# Package 'azuremlsdk'

October 12, 2022

**Type** Package

**Title** Interface to the 'Azure Machine Learning' 'SDK'

**Version** 1.10.0

**URL** https://github.com/azure/azureml-sdk-for-r

**BugReports** https://github.com/azure/azureml-sdk-for-r/issues

**Description** Interface to the 'Azure Machine Learning' Software Development Kit ('SDK'). Data scientists can use the 'SDK' to train, deploy, automate, and manage machine learning models on the 'Azure Machine Learning' service. To learn more about 'Azure Machine Learning' visit the website: <https://docs.microsoft.com/en-us/azure/machine-learning/service/overview-what-is-azure-ml>.

**Encoding** UTF-8

**License** MIT + file LICENSE

**RoxygenNote** 7.1.1

**Depends** R (>= 3.5.0)

**Imports** ggplot2, reticulate (>= 1.12), plyr (>= 1.8), DT, rstudioapi (>= 0.7), htmltools, servr, shiny, shinycssloaders

**Suggests** rmarkdown, knitr, testthat, dplyr, jsonlite, foreach, iterators, utils

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Diondra Peck [cre, aut],
Minna Xiao [aut],
AzureML R SDK Team [ctb],
Microsoft [cph, fnd],
Google Inc. [cph] (Examples and Tutorials),
The TensorFlow Authors [cph] (Examples and Tutorials),
RStudio Inc. [cph] (Examples and Tutorials)

**Maintainer** Diondra Peck <Diondra.Peck@microsoft.com>

**Repository** CRAN

**Date/Publication** 2020-09-22 15:40:07 UTC

# R topics documented:

---

`aci_webservice_deployment_config`
*Create a deployment config for deploying an ACI web service*

---

## Description

Deploy a web service to Azure Container Instances for testing or debugging. Use ACI for low-scale CPU-based workloads that require less than 48 GB of RAM.

Deploy to ACI if one of the following conditions is true:

- You need to quickly deploy and validate your model. You do not need to create ACI containers ahead of time. They are created as part of the deployment process.
- You are testing a model that is under development.

## Usage

```
aci_webservice_deployment_config(
  cpu_cores = NULL,
  memory_gb = NULL,
  tags = NULL,
  properties = NULL,
  description = NULL,
  location = NULL,
  auth_enabled = NULL,
  ssl_enabled = NULL,
  enable_app_insights = NULL,
  ssl_cert_pem_file = NULL,
  ssl_key_pem_file = NULL,
  ssl_cname = NULL,
  dns_name_label = NULL
)
```

## Arguments

| | |
|---|---|
| `cpu_cores` | The number of cpu cores to allocate for the web service. Can be a decimal. Defaults to `0.1`. |
| `memory_gb` | The amount of memory (in GB) to allocate for the web service. Can be a decimal. Defaults to `0.5`. |
| `tags` | A named list of key-value tags for the web service, e.g. `list("key" = "value")`. |
| `properties` | A named list of key-value properties for the web service, e.g. `list("key" = "value")`. These properties cannot be changed after deployment, but new key-value pairs can be added. |
| `description` | A string of the description to give the web service. |
| `location` | A string of the Azure region to deploy the web service to. If not specified the workspace location will be used. More details on available regions can be found here. |
| `auth_enabled` | If `TRUE` enable key-based authentication for the web service. Defaults to `FALSE`. |
| `ssl_enabled` | If `TRUE` enable SSL for the web service. Defaults to `FALSE`. |
| `enable_app_insights` | If `TRUE` enable AppInsights for the web service. Defaults to `FALSE`. |
| `ssl_cert_pem_file` | A string of the cert file needed if SSL is enabled. |
| `ssl_key_pem_file` | A string of the key file needed if SSL is enabled. |
| `ssl_cname` | A string of the cname if SSL is enabled. |
| `dns_name_label` | A string of the dns name label for the scoring endpoint. If not specified a unique dns name label will be generated for the scoring endpoint. |

## Value

The `AciServiceDeploymentConfiguration` object.

## See Also

```
deploy_model()
```

## Examples

```
## Not run:
deployment_config <- aci_webservice_deployment_config(cpu_cores = 1, memory_gb = 1)

## End(Not run)
```

---

# aks_webservice_deployment_config

*Create a deployment config for deploying an AKS web service*

---

## Description

Deploy a web service to Azure Kubernetes Service for high-scale prodution deployments. Provides fast response time and autoscaling of the deployed service. Using GPU for inference when deployed as a web service is only supported on AKS.

Deploy to AKS if you need one or more of the following capabilities:

- Fast response time
- Autoscaling of the deployed service
- Hardware acceleration options

## Usage

```
aks_webservice_deployment_config(
  autoscale_enabled = NULL,
  autoscale_min_replicas = NULL,
  autoscale_max_replicas = NULL,
  autoscale_refresh_seconds = NULL,
  autoscale_target_utilization = NULL,
  auth_enabled = NULL,
  cpu_cores = NULL,
  memory_gb = NULL,
  enable_app_insights = NULL,
  scoring_timeout_ms = NULL,
  replica_max_concurrent_requests = NULL,
  max_request_wait_time = NULL,
  num_replicas = NULL,
  primary_key = NULL,
  secondary_key = NULL,
  tags = NULL,
  properties = NULL,
  description = NULL,
```

```
  gpu_cores = NULL,
  period_seconds = NULL,
  initial_delay_seconds = NULL,
  timeout_seconds = NULL,
  success_threshold = NULL,
  failure_threshold = NULL,
  namespace = NULL,
  token_auth_enabled = NULL
)
```

## Arguments

| | |
|---|---|
| autoscale_enabled | If TRUE enable autoscaling for the web service. Defaults to TRUE if num_replicas = NULL. |
| autoscale_min_replicas | An int of the minimum number of containers to use when autoscaling the web service. Defaults to 1. |
| autoscale_max_replicas | An int of the maximum number of containers to use when autoscaling the web service. Defaults to 10. |
| autoscale_refresh_seconds | An int of how often in seconds the autoscaler should attempt to scale the web service. Defaults to 1. |
| autoscale_target_utilization | An int of the target utilization (in percent out of 100) the autoscaler should attempt to maintain for the web service. Defaults to 70. |
| auth_enabled | If TRUE enable key-based authentication for the web service. Defaults to TRUE. |
| cpu_cores | The number of cpu cores to allocate for the web service. Can be a decimal. Defaults to 0.1. |
| memory_gb | The amount of memory (in GB) to allocate for the web service. Can be a decimal. Defaults to 0.5. |
| enable_app_insights | If TRUE enable AppInsights for the web service. Defaults to FALSE. |
| scoring_timeout_ms | An int of the timeout (in milliseconds) to enforce for scoring calls to the web service. Defaults to 60000. |
| replica_max_concurrent_requests | An int of the number of maximum concurrent requests per node to allow for the web service. Defaults to 1. |
| max_request_wait_time | An int of the maximum amount of time a request will stay in the queue (in milliseconds) before returning a 503 error. Defaults to 500. |
| num_replicas | An int of the number of containers to allocate for the web service. If this parameter is not set then the autoscaler is enabled by default. |
| primary_key | A string of the primary auth key to use for the web service. |
| secondary_key | A string of the secondary auth key to use for the web service. |

`tags` A named list of key-value tags for the web service, e.g. `list("key" = "value")`.

`properties` A named list of key-value properties for the web service, e.g. `list("key" = "value")`. These properties cannot be changed after deployment, but new key-value pairs can be added.

`description` A string of the description to give the web service

`gpu_cores` An int of the number of gpu cores to allocate for the web service. Defaults to `1`.

`period_seconds` An int of how often in seconds to perform the liveness probe. Default to `10`. Minimum value is `1`.

`initial_delay_seconds`
An int of the number of seconds after the container has started before liveness probes are initiated. Defaults to `310`.

`timeout_seconds`
An int of the number of seconds after which the liveness probe times out. Defaults to `2`. Minimum value is `1`.

`success_threshold`
An int of the minimum consecutive successes for the liveness probe to be considered successful after having failed. Defaults to `1`. Minimum value is `1`.

`failure_threshold`
An int of the number of times Kubernetes will try the liveness probe when a Pod starts and the probe fails, before giving up. Defaults to `3`. Minimum value is `1`.

`namespace` A string of the Kubernetes namespace in which to deploy the web service: up to 63 lowercase alphanumeric ('a'-'z', '0'-'9') and hyphen ('-') characters. The first last characters cannot be hyphens.

`token_auth_enabled`
If `TRUE`, enable token-based authentication for the web service. If enabled, users can access the web service by fetching an access token using their Azure Active Directory credentials. Defaults to `FALSE`. Both `token_auth_enabled` and `auth_enabled` cannot be set to `TRUE`.

## Details

**AKS compute target:** When deploying to AKS, you deploy to an AKS cluster that is connected to your workspace. There are two ways to connect an AKS cluster to your workspace:

- Create the AKS cluster using Azure ML (see `create_aks_compute()`).
- Attach an existing AKS cluster to your workspace (see `attach_aks_compute()`).

Pass the `AksCompute` object to the `deployment_target` parameter of `deploy_model()`.

**Token-based authentication:** We strongly recommend that you create your Azure ML workspace in the same region as your AKS cluster. To authenticate with a token, the web service will make a call to the region in which your workspace is created. If your workspace's region is unavailable, then you will not be able to fetch a token for your web service, even if your cluster is in a different region than your workspace. This effectively results in token-based auth being unavailable until your workspace's region is available again. In addition, the greater the distance between your cluster's region and your workspace's region, the longer it will take to fetch a token.

## Value

The AksServiceDeploymentConfiguration object.

## See Also

deploy_model()

## Examples

```
## Not run:
deployment_config <- aks_webservice_deployment_config(cpu_cores = 1, memory_gb = 1)

## End(Not run)
```

---

attach_aks_compute *Attach an existing AKS cluster to a workspace*

---

## Description

If you already have an AKS cluster in your Azure subscription, and it is version 1.12.##, you can attach it to your workspace to use for deployments. The existing AKS cluster can be in a different Azure region than your workspace.

If you want to secure your AKS cluster using an Azure Virtual Network, you must create the virtual network first. For more information, see Secure Azure ML experimentation and inference jobs within an Azure Virtual Network

If you want to re-attach an AKS cluster, for example to to change SSL or other cluster configuration settings, you must first remove the existing attachment with detach_aks_compute().

Attaching a cluster will take approximately 5 minutes.

## Usage

```
attach_aks_compute(
  workspace,
  resource_group,
  cluster_name,
  cluster_purpose = c("FastProd", "DevTest")
)
```

## Arguments

| | |
|---|---|
| workspace | The Workspace object to attach the AKS cluster to. |
| resource_group | A string of the resource group in which the AKS cluster is located. |
| cluster_name | A string of the name of the AKS cluster. |

cluster_purpose
: The targeted usage of the cluster. The possible values are "DevTest" or "FastProd". This is used to provision Azure Machine Learning components to ensure the desired level of fault-tolerance and QoS. If your cluster has less than 12 virtual CPUs, you will need to specify "DevTest" for this argument. We recommend that your cluster have at least 2 virtual CPUs for dev/test usage.

## Value

The `AksCompute` object.

## Examples

```
ws <- load_workspace_from_config()
compute_target <- attach_aks_compute(ws,
                                     resource_group = 'myresourcegroup',
                                     cluster_name = 'myakscluster')
```

If the cluster has less than 12 virtual CPUs, you will need to also specify the `cluster_purpose` parameter in the `attach_aks_compute()` call: `cluster_purpose = 'DevTest'`.

## See Also

`detach_aks_compute()`

---

# azureml *azureml module User can access functions/modules in azureml that are not exposed through the exported R functions.*

---

## Description

azureml module User can access functions/modules in azureml that are not exposed through the exported R functions.

## Usage

```
azureml
```

## Format

An object of class `python.builtin.module` (inherits from `python.builtin.object`) of length 5.

`bandit_policy` *Define a Bandit policy for early termination of HyperDrive runs*

## Description

Bandit is an early termination policy based on slack factor/slack amount and evaluation interval. The policy early terminates any runs where the primary metric is not within the specified slack factor/slack amount with respect to the best performing training run.

## Usage

```
bandit_policy(
  slack_factor = NULL,
  slack_amount = NULL,
  evaluation_interval = 1L,
  delay_evaluation = 0L
)
```

## Arguments

| | |
|---|---|
| `slack_factor` | A double of the ratio of the allowed distance from the best performing run. |
| `slack_amount` | A double of the absolute distance allowed from the best performing run. |
| `evaluation_interval` | An integer of the frequency for applying policy. |
| `delay_evaluation` | An integer of the number of intervals for which to delay the first evaluation. |

## Value

The `BanditPolicy` object.

## Details

The Bandit policy takes the following configuration parameters:

- `slack_factor` or `slack_amount`: The slack allowed with respect to the best performing training run. `slack_factor` specifies the allowable slack as a ration. `slack_amount` specifies the allowable slack as an absolute amount, instead of a ratio.
- `evaluation_interval`: Optional. The frequency for applying the policy. Each time the training script logs the primary metric counts as one interval.
- `delay_evaluation`: Optional. The number of intervals to delay the policy evaluation. Use this parameter to avoid premature termination of training runs. If specified, the policy applies every multiple of `evaluation_interval` that is greater than or equal to `delay_evaluation`.

Any run that doesn't fall within the slack factor or slack amount of the evaluation metric with respect to the best performing run will be terminated.

Consider a Bandit policy with `slack_factor = 0.2` and `evaluation_interval = 100`. Assume that run X is the currently best performing run with an AUC (performance metric) of 0.8 after 100 intervals. Further, assume the best AUC reported for a run is Y. This policy compares the value `(Y + Y * 0.2)` to 0.8, and if smaller, cancels the run. If `delay_evaluation = 200`, then the first time the policy will be applied is at interval 200.

Now, consider a Bandit policy with `slack_amount = 0.2` and `evaluation_interval = 100`. If run 3 is the currently best performing run with an AUC (performance metric) of 0.8 after 100 intervals, then any run with an AUC less than 0.6 (`0.8 - 0.2`) after 100 iterations will be terminated. Similarly, the `delay_evaluation` can also be used to delay the first termination policy evaluation for a specific number of sequences.

