Package 'AzureContainers'

May 6, 2026

Title Interface to 'Container Instances', 'Docker Registry' and 'Kubernetes' in 'Azure'

Version 1.3.3

Description An interface to container functionality in Microsoft's 'Azure' cloud: <https://azure.microsoft.com/en-us/products/category/containers/>. Manage 'Azure Container Instance' (ACI), 'Azure Container Registry' (ACR) and 'Azure Kubernetes Service' (AKS) resources, push and pull images, and deploy services. On the client side, lightweight shells to the 'docker', 'docker-compose', 'kubectl' and 'helm' commandline tools are provided. Part of the 'AzureR' family of packages.

URL https://github.com/Azure/AzureContainers
https://github.com/Azure/AzureR

BugReports https://github.com/Azure/AzureContainers/issues

License MIT + file LICENSE

VignetteBuilder knitr

Depends R (>= 3.3)

Imports utils, AzureRMR (>= 2.0.0), AzureGraph (>= 1.1.0), openssl, httr, R6, processx

Suggests knitr, rmarkdown, testthat, uuid, MASS, bcrypt, randomForest, plumber, RestRserve, AzureKeyVault

RoxygenNote 7.3.1

NeedsCompilation no

Author Hong Ooi [aut, cre],
Bill Liang [ctb] (Assistance debugging MMLS on Kubernetes),
Ramkumar Chandrasekaran [ctb] (Original blog article on Dockerising MMLS),
Microsoft [cph]

Maintainer Hong Ooi <hongooi73@gmail.com>

Repository CRAN

Date/Publication 2025-04-12 08:30:08 UTC

Contents

`aci` *Azure Container Instance class*

Description

Class representing an Azure Container Instance (ACI) resource.

Methods

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

- `new(...)`: Initialize a new ACI object.
- `restart()`, `start()`: Start a stopped container. These methods are synonyms for each other.
- `stop()`: Stop a container.

Details

Initializing a new object of this class can either retrieve an existing ACI resource, or create a new resource on the host. Generally, the best way to initialize an object is via the `get_aci`, `create_aci` or `list_acis` methods of the AzureRMR::az_resource_group class, which handle the details automatically.

See Also

acr, aks

ACI documentation and API reference

Docker commandline reference

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

myaci <- rg$get_aci("mycontainer")

myaci$stop()
myaci$restart()


## End(Not run)
```

---

aci_ports *Utilities for specifying ACI configuration information*

Description

Utilities for specifying ACI configuration information

Usage

```
aci_ports(port = c(80L, 443L), protocol = "TCP")

aci_creds(server, username = NULL, password = NULL)

get_aci_credentials_list(lst)
```

Arguments

`port`, `protocol`	For `aci_ports`, vectors of the port numbers and protocols to open for the instance.
`server`, `username`, `password`	For `aci_creds`, the authentication details for a Docker registry. See docker_registry.
`lst`	for `get_aci_credentials_list`, a list of objects.

Details

These are helper functions to be used in specifying the configuration for a container instance. Only `aci_ports` and `aci_creds` are meant to be called by the user; `get_aci_credentials_list` is exported to workaround namespacing issues on startup.

See Also

create_aci, aci, docker_registry

---

acr *Azure Container Registry class*

Description

Class representing an Azure Container Registry (ACR) resource. For working with the registry endpoint itself, including uploading and downloading images etc, see docker_registry.

Methods

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

- `new(...)`: Initialize a new ACR object. See 'Details'.
- `add_role_assignment(principal, role, scope=NULL, ...)`: Adds a role for the specified principal. This is an override mainly to handle AKS objects, so that the Kubernetes cluster can be granted access to the registry. You can use the `...` arguments to supply authentication details for AzureGraph, which is used to retrieve the cluster service principal.
- `list_credentials`: Return the username and passwords for this registry. Only valid if the Admin user for the registry has been enabled.
- `list_policies`: Return the policies for this registry.
- `list_usages`: Return the usage for this registry.
- `get_docker_registry(username, password)`: Return an object representing the Docker registry endpoint.

Details

Initializing a new object of this class can either retrieve an existing registry resource, or create a new registry on the host. Generally, the best way to initialize an object is via the `get_acr`, `create_acr` or `list_acrs` methods of the AzureRMR::az_resource_group class, which handle the details automatically.

Note that this class is separate from the Docker registry itself. This class exposes methods for working with the Azure resource: listing credentials, updating resource tags, updating and deleting the resource, and so on.

For working with the registry, including uploading and downloading images, updating tags, deleting layers and images etc, use the endpoint object generated with `get_docker_registry`. This method takes two optional arguments:

- `username`: The username that Docker will use to authenticate with the registry.
- `password`: The password that Docker will use to authenticate with the registry.

By default, these arguments will be retrieved from the ACR resource. They will only exist if the resource was created with `admin_user_enabled=TRUE`. Currently AzureContainers does not support authentication methods other than a username/password combination.

See Also

create_acr, get_acr, delete_acr, list_acrs

docker_registry for interacting with the Docker registry endpoint

Azure Container Registry and API reference

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

myacr <- rg$get_acr("myregistry")

myacr$list_credentials()
myacr$list_policies()

# see who has push and pull access
myacr$list_role_assignments()

# grant a Kubernetes cluster pull access
myaks <- rg$get_aks("myaks")
myacr$add_role_assignment(myaks, "Acrpull")

# get the registry endpoint (for interactive use)
myacr$get_docker_registry()

# get the registry endpoint (admin user account)
```

```
myacr$get_docker_registry(as_admin=TRUE)


## End(Not run)
```

