not support SSH logins.
- **size**  For datadisk_config, the size of the data disk in GB. Set this to NULL for a disk that will be created from an image.
- **name**  For datadisk_config, the disk name. Duplicate names will automatically be disambiguated prior to VM deployment.
- **create**  For datadisk_config, the creation method. Can be "empty" (the default) to create a blank disk, or "fromImage" to use an image.
- **type**  For datadisk_config, the disk type (SKU). Can be "Standard_LRS", "StandardSSD_LRS" (the default), "Premium_LRS" or "UltraSSD_LRS". Of these, "Standard_LRS" uses hard disks and the others use SSDs as the underlying hardware.
vnet_config

### write_accelerator
For `datadisk_config`, whether the disk should have write acceleration enabled.

### publisher, offer, sku, version
For `image_config`, the details for a marketplace image.

### id
For `image_config`, the resource ID for a disk to use as a custom image.

#### Examples

```r
# Not run:

## user_config: SSH public key resource in Azure
# create the resource
keyres <- rg$create_resource(type="Microsoft.Compute/sshPublicKeys", name="mysshkey")

# generate the public and private keys
keys <- keyres$do_operation("generateKeyPair", http_verb="POST")
keyres$sync_fields()

# save the private key (IMPORTANT)
writeBin(keys$privateKey, "mysshkey.pem")

# create a new VM using the public key resource for authentication
# you can then login to the VM with ssh -i mysshkey.pem <username@vmaddress>
rg$create_vm("myvm", user_config("username", sshkey=keyres), config="ubuntu_20.04")

## user_config: SSH public key as a file
rg$create_vm("myvm", user_config("username", sshkey="mysshkey.pub"), config="ubuntu_20.04")

## user_config: SSH public key as a string (read from a file)
pubkey <- readLines("mysshkey.pub")
rg$create_vm("myvm", user_config("username", sshkey=pubkey), config="ubuntu_20.04")
```

## End(Not run)

---

### vnet_config

**Virtual network configuration**

#### Description

Virtual network configuration

#### Usage

```r
vnet_config(address_space = "10.0.0.0/16", subnets = list(subnet_config()),
...
)```
subnet_config(name = "subnet", addresses = "10.0.0.0/16",
               nsg = "[variables('nsgId')]", ...)
Index

* datasets
  lb_rule_ssh, 33
  nsg_rule_allow_ssh, 38
autoscaler_config, 2, 23
autoscaler_profile (autoscaler_config), 2
az_vm_resource, 4, 6, 7, 10, 28, 30, 32
az_vm_template, 9, 9, 26, 28, 30, 32
az_vmss_resource, 3, 6, 30, 32
az_vmss_template, 5, 5, 26, 30, 32
AzureRMR::az_resource, 4–6, 8–10
AzureRMR::az_resource_group, 24, 26, 28–30, 35
AzureRMR::az_subscription, 24, 26, 28–30, 35
AzureRMR::init_pool, 5, 7
build_template, 16, 23
build_template
  (build_template_definition.vm_config), 11
build_template_definition.vm_config, 11
build_template_definition.vmss_config
  (build_template_definition.vm_config), delete_vm_scaleset (delete_vm), 28
build_template_parameters.vm_config
  (build_template_definition.vm_config), 11
build_template_parameters.vmss_config
  (build_template_definition.vm_config), 11
centos_7.5, 12
centos_7.5_ss, 18
centos_7.6 (centos_7.5), 12
centos_7.6 ss (centos_7.5 ss), 18
centos_8.1 (centos_7.5), 12
centos_8.1 ss (centos_7.5 ss), 18
create_vm, 10, 12, 16, 24, 28, 31, 32, 35–37, 42
create_vm_cluster (defunct), 27
create_vm_scaleset, 3, 7, 23, 28, 32, 33
create_vm_scaleset (create_vm), 24
datadisk_config, 16, 26
datadisk_config (user_config), 40
debian_10_backports (centos_7.5), 12
debian_10_backports_gen2 (centos_7.5), 12
debian_10_backports_gen2 ss
  (centos_7.5 ss), 18
debian_10_backports_ss (centos_7.5 ss), 18
debian_8_backports (centos_7.5), 12
debian_8_backports_ss (centos_7.5 ss), 18
debian_9_backports (centos_7.5), 12
debian_9_backports_ss (centos_7.5 ss),
defunct, 27
delete_vm, 10, 28
delete_vm_cluster (defunct), 27
delete_vm_scaleset, 7, 28
get_vm, 10, 29
get_vm_cluster (defunct), 27
get_vm_resource, 9
get_vm_resource (get_vm), 29
get_vm_scaleset, 7, 28
get_vm_scaleset (get_vm), 29
get_vm_scaleset_resource, 5
get_vm_scaleset_resource (get_vm), 29
image_config, 16
image_config (user_config), 40
ip_config, 16, 23, 30