## Examples

```
# In this example, the early termination policy is applied at every interval
# when metrics are reported, starting at evaluation interval 5. Any run whose
# best metric is less than (1 / (1 + 0.1)) or 91\% of the best performing run will
# be terminated
## Not run:
early_termination_policy = bandit_policy(slack_factor = 0.1,
                                         evaluation_interval = 1L,
                                         delay_evaluation = 5L)

## End(Not run)
```

---

# bayesian_parameter_sampling

*Define Bayesian sampling over a hyperparameter search space*

---

## Description

Bayesian sampling is based on the Bayesian optimization algorithm and makes intelligent choices on the hyperparameter values to sample next. It picks the sample based on how the previous samples performed, such that the new sample improves the reported primary metric.

## Usage

```
bayesian_parameter_sampling(parameter_space)
```

## Arguments

`parameter_space`
: A named list containing each parameter and its distribution, e.g. `list("parameter" = distribution)`.

## Value

The `BayesianParameterSampling` object.

## Details

When you use Bayesian sampling, the number of concurrent runs has an impact on the effectiveness of the tuning process. Typically, a smaller number of concurrent runs can lead to better sampling convergence, since the smaller degree of parallelism increases the number of runs that benefit from previously completed runs.

Bayesian sampling only supports `choice()`, `uniform()`, and `quniform()` distributions over the search space.

Bayesian sampling does not support any early termination policy. When using Bayesian parameter sampling, `early_termination_policy` must be `NULL`.

## See Also

`choice()`, `uniform()`, `quniform()`

## Examples

```
## Not run:
param_sampling <- bayesian_parameter_sampling(list("learning_rate" = uniform(0.05, 0.1),
                                                   "batch_size" = choice(c(16, 32, 64, 128))))

## End(Not run)
```

---

`cancel_run` *Cancel a run*

---

## Description

Cancel an ongoing run.

## Usage

```
cancel_run(run)
```

## Arguments

| | |
|---|---|
| `run` | The `Run` object. |

## Value

`TRUE` if cancellation was successful, else `FALSE`.

## choice *Specify a discrete set of options to sample from*

### Description

Specify a discrete set of options to sample the hyperparameters from.

### Usage

```
choice(options)
```

### Arguments

| | |
|---|---|
| options | A vector of values to choose from. |

### Value

A list of the stochastic expression.

### See Also

`random_parameter_sampling()`, `grid_parameter_sampling()`, `bayesian_parameter_sampling()`

## complete_run *Mark a run as completed.*

### Description

Mark the run as completed. Use for an interactive logging run.

### Usage

```
complete_run(run)
```

### Arguments

| | |
|---|---|
| run | The Run object. |

### Value

None

### See Also

`start_logging_run()`

## container_registry *Specify Azure Container Registry details*

### Description

Returns a `ContainerRegistry` object with the details for an Azure Container Registry (ACR). This is needed when a custom Docker image used for training or deployment is located in a private image registry. Provide a `ContainerRegistry` object to the `image_registry_details` parameter of either `r_environment()` or `estimator()`.

### Usage

```
container_registry(address = NULL, username = NULL, password = NULL)
```

### Arguments

| | |
|---|---|
| `address` | A string of the DNS name or IP address of the Azure Container Registry (ACR). |
| `username` | A string of the username for ACR. |
| `password` | A string of the password for ACR. |

### Value

The `ContainerRegistry` object.

### See Also

`r_environment()`, `estimator()`

## convert_to_dataset_with_csv_files *Convert the current dataset into a FileDataset containing CSV files.*

### Description

Convert the current dataset into a FileDataset containing CSV files.

### Usage

```
convert_to_dataset_with_csv_files(dataset, separator = ",")
```

### Arguments

| | |
|---|---|
| `dataset` | The Tabular Dataset object. |
| `separator` | The separator to use to separate values in the resulting file. |

### Value

A new FileDataset object with a set of CSV files containing the data in this dataset.

---

`convert_to_dataset_with_parquet_files` *Convert the current dataset into a FileDataset containing Parquet files.*

---

### Description

Convert the current dataset into a FileDataset containing Parquet files. The resulting dataset will contain one or more Parquet files, each corresponding to a partition of data from the current dataset. These files are not materialized until they are downloaded or read from.

### Usage

```
convert_to_dataset_with_parquet_files(dataset)
```

### Arguments

| | |
|---|---|
| `dataset` | The Tabular Dataset object. |

### Value

A new FileDataset object with a set of Parquet files containing the data in this dataset.

---

`cran_package` *Specifies a CRAN package to install in environment*

---

### Description

Specifies a CRAN package to install in run environment

### Usage

```
cran_package(name, version = NULL, repo = "https://cloud.r-project.org")
```

### Arguments

| | |
|---|---|
| `name` | The package name |
| `version` | A string of the package version. If not provided, version will default to latest |
| `repo` | The base URL of the repository to use, e.g., the URL of a CRAN mirror. If not provided, the package will be pulled from "https://cloud.r-project.org". |

### Value

A named list containing the package specifications

## Examples

```
pkg1 <- cran_package("ggplot2", version = "3.3.0")
pkg2 <- cran_package("stringr")
pkg3 <- cran_package("ggplot2", version = "0.9.1",
                     repo = "http://cran.us.r-project.org")

env <- r_environment(name = "r_env",
                     cran_packages = list(pkg1, pkg2, pkg3))
```

## See Also

r_environment()

---

# create_aks_compute *Create an AksCompute cluster*

---

## Description

Provision an Azure Kubernetes Service instance (AksCompute) as a compute target for web service deployment. AksCompute is recommended for high-scale production deployments and provides fast response time and autoscaling of the deployed service. Cluster autoscaling isn't supported through the Azure ML R SDK. To change the nodes in the AksCompute cluster, use the UI for the cluster in the Azure portal. Once created, the cluster can be reused for multiple deployments.

## Usage

```
create_aks_compute(
  workspace,
  cluster_name,
  agent_count = NULL,
  vm_size = NULL,
  ssl_cname = NULL,
  ssl_cert_pem_file = NULL,
  ssl_key_pem_file = NULL,
  location = NULL,
  vnet_resourcegroup_name = NULL,
  vnet_name = NULL,
  subnet_name = NULL,
  service_cidr = NULL,
  dns_service_ip = NULL,
  docker_bridge_cidr = NULL,
  cluster_purpose = c("FastProd", "DevTest")
)
```

## Arguments

| | |
|---|---|
| workspace | The Workspace object. |
| cluster_name | A string of the name of the cluster. |
| agent_count | An integer of the number of agents (VMs) to host containers. Defaults to 3. |
| vm_size | A string of the size of agent VMs. More details can be found here. Note that not all sizes are available in all regions, as detailed in the aformentioned link. Defaults to 'Standard_D3_v2'. |
| ssl_cname | A string of a CName to use if enabling SSL validation on the cluster. Must provide all three - CName, cert file, and key file - to enable SSL validation. |
| ssl_cert_pem_file | A string of a file path to a file containing cert information for SSL validation. Must provide all three - CName, cert file, and key file - to enable SSL validation. |
| ssl_key_pem_file | A string of a file path to a file containing key information for SSL validation. Must provide all three - CName, cert file, and key file - to enable SSL validation. |
| location | A string of the location to provision the cluster in. If not specified, defaults to the workspace location. Available regions for this compute can be found here: "https://azure.microsoft.com/global-infrastructure/services/?regions=all&products=kubernetes-service". |
| vnet_resourcegroup_name | A string of the name of the resource group where the virtual network is located. |
| vnet_name | A string of the name of the virtual network. |
| subnet_name | A string of the name of the subnet inside the vnet. |
| service_cidr | A string of a CIDR notation IP range from which to assign service cluster IPs. |
| dns_service_ip | A string of the container's DNS server IP address. |
| docker_bridge_cidr | A string of a CIDR notation IP for Docker bridge. |
| cluster_purpose | A string describing targeted usage of the cluster. This is used to provision Azure Machine Learning components to ensure the desired level of fault-tolerance and QoS. 'FastProd' will provision components to handle higher levels of traffic with production quality fault-tolerance. This will default the AKS cluster to have 3 nodes. 'DevTest' will provision components at a minimal level for testing. This will default the AKS cluster to have 1 node. 'FastProd'is the default value. |

## Value

An AksCompute object.

## Details

For more information on using an AksCompute resource within a virtual network, see Secure Azure ML experimentation and inference jobs within an Azure Virtual Network.

## Examples

```
# Create an AksCompute cluster using the default configuration (you can also
# provide parameters to customize this)

ws <- load_workspace_from_config()

compute_target <- create_aks_compute(ws, cluster_name = 'mycluster')
wait_for_provisioning_completion(compute_target)
```

---

`create_aml_compute` *Create an AmlCompute cluster*

---

## Description

Provision Azure Machine Learning Compute (AmlCompute) as a compute target for training. AmlCompute is a managed-compute infrastructure that allows the user to easily create a single or multi-node compute. To create a persistent AmlCompute resource that can be reused across jobs, make sure to specify the `vm_size` and `max_nodes` parameters. The compute can then be shared with other users in the workspace and is kept between jobs. If `min_nodes = 0`, the compute autoscales down to zero nodes when it isn't used, and scales up automatically when a job is submitted.

AmlCompute has default limits, such as the number of cores that can be allocated. For more information, see Manage and request quotas for Azure resources.

## Usage

```
create_aml_compute(
  workspace,
  cluster_name,
  vm_size,
  vm_priority = "dedicated",
  min_nodes = 0,
  max_nodes = NULL,
  idle_seconds_before_scaledown = NULL,
  admin_username = NULL,
  admin_user_password = NULL,
  admin_user_ssh_key = NULL,
  vnet_resourcegroup_name = NULL,
  vnet_name = NULL,
  subnet_name = NULL,
  tags = NULL,
  description = NULL
)
```

## Arguments

| | |
|---|---|
| workspace | The Workspace object. |
| cluster_name | A string of the name of the cluster. |
| vm_size | A string of the size of agent VMs. More details can be found here. Note that not all sizes are available in all regions, as detailed in the aformentioned link. Defaults to 'Standard_NC6'. |
| vm_priority | A string of either 'dedicated' or 'lowpriority' to use either dedicated or low-priority VMs. Defaults to 'dedicated'. |
| min_nodes | An integer of the minimum number of nodes to use on the cluster. If not specified, will default to 0. |
| max_nodes | An integer of the maximum number of nodes to use on the cluster. |
| idle_seconds_before_scaledown | An integer of the node idle time in seconds before scaling down the cluster. Defaults to 120. |
| admin_username | A string of the name of the administrator user account that can be used to SSH into nodes. |
| admin_user_password | A string of the password of the administrator user account. |
| admin_user_ssh_key | A string of the SSH public key of the administrator user account. |
| vnet_resourcegroup_name | A string of the name of the resource group where the virtual network is located. |
| vnet_name | A string of the name of the virtual network. |
| subnet_name | A string of the name of the subnet inside the vnet. |
| tags | A named list of tags for the cluster, e.g. list("tag" = "value").‘ |
| description | A string of the description for the cluster. |

## Value

The AmlCompute object.

## Details

For more information on using an Azure Machine Learning Compute resource in a virtual network, see Secure Azure ML experimentation and inference jobs within an Azure Virtual Network.

## See Also

wait_for_provisioning_completion()

## Examples

```
## Not run:
ws <- load_workspace_from_config()
compute_target <- create_aml_compute(ws,
                                     cluster_name = 'mycluster',
                                     vm_size = 'STANDARD_D2_V2',
                                     max_nodes = 1)
wait_for_provisioning_completion(compute_target, show_output = TRUE)

## End(Not run)
```

---

# create_child_run *Create a child run*

---

## Description

Create a child run. This is used to isolate part of a run into a subsection.

## Usage

```
create_child_run(parent_run, name = NULL, run_id = NULL, outputs = NULL)
```

## Arguments

| | |
|---|---|
| parent_run | The parent Run object. |
| name | An optional name for the child run, typically specified for a "part" |
| run_id | An optional run ID for the child, otherwise it is auto-generated. Typically this parameter is not set. |
| outputs | Optional outputs directory to track for the child. |

## Value

The child run, a Run object.

---

# create_child_runs *Create one or many child runs*

---

## Description

Create one or many child runs.

## Usage

```
create_child_runs(parent_run, count = NULL, tag_key = NULL, tag_values = NULL)
```

### Arguments

| | |
|---|---|
| parent_run | The parent Run object. |
| count | An optional number of children to create. |
| tag_key | An optional key to populate the Tags entry in all created children. |
| tag_values | An optional list of values that will map onto Tags for the list of runs created. |

### Value

The list of child runs, Run objects.

---

## create_file_dataset_from_files

*Create a FileDataset to represent file streams.*

---

### Description

Create a FileDataset to represent file streams.

### Usage

```
create_file_dataset_from_files(path, validate = TRUE)
```

### Arguments

| | |
|---|---|
| path | A data path in a registered datastore or a local path. |
| validate | Indicates whether to validate if data can be loaded from the returned dataset. Defaults to True. Validation requires that the data source is accessible from the current compute. |

### Value

The FileDataset object

### See Also

data_path

`create_tabular_dataset_from_delimited_files` *Create an unregistered, in-memory Dataset from delimited files.*

## Description

Create an unregistered, in-memory Dataset from delimited files. Use this method to read delimited text files when you want to control the options used.

## Usage

```
create_tabular_dataset_from_delimited_files(
  path,
  validate = TRUE,
  include_path = FALSE,
  infer_column_types = TRUE,
  set_column_types = NULL,
  separator = ",",
  header = TRUE,
  partition_format = NULL,
  support_multi_line = FALSE,
  empty_as_string = FALSE
)
```

## Arguments

| | |
|---|---|
| `path` | A data path in a registered datastore, a local path, or an HTTP URL. |
| `validate` | Boolean to validate if data can be loaded from the returned dataset. Defaults to True. Validation requires that the data source is accessible from the current compute. |
| `include_path` | Whether to include a column containing the path of the file from which the data was read. This is useful when you are reading multiple files, and want to know which file a particular record originated from, or to keep useful information in file path. |
| `infer_column_types` | Indicates whether column data types are inferred. |
| `set_column_types` | A named list to set column data type, where key is column name and value is data type. |
| `separator` | The separator used to split columns. |
| `header` | Controls how column headers are promoted when reading from files. Defaults to True for all files having the same header. Files will read as having no header When header=False. More options can be specified using `PromoteHeadersBehavior`. |

`partition_format`
: Specify the partition format in path and create string columns from format 'x' and datetime column from format 'x:yyyy/MM/dd/HH/mm/ss', where 'yyyy', 'MM', 'dd', 'HH', 'mm' and 'ss' are used to extrat year, month, day, hour, minute and second for the datetime type. The format should start from the postition of first partition key until the end of file path. For example, given a file path '../USA/2019/01/01/data.csv' and data is partitioned by country and time, we can define '/Country/PartitionDate:yyyy/MM/dd/data.csv' to create columns 'Country' of string type and 'PartitionDate' of datetime type.