---

agent_pool *Utility function for specifying Kubernetes agent pools*

Description

Utility function for specifying Kubernetes agent pools

Usage

```
agent_pool(name, count, size = "Standard_DS2_v2", os = "Linux",
  disksize = 0, use_scaleset = TRUE, low_priority = FALSE,
  autoscale_nodes = FALSE, ...)
```

Arguments

name	The name(s) of the pool(s).
count	The number of nodes per pool.
size	The VM type (size) to use for the pool. To see a list of available VM sizes, use the list_vm_sizes method for the resource group or subscription classes.
os	The operating system to use for the pool. Can be "Linux" or "Windows".
disksize	The OS disk size in gigabytes for each node in the pool. A value of 0 means to use the default disk size for the VM type.
use_scaleset	Whether to use a VM scaleset instead of individual VMs for this pool. A scaleset offers greater flexibility than individual VMs, and is the recommended method of creating an agent pool.
low_priority	If this pool uses a scaleset, whether it should be made up of spot (low-priority) VMs. A spot VM pool is cheaper, but is subject to being evicted to make room for other, higher-priority workloads. Ignored if use_scaleset=FALSE.
autoscale_nodes	The cluster autoscaling parameters for the pool. To enable autoscaling, set this to a vector of 2 numbers giving the minimum and maximum size of the agent pool. Ignored if use_scaleset=FALSE.
...	Other named arguments, to be used as parameters for the agent pool.

Details

agent_pool is a convenience function to simplify the task of specifying the agent pool for a Kubernetes cluster.

Value

An object of class `agent_pool`, suitable for passing to the `create_aks` constructor method.

See Also

create_aks, list_vm_sizes

Agent pool parameters on Microsoft Docs

Examples

```
# pool of 5 Linux GPU-enabled VMs
agent_pool("pool1", 5, size="Standard_NC6s_v3")

# pool of 3 Windows Server VMs, 500GB disk size each
agent_pool("pool1", 3, os="Windows", disksize=500)

# enable cluster autoscaling, with a minimum of 1 and maximum of 10 nodes
agent_pool("pool1", 5, autoscale_nodes=c(1, 10))

# use individual VMs rather than scaleset
agent_pool("vmpool1", 3, use_scaleset=FALSE)
```

---

aks *Azure Kubernetes Service class*

Description

Class representing an Azure Kubernetes Service (AKS) resource. For working with the cluster endpoint itself, including deploying images, creating services etc, see kubernetes_cluster.

Methods

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

- `new(...)`: Initialize a new AKS object.
- `get_cluster(config, role)`: Return an object representing the Docker registry endpoint.
- `get_agent_pool(pollname)`: Returns an object of class `az_agent_pool` representing an agent pool. This class inherits from AzureRMR::az_resource; it currently has no extra methods, but these may be added in the future.
- `list_agent_pools()`: Returns a list of agent pool objects.
- `create_agent_pool(poolname, ..., wait=FALSE)`: Creates a new agent pool. See the agent_pool function for further arguments to this method.
- `delete_agent_pool(poolname, confirm=TRUE. wait=FALSE)`: Deletes an agent pool.
- `list_cluster_resources()`: Returns a list of all the Azure resources managed by the cluster.

- `update_aad_password(name=NULL, duration=NULL, ...)`: Update the password for Azure Active Directory integration, returning the new password invisibly. See 'Updating credentials' below.
- `update_service_password(name=NULL, duration=NULL, ...)`: Update the password for the service principal used to manage the cluster resources, returning the new password invisibly. See 'Updating credentials' below.

Details

Initializing a new object of this class can either retrieve an existing AKS resource, or create a new resource on the host. Generally, the best way to initialize an object is via the `get_aks`, `create_aks` or `list_aks` methods of the AzureRMR::az_resource_group class, which handle the details automatically.

Note that this class is separate from the Kubernetes cluster itself. This class exposes methods for working with the Azure resource: updating resource tags, updating and deleting the resource (including updating the Kubernetes version), and so on.

For working with the cluster, including deploying images, services, etc use the object generated with the `get_cluster` method. This method takes two optional arguments:

- `config`: The file in which to store the cluster configuration details. By default, this will be located in the AzureR configuration directory if it exists (see AzureAuth::AzureR_dir); otherwise, in the R temporary directory. To use the Kubernetes default `~/.kube/config` file, set this argument to NULL. Any existing file in the given location will be overwritten.
- `role`: This can be `"User"` (the default) or `"Admin"`.

Updating credentials

An AKS resource can have up to three service principals associated with it. Two of these are for Azure Active Directory (AAD) integration. The third is used to manage the subsidiary resources (VMs, networks, disks, etc) used by the cluster, if it doesn't have a service identity.

Any service principals used by the AKS resource will have secret passwords, which have to be refreshed as they expire. The `update_aad_password()` and `update_service_password()` methods let you refresh the passwords for the cluster's service principals. Their arguments are:

- `name`: An optional friendly name for the password.
- `duration`: The duration for which the new password is valid. Defaults to 2 years.
- `...`: Other arguments passed to `AzureGraph::create_graph_login`. Note that these are used to authenticate with Microsoft Graph, which does the actual work of updating the service principals, not to the cluster itself.