`support_multi_line`
: By default (support_multi_line=FALSE), all line breaks, including those in quoted field values, will be interpreted as a record break. Reading data this way is faster and more optimized for parallel execution on multiple CPU cores. However, it may result in silently producing more records with misaligned field values. This should be set to TRUE when the delimited files are known to contain quoted line breaks.

`empty_as_string`
: Specify if empty field values should be loaded as empty strings. The default (FALSE) will read empty field values as nulls. Passing this as TRUE will read empty field values as empty strings. If the values are converted to numeric or datetime then this has no effect, as empty values will be converted to nulls.

## Value

The Tabular Dataset object.

## See Also

`data_path`

---

`create_tabular_dataset_from_json_lines_files`
: *Create a TabularDataset to represent tabular data in JSON Lines files (http://jsonlines.org/).*

---

## Description

Create a TabularDataset to represent tabular data in JSON Lines files (http://jsonlines.org/). "from_json_lines_files"' creates a Tabular Dataset object , which defines the operations to load data from JSON Lines files into tabular representation. For the data to be accessible by Azure Machine Learning, the JSON Lines files specified by `path` must be located in a Datastore or behind public web urls. Column data types are read from data types saved in the JSON Lines files. Providing 'set_column_types' will override the data type for the specified columns in the returned Tabular Dataset.

## Usage

```
create_tabular_dataset_from_json_lines_files(
  path,
  validate = TRUE,
  include_path = FALSE,
  set_column_types = NULL,
  partition_format = NULL
)
```

## Arguments

| | |
|---|---|
| path | The path to the source files, which can be single value or list of http url string or tuple of Datastore and relative path. |
| validate | Boolean to validate if data can be loaded from the returned dataset. Defaults to True. Validation requires that the data source is accessible from the current compute. |
| include_path | Boolean to keep path information as column in the dataset. Defaults to False. This is useful when reading multiple files, and want to know which file a particular record originated from, or to keep useful information in file path. |
| set_column_types | A named list to set column data type, where key is column name and value is data type. |
| partition_format | Specify the partition format in path and create string columns from format 'x' and datetime column from format 'x:yyyy/MM/dd/HH/mm/ss', where 'yyyy', 'MM', 'dd', 'HH', 'mm' and 'ss' are used to extrat year, month, day, hour, minute and second for the datetime type. The format should start from the postition of first partition key until the end of file path. For example, given a file path '../USA/2019/01/01/data.csv' and data is partitioned by country and time, we can define '/Country/PartitionDate:yyyy/MM/dd/data.csv' to create columns 'Country' of string type and 'PartitionDate' of datetime type. |

## Value

The Tabular Dataset object.

## See Also

data_path

---

`create_tabular_dataset_from_parquet_files`
*Create an unregistered, in-memory Dataset from parquet files.*

---

## Description

Create an unregistered, in-memory Dataset from parquet files.

## Usage

```
create_tabular_dataset_from_parquet_files(
  path,
  validate = TRUE,
  include_path = FALSE,
  set_column_types = NULL,
  partition_format = NULL
)
```

## Arguments

| | |
|---|---|
| path | A data path in a registered datastore or a local path. |
| validate | Boolean to validate if data can be loaded from the returned dataset. Defaults to True. Validation requires that the data source is accessible from the current compute. |
| include_path | Whether to include a column containing the path of the file from which the data was read. This is useful when you are reading multiple files, and want to know which file a particular record originated from, or to keep useful information in file path. |
| set_column_types | A named list to set column data type, where key is column name and value is data type. |
| partition_format | Specify the partition format in path and create string columns from format 'x' and datetime column from format 'x:yyyy/MM/dd/HH/mm/ss', where 'yyyy', 'MM', 'dd', 'HH', 'mm' and 'ss' are used to extrat year, month, day, hour, minute and second for the datetime type. The format should start from the postition of first partition key until the end of file path. For example, given a file path '../USA/2019/01/01/data.csv' and data is partitioned by country and time, we can define '/Country/PartitionDate:yyyy/MM/dd/data.csv' to create columns 'Country' of string type and 'PartitionDate' of datetime type. |

## Value

The Tabular Dataset object.

## See Also

data_path

`create_tabular_dataset_from_sql_query` *Create a TabularDataset to represent tabular data in SQL databases.*

## Description

Create a TabularDataset to represent tabular data in SQL databases. "from_sql_query"' creates a Tabular Dataset object , which defines the operations to load data from SQL databases into tabular representation. For the data to be accessible by Azure Machine Learning, the SQL database specified by `query` must be located in a Datastore and the datastore type must be of a SQL kind. Column data types are read from data types in SQL query result. Providing 'set_column_types' will override the data type for the specified columns in the returned Tabular Dataset.

## Usage

```
create_tabular_dataset_from_sql_query(
  query,
  validate = TRUE,
  set_column_types = NULL,
  query_timeout = 30L
)
```

## Arguments

| | |
|---|---|
| `query` | A SQL-kind datastore and a query |
| `validate` | Boolean to validate if data can be loaded from the returned dataset. Defaults to True. Validation requires that the data source is accessible from the current compute. |
| `set_column_types` | A named list to set column data type, where key is column name and value is data type. |
| `query_timeout` | Sets the wait time (as an int, in seconds) before terminating the attempt to execute a command and generating an error. The default is 30 seconds. |

## Value

A `TabularDataset` object

## Examples

```
# create tabular dataset from a SQL database in datastore
datastore <- get_datastore(ws, 'sql-db')
query <- data_path(datastore, 'SELECT * FROM my_table')
tab_ds <- create_tabular_dataset_from_sql_query(query, query_timeout = 10)

# use `set_column_types` param to set column data types
```

```
data_types <- list(ID = data_type_string(),
                   Date = data_type_datetime('%d/%m/%Y %I:%M:%S %p'),
                   Count = data_type_long(),
                   Latitude = data_type_double(),
                   Found = data_type_bool())

set_tab_ds <- create_tabular_dataset_from_sql_query(query, set_column_types = data_types)
```

## See Also

data_path() data_type_datetime() data_type_bool() data_type_double() data_type_string() data_type_long()

---

create_workspace *Create a new Azure Machine Learning workspace*

---

## Description

Create a new Azure Machine Learning workspace. Throws an exception if the workspace already exists or any of the workspace requirements are not satisfied. When you create new workspace, it automatically creates several Azure resources that are used in the workspace:

- Azure Container Registry: Registers Docker containers that you use during training and when you deploy a model. To minimize costs, ACR is lazy-loaded until deployment images are created.
- Azure Storage account: Used as the default datastore for the workspace.
- Azure Application Insights: Stores monitoring information about your models.
- Azure Key Vault: Stores secrets that are used by compute targets and other sensitive information that's needed by the workspace.

## Usage

```
create_workspace(
  name,
  auth = NULL,
  subscription_id = NULL,
  resource_group = NULL,
  location = NULL,
  create_resource_group = TRUE,
  friendly_name = NULL,
  storage_account = NULL,
  key_vault = NULL,
  app_insights = NULL,
  container_registry = NULL,
  cmk_keyvault = NULL,
  resource_cmk_uri = NULL,
  hbi_workspace = FALSE,
```

```
  exist_ok = FALSE,
  show_output = TRUE,
  sku = "basic"
)
```

## Arguments

| | |
|---|---|
| name | A string of the new workspace name. Workspace name has to be between 2 and 32 characters of letters and numbers. |
| auth | The ServicePrincipalAuthentication or InteractiveLoginAuthentication object. For more details refer to https://aka.ms/aml-notebook-auth. If NULL, the default Azure CLI credentials will be used or the API will prompt for credentials. |
| subscription_id | A string of the subscription ID of the containing subscription for the new workspace. The parameter is required if the user has access to more than one subscription. |
| resource_group | A string of the Azure resource group that is containing the workspace. The parameter defaults to a mutation of the workspace name. |
| location | A string of the location of the workspace. The parameter defaults to the resource group location. The location has to be a supported region for Azure Machine Learning Services. |
| create_resource_group | If TRUE the resource group will be created if it doesn't exist. |
| friendly_name | A string of the friendly name for the workspace that can be displayed in the UI. |
| storage_account | A string of an existing storage account in the Azure resource ID format. The storage will be used by the workspace to save run outputs, code, logs etc. If NULL a new storage will be created. |
| key_vault | A string of an existing key vault in the Azure resource ID format. The key vault will be used by the workspace to store credentials added to the workspace by the users. If NULL a new key vault will be created. |
| app_insights | A string of an existing Application Insights in the Azure resource ID format. The Application Insights will be used by the workspace to log webservices events. If NULL a new Application Insights will be created. |
| container_registry | A string of an existing container registry in the Azure resource ID format. The container registry will be used by the workspace to pull and push both experimentation and webservices images. If NULL a new container registry will be created. |
| cmk_keyvault | A string representing the key vault containing the customer managed key in the Azure resource ID format: '/subscriptions//resourcegroups//providers/microsoft.keyvault/vaults/'. For example: '/subscriptions/d139f240-94e6-4175-87a7-954b9d27db16/resourcegroups/myresourcegrou |
| resource_cmk_uri | The key URI of the customer managed key to encrypt the data at rest. The URI format is: 'https://<keyvault-dns-name>/keys/<key-name>/<key-version>'. For example, 'https://mykeyvault.vault.azure.net/keys/mykey/bc5dce6d01df49w2na7ffb11a2ee008b'. |

| | |
|---|---|
| | Refer to https://docs.microsoft.com/azure-stack/user/azure-stack-key-vault-manage-portal for steps on how to create a key and get its URI. |
| hbi_workspace | Specifies whether the customer data is of High Business Impact(HBI), i.e., contains sensitive business information. The default value is FALSE. When set to TRUE, downstream services will selectively disable logging. |
| exist_ok | If TRUE the method will not fail if the workspace already exists. |
| show_output | If TRUE the method will print out incremental progress of method. |
| sku | A string indicating if the workspace will be "basic" or "enterprise" edition. |

## Value

The Workspace object.

## Examples

This example requires only minimal specification, and all dependent resources as well as the resource group will be created automatically.

```
ws <- create_workspace(name = 'myworkspace',
                       subscription_id = '<azure-subscription-id>',
                       resource_group = 'myresourcegroup',
                       location = 'eastus2')
```

This example shows how to reuse existing Azure resources by making use of all parameters utilizing the Azure resource ID format. The specific Azure resource IDs can be retrieved through the Azure Portal or SDK. This assumes that the resource group, storage account, key vault, App Insights and container registry already exist.

```
prefix = "subscriptions/<azure-subscription-id>/resourcegroups/myresourcegroup/providers/"
ws <- create_workspace(
       name = 'myworkspace',
       subscription_id = '<azure-subscription-id>',
       resource_group = 'myresourcegroup',
       create_resource_group = FALSE,
       location = 'eastus2',
       friendly_name = 'My workspace',
     storage_account = paste0(prefix, 'microsoft.storage/storageaccounts/mystorageaccount'),
       key_vault = paste0(prefix, 'microsoft.keyvault/vaults/mykeyvault'),
     app_insights = paste0(prefix, 'microsoft.insights/components/myappinsights'),
       container_registry = paste0(
         prefix,
         'microsoft.containerregistry/registries/mycontainerregistry'))
```

## See Also

get_workspace() service_principal_authentication() interactive_login_authentication()

dataset_consumption_config *Represent how to deliver the dataset to a compute target.*

## Description

Represent how to deliver the dataset to a compute target.

## Usage

```
dataset_consumption_config(
  name,
  dataset,
  mode = "direct",
  path_on_compute = NULL
)
```

## Arguments

| | |
|---|---|
| name | The name of the dataset in the run, which can be different to the registered name. The name will be registered as environment variable and can be used in data plane. |
| dataset | The dataset that will be consumed in the run. |
| mode | Defines how the dataset should be delivered to the compute target. There are three modes:<br>'direct': consume the dataset as dataset. 'download': download the dataset and consume the dataset as downloaded path. 'mount': mount the dataset and consume the dataset as mount path. |
| path_on_compute | The target path on the compute to make the data available at. The folder structure of the source data will be kept, however, we might add prefixes to this folder structure to avoid collision. |

## Value

The DatasetConsumptionConfig object.

## Examples

```
est <- estimator(source_directory = ".",
                 entry_script = "train.R",
           inputs = list(dataset_consumption_config('mydataset', dataset, mode = 'download')),
                 compute_target = compute_target)
```

## See Also

estimator

data_path *Represents a path to data in a datastore.*

## Description

The path represented by DataPath object can point to a directory or a data artifact (blob, file).

## Usage

```
data_path(datastore, path_on_datastore = NULL, name = NULL)
```

## Arguments

| | |
|---|---|
| datastore | The Datastore to reference. |
| path_on_datastore | The relative path in the backing storage for the data reference. |
| name | An optional name for the DataPath. |

## Value

The DataPath object.

## Examples

```
my_data <- register_azure_blob_container_datastore(
    workspace = ws,
    datastore_name = blob_datastore_name,
    container_name = ws_blob_datastore$container_name,
    account_name = ws_blob_datastore$account_name,
    account_key = ws_blob_datastore$account_key,
    create_if_not_exists = TRUE)

datapath <- data_path(my_data, <path_on_my_datastore>)
dataset <- create_file_dataset_from_files(datapath)
```

## See Also

create_file_dataset_from_files create_tabular_dataset_from_parquet_files create_tabular_dataset_from_
create_tabular_dataset_from_json_lines_files create_tabular_dataset_from_sql_query

## data_type_bool *Configure conversion to bool.*

### Description

Configure conversion to bool.