See Also

create_aks, get_aks, delete_aks, list_aks, AzureAuth::AzureR_dir, AzureGraph::create_graph_login

kubernetes_cluster for interacting with the cluster endpoint

AKS documentation and API reference

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

myaks <- rg$get_aks("mycluster")

# sync with Azure: AKS resource creation can take a long time, use this to track status
myaks$sync_fields()

# get the cluster endpoint
kubclus <- myaks$get_cluster()

# list of agent pools
myaks$list_agent_pools()

# create a new agent pool, then delete it
pool <- myaks$create_agent_pool("pool2", 3, size="Standard_DS3_v2")
pool$delete()

# refresh the service principal password (mostly for legacy clusters without a managed identity)
myaks$update_service_password()

# refresh the service principal password, using custom credentials to authenticate with MS Graph
# arguments here are for Graph, not AKS!
myaks$update_service_password(app="app_id", password="app_password")


## End(Not run)
```

---

aks_pools *Vectorised utility function for specifying Kubernetes agent pools*

Description

Vectorised utility function for specifying Kubernetes agent pools

Usage

```
aks_pools(name, count, size = "Standard_DS2_v2", os = "Linux")
```

Arguments

name	The name(s) of the pool(s).
count	The number of nodes per pool.
size	The VM type (size) to use for the pool. To see a list of available VM sizes, use the list_vm_sizes method for the resource group or subscription classes.
os	The operating system to use for the pool. Can be "Linux" or "Windows".

Details

This is a convenience function to simplify the task of specifying the agent pool for a Kubernetes cluster. You can specify multiple pools by providing vectors as input arguments; any scalar inputs will be replicated to match.

`aks_pools` is deprecated; please use agent_pool going forward.

Value

A list of lists, suitable for passing to the `create_aks` constructor method.

See Also

list_vm_sizes, agent_pool

Examples

```
# 1 pool of 5 Linux VMs
aks_pools("pool1", 5)

# 1 pool of 3 Windows Server VMs
aks_pools("pool1", 3, os="Windows")

# 2 pools with different VM sizes per pool
aks_pools(c("pool1", "pool2"), count=c(3, 3), size=c("Standard_DS2_v2", "Standard_DS3_v2"))
```

---

call_docker *Call the docker commandline tool*

Description

Call the docker commandline tool

Usage

```
call_docker(cmd = "", ..., echo = getOption("azure_containers_tool_echo",
  TRUE))
```

Arguments

`cmd`	The docker command. This should be a *vector* of individual docker arguments, but can also be a single commandline string. See below.
`...`	Other arguments to pass to processx::run.
`echo`	Whether to echo the output of the command to the console.

Details

This function calls the `docker` binary, which must be located in your search path. AzureContainers will search for the binary at package startup, and print a warning if it is not found.

The docker command should be specified as a vector of the individual arguments, which is what `processx::run` expects. If a single string is passed, for convenience and back-compatibility reasons `call_docker` will split it into arguments for you. This is prone to error, for example if you are working with pathnames that contain spaces, so it's strongly recommended to pass a vector of arguments as a general practice.

Value

A list with the following components:

- `status`: The exit status of the docker tool. If this is `NA`, then the process was killed and had no exit status.
- `stdout`: The standard output of the command, in a character scalar.
- `stderr`: The standard error of the command, in a character scalar.
- `timeout`: Whether the process was killed because of a timeout.
- `cmdline`: The command line.

The first four components are from `processx::run`; AzureContainers adds the last to make it easier to construct scripts that can be run outside R.

See Also

processx::run, call_docker_compose, call_kubectl for the equivalent interface to the `kubectl` Kubernetes tool

docker_registry

Docker command line reference

Examples

```
## Not run:

# without any args, prints the docker help screen
call_docker()

# build an image: recommended usage
call_docker(c("build", "-t", "myimage", "."))

# alternative usage, will be split into individual arguments
call_docker("build -t myimage .")

# list running containers
call_docker(c("container", "ls"))

# prune unused containers and images
call_docker(c("container", "prune", "-f"))
```

```
call_docker(c("image", "prune", "-f"))


## End(Not run)
```

`call_docker_compose` *Call the docker-compose commandline tool*

Description

Call the docker-compose commandline tool

Usage

```
call_docker_compose(cmd = "", ...,
  echo = getOption("azure_containers_tool_echo", TRUE))
```

Arguments

`cmd`	The docker-compose command line to execute. This should be a *vector* of individual docker-compose arguments, but can also be a single commandline string. See below.
`...`	Other arguments to pass to processx::run.
`echo`	Whether to echo the output of the command to the console.

Details

This function calls the `docker-compose` binary, which must be located in your search path. AzureContainers will search for the binary at package startup, and print a warning if it is not found.

The docker-compose command should be specified as a vector of the individual arguments, which is what `processx::run` expects. If a single string is passed, for convenience and back-compatibility reasons `call_docker_compose` will split it into arguments for you. This is prone to error, for example if you are working with pathnames that contain spaces, so it's strongly recommended to pass a vector of arguments as a general practice.

Value

A list with the following components:

- `status`: The exit status of the docker-compose tool. If this is `NA`, then the process was killed and had no exit status.
- `stdout`: The standard output of the command, in a character scalar.
- `stderr`: The standard error of the command, in a character scalar.
- `timeout`: Whether the process was killed because of a timeout.
- `cmdline`: The command line.

The first four components are from `processx::run`; AzureContainers adds the last to make it easier to construct scripts that can be run outside R.

See Also

processx::run, call_docker, call_kubectl for the equivalent interface to the `kubectl` Kubernetes tool

docker_registry

Docker-compose command line reference

`call_helm` *Call the Helm commandline tool*

Description

Call the Helm commandline tool

Usage