### Usage

```
data_type_bool()
```

### Value

Converted DataType object.

## data_type_datetime *Configure conversion to datetime.*

### Description

Configure conversion to datetime.

### Usage

```
data_type_datetime(formats = NULL)
```

### Arguments

| | |
|---|---|
| formats | Formats to try for datetime conversion. For example `%d-%m-%Y` for data in "day-month-year", and `%Y-%m-%dT%H:%M:%S.%f` for "combined date an time representation" according to ISO 8601.<br>• %Y: Year with 4 digits<br>• %y: Year with 2 digits<br>• %m: Month in digits<br>• %b: Month represented by its abbreviated name in 3 letters, like Aug<br>• %B: Month represented by its full name, like August<br>• %d: Day in digits<br>• %H: Hour as represented in 24-hour clock time<br>• %I: Hour as represented in 12-hour clock time<br>• %M: Minute in 2 digits<br>• %S: Second in 2 digits<br>• %f: Microsecond<br>• %p: AM/PM designator<br>• %z: Timezone, for example: -0700<br>Format specifiers will be inferred if not specified. Inference requires that the data source is accessible from current compute. |

### Value

Converted DataType object.

---

`data_type_double` *Configure conversion to 53-bit double.*

---

### Description

Configure conversion to 53-bit double.

### Usage

```
data_type_double()
```

### Value

Converted DataType object.

---

`data_type_long` *Configure conversion to 64-bit integer.*

---

### Description

Configure conversion to 64-bit integer.

### Usage

```
data_type_long()
```

### Value

Converted DataType object.

---

`data_type_string` *Configure conversion to string.*

---

### Description

Configure conversion to string.

### Usage

```
data_type_string()
```

### Value

Converted DataType object.

## define_timestamp_columns_for_dataset *Define timestamp columns for the dataset.*

### Description

Define timestamp columns for the dataset. The method defines columns to be used as timestamps. Timestamp columns on a dataset make it possible to treat the data as time-series data and enable additional capabilities. When a dataset has both `fine_grain_timestamp` and `coarse_grain_timestamp` defined specified, the two columns should represent the same timeline.

### Usage

```
define_timestamp_columns_for_dataset(
  dataset,
  fine_grain_timestamp,
  coarse_grain_timestamp = NULL,
  validate = FALSE
)
```

### Arguments

| | |
|---|---|
| `dataset` | The Tabular Dataset object. |
| `fine_grain_timestamp` | The name of column as fine grain timestamp. Use None to clear it. |
| `coarse_grain_timestamp` | The name of column coarse grain timestamp (optional). The default is None. |
| `validate` | Indicates whether to validate if specified columns exist in dataset. The default is False. Validation requires that the data source is accessible from the current compute. |

### Value

The Tabular Dataset with timestamp columns defined.

## delete_compute *Delete a cluster*

### Description

Remove the compute object from its associated workspace and delete the corresponding cloud-based resource.

### Usage

```
delete_compute(cluster)
```

### Arguments

| | |
|---|---|
| cluster | The AmlCompute or AksCompute object. |

### Value

None

### Examples

```
## Not run:
ws <- load_workspace_from_config()
compute_target <- get_compute(ws, cluster_name = 'mycluster')
delete_compute(compute_target)

## End(Not run)
```

---

## delete_local_webservice

*Delete a local web service from the local machine*

---

### Description

Delete a local web service from the local machine. This function call is not asynchronous; it runs until the service is deleted.

### Usage

```
delete_local_webservice(webservice, delete_cache = TRUE, delete_image = FALSE)
```

### Arguments

| | |
|---|---|
| webservice | The LocalWebservice object. |
| delete_cache | If TRUE, delete the temporary files cached for the service. |
| delete_image | If TRUE, delete the service's Docker image. |

### Value

None

## delete_model *Delete a model from its associated workspace*

### Description

Delete the registered model from its associated workspace. Note that you cannot delete a registered model that is being used by an active web service deployment.

### Usage

```
delete_model(model)
```

### Arguments

| | |
|---|---|
| model | The Model object. |

### Value

None

## delete_secrets *Delete secrets from a keyvault*

### Description

Delete secrets from the keyvault associated with the workspace for a specified set of secret names.

### Usage

```
delete_secrets(keyvault, secrets)
```

### Arguments

| | |
|---|---|
| keyvault | The Keyvault object. |
| secrets | A vector of secret names. |

### Value

None

## delete_webservice *Delete a web service from a given workspace*

### Description

Delete a deployed ACI or AKS web service from the given workspace. This function call is not asynchronous; it runs until the resource is deleted.

To delete a `LocalWebservice` see `delete_local_webservice()`.

### Usage

```
delete_webservice(webservice)
```

### Arguments

| | |
|---|---|
| `webservice` | The `AciWebservice` or `AksWebservice` object. |

### Value

None

## delete_workspace *Delete a workspace*

### Description

Delete the Azure Machine Learning workspace resource. `delete_workspace()` can also delete the workspace's associated resources.

### Usage

```
delete_workspace(
  workspace,
  delete_dependent_resources = FALSE,
  no_wait = FALSE
)
```

### Arguments

| | |
|---|---|
| `workspace` | The `Workspace` object of the workspace to delete. |
| `delete_dependent_resources` | If `TRUE` the workspace's associated resources, i.e. ACR, storage account, key value, and application insights will also be deleted. |
| `no_wait` | If `FALSE` do not wait for the workspace deletion to complete. |

## Value

None

---

`deploy_model` *Deploy a web service from registered model(s)*

---

## Description

Deploy a web service from zero or more registered models. Types of web services that can be deployed are `LocalWebservice`, which will deploy a model locally, and `AciWebservice` and `AksWebservice`, which will deploy a model to Azure Container Instances (ACI) and Azure Kubernetes Service (AKS), respectively.The type of web service deployed will be determined by the `deployment_config` specified. Returns a `Webservice` object corresponding to the deployed web service.

## Usage

```
deploy_model(
  workspace,
  name,
  models,
  inference_config,
  deployment_config = NULL,
  deployment_target = NULL
)
```

## Arguments

| | |
|---|---|
| `workspace` | The `Workspace` object. |
| `name` | A string of the name to give the deployed service. Must be unique to the workspace, only consist of lowercase letters, numbers, or dashes, start with a letter, and be between 3 and 32 characters long. |
| `models` | A list of `Model` objects. Can be an empty list. |
| `inference_config` | The `InferenceConfig` object used to describe how to configure the model to make predictions. |
| `deployment_config` | The deployment configuration of type `LocalWebserviceDeploymentConfiguration`, `AciServiceDeploymentConfiguration`, or `AksServiceDeploymentConfiguration` used to configure the web service. The deployment configuration is specific to the compute target that will host the web service. For example, when you deploy a model locally, you must specify the port where the service accepts requests. If `NULL`, an empty configuration object will be used based on the desired target specified by `deployment_target`. |
| `deployment_target` | The compute target to deploy the model to. You will only need to specify this parameter if you are deploy to AKS, in which case provide an `AksCompute` object. If you are deploying locally or to ACI, leave this parameter as `NULL`. |

### Value

The LocalWebservice, AciWebservice, or AksWebservice object.

### Details

If you encounter any issue in deploying your web service, please visit this troubleshooting guide.

### See Also

inference_config(), aci_webservice_deployment_config(), aks_webservice_deployment_config(), local_webservice_deployment_config()

### Examples

```
## Not run:
ws <- load_workspace_from_config()
model <- get_model(ws, name = "my_model")
r_env <- r_environment(name = "r_env")
inference_config <- inference_config(entry_script = "score.R",
                                     source_directory = ".",
                                     environment = r_env)
deployment_config <- aci_webservice_deployment_config(cpu_cores = 1, memory_gb = 1)
service <- deploy_model(ws,
                        name = "my_webservice",
                        models = list(model),
                        inference_config = inference_config,
                        deployment_config = deployment_config)
wait_for_deployment(service, show_output = TRUE)

## End(Not run)
```

---

## detach_aks_compute *Detach an AksCompute cluster from its associated workspace*

---

### Description

Detach the AksCompute cluster from its associated workspace. No underlying cloud resource will be deleted; the association will just be removed.

### Usage

```
detach_aks_compute(cluster)
```

### Arguments

| | |
|---|---|
| cluster | The AksCompute object. |

### Value

None

download_files_from_run *Download files from a run*

## Description

Download files from the run record. You can download any files that were uploaded to the run record via `upload_files_to_run()` or `upload_folder_to_run()`, or any files that were written out to the `./outputs` or `./logs` folders during a run.

## Usage

```
download_files_from_run(
  run,
  prefix = NULL,
  output_directory = NULL,
  output_paths = NULL,
  batch_size = 100L
)
```

## Arguments

| | |
|---|---|
| `run` | The Run object. |
| `prefix` | A string of the the filepath prefix (folder name) from which to download all artifacts. If not specified, all the artifacts in the run record will be downloaded. |
| `output_directory` | (Optional) A string of the directory that all artifact paths use as a prefix. |
| `output_paths` | (Optional) A list of strings of the local filepaths where the artifacts will be downloaded to. |
| `batch_size` | An int of the number of files to download per batch. |

## Value

None

## See Also

`download_file_from_run()`

download_file_from_run *Download a file from a run*

## Description

Download a file from the run record. You can download any file that was uploaded to the run record via `upload_files_to_run()` or `upload_folder_to_run()`, or any file that was written out to the `./outputs` or `./logs` folders during a run.

You can see what files are available to download from the run record by calling `get_run_file_names()`.

## Usage

```
download_file_from_run(run, name, output_file_path = NULL)
```

## Arguments

| | |
|---|---|
| `run` | The Run object. |
| `name` | A string of the name of the artifact to be downloaded. |
| `output_file_path` | A string of the local path where to download the artifact to. |

## Value

None

## See Also

`download_files_from_run()`

download_from_datastore *Download data from a datastore to the local file system*

## Description

Download data from the datastore to the local file system.

## Usage

```
download_from_datastore(
  datastore,
  target_path,
  prefix = NULL,
  overwrite = FALSE,
  show_progress = TRUE
)
```

### Arguments

| | |
|---|---|
| datastore | The AzureBlobDatastore or AzureFileDatastore object. |
| target_path | A string of the local directory to download the file to. |
| prefix | A string of the path to the folder in the blob container or file store to download. If NULL, will download everything in the blob container or file share |
| overwrite | If TRUE, overwrites any existing data at target_path. |
| show_progress | If TRUE, show progress of upload in the console. |

### Value

An integer of the number of files successfully downloaded.

---

## download_from_file_dataset

*Download file streams defined by the dataset as local files.*

---

### Description

Download file streams defined by the dataset as local files. If target_path starts with a /, then it will be treated as an absolute path. If it doesn't start with a /, then it will be treated as a relative path relative to the current working directory.

### Usage

```
download_from_file_dataset(dataset, target_path = NULL, overwrite = FALSE)
```

### Arguments

| | |
|---|---|
| dataset | The Dataset object |
| target_path | The local directory to download the files to. If NULL, the data will be downloaded into a temporary directory. |
| overwrite | Indicates whether to overwirte existing files. The default is FALSE. Existing files will be overwritten if overwrite is set to TRUE; otherwise an exception will be raised. |

### Value

A list of file paths for each file downloaded.

download_model *Download a model to the local file system*

## Description

Download a registered model to the target_dir of your local file system.

## Usage

```
download_model(model, target_dir = ".", exist_ok = FALSE)
```

## Arguments

| | |
|---|---|
| model | The Model object. |
| target_dir | A string of the path to the directory on your local file system for where to download the model to. Defaults to ".". |
| exist_ok | If FALSE, replace the downloaded folder/file if they already exist. |

## Value

A string of the path to the file or folder of the downloaded model.

## Examples

```
## Not run:
ws <- load_workspace_from_config()
model <- get_model(ws, name = "my_model", version = 2)
download_model(model, target_dir = tempdir(), exist_ok = TRUE)

## End(Not run)
```

---

drop_columns_from_dataset

*Drop the specified columns from the dataset.*

## Description

Drop the specified columns from the dataset. If a timeseries column is dropped, the corresponding capabilities will be dropped for the returned dataset as well.

## Usage

```
drop_columns_from_dataset(dataset, columns)
```

### Arguments

| | |
|---|---|
| dataset | The Tabular Dataset object. |
| columns | A name or a list of names for the columns to drop. |

### Value

A new TabularDataset object with the specified columns dropped.

---

## estimator *Create an estimator*

---

### Description

An Estimator wraps run configuration information for specifying details of executing an R script. Running an Estimator experiment (using `submit_experiment()`) will return a `ScriptRun` object and execute your training script on the specified compute target.

To define the environment to use for training, you can either directly provide the environment-related parameters (e.g. `cran_packages`, `custom_docker_image`) to `estimator()`, or you can provide an `Environment` object to the `environment` parameter. For more information on the pre-defined Docker images that are used for training if `custom_docker_image` is not specified, see the documentation here.

### Usage

```
estimator(
  source_directory,
  compute_target = NULL,
  vm_size = NULL,
  vm_priority = NULL,
  entry_script = NULL,
  script_params = NULL,
  cran_packages = NULL,
  github_packages = NULL,
  custom_url_packages = NULL,
  custom_docker_image = NULL,
  image_registry_details = NULL,
  use_gpu = FALSE,
  environment_variables = NULL,
  shm_size = NULL,
  max_run_duration_seconds = NULL,
  environment = NULL,
  inputs = NULL
)
```

## Arguments

`source_directory`
: A string of the local directory containing experiment configuration and code files needed for the training job.

`compute_target`
: The `AmlCompute` object for the compute target where training will happen.

`vm_size`
: A string of the VM size of the compute target that will be created for the training job. The list of available VM sizes are listed here. Provide this parameter if you want to create AmlCompute as the compute target at run time, instead of providing an existing cluster to the `compute_target` parameter. If `vm_size` is specified, a single-node cluster is automatically created for your run and is deleted automatically once the run completes.

`vm_priority`
: A string of either `'dedicated'` or `'lowpriority'` to specify the VM priority of the compute target that will be created for the training job. Defaults to `'dedicated'`. This takes effect only when the `vm_size` parameter is specified.