```
call_helm(cmd = "", config = NULL, ...,
  echo = getOption("azure_containers_tool_echo", TRUE))
```

Arguments

`cmd`	The Helm command line to execute. This should be a *vector* of individual helm arguments, but can also be a single commandline string. See below.
`config`	The pathname of the cluster config file, if required.
`...`	Other arguments to pass to processx::run.
`echo`	Whether to echo the output of the command to the console.

Details

This function calls the `helm` binary, which must be located in your search path. AzureContainers will search for the binary at package startup, and print a warning if it is not found.

The helm command should be specified as a vector of the individual arguments, which is what `processx::run` expects. If a single string is passed, for convenience and back-compatibility reasons `call_docker_compose` will split it into arguments for you. This is prone to error, for example if you are working with pathnames that contain spaces, so it's strongly recommended to pass a vector of arguments as a general practice.

Value

A list with the following components:

- `status`: The exit status of the helm tool. If this is `NA`, then the process was killed and had no exit status.
- `stdout`: The standard output of the command, in a character scalar.
- `stderr`: The standard error of the command, in a character scalar.
- `timeout`: Whether the process was killed because of a timeout.
- `cmdline`: The command line.

The first four components are from `processx::run`; AzureContainers adds the last to make it easier to construct scripts that can be run outside R.

See Also

processx::run, call_docker, call_kubectl

kubernetes_cluster

Kubectl command line reference

`call_kubectl` *Call the Kubernetes commandline tool, kubectl*

Description

Call the Kubernetes commandline tool, kubectl

Usage

```
call_kubectl(cmd = "", config = NULL, ...,
  echo = getOption("azure_containers_tool_echo", TRUE))
```

Arguments

`cmd`	The kubectl command line to execute. This should be a *vector* of individual kubectl arguments, but can also be a single commandline string. See below.
`config`	The pathname of the cluster config file, if required.
`...`	Other arguments to pass to processx::run.
`echo`	Whether to echo the output of the command to the console.

Details

This function calls the `kubectl` binary, which must be located in your search path. AzureContainers will search for the binary at package startup, and print a warning if it is not found.

The kubectl command should be specified as a vector of the individual arguments, which is what `processx::run` expects. If a single string is passed, for convenience and back-compatibility reasons `call_docker_compose` will split it into arguments for you. This is prone to error, for example if you are working with pathnames that contain spaces, so it's strongly recommended to pass a vector of arguments as a general practice.

Value

A list with the following components:

- `status`: The exit status of the kubectl tool. If this is `NA`, then the process was killed and had no exit status.
- `stdout`: The standard output of the command, in a character scalar.
- `stderr`: The standard error of the command, in a character scalar.
- `timeout`: Whether the process was killed because of a timeout.
- `cmdline`: The command line.

The first four components are from `processx::run`; AzureContainers adds the last to make it easier to construct scripts that can be run outside R.

See Also

processx::run, call_docker, call_helm

kubernetes_cluster

Kubectl command line reference

Examples

```
## Not run:

# without any args, prints the kubectl help screen
call_kubectl()

# append "--help" to get help for a command
call_kubectl(c("create", "--help"))

# deploy a service from a yaml file
call_kubectl(c("create", "-f", "deployment.yaml"))

# get deployment and service status
call_kubectl(c("get", "deployment"))
call_kubectl(c("get", "service"))


## End(Not run)
```

`create_aci` *Create Azure Container Instance (ACI)*

Description

Method for the AzureRMR::az_resource_group class.

Usage

```
create_aci(name, location = self$location,
           container = name, image,
           registry_creds = list(),
           cores = 1, memory = 8,
           os = c("Linux", "Windows"),
           command = list(), env_vars = list(),
           ports = aci_ports(), dns_name = name, public_ip = TRUE,
          restart = c("Always", "OnFailure", "Never"), managed_identity = TRUE,
           ...)
```

Arguments

- `name`: The name of the ACI service.
- `location`: The location/region in which to create the ACI service. Defaults to this resource group's location.
- `container`: The name of the running container.
- `image`: The name of the image to run.
- `registry_creds`: Docker registry authentication credentials, if the image is stored in a private registry. See 'Details'.
- `cores`: The number of CPU cores for the instance.
- `memory`: The memory size in GB for the instance.
- `os`: The operating system to run in the instance.
- `command`: A list of commands to run in the instance. This is similar to the `--entrypoint` commandline argument to `docker run`; see here for some examples.
- `env_vars`: A list of name-value pairs to set as environment variables in the instance.
- `secure_env_vars`: A list of name-value pairs to set as *secure* environment variables in the instance. The values of these variables are not visible in the container's properties, eg when viewed in the Azure portal or via the CLI.
- `ports`: The network ports to open. By default, opens ports 80 and 443. See 'Details'.
- `dns_name`: The domain name prefix for the instance. Only takes effect if `public_ip=TRUE`.
- `public_ip`: Whether the instance should be publicly accessible.
- `restart`: Whether to restart the instance should an event occur.
- `managed_identity`: Whether to assign the container instance a managed identity.
- `...`: Other named arguments to pass to the AzureRMR::az_resource initialization function.

Details

An ACI resource is a running container hosted in Azure. See the documentation for the resource for more information. Currently ACI only supports a single image in an instance.

To supply the registry authentication credentials, the `registry_creds` argument should contain either an ACR object, a docker_registry object, or the result of a call to the aci_creds function.

The ports to open should be obtained by calling the aci_ports function. This takes a vector of port numbers as well as the protocol (TCP or UDP) for each port.

Value

An object of class `az_container_instance` representing the instance.

See Also

get_aci, delete_aci, list_acis

az_container_instance

ACI documentation and API reference

Docker commandline reference

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

# get the ACR resource that contains the image
myacr <- rg$get_acr("myregistry", as_admin=TRUE)

rg$create_aci("mycontainer",
    image="myregistry.azurecr.io/myimage:latest",
    registry_creds=myacr)


## End(Not run)
```

`create_acr` *Create Azure Container Registry (ACR)*

Description

Method for the AzureRMR::az_resource_group class.

Usage

```
create_acr(name, location = self$location,
           admin_user_enabled = TRUE, sku = "Standard", ...)
```

Arguments

- `name`: The name of the container registry.
- `location`: The location/region in which to create the container registry. Defaults to this resource group's location.
- `admin_user_enabled`: Whether to enable the Admin user. Currently this must be `TRUE` for ACI to pull from the registry.
- `sku`: Either "Basic", "Standard" (the default) or "Premium".
- `wait`: Whether to wait until the ACR resource provisioning is complete.
- `...`: Other named arguments to pass to the AzureRMR::az_resource initialization function.

Details

An ACR resource is a Docker registry hosted in Azure. See the documentation for the resource for more information. To work with the registry (transfer images, retag images, etc) see the documentation for the registry endpoint.

Value

An object of class `az_container_registry` representing the registry resource.

See Also

get_acr, delete_acr, list_acrs

az_container_registry

docker_registry for the registry endpoint

ACR documentation and API reference

Docker registry API

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

rg$create_acr("myregistry")


## End(Not run)
```

`create_aks` *Create Azure Kubernetes Service (AKS)*

Description

Method for the AzureRMR::az_resource_group class.

Usage

```
create_aks(name, location = self$location,
           dns_prefix = name, kubernetes_version = NULL,
           enable_rbac = FALSE, agent_pools = agent_pool("pool1", 3),
           login_user = "", login_passkey = "",
           cluster_service_principal = NULL, managed_identity = TRUE,
           private_cluster = FALSE,
           properties = list(), ..., wait = TRUE)
```

Arguments

- `name`: The name of the Kubernetes service.
- `location`: The location/region in which to create the service. Defaults to this resource group's location.
- `dns_prefix`: The domain name prefix to use for the cluster endpoint. The actual domain name will start with this argument, followed by a string of pseudorandom characters.
- `kubernetes_version`: The Kubernetes version to use. If not specified, uses the most recent version of Kubernetes available.
- `enable_rbac`: Whether to enable Kubernetes role-based access controls (which is distinct from Azure AD RBAC).
- `agent_pools`: The pool specification(s) for the cluster. See 'Details'.
- `login_user,login_passkey`: Optionally, a login username and public key (on Linux). Specify these if you want to be able to ssh into the cluster nodes.
- `cluster_service_principal`: The service principal that AKS will use to manage the cluster resources. This should be a list, with the first component being the client ID and the second the client secret. If not supplied, a new service principal will be created (requires an interactive session). Ignored if `managed_identity=TRUE`, which is the default.
- `managed_identity`: Whether the cluster should have a managed identity assigned to it. If `FALSE`, a service principal will be used to manage the cluster's resources; see 'Details' below.
- `private_cluster`: Whether this cluster is private (not visible from the public Internet). A private cluster is accessible only to hosts on its virtual network.
- `properties`: A named list of further Kubernetes-specific properties to pass to the initialization function.
- `wait`: Whether to wait until the AKS resource provisioning is complete. Note that provisioning a Kubernetes cluster can take several minutes.
- `...`: Other named arguments to pass to the initialization function.

Details

An AKS resource is a Kubernetes cluster hosted in Azure. See the documentation for the resource for more information. To work with the cluster (deploy images, define and start services, etc) see the documentation for the cluster endpoint.

The nodes for an AKS cluster are organised into *agent pools*, also known as *node pools*, which are homogenous groups of virtual machines. To specify the details for a single agent pool, use the `agent_pool` function, which returns an S3 object of that class. To specify the details for multiple pools, you can supply a list of such objects, or a single call to the `aks_pools` function; see the examples below. Note that `aks_pools` is older, and does not support all the possible parameters for an agent pool.

Of the agent pools in a cluster, at least one must be a *system pool*, which is used to host critical system pods such as CoreDNS and tunnelfront. If you specify more than one pool, the first pool will be treated as the system pool. Note that there are certain extra requirements for the system pool.

An AKS cluster requires an identity to manage the low-level resources it uses, such as virtual machines and networks. The default and recommended method is to use a *managed identity*, in

which all the details of this process are handled by AKS. In AzureContainers version 1.2.1 and older, a *service principal* was used instead, which is an older and less automated method. By setting `managed_identity=FALSE`, you can continue using a service principal instead of a managed identity.

One thing to be aware of with service principals is that they have a secret password that will expire eventually. By default, the password for a newly-created service principal will expire after one year. You should run the `update_service_password` method of the AKS object to reset/update the password before it expires.

Value

An object of class `az_kubernetes_service` representing the service.

See Also

get_aks, delete_aks, list_aks, agent_pool, aks_pools

az_kubernetes_service

kubernetes_cluster for the cluster endpoint

AKS documentation and API reference

Kubernetes reference

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

rg$create_aks("mycluster", agent_pools=agent_pool("pool1", 5))

# GPU-enabled cluster
rg$create_aks("mygpucluster", agent_pools=agent_pool("pool1", 5, size="Standard_NC6s_v3"))

# multiple agent pools
rg$create_aks("mycluster", agent_pools=list(
    agent_pool("pool1", 2),
    agent_pool("pool2", 3, size="Standard_NC6s_v3")
))

# deprecated alternative for multiple pools
rg$create_aks("mycluster",
   agent_pools=aks_pools(c("pool1", "pool2"), c(2, 3), c("Standard_DS2_v2", "Standard_NC6s_v3")))


## End(Not run)
```