`entry_script`
: A string representing the relative path to the file used to start training.

`script_params`
: A named list of the command-line arguments to pass to the training script specified in `entry_script`.

`cran_packages`
: A list of `cran_package` objects to be installed.

`github_packages`
: A list of `github_package` objects to be installed.

`custom_url_packages`
: A character vector of packages to be installed from local directory or custom URL.

`custom_docker_image`
: A string of the name of the Docker image from which the image to use for training will be built. If not set, a predefined image will be used as the base image. To use an image from a private Docker repository, you will also have to specify the `image_registry_details` parameter.

`image_registry_details`
: A `ContainerRegistry` object of the details of the Docker image registry for the custom Docker image.

`use_gpu`
: Indicates whether the environment to run the experiment should support GPUs. If `TRUE`, a predefined GPU-based Docker image will be used in the environment. If `FALSE`, a predefined CPU-based image will be used. Predefined Docker images (CPU or GPU) will only be used if the `custom_docker_image` parameter is not set.

`environment_variables`
: A named list of environment variables names and values. These environment variables are set on the process where the user script is being executed.

`shm_size`
: A string for the size of the Docker container's shared memory block. For more information, see Docker run reference. If not set, a default value of `'2g'` is used.

`max_run_duration_seconds`
: An integer of the maximum allowed time for the run. Azure ML will attempt to automatically cancel the run if it takes longer than this value.

| | |
|---|---|
| environment | The Environment object that configures the R environment where the experiment is executed. This parameter is mutually exclusive with the other environment-related parameters custom_docker_image , image_registry_details, use_gpu, environment_variables, shm_size, cran_packages, github_packages, and custom_url_packages and if set will take precedence over those parameters. |
| inputs | A list of DataReference objects or DatasetConsumptionConfig objects to use as input. |

## Value

The Estimator object.

## Examples

```
r_env <- r_environment(name = "r-env",
                       cran_packages = list(cran_package("dplyr"),
                                            cran_package("ggplot2")))
est <- estimator(source_directory = ".",
                 entry_script = "train.R",
                 compute_target = compute_target,
                 environment = r_env)
```

## See Also

r_environment(), container_registry(), submit_experiment(), dataset_consumption_config(), cran_package()

---

# experiment *Create an Azure Machine Learning experiment*

---

## Description

An experiment is a grouping of many runs from a specified script.

## Usage

```
experiment(workspace, name)
```

## Arguments

| | |
|---|---|
| workspace | The Workspace object. |
| name | A string of the experiment name. The name must be between 3-36 characters, start with a letter or number, and can only contain letters, numbers, underscores, and dashes. |

## Value

The Experiment object.

## See Also

submit_experiment()

## Examples

```
## Not run:
ws <- load_workspace_from_config()
exp <- experiment(ws, name = 'myexperiment')

## End(Not run)
```

---

# filter_dataset_after_time

*Filter Tabular Dataset with time stamp columns after a specified start time.*

---

## Description

Filter Tabular Dataset with time stamp columns after a specified start time.

## Usage

```
filter_dataset_after_time(dataset, start_time, include_boundary = TRUE)
```

## Arguments

| | |
|---|---|
| dataset | The Tabular Dataset object |
| start_time | The lower bound for filtering data. |
| include_boundary | Boolean indicating if the row associated with the boundary time (start_time) should be included. |

## Value

The filtered Tabular Dataset

`filter_dataset_before_time` *Filter Tabular Dataset with time stamp columns before a specified end time.*

## Description

Filter Tabular Dataset with time stamp columns before a specified end time.

## Usage

```
filter_dataset_before_time(dataset, end_time, include_boundary = TRUE)
```

## Arguments

| | |
|---|---|
| `dataset` | The Tabular Dataset object |
| `end_time` | The upper bound for filtering data. |
| `include_boundary` | Boolean indicating if the row associated with the boundary time (`start_time`) should be included. |

## Value

The filtered Tabular Dataset

`filter_dataset_between_time` *Filter Tabular Dataset between a specified start and end time.*

## Description

Filter Tabular Dataset between a specified start and end time.

## Usage

```
filter_dataset_between_time(
  dataset,
  start_time,
  end_time,
  include_boundary = TRUE
)
```

### Arguments

| | |
|---|---|
| dataset | The Tabular Dataset object |
| start_time | The lower bound for filtering data. |
| end_time | The upper bound for filtering data. |
| include_boundary | Boolean indicating if the row associated with the boundary time (start_time and end_time) should be included. |

### Value

The filtered Tabular Dataset

---

## filter_dataset_from_recent_time

*Filter Tabular Dataset to contain only the specified duration (amount) of recent data.*

---

### Description

Filter Tabular Dataset to contain only the specified duration (amount) of recent data.

### Usage

```
filter_dataset_from_recent_time(dataset, time_delta, include_boundary = TRUE)
```

### Arguments

| | |
|---|---|
| dataset | The Tabular Dataset object |
| time_delta | The duration (amount) of recent data to retrieve. |
| include_boundary | Boolean indicating if the row associated with the boundary time (time_delta) should be included. |

### Value

The filtered Tabular Dataset

## generate_entry_script *Generates the control script for the experiment.*

### Description

Generates the control script for the experiment.

### Usage

```
generate_entry_script(source_directory)
```

### Arguments

| | |
|---|---|
| source_directory | The directory which contains all the files needed for the experiment. |

## generate_new_webservice_key *Regenerate one of a web service's keys*

### Description

Regenerate either the primary or secondary authentication key for an `AciWebservice` or `AksWebservice`.The web service must have been deployed with key-based authentication enabled.

Not supported for `LocalWebservice` deployments.

### Usage

```
generate_new_webservice_key(webservice, key_type)
```

### Arguments

| | |
|---|---|
| webservice | The `AciWebservice` or `AksWebservice` object. |
| key_type | A string of which key to regenerate. Options are "Primary" or "Secondary". |

### Value

None

## get_aks_compute_credentials

*Get the credentials for an AksCompute cluster*

### Description

Retrieve the credentials for an AksCompute cluster.

### Usage

```
get_aks_compute_credentials(cluster)
```

### Arguments

| | |
|---|---|
| cluster | The AksCompute object. |

### Value

Named list of the cluster details.

## get_best_run_by_primary_metric

*Return the best performing run amongst all completed runs*

### Description

Find and return the run that corresponds to the best performing run amongst all the completed runs.

The best performing run is identified solely based on the primary metric parameter specified in the HyperDriveConfig (primary_metric_name). The PrimaryMetricGoal governs whether the minimum or maximum of the primary metric is used. To do a more detailed analysis of all the run metrics launched by this HyperDrive run, use get_child_run_metrics(). Only one of the runs is returned from get_best_run_by_primary_metric(), even if several of the runs launched by this HyperDrive run reached the same best metric.

### Usage

```
get_best_run_by_primary_metric(
  hyperdrive_run,
  include_failed = TRUE,
  include_canceled = TRUE
)
```

### Arguments

| | |
|---|---|
| hyperdrive_run | The HyperDriveRun object. |
| include_failed | If TRUE, include the failed runs. |
| include_canceled | If TRUE, include the canceled runs. |

### Value

The Run object.

---

## get_child_runs *Get all children for the current run selected by specified filters*

---

### Description

Get all children for the current run selected by specified filters.

### Usage

```
get_child_runs(
  parent_run,
  recursive = FALSE,
  tags = NULL,
  properties = NULL,
  type = NULL,
  status = NULL
)
```

### Arguments

| | |
|---|---|
| parent_run | The parent Run object. |
| recursive | Boolean indicating whether to recurse through all descendants. |
| tags | If specified, returns runs matching specified "tag" or list(tag = value). |
| properties | If specified, returns runs matching specified "property" or list(property = value). |
| type | If specified, returns runs matching this type. |
| status | If specified, returns runs with status specified "status". |

### Value

A list of child runs, Run objects.

`get_child_runs_sorted_by_primary_metric` *Get the child runs sorted in descending order by best primary metric*

## Description

Return a list of child runs of the HyperDrive run sorted by their best primary metric. The sorting is done according to the primary metric and its goal: if it is maximize, then the child runs are returned in descending order of their best primary metric. If `reverse = TRUE`, the order is reversed. Each child in the result has run id, hyperparameters, best primary metric value, and status.

Child runs without the primary metric are discarded when `discard_no_metric = TRUE`. Otherwise, they are appended to the list behind other child runs with the primary metric. Note that the reverse option has no impact on them.

## Usage

```
get_child_runs_sorted_by_primary_metric(
  hyperdrive_run,
  top = 0L,
  reverse = FALSE,
  discard_no_metric = FALSE
)
```

## Arguments

| | |
|---|---|
| `hyperdrive_run` | The `HyperDriveRun` object. |
| `top` | An integer of the number of top child runs to be returned. If `0` (the default value), all child runs will be returned. |
| `reverse` | If `TRUE`, the order will be reversed. This sorting only impacts child runs with the primary metric. |
| `discard_no_metric` | If `FALSE`, child runs without the primary metric will be appended to the list returned. |

## Value

The named list of child runs.

## get_child_run_hyperparameters *Get the hyperparameters for all child runs*

### Description

Return the hyperparameters for all the child runs of the HyperDrive run.

### Usage

```
get_child_run_hyperparameters(hyperdrive_run)
```

### Arguments

`hyperdrive_run` The `HyperDriveRun` object.

### Value

The named list of hyperparameters where element name is the run_id, e.g. `list("run_id" = hyperparameters)`.

## get_child_run_metrics *Get the metrics from all child runs*

### Description

Return the metrics from all the child runs of the HyperDrive run.

### Usage

```
get_child_run_metrics(hyperdrive_run)
```

### Arguments

`hyperdrive_run` The `HyperDriveRun` object.

### Value

The named list of metrics where element name is the run_id, e.g. `list("run_id" = metrics)`.

## get_compute *Get an existing compute cluster*

### Description

Returns an AmlCompute or AksCompute object for an existing compute resource. If the compute target doesn't exist, the function will return NULL.

### Usage

```
get_compute(workspace, cluster_name)
```

### Arguments

| | |
|---|---|
| workspace | The Workspace object. |
| cluster_name | A string of the name of the cluster. |

### Value

The AmlCompute or AksCompute object.

### Examples

```
## Not run:
ws <- load_workspace_from_config()
compute_target <- get_compute(ws, cluster_name = 'mycluster')

## End(Not run)
```

## get_current_run *Get the context object for a run*

### Description

This function is commonly used to retrieve the authenticated run object inside of a script to be submitted for execution via submit_experiment(). Note that the logging functions (log_* methods, upload_files_to_run(), upload_folder_to_run()) will by default log the specified metrics or files to the run returned from get_current_run().

### Usage

```
get_current_run(allow_offline = TRUE)
```

### Arguments

| | |
|---|---|
| allow_offline | If TRUE, allow the service context to fall back to offline mode so that the training script can be tested locally without submitting a job with the SDK. |

### Value

The Run object.

---

`get_dataset_by_id` *Get Dataset by ID.*

---

### Description

Get a Dataset which is saved to the workspace using its ID.

### Usage

```
get_dataset_by_id(workspace, id)
```

### Arguments

| | |
|---|---|
| `workspace` | The existing AzureML workspace in which the Dataset is saved. |
| `id` | The ID of the dataset |

### Value

The Dataset object

---

`get_dataset_by_name` *Get a registered Dataset from the workspace by its registration name.*

---

### Description

Get a registered Dataset from the workspace by its registration name.

### Usage

```
get_dataset_by_name(workspace, name, version = "latest")
```

### Arguments

| | |
|---|---|
| `workspace` | The existing AzureML workspace in which the Dataset was registered. |
| `name` | The registration name. |
| `version` | The registration version. Defaults to "latest". |

### Value

The registered Dataset object.

## get_datastore *Get an existing datastore*

### Description

Get the corresponding datastore object for an existing datastore by name from the given workspace.

### Usage

```
get_datastore(workspace, datastore_name)
```

### Arguments

| | |
|---|---|
| workspace | The Workspace object. |
| datastore_name | A string of the name of the datastore. |

### Value

The azureml.data.azure_sql_database.AzureBlobDatastore, azureml.data.azure_sql_database.AzureFileData
azureml.data.azure_sql_database.AzureSqlDatabaseDatastore, azureml.data.azure_data_lake_datastore.Az
azureml.data.azure_postgre_sql_datastore.AzurePostgreSqlDatastore, or azureml.data.azure_sql_database
object.

## get_default_datastore *Get the default datastore for a workspace*

### Description

Returns the default datastore associated with the workspace.

When you create a workspace, an Azure blob container and Azure file share are registered to the workspace with the names workspaceblobstore and workspacefilestore, respectively. They store the connection information of the blob container and the file share that is provisioned in the storage account attached to the workspace. The workspaceblobstore is set as the default datastore, and remains the default datastore unless you set a new datastore as the default with set_default_datastore().

### Usage

```
get_default_datastore(workspace)
```

### Arguments

| | |
|---|---|
| workspace | The Workspace object. |

### Value

The default `Datastore` object.

### Examples

Get the default datastore for the datastore:

```
ws <- load_workspace_from_config()
ds <- get_default_datastore(ws)
```

If you have not changed the default datastore for the workspace, the following code will return the same datastore object as the above example:

```
ws <- load_workspace_from_config()
ds <- get_datastore(ws, datastore_name = 'workspaceblobstore')
```

### See Also

`set_default_datastore()`

---

## `get_default_keyvault` *Get the default keyvault for a workspace*

---

### Description

Returns a `Keyvault` object representing the default Azure Key Vault associated with the workspace.

### Usage

```
get_default_keyvault(workspace)
```

### Arguments

| | |
|---|---|
| `workspace` | The `Workspace` object. |

### Value

The `Keyvault` object.

### See Also

`set_secrets()` `get_secrets()` `list_secrets()` `delete_secrets()`

## get_environment *Get an existing environment*

### Description

Returns an `Environment` object for an existing environment in the workspace.

### Usage

```
get_environment(workspace, name, version = NULL)
```

### Arguments

| | |
|---|---|
| `workspace` | The `Workspace` object. |
| `name` | A string of the name of the environment. |
| `version` | A string of the version of the environment. |

### Value

The `Environment` object.