`delete_aci` *Delete an Azure Container Instance (ACI)*

Description

Method for the AzureRMR::az_resource_group class.

Usage

```
delete_aci(name, confirm=TRUE, wait=FALSE)
```

Arguments

- `name`: The name of the container instance.
- `confirm`: Whether to ask for confirmation before deleting.
- `wait`: Whether to wait until the deletion is complete.

Value

NULL on successful deletion.

See Also

create_aci, get_aci

az_container_instance

ACI documentation and API reference

Docker commandline reference

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

rg$delete_aci("mycontainer")


## End(Not run)
```

`delete_acr` *Delete an Azure Container Registry (ACR)*

Description

Method for the AzureRMR::az_resource_group class.

Usage

```
delete_acr(name, confirm=TRUE, wait=FALSE)
```

Arguments

- `name`: The name of the container registry.
- `confirm`: Whether to ask for confirmation before deleting.
- `wait`: Whether to wait until the deletion is complete.

Value

NULL on successful deletion.

See Also

create_acr, get_acr

az_container_registry

docker_registry for the registry endpoint

ACR documentation and API reference

Docker registry API

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

rg$delete_acr("myregistry")


## End(Not run)
```

`delete_aks` *Delete an Azure Kubernetes Service (AKS)*

Description

Method for the AzureRMR::az_resource_group class.

Usage

```
delete_aks(name, confirm=TRUE, wait=FALSE)
```

Arguments

- `name`: The name of the Kubernetes service.
- `confirm`: Whether to ask for confirmation before deleting.
- `wait`: Whether to wait until the deletion is complete.

Value

NULL on successful deletion.

See Also

create_aks, get_aks

az_kubernetes_service

kubernetes_cluster for the cluster endpoint

AKS documentation and API reference

Kubernetes reference

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

rg$delete_aks("mycluster")


## End(Not run)
```

`DockerRegistry` *Docker registry class*

Description

Class representing a Docker registry. Note that this class can be used to interface with any Docker registry that supports the HTTP V2 API, not just those created via the Azure Container Registry service. Use the docker_registry function to instantiate new objects of this class.

Methods

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

- `login(...)`: Do a local login to the registry via `docker login`; necessary if you want to push and pull images. By default, instantiating a new object of this class will also log you in. See 'Details' below.
- `push(src_image, dest_image, ...)`: Push an image to the registry, using `docker tag` and `docker push`.
- `pull(image, ...)`: Pull an image from the registry, using `docker pull`.
- `get_image_manifest(image, tag="latest")`: Gets the manifest for an image.
- `get_image_digest(image, tag="latest")`: Gets the digest (SHA hash) for an image.
- `delete_image(image, digest, confirm=TRUE)`: Deletes an image from the registry.
- `list_repositories()`: Lists the repositories (images) in the registry.

Details

The arguments to the `login()` method are:

- `tenant`: The Azure Active Directory (AAD) tenant for the registry.
- `username`: The username that Docker will use to authenticate with the registry. This can be either the admin username, if the registry was created with an admin account, or the ID of a registered app that has access to the registry.
- `password`: The password that Docker will use to authenticate with the registry.
- `app`: The app ID to use to authenticate with the registry. Set this to NULL to authenticate with a username and password, rather than via AAD.
- `...`: Further arguments passed to AzureAuth::get_azure_token.
- `token`: An Azure token object. If supplied, all authentication details will be inferred from this.

The `login()`, `push()` and `pull()` methods for this class call the `docker` commandline tool under the hood. This allows all the features supported by Docker to be available immediately, with a minimum of effort. Any calls to the `docker` tool will also contain the full commandline as the `cmdline` attribute of the (invisible) returned value; this allows scripts to be developed that can be run outside R.

See Also

acr, docker_registry, call_docker

Docker commandline reference

Docker registry API

Examples

```
## Not run:

reg <- docker_registry("myregistry")

reg$list_repositories()

# create an image from a Dockerfile in the current directory
call_docker(c("build", "-t", "myimage", "."))

# push the image
reg$push("myimage")

reg$get_image_manifest("myimage")
reg$get_image_digest("myimage")


## End(Not run)
```

---

docker_registry *Create a new Docker registry object*

Description

Create a new Docker registry object

Usage

```
docker_registry(server, tenant = "common", username = NULL,
  password = NULL, app = .az_cli_app_id, ..., domain = "azurecr.io",
  token = NULL, login = TRUE)
```

Arguments

server	The registry server. This can be a URL ("https://myregistry.azurecr.io") or a domain name label ("myregistry"); if the latter, the value of the domain argument is appended to obtain the full hostname.
tenant, username, password, app, ...	Authentication arguments to AzureAuth::get_azure_token. See 'Details' below.
domain	The default domain for the registry server.

token	An OAuth token, of class AzureAuth::AzureToken. If supplied, the authentication details for the registry will be inferred from this.
login	Whether to perform a local login (requires that you have Docker installed). This is necessary if you want to push or pull images.

Details

There are two ways to authenticate with an Azure Docker registry: via Azure Active Directory (AAD), or with a username and password. The latter is simpler, while the former is more complex but also more flexible and secure.

The default method of authenticating is via AAD. Without any arguments, `docker_registry` will authenticate using the AAD credentials of the currently logged-in user. You can change this by supplying the appropriate arguments to `docker_registry`, which will be passed to `AzureAuth::get_azure_token`; alternatively, you can provide an existing token object.

To authenticate via the admin user account, set `app=NULL` and supply the admin username and password in the corresponding arguments. Note that for this to work, the registry must have been created with the admin account enabled.

Authenticating with a service principal can be done either indirectly via AAD, or via a username and password. See the examples below. The latter method is recommended, as it is both faster and allows easier interoperability with AKS and ACI.

Value

An R6 object of class `DockerRegistry`.

See Also

DockerRegistry for methods available for interacting with the registry, call_docker

kubernetes_cluster for the corresponding function to create a Kubernetes cluster object

Examples

```
## Not run:

# connect to the Docker registry 'myregistry.azurecr.io', authenticating as the current user
docker_registry("myregistry")

# same, but providing a full URL
docker_registry("https://myregistry.azurecr.io")

# authenticating via the admin account
docker_registry("myregistry", username="admin", password="password", app=NULL)

# authenticating with a service principal, method 1: recommended
docker_registry("myregistry", username="app_id", password="client_creds", app=NULL)

# authenticating with a service principal, method 2
docker_registry("myregistry", app="app_id", password="client_creds")
```

```
# authenticating from a managed service identity (MSI)
token <- AzureAuth::get_managed_token("https://management.azure.com/")
docker_registry("myregistry", token=token)

# you can also interact with a registry outside Azure
# note that some registry methods, and AAD authentication, may not work in this case
docker_registry("https://hub.docker.com", username="mydockerid", password="password", app=NULL)


## End(Not run)
```

`get_aci` *Get Azure Container Instance (ACI)*

Description

Method for the AzureRMR::az_resource_group class.

Usage

```
get_aci(name)
list_acis()
```

Arguments

- `name`: For `get_aci()`, the name of the container instance resource.

Details

The `AzureRMR::az_resource_group` class has both `get_aci()` and `list_acis()` methods, while the `AzureRMR::az_subscription` class only has the latter.

Value

For `get_aci()`, an object of class `az_container_instance` representing the instance resource.

For `list_acis()`, a list of such objects.

See Also

create_aci, delete_aci

az_container_instance

ACI documentation and API reference

Docker commandline reference

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

rg$get_aci("mycontainer")


## End(Not run)
```

---

get_acr *Get Azure Container Registry (ACR)*

Description

Method for the AzureRMR::az_resource_group class.

Usage

```
get_acr(name)
list_acrs()
```

Arguments

- `name`: For `get_acr()`, the name of the container registry resource.

Details

The `AzureRMR::az_resource_group` class has both `get_acr()` and `list_acrs()` methods, while the `AzureRMR::az_subscription` class only has the latter.

Value

For `get_acr()`, an object of class `az_container_registry` representing the registry resource.

For `list_acrs()`, a list of such objects.

See Also

create_acr, delete_acr

az_container_registry

docker_registry for the registry endpoint

ACR documentation and API reference

Docker registry API

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

rg$get_acr("myregistry")


## End(Not run)
```

`get_aks` *Get Azure Kubernetes Service (AKS)*

Description

Method for the AzureRMR::az_resource_group class.

Usage

```
get_aks(name)
list_aks()
```

Arguments

- `name`: For `get_aks()`, the name of the Kubernetes service.

Details

The `AzureRMR::az_resource_group` class has both `get_aks()` and `list_aks()` methods, while the `AzureRMR::az_subscription` class only has the latter.

Value

For `get_aks()`, an object of class `az_kubernetes_service` representing the service.

For `list_aks()`, a list of such objects.

See Also

create_aks, delete_aks

az_kubernetes_service

kubernetes_cluster for the cluster endpoint

AKS documentation and API reference

Kubernetes reference

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

rg$get_aks("mycluster")


## End(Not run)
```

---

is_acr *Utility functions to test whether an object is of the given class.*

Description

Utility functions to test whether an object is of the given class.

Usage

```
is_acr(object)

is_aks(object)

is_aci(object)

is_docker_registry(object)

is_kubernetes_cluster(object)
```

Arguments

object	An R object

Details

These functions are simple wrappers around `R6::is.R6` and `inherits`.