### Examples

```
## Not run:
ws <- load_workspace_from_config()
env <- get_environment(ws, name = 'myenv', version = '1')

## End(Not run)
```

## get_file_dataset_paths *Get a list of file paths for each file stream defined by the dataset.*

### Description

Get a list of file paths for each file stream defined by the dataset. The file paths are relative paths for local files when the file srteam are downloaded or mounted. A common prefix will be removed from the file paths based on how data source was specified to create the dataset.

### Usage

```
get_file_dataset_paths(dataset)
```

### Arguments

| | |
|---|---|
| `dataset` | The Dataset object. |

## Value

A list of file paths.

---

`get_input_dataset_from_run` *Return the named list for input datasets.*

---

## Description

Return the named list for input datasets.

## Usage

```
get_input_dataset_from_run(name, run = NULL)
```

## Arguments

| | |
|---|---|
| `name` | The name of the input dataset |
| `run` | The run taking the dataset as input |

## Value

A dataset object corresponding to the "name"

---

`get_model` *Get a registered model*

---

## Description

Returns a `Model` object for an existing model that has been previously registered to the given workspace.

## Usage

```
get_model(
  workspace,
  name = NULL,
  id = NULL,
  tags = NULL,
  properties = NULL,
  version = NULL,
  run_id = NULL
)
```

### Arguments

| | |
|---|---|
| workspace | The Workspace object. |
| name | Retrieve the latest model with the corresponding name (a string), if it exists. |
| id | Retrieve the model with the corresponding ID (a string), if it exists. |
| tags | (Optional) Retrieve the model filtered based on the provided tags (a list), searching by either 'key' or 'list(key, value)'. |
| properties | (Optional) Retrieve the model filter based on the provided properties (a list), searching by either 'key' or 'list(key, value)'. |
| version | (Optional) An int of the version of a model to retrieve, when provided along with name. The specific version of the specified named model will be returned, if it exists. |
| run_id | (Optional) Retrieve the model filterd by the provided run ID (a string) the model was registered from, if it exists. |

### Value

The Model object.

---

## get_model_package_container_registry

*Get the Azure container registry that a packaged model uses*

---

### Description

Return a ContainerRegistry object for where the image (or base image, for Dockerfile packages) is stored in an Azure container registry.

### Usage

```
get_model_package_container_registry(package)
```

### Arguments

| | |
|---|---|
| package | The ModelPackage object. |

### Value

The ContainerRegistry object.

### See Also

container_registry()

## Examples

```
# Given a ModelPackage object,
# get the container registry information
## Not run:
container_registry <- get_model_package_container_registry(package)
address <- container_registry$address
username <- container_registry$username
password <- container_registry$password

## End(Not run)

# To then authenticate Docker with the Azure container registry from
# a shell or command-line session, use the following command, replacing
# <address>, <username>, and <password> with the values retrieved
# from above:
# ```bash
# docker login <address> -u <username> -p <password>
# ```
```

---

`get_model_package_creation_logs` *Get the model package creation logs*

---

## Description

Retrieve the creation logs from packaging a model with `package_model()`.

## Usage

```
get_model_package_creation_logs(package, decode = TRUE, offset = 0)
```

## Arguments

| | |
|---|---|
| `package` | The `ModelPackage` object. |
| `decode` | If `TRUE`, decode the raw log bytes to a string. |
| `offset` | An int of the byte offset from which to start reading the logs. |

## Value

A string or character vector of package creation logs.

## get_run *Get an experiment run*

### Description

Given the associated experiment and run ID, return the run object for a previously submitted/tracked run.

### Usage

```
get_run(experiment, run_id)
```

### Arguments

| | |
|---|---|
| experiment | The Experiment object. |
| run_id | A string of the run ID for the run. |

### Value

The Run object.

## get_runs_in_experiment *Return a generator of the runs for an experiment*

### Description

Return a generator of the runs for an experiment, in reverse chronological order.

### Usage

```
get_runs_in_experiment(
  experiment,
  type = NULL,
  tags = NULL,
  properties = NULL,
  include_children = FALSE
)
```

### Arguments

| | |
|---|---|
| experiment | The Experiment object. |
| type | Filter the returned generator of runs by the provided type. |
| tags | Filter runs by tags. A named list eg. list("tag" = "value"). |
| properties | Filter runs by properties. A named list eg. list("property" = "value"). |
| include_children | By default, fetch only top-level runs. Set to TRUE to list all runs. |

## Value

The list of runs matching supplied filters.

---

`get_run_details` *Get the details of a run*

---

## Description

Get the definition, status information, current log files, and other details of the run.

## Usage

```
get_run_details(run)
```

## Arguments

| | |
|---|---|
| `run` | The `Run` object. |

## Details

The returned list contains the following named elements:

- *runId*: ID of the run.
- *target*: The compute target of the run.
- *status*: The run's current status.
- *startTimeUtc*: UTC time of when the run was started, in ISO8601.
- *endTimeUtc*: UTC time of when the run was finished (either Completed or Failed), in ISO8601. This element does not exist if the run is still in progress.
- *properties*: Immutable key-value pairs associated with the run.
- *logFiles*: Log files from the run.

## Value

A named list of the details for the run.

## See Also

`get_run_details_with_logs()`

## get_run_details_with_logs *Get the details of a run along with the log files' contents*

### Description

Get the details of a run along with the log files' contents

### Usage

```
get_run_details_with_logs(run)
```

### Arguments

| | |
|---|---|
| run | The Run object. |

### Value

A named list of the run details and log file contents.

### See Also

get_run_details()

## get_run_file_names *List the files that are stored in association with a run*

### Description

Get the list of files stored in a run record.

### Usage

```
get_run_file_names(run)
```

### Arguments

| | |
|---|---|
| run | The Run object. |

### Value

A list of strings of the paths for existing artifacts in the run record.

### See Also

download_file_from_run() download_files_from_run()

`get_run_metrics` *Get the metrics logged to a run*

## Description

Retrieve the metrics logged to a run that were logged with the `log_*()` methods.

## Usage

```
get_run_metrics(
  run,
  name = NULL,
  recursive = FALSE,
  run_type = NULL,
  populate = FALSE
)
```

## Arguments

| | |
|---|---|
| `run` | The `Run` object. |
| `name` | The name of the metric. |
| `recursive` | If specified, returns runs matching specified *"property"* or *"property"*: *"value"*. |
| `run_type` | run type |
| `populate` | Boolean indicating whether to fetch the contents of external data linked to the metric. |

## Value

A named list of the metrics associated with the run, e.g. `list("metric_name" = metric)`.

## Examples

```
ws <- load_workspace_from_config()
exp <- experiment(ws, name = 'myexperiment')
run <- get_run(exp, run_id = "myrunid")
metrics <- get_run_metrics(run)
```

## get_secrets *Get secrets from a keyvault*

### Description

Returns the secret values from the keyvault associated with the workspace for a given set of secret names. For runs submitted using `submit_experiment()`, you can use `get_secrets_from_run()` instead, as that method shortcuts workspace instantiation (since a submitted run is aware of its workspace).

### Usage

```
get_secrets(keyvault, secrets)
```

### Arguments

| | |
|---|---|
| `keyvault` | The `Keyvault` object. |
| `secrets` | A vector of secret names. |

### Value

A named list of found and not found secrets, where element name corresponds to the secret name. If a secret was not found, the corresponding element will be `NULL`.

## get_secrets_from_run *Get secrets from the keyvault associated with a run's workspace*

### Description

From within the script of a run submitted using `submit_experiment()`, you can use `get_secrets_from_run()` to get secrets that are stored in the keyvault of the associated workspace.

Note that this method is slightly different than `get_secrets()`, which first requires you to instantiate the workspace object. Since a submitted run is aware of its workspace, `get_secrets_from_run()` shortcuts workspace instantiation and returns the secret value directly.

Be careful not to expose the secret(s) values by writing or printing them out.

### Usage

```
get_secrets_from_run(run, secrets)
```

### Arguments

| | |
|---|---|
| `run` | The `Run` object. |
| `secrets` | A vector of strings of secret names to retrieve the values for. |

## Value

A named list of found and not found secrets. If a secret was not found, the corresponding element will be NULL.

## See Also

set_secrets()

---

# get_webservice *Get a deployed web service*

---

## Description

Return the corresponding Webservice object of a deployed web service from a given workspace.

## Usage

```
get_webservice(workspace, name)
```

## Arguments

| | |
|---|---|
| workspace | The Workspace object. |
| name | A string of the name of the web service to retrieve. |

## Value

The LocalWebservice, AciWebservice, or AksWebservice object.

---

# get_webservice_keys *Retrieve auth keys for a web service*

---

## Description

Get the authentication keys for a web service that is deployed with key-based authentication enabled. In order to enable key-based authentication, set the auth_enabled = TRUE parameter when you are creating or updating a deployment (either aci_webservice_deployment_config() or aks_webservice_deployment_config() for creation and update_aci_webservice() or update_aks_webservice() for updating). Note that key-based auth is enabled by default for AksWebservice but not for AciWebservice.

To check if a web service has key-based auth enabled, you can access the following boolean property from the Webservice object: service$auth_enabled

Not supported for LocalWebservice deployments.

### Usage

```
get_webservice_keys(webservice)
```

### Arguments

| | |
|---|---|
| webservice | The AciWebservice or AksWebservice object. |

### Value

A list of two strings corresponding to the primary and secondary authentication keys.

### See Also

```
generate_new_webservice_key()
```

---

## get_webservice_logs *Retrieve the logs for a web service*

---

### Description

You can get the detailed Docker engine log messages from your web service deployment. You can view the logs for local, ACI, and AKS deployments.

For example, if your web service deployment fails, you can inspect the logs to help troubleshoot.

### Usage

```
get_webservice_logs(webservice, num_lines = 5000L)
```

### Arguments

| | |
|---|---|
| webservice | The LocalWebservice, AciWebservice, or AksWebservice object. |
| num_lines | An int of the maximum number of log lines to retrieve. |

### Value

A string of the logs for the web service.

## get_webservice_token *Retrieve the auth token for a web service*

### Description

Get the authentication token, scoped to the current user, for a web service that was deployed with token-based authentication enabled. Token-based authentication requires clients to use an Azure Active Directory account to request an authentication token, which is used to make requests to the deployed service. Only available for AKS deployments.

In order to enable token-based authentication, set the `token_auth_enabled = TRUE` parameter when you are creating or updating a deployment (`aks_webservice_deployment_config()` for creation or `update_aks_webservice()` for updating). Note that you cannot have both key-based authentication and token-based authentication enabled. Token-based authentication is not enabled by default.

To check if a web service has token-based auth enabled, you can access the following boolean property from the Webservice object: `service$token_auth_enabled`

### Usage

```
get_webservice_token(webservice)
```

### Arguments

| | |
|---|---|
| `webservice` | The `AksWebservice` object. |

### Value

An `AksServiceAccessToken` object.

## get_workspace *Get an existing workspace*

### Description

Returns a `Workspace` object for an existing Azure Machine Learning workspace. Throws an exception if the workpsace doesn't exist or the required fields don't lead to a uniquely identifiable workspace.

### Usage

```
get_workspace(name, auth = NULL, subscription_id = NULL, resource_group = NULL)
```

### Arguments

| | |
|---|---|
| name | A string of the workspace name to get. |
| auth | The ServicePrincipalAuthentication or InteractiveLoginAuthentication object. For more details refer to https://aka.ms/aml-notebook-auth. If NULL, the default Azure CLI credentials will be used or the API will prompt for credentials. |
| subscription_id | A string of the subscription ID to use. The parameter is required if the user has access to more than one subscription. |
| resource_group | A string of the resource group to use. If NULL the method will search all resource groups in the subscription. |

### Value

The Workspace object.

### See Also

create_workspace() service_principal_authentication() interactive_login_authentication()

---

## get_workspace_details *Get the details of a workspace*

---

### Description

Returns the details of the workspace.

### Usage

```
get_workspace_details(workspace)
```

### Arguments

| | |
|---|---|
| workspace | The Workspace object. |

### Value

Named list of the workspace details.

### Details

The returned named list contains the following elements:

- *id*: URI pointing to the workspace resource, containing subscription ID, resource group, and workspace name.
- *name*: Workspace name.
- *location*: Workspace region.

- *type*: URI of the format `"{providerName}/workspaces"`.
- *workspaceid*: Workspace ID.
- *description*: Workspace description.
- *friendlyName*: Workspace friendly name.
- *creationTime*: Time the workspace was created, in ISO8601.
- *containerRegistry*: Workspace container registry.
- *keyVault*: Workspace key vault.
- *applicationInsights*: Workspace App Insights.
- *identityPrincipalId*: Workspace identity principal ID.
- *identityTenantId*: Workspace tenant ID.
- *identityType*: Workspace identity type.
- *storageAccount*: Workspace storage account.

---

`github_package` *Specifies a Github package to install in environment*

---

## Description

Specifies a Github package to install in run environment

## Usage

```
github_package(repository, auth_token = NULL)
```

## Arguments

| | |
|---|---|
| `repository` | Repository address of the github package |
| `auth_token` | Personal access token to install from a private repo. |

## Value

A named list containing the package specifications

## Examples

```
pkg1 <- github_package("Azure/azureml-sdk-for-r")

env <- r_environment(name = "r_env",
                     github_packages = list(pkg1))
```

## See Also

`r_environment()`

## grid_parameter_sampling *Define grid sampling over a hyperparameter search space*

### Description

Grid sampling performs a simple grid search over all feasible values in the defined search space. It can only be used with hyperparameters specified using `choice()`.

### Usage

```
grid_parameter_sampling(parameter_space)
```

### Arguments

| | |
|---|---|
| `parameter_space` | A named list containing each parameter and its distribution, e.g. `list("parameter" = distribution)`. |

### Value

The `GridParameterSampling` object.

### See Also

`choice()`

### Examples

```
## Not run:
param_sampling <- grid_parameter_sampling(list("num_hidden_layers" = choice(c(1, 2, 3)),
                                               "batch_size" = choice(c(16, 32))))

## End(Not run)
```

## hyperdrive_config *Create a configuration for a HyperDrive run*

### Description

The HyperDrive configuration includes information about hyperparameter space sampling, termination policy, primary metric, estimator, and the compute target to execute the experiment runs on.

To submit the HyperDrive experiment, pass the `HyperDriveConfig` object returned from this method to `submit_experiment()`.

## Usage

```
hyperdrive_config(
  hyperparameter_sampling,
  primary_metric_name,
  primary_metric_goal,
  max_total_runs,
  max_concurrent_runs = NULL,
  max_duration_minutes = 10080L,
  policy = NULL,
  estimator = NULL
)
```

## Arguments

| | |
|---|---|
| hyperparameter_sampling | The hyperparameter sampling space. Can be a RandomParameterSampling, GridParameterSampling, or BayesianParameterSampling object. |
| primary_metric_name | A string of the name of the primary metric reported by the experiment runs. |
| primary_metric_goal | The PrimaryMetricGoal object. This parameter determines if the primary metric is to be minimized or maximized when evaluating runs. |
| max_total_runs | An integer of the maximum total number of runs to create. This is the upper bound; there may be fewer runs when the sample space is smaller than this value. If both max_total_runs and max_duration_minutes are specified, the hyperparameter tuning experiment terminates when the first of these two thresholds is reached. |
| max_concurrent_runs | An integer of the maximum number of runs to execute concurrently. If NULL, all runs are launched in parallel. The number of concurrent runs is gated on the resources available in the specified compute target. Hence, you need to ensure that the compute target has the available resources for the desired concurrency. |
| max_duration_minutes | An integer of the maximum duration of the HyperDrive run. Once this time is exceeded, any runs still executing are cancelled. If both max_total_runs and max_duration_minutes are specified, the hyperparameter tuning experiment terminates when the first of these two thresholds is reached. |
| policy | The early termination policy to use. Can be either a BanditPolicy, MedianStoppingPolicy, or TruncationSelectionPolicy object. If NULL (the default), no early termination policy will be used.<br>The MedianStoppingPolicy with delay_evaluation of = 5 is a good termination policy to start with. These are conservative settings that can provide 25%-35% savings with no loss on primary metric (based on our evaluation data). |
| estimator | The Estimator object. |

## Value

The HyperDriveConfig object.

## See Also

submit_experiment()

## Examples

```
## Not run:
# Load the workspace
ws <- load_workspace_from_config()

# Get the compute target
compute_target <- get_compute(ws, cluster_name = 'mycluster')

# Define the primary metric goal
goal = primary_metric_goal("MAXIMIZE")

# Define the early termination policy
early_termination_policy = median_stopping_policy(evaluation_interval = 1L,
                                                  delay_evaluation = 5L)

# Create the estimator
est <- estimator(source_directory = '.',
                 entry_script = 'train.R',
                 compute_target = compute_target)

# Create the HyperDrive configuration
hyperdrive_run_config = hyperdrive_config(
                                  hyperparameter_sampling = param_sampling,
                                  primary_metric_name = 'accuracy',
                                  primary_metric_goal = goal,
                                  max_total_runs = 100,
                                  max_concurrent_runs = 4,
                                  policy = early_termination_policy,
                                  estimator = est)

# Submit the HyperDrive experiment
exp <- experiment(ws, name = 'myexperiment')
run = submit_experiment(exp, hyperdrive_run_config)

## End(Not run)
```

---

# inference_config *Create an inference configuration for model deployments*

---

## Description

The inference configuration describes how to configure the model to make predictions. It references your scoring script (entry_script) and is used to locate all the resources required for the deployment. Inference configurations use Azure Machine Learning environments (see r_environment()) to define the software dependencies needed for your deployment.

## Usage

```
inference_config(
  entry_script,
  source_directory = ".",
  description = NULL,
  environment = NULL
)
```

## Arguments

| | |
|---|---|
| entry_script | A string of the path to the local file that contains the code to run for making predictions. |
| source_directory | A string of the path to the local folder that contains the files to package and deploy alongside your model, such as helper files for your scoring script (entry_script). The folder must contain the entry_script. |
| description | (Optional) A string of the description to give this configuration. |
| environment | An Environment object to use for the deployment. The environment does not have to be registered. |

## Value

The InferenceConfig object.

## Defining the entry script

To deploy a model, you must provide an entry script that accepts requests, scores the requests by using the model, and returns the results. The entry script is specific to your model. It must understand the format of the incoming request data, the format of the data expected by your model, and the format of the data returned to clients. If the request data is in a format that is not usable by your model, the script can transform it into an acceptable format. It can also transform the response before returning it to the client.

The entry script must contain an init() method that loads your model and then returns a function that uses the model to make a prediction based on the input data passed to the function. Azure ML runs the init() method once, when the Docker container for your web service is started. The prediction function returned by init() will be run every time the service is invoked to make a prediction on some input data. The inputs and outputs of this prediction function typically use JSON for serialization and deserialization.

To locate the model in your entry script (when you load the model in the script's init() method), use AZUREML_MODEL_DIR, an environment variable containing the path to the model location. The environment variable is created during service deployment, and you can use it to find the location of your deployed model(s).

To get the path to a file in a model, combine the environment variable with the filename you're looking for. The filenames of the model files are preserved during registration and deployment.

Single model example:

```
model_path <- file.path(Sys.getenv("AZUREML_MODEL_DIR"), "my_model.rds")
```

Multiple model example:

```
model1_path <- file.path(Sys.getenv("AZUREML_MODEL_DIR"), "my_model/1/my_model.rds")
```

## See Also

r_environment(), deploy_model()

---

`install_azureml` *Install azureml sdk package*

---

## Description

Install azureml sdk package

## Usage

```
install_azureml(
  version = "1.10.0",
  envname = "r-reticulate",
  conda_python_version = "3.6",
  restart_session = TRUE,
  remove_existing_env = FALSE
)
```

## Arguments

| | |
|---|---|
| version | azureml sdk package version |
| envname | name of environment to create, if environment other than default is desired |
| conda_python_version | version of python for conda environment |
| restart_session | restart R session after installation |
| remove_existing_env | delete the conda environment if already exists |

## Value

None

`interactive_login_authentication` *Manages authentication and acquires an authorization token in interactive login workflows.*

## Description

Interactive login authentication is suitable for local experimentation on your own computer, and is the default authentication model when using Azure Machine Learning SDK. The constructor of the class will prompt you to login. The constructor then will save the credentials for any subsequent attempts. If you are already logged in with the Azure CLI or have logged-in before, the constructor will load the existing credentials without prompt.

## Usage

```
interactive_login_authentication(
  force = FALSE,
  tenant_id = NULL,
  cloud = "AzureCloud"
)
```

## Arguments

| | |
|---|---|
| `force` | Indicates whether "az login" will be run even if the old "az login" is still valid. |
| `tenant_id` | The string id of the active directory tenant that the service identity belongs to. This is can be used to specify a specific tenant when you have access to multiple tenants. If unspecified, the default tenant will be used. |
| `cloud` | The name of the target cloud. Can be one of "AzureCloud", "AzureChinaCloud", or "AzureUSGovernment". If no cloud is specified, "AzureCloud" is used. |

## Value

`InteractiveLoginAuthentication` object

## Examples

```
interactive_auth <- interactive_login_authentication(tenant_id="your-tenant-id")

ws <- get_workspace("<your workspace name>",
                    "<your subscription ID>",
                    "<your resource group>",
                    auth = interactive_auth)
```

## See Also

`get_workspace()` `service_principal_authentication()`

## invoke_webservice *Call a web service with the provided input*

### Description

Invoke the web service with the provided input and to receive predictions from the deployed model. The structure of the provided input data needs to match what the service's scoring script and model expect. See the "Details" section of `inference_config()`.

### Usage

```
invoke_webservice(webservice, input_data)
```

### Arguments

| | |
|---|---|
| `webservice` | The `LocalWebservice`, `AciWebservice`, or `AksWebservice` object. |
| `input_data` | The input data to invoke the web service with. This is the data your model expects as an input to run predictions. |

### Details

Instead of invoking the web service using `invoke_webservice()`, you can also consume the web service using the service's REST API. If you've enabled key-based authentication for your service, you will need to provide a service key as a token in your request header (see `get_webservice_keys()`). If you've enabled token-based authentication, you will need to provide an JWT token as a bearer token in your request header (see `get_webservice_token()`).

To get the REST API address for the service's scoring endpoint, you can access the following property from the Webservice object: `service$scoring_uri`

### Value

A named list of the result of calling the web service. This will return the predictions run from your model.

## keep_columns_from_dataset *Keep the specified columns and drops all others from the dataset.*

### Description

Keep the specified columns and drops all others from the dataset. If a timeseries column is dropped, the corresponding capabilities will be dropped for the returned dataset as well.

### Usage

```
keep_columns_from_dataset(dataset, columns, validate = FALSE)
```

### Arguments

| | |
|---|---|
| dataset | The Tabular Dataset object |
| columns | The name or a list of names for the columns to keep. |
| validate | Indicates whether to validate if data can be loaded from the returned dataset. The default is False. Validation requires that the data source is accessible from current compute. |

### Value

A new Tabular Dataset object with only the specified columns kept.

---

## list_nodes_in_aml_compute

*Get the details (e.g IP address, port etc) of all the compute nodes in the compute target*

---

### Description

Get the details (e.g IP address, port etc) of all the compute nodes in the compute target

### Usage

```
list_nodes_in_aml_compute(cluster)
```

### Arguments

| | |
|---|---|
| cluster | cluster object |

### Value

Details of all the compute nodes in the cluster in data frame

---

## list_secrets

*List the secrets in a keyvault*

---

### Description

Returns the list of secret names for all the secrets in the keyvault associated with the workspace.

### Usage

```
list_secrets(keyvault)
```

### Arguments

| | |
|---|---|
| keyvault | The Keyvault object. |

### Value

A list of secret names.

---

`list_supported_vm_sizes` *List the supported VM sizes in a region*

---

### Description

List the supported VM sizes in a region

### Usage

```
list_supported_vm_sizes(workspace, location = NULL)
```

### Arguments

| | |
|---|---|
| `workspace` | The `Workspace` object. |
| `location` | A string of the location of the cluster. If not specified, will default to the workspace location. |

### Value

A data frame of supported VM sizes in a region with name of the VM, VCPUs, RAM.

---

`list_workspaces` *List all workspaces that the user has access to in a subscription ID*

---

### Description

List all workspaces that the user has access to in the specified `subscription_id` parameter. The list of workspaces can be filtered based on the resource group.

### Usage

```
list_workspaces(subscription_id, resource_group = NULL)
```

### Arguments

| | |
|---|---|
| `subscription_id` | A string of the specified subscription ID to list the workspaces in. |
| `resource_group` | A string of the specified resource group to list the workspaces. If `NULL` the method will list all the workspaces within the specified subscription in. |

### Value

A named list of `Workspace` objects where element name corresponds to the workspace name.

load_dataset_into_data_frame
*Load all records from the dataset into a dataframe.*

## Description

Load all records from the dataset into a dataframe.

## Usage

```
load_dataset_into_data_frame(
  dataset,
  on_error = "null",
  out_of_range_datetime = "null"
)
```

## Arguments

| | |
|---|---|
| dataset | The Tabular Dataset object. |
| on_error | How to handle any error values in the dataset, such as those produced by an error while parsing values. Valid values are 'null' which replaces them with NULL; and 'fail' which will result in an exception. |
| out_of_range_datetime | How to handle date-time values that are outside the range supported by Pandas. Valid values are 'null' which replaces them with NULL; and 'fail' which will result in an exception. |

## Value

A data.frame.

load_workspace_from_config
*Load workspace configuration details from a config file*

## Description

Returns a `Workspace` object for an existing Azure Machine Learning workspace by reading the workspace configuration from a file. The method provides a simple way of reusing the same workspace across multiple files or projects. Users can save the workspace ARM properties using `write_workspace_config()`, and use this method to load the same workspace in different files or projects without retyping the workspace ARM properties.

## Usage

```
load_workspace_from_config(path = NULL, file_name = NULL)
```

### Arguments

| | |
|---|---|
| path | A string of the path to the config file or starting directory for search. The parameter defaults to starting the search in the current directory. |
| file_name | A string that will override the config file name to search for when path is a directory path. |

### Value

The Workspace object.

### See Also

write_workspace_config()

---

## local_webservice_deployment_config

*Create a deployment config for deploying a local web service*

---

### Description

You can deploy a model locally for limited testing and troubleshooting. To do so, you will need to have Docker installed on your local machine.

If you are using an Azure Machine Learning Compute Instance for development, you can also deploy locally on your compute instance.

### Usage

```
local_webservice_deployment_config(port = NULL)
```

### Arguments

| | |
|---|---|
| port | An int of the local port on which to expose the service's HTTP endpoint. |

### Value

The LocalWebserviceDeploymentConfiguration object.

### Examples

```
## Not run:
deployment_config <- local_webservice_deployment_config(port = 8890)

## End(Not run)
```

## lognormal *Specify a normal distribution of the form* `exp(normal(mu, sigma))`

### Description

Specify a normal distribution of the form `exp(normal(mu, sigma))`.

The logarithm of the return value is normally distributed. When optimizing, this variable is constrained to be positive.

### Usage

```
lognormal(mu, sigma)
```

### Arguments

| | |
|---|---|
| `mu` | A double of the mean of the normal distribution. |
| `sigma` | A double of the standard deviation of the normal distribution. |

### Value

A list of the stochastic expression.

### See Also

`random_parameter_sampling()`, `grid_parameter_sampling()`, `bayesian_parameter_sampling()`

## loguniform *Specify a log uniform distribution*

### Description

Specify a log uniform distribution.

A value is drawn according to `exp(uniform(min_value, max_value))` so that the logarithm of the return value is uniformly distributed. When optimizing, this variable is constrained to the interval `[exp(min_value), exp(max_value)]`.

### Usage

```
loguniform(min_value, max_value)
```

### Arguments

| | |
|---|---|
| `min_value` | A double where the minimum value in the range will be `exp(min_value)` (inclusive). |
| `max_value` | A double where the maximum value in the range will be `exp(min_value)` (inclusive). |

### Value

A list of the stochastic expression.

### See Also

`random_parameter_sampling()`, `grid_parameter_sampling()`, `bayesian_parameter_sampling()`

---

`log_accuracy_table_to_run`

*Log an accuracy table metric to a run*

---

### Description

The accuracy table metric is a multi-use non-scalar metric that can be used to produce multiple types of line charts that vary continuously over the space of predicted probabilities. Examples of these charts are ROC, precision-recall, and lift curves.

### Usage

```
log_accuracy_table_to_run(name, value, description = "", run = NULL)
```

### Arguments

| | |
|---|---|
| `name` | A string of the name of the metric. |
| `value` | A named list containing name, version, and data properties. |
| `description` | (Optional) A string of the metric description. |
| `run` | The `Run` object. If not specified, will default to the current run from the service context. |

### Details

The calculation of the accuracy table is similar to the calculation of an ROC curve. An ROC curve stores true positive rates and false positive rates at many different probability thresholds. The accuracy table stores the raw number of true positives, false positives, true negatives, and false negatives at many probability thresholds.

There are two methods used for selecting thresholds: "probability" and "percentile." They differ in how they sample from the space of predicted probabilities.

Probability thresholds are uniformly spaced thresholds between 0 and 1. If NUM_POINTS were 5 the probability thresholds would be c(0.0, 0.25, 0.5, 0.75, 1.0).

Percentile thresholds are spaced according to the distribution of predicted probabilities. Each threshold corresponds to the percentile of the data at a probability threshold. For example, if NUM_POINTS were 5, then the first threshold would be at the 0th percentile, the second at the 25th percentile, the third at the 50th, and so on.

The probability tables and percentile tables are both 3D lists where the first dimension represents the class label, the second dimension represents the sample at one threshold (scales with

NUM_POINTS), and the third dimension always has 4 values: TP, FP, TN, FN, and always in that order.

The confusion values (TP, FP, TN, FN) are computed with the one vs. rest strategy. See the following link for more details: https://en.wikipedia.org/wiki/Multiclass_classification.

N = # of samples in validation dataset (200 in example), M = # thresholds = # samples taken from the probability space (5 in example), C = # classes in full dataset (3 in example)

Some invariants of the accuracy table:

- TP + FP + TN + FN = N for all thresholds for all classes
- TP + FN is the same at all thresholds for any class
- TN + FP is the same at all thresholds for any class
- Probability tables and percentile tables have shape (C, M, 4)

Note: M can be any value and controls the resolution of the charts. This is independent of the dataset, is defined when calculating metrics, and trades off storage space, computation time, and resolution.

Class labels should be strings, confusion values should be integers, and thresholds should be doubles.

## Value

None

---

`log_confusion_matrix_to_run` *Log a confusion matrix metric to a run*

---

## Description

Log a confusion matrix metric to a run

## Usage

```
log_confusion_matrix_to_run(name, value, description = "", run = NULL)
```

## Arguments

| | |
|---|---|
| `name` | A string of the name of the metric. |
| `value` | A named list containing name, version, and data properties. |
| `description` | (Optional) A string of the metric description. |
| `run` | The `Run` object. If not specified, will default to the current run from the service context. |

## Value

None

## log_image_to_run *Log an image metric to a run*

### Description

Log an image to the run with the give metric name. Use `log_image_to_run()` to log an image file or ggplot2 plot to the run. These images will be visible and comparable in the run record.

### Usage

```
log_image_to_run(name, path = NULL, plot = NULL, description = "", run = NULL)
```

### Arguments

| | |
|---|---|
| `name` | A string of the name of the metric. |
| `path` | A string of the path or stream of the image. |
| `plot` | The ggplot2 plot to log as an image. |
| `description` | (Optional) A string of the metric description. |
| `run` | The `Run` object. If not specified, will default to the current run from the service context. |

### Value

None

## log_list_to_run *Log a vector metric value to a run*

### Description

Log a vector with the given metric name to the run.

### Usage

```
log_list_to_run(name, value, description = "", run = NULL)
```

### Arguments

| | |
|---|---|
| `name` | A string of the name of metric. |
| `value` | The vector of elements to log. |
| `description` | (Optional) A string of the metric description. |
| `run` | The `Run` object. If not specified, will default to the current run from the service context. |

## Value

None

## Examples

```
log_list_to_run("Accuracies", c(0.6, 0.7, 0.87))
```

---

`log_metric_to_run` *Log a metric to a run*

---

## Description

Log a numerical or string value with the given metric name to the run. Logging a metric to a run causes that metric to be stored in the run record in the experiment. You can log the same metric multiple times within a run, the result being considered a vector of that metric.

## Usage

```
log_metric_to_run(name, value, run = NULL)
```

## Arguments

| | |
|---|---|
| `name` | A string of the name of the metric. |
| `value` | The value of the metric. |
| `run` | The `Run` object. If not specified, will default to the current run from the service context. |

## Value

None

## Examples

```
log_metric_to_run("Accuracy", 0.95)
```

log_predictions_to_run *Log a predictions metric to a run*

## Description

`log_predictions_to_run()` logs a metric score that can be used to compare the distributions of true target values to the distribution of predicted values for a regression task.

The predictions are binned and standard deviations are calculated for error bars on a line chart.

## Usage

```
log_predictions_to_run(name, value, description = "", run = NULL)
```

## Arguments

| | |
|---|---|
| `name` | A string of the name of the metric. |
| `value` | A named list containing name, version, and data properties. |
| `description` | (Optional) A string of the metric description. |
| `run` | The `Run` object. If not specified, will default to the current run from the service context. |

## Value

None

## Examples

```
data <- list("bin_averages" = c(0.25, 0.75),
             "bin_errors" = c(0.013, 0.042),
             "bin_counts" = c(56, 34),
             "bin_edges" = c(0.0, 0.5, 1.0))
predictions <- list("schema_type" = "predictions",
                    "schema_version" = "v1",
                    "data" = data)
log_predictions_to_run("mypredictions", predictions)
```

## log_residuals_to_run *Log a residuals metric to a run*

### Description

log_residuals_to_run() logs the data needed to display a histogram of residuals for a regression task. The residuals are predicted - actual.

There should be one more edge than the number of counts.

### Usage

```
log_residuals_to_run(name, value, description = "", run = NULL)
```

### Arguments

| | |
|---|---|
| name | A string of the name of the metric. |
| value | A named list containing name, version, and data properties. |
| description | (Optional) A string of the metric description. |
| run | The Run object. If not specified, will default to the current run from the service context. |

### Value

None

### Examples

```
data <- list("bin_edges" = c(50, 100, 200, 300, 350),
             "bin_counts" = c(0.88, 20, 30, 50.99))
residuals <- list("schema_type" = "residuals",
                  "schema_version" = "v1",
                  "data" = data)
log_predictions_to_run("myresiduals", predictions)
```

## log_row_to_run *Log a row metric to a run*

### Description

Using log_row_to_run() creates a metric with multiple columns as described in .... Each named parameter generates a column with the value specified. log_row_to_run() can be called once to log an arbitrary tuple, or multiple times in a loop to generate a complete table.

### Usage

```
log_row_to_run(name, description = "", run = NULL, ...)
```

### Arguments

| | |
|---|---|
| name | A string of the name of metric. |
| description | (Optional) A string of the metric description. |
| run | The Run object. If not specified, will default to the current run from the service context. |
| ... | Each named parameter generates a column with the value specified. |

### Value

None

### Examples

Log an arbitrary tuple:

```
log_row_to_run("Y over X", x = 1, y = 0.4)
```

Log the complete table:

```
citrus <- c("orange", "lemon", "lime")
sizes <- c(10, 7, 3)
for (i in seq_along(citrus)) {
    log_row_to_run("citrus", fruit = citrus[i], size = sizes[i])
}
```

---

## log_table_to_run *Log a table metric to a run*

---

### Description

Log a table metric with the given metric name to the run. The table value is a named list where each element corresponds to a column of the table.

### Usage

```
log_table_to_run(name, value, description = "", run = NULL)
```

### Arguments

| | |
|---|---|
| name | A string of the name of metric. |
| value | The table value of the metric (a named list where the element name corresponds to the column name). |
| description | (Optional) A string of the metric description. |
| run | The Run object. If not specified, will default to the current run from the service context. |

## Value

None

## Examples

```
log_table_to_run("Y over X",
                 list("x" = c(1, 2, 3), "y" = c(0.6, 0.7, 0.89)))
```

---

`median_stopping_policy` *Define a median stopping policy for early termination of HyperDrive runs*

---

## Description

Median stopping is an early termination policy based on running averages of primary metrics reported by the runs. This policy computes running averages across all training runs and terminates runs whose performance is worse than the median of the running averages. Specifically, a run will be canceled at interval N if its best primary metric reported up to interval N is worse than the median of the running averages for intervals 1:N across all runs.

## Usage

```
median_stopping_policy(evaluation_interval = 1L, delay_evaluation = 0L)
```

## Arguments

`evaluation_interval`
: An integer of the frequency for applying policy.

`delay_evaluation`
: An integer of the number of intervals for which to delay the first evaluation.

## Value

The `MedianStoppingPolicy` object.

## Details

The median stopping policy takes the following optional configuration parameters:

- `evaluation_interval`: Optional. The frequency for applying the policy. Each time the training script logs the primary metric counts as one interval.
- `delay_evaluation`: Optional. The number of intervals to delay the policy evaluation. Use this parameter to avoid premature termination of training runs. If specified, the policy applies every multiple of `evaluation_interval` that is greater than or equal to `delay_evaluation`.

This policy is inspired from the research publication Google Vizier: A Service for Black-Box Optimization.

If you are looking for a conservative policy that provides savings without terminating promising jobs, you can use a `MedianStoppingPolicy` with `evaluation_interval` = 1 and `delay_evaluation` = 5. These are conservative settings that can provide approximately 25%-35% savings with no loss on the primary metric (based on our evaluation data).

## Examples

```
# In this example, the early termination policy is applied at every
# interval starting at evaluation interval 5. A run will be terminated at
# interval 5 if its best primary metric is worse than the median of the
# running averages over intervals 1:5 across all training runs
## Not run:
early_termination_policy = median_stopping_policy(evaluation_interval = 1L,
                                                  delay_evaluation = 5L)

## End(Not run)
```

---

# merge_results *Combine the results from the parallel training.*

---

## Description

Combine the results from the parallel training.

## Usage

```
merge_results(node_count, process_count_per_node, run, source_directory)
```

## Arguments

| | |
|---|---|
| `node_count` | Number of nodes in the AmlCompute cluster. |
| `process_count_per_node` | Number of processes per node. |
| `run` | The run object whose output needs to be combined. |
| `source_directory` | The directory where the output from the run would be downloaded. |

## mount_file_dataset *Create a context manager for mounting file streams defined by the dataset as local files.*

### Description

Create a context manager for mounting file streams defined by the dataset as local files. A context manager will be returned to manage the lifecycle of the mount. To mount, you will need to enter the context manager and to unmount, exit from the context manager. Mount is only supported on Unix or Unix-like operating systems and libfuse must be present. If you are running inside a docker container, the docker container must be started with the `--privileged` flag or started with `--cap-add SYS_ADMIN --device /dev/fuse`.

### Usage

```
mount_file_dataset(dataset, mount_point = NULL)
```

### Arguments

| | |
|---|---|
| `dataset` | The Dataset object. |
| `mount_point` | The local directory to mount the files to. If NULL, the data will be mounted into a temporary directory. |

### Value

Returns a context manager for managing the lifecycle of the mount of type `azureml.dataprep.fuse.daemon.MountContext`

## normal *Specify a real value that is normally-distributed with mean* `mu` *and standard deviation* `sigma`

### Description

Specify a real value that is normally-distributed with mean `mu` and standard deviation `sigma`.

When optimizing, this is an unconstrained variable.

### Usage

```
normal(mu, sigma)
```

### Arguments

| | |
|---|---|
| `mu` | A double of the mean of the normal distribution. |
| `sigma` | A double of the standard deviation of the normal distribution. |

### Value

A list of the stochastic expression.

### See Also

`random_parameter_sampling()`, `grid_parameter_sampling()`, `bayesian_parameter_sampling()`

---

`package_model` *Create a model package that packages all the assets needed to host a model as a web service*

---

### Description

In some cases, you might want to create a Docker image without deploying the model (for example, if you plan to deploy to Azure App Service). Or you might want to download the image and run it on a local Docker installation. You might even want to download the files used to build the image, inspect them, modify them, and build the image manually.

Model packaging enables you to do these things. `package_model()` packages all the assets needed to host a model as a web service and allows you to download either a fully built Docker image or the files needed to build one. There are two ways to use model packaging:

- **Download a packaged model**: Download a Docker image that contains the model and other files needed to host it as a web service.
- **Generate a Dockerfile**: Download the Dockerfile, model, entry script, and other assets needed to build a Docker image. You can then inspect the files or make changes before you build the image locally. To use this method, make sure to set `generate_dockerfile = TRUE`. With either scenario, you will need to have Docker installed in your development environment.

### Usage

```
package_model(workspace, models, inference_config, generate_dockerfile = FALSE)
```

### Arguments

| | |
|---|---|
| `workspace` | The `Workspace` object. |
| `models` | A list of `Model` objects to include in the package. Can be an empty list. |
| `inference_config` | The `InferenceConfig` object to configure the operation of the models. |
| `generate_dockerfile` | If `TRUE`, will create a Dockerfile that can be run locally instead of building an image. |

### Value

The `ModelPackage` object.

## See Also

wait_for_model_package_creation(), get_model_package_container_registry(), get_model_package_creation_
pull_model_package_image(), save_model_package_files()

## Examples

```
# Package a registered model
## Not run:
ws <- load_workspace_from_config()
model <- get_model(ws, name = "my_model")
r_env <- r_environment(name = "r_env")
inference_config <- inference_config(entry_script = "score.R",
                                     source_directory = ".",
                                     environment = r_env)
package <- package_model(ws,
                         models = list(model),
                         inference_config = inference_config)
wait_for_model_package_creation(show_output = TRUE)

## End(Not run)
```

---

# plot_run_details *Generate table of run details*

---

## Description

Plot a table of run details including

- ID
- Status
- Start Time
- Duration
- Script Name
- Arguments
- Link to Web Portal view
- Errors

## Usage

```
plot_run_details(run)
```

## Arguments

| | |
|---|---|
| run | The Run object. |

## Value

Datatable containing run details

## primary_metric_goal *Define supported metric goals for hyperparameter tuning*

### Description

A metric goal is used to determine whether a higher value for a metric is better or worse. Metric goals are used when comparing runs based