Value

TRUE or FALSE depending on whether the object is an R6 object of the specified class.

KubernetesCluster *Kubernetes cluster class*

Description

Class representing a Kubernetes cluster. Note that this class can be used to interface with any Docker registry that supports the HTTP V2 API, not just those created via the Azure Container Registry service. Use the kubernetes_cluster function to instantiate new objects of this class.

Methods

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

- `new(...)`: Initialize a new registry object. See 'Initialization' below.
- `create_registry_secret(registry, secret_name, email)`: Provide authentication secret for a Docker registry. See 'Secrets' below.
- `delete_registry_secret(secret_name)`: Delete a registry authentication secret.
- `create(file, ...)`: Creates a deployment or service from a file, using `kubectl create -f`.
- `get(type, ...)`: Get information about resources, using `kubectl get`.
- `run(name, image, ...)`: Run an image using `kubectl run --image`.
- `expose(name, type, file, ...)`: Expose a service using `kubectl expose`. If the `file` argument is provided, read service information from there.
- `delete(type, name, file, ...)`: Delete a resource (deployment or service) using `kubectl delete`. If the `file` argument is provided, read resource information from there.
- `apply(file, ...)`: Apply a configuration file, using `kubectl apply -f`.
- `show_dashboard(port, ...)`: Display the cluster dashboard. By default, use local port 30000.
- `kubectl(cmd, ...)`: Run an arbitrary `kubectl` command on this cluster. Called by the other methods above.
- `helm(cmd, ...)`: Run a `helm` command on this cluster.

Initialization

The `new()` method takes one argument: `config`, the name of the file containing the configuration details for the cluster. This should be a YAML or JSON file in the standard Kubernetes configuration format. Set this to NULL to use the default `~/.kube/config` file.

Secrets

The recommended way to allow a cluster to authenticate with a Docker registry is to give its service principal the appropriate role-based access. However, you can also authenticate with a username and password. To do this, call the `create_registry_secret` method with the following arguments:

- `registry`: An object of class either acr representing an Azure Container Registry service, or DockerRegistry representing the registry itself.
- `secret_name`: The name to give the secret. Defaults to the name of the registry server.
- `email`: The email address for the Docker registry.

Kubectl and helm

The methods for this class call the `kubectl` and `helm` commandline tools, passing the `--config` option to specify the configuration information for the cluster. This allows all the features supported by Kubernetes to be available immediately and with a minimum of effort, although it does require that the tools be installed. The returned object from a call to `kubectl` or `helm` will contain the following components:

- `status`: The exit status. If this is `NA`, then the process was killed and had no exit status.
- `stdout`: The standard output of the command, in a character scalar.
- `stderr`: The standard error of the command, in a character scalar.
- `timeout`: Whether the process was killed because of a timeout.
- `cmdline`: The command line.

The first four components are from `processx::run`; AzureContainers adds the last to make it easier to construct scripts that can be run outside R.

See Also

aks, call_kubectl, call_helm

Kubectl commandline reference

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

# get the cluster endpoint
kubclus <- rg$get_aks("mycluster")$get_cluster()

# get registry authentication secret
kubclus$create_registry_secret(rg$get_acr("myregistry"))

# deploy a service
kubclus$create("deployment.yaml")

# deploy a service from an Internet URL
kubclus$create("https://example.com/deployment.yaml")

# can also supply the deployment parameters inline
kubclus$create("
```

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: model1
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: model1
    spec:
      containers:
      - name: model1
        image: myregistry.azurecr.io/model1
        ports:
        - containerPort: 8000
      imagePullSecrets:
      - name: myregistry.azurecr.io
---
apiVersion: v1
kind: Service
metadata:
  name: model1-svc
spec:
  selector:
    app: model1
  type: LoadBalancer
  ports:
  - protocol: TCP
    port: 8000")

# track status
kubclus$get("deployment")
kubclus$get("service")


## End(Not run)
```

`kubernetes_cluster` *Create a new Kubernetes cluster object*

Description

Create a new Kubernetes cluster object

Usage

```
kubernetes_cluster(config = NULL)
```

Arguments

config	The name of the file containing the configuration details for the cluster. This should be a YAML or JSON file in the standard Kubernetes configuration format. Set this to NULL to use the default ~/.kube/config file.

Details

Use this function to instantiate a new object of the KubernetesCluster class, for interacting with a Kubernetes cluster.

Value

An R6 object of class KubernetesCluster.

See Also

KubernetesCluster for methods for working with the cluster, call_kubectl, call_helm

docker_registry for the corresponding function to create a Docker registry object

Examples

```
## Not run:

kubernetes_cluster()
kubernetes_cluster("myconfig.yaml")


## End(Not run)
```

---

list_kubernetes_versions

List available Kubernetes versions

Description

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

Usage

```
## R6 method for class 'az_subscription'
list_kubernetes_versions(location)

## R6 method for class 'az_resource_group'
list_kubernetes_versions()
```

Arguments

- location: For the az_subscription class method, the location for which to obtain available Kubernetes versions.

Value

A vector of strings, which are the Kubernetes versions that can be used when creating a cluster.

See Also

create_aks

Kubernetes reference

Examples

```
## Not run:

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

rg$list_kubernetes_versions()


## End(Not run)
```

---

list_vm_sizes *List available VM sizes*

Description

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

Usage

```
## 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. 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.

Examples

```
## 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)
```

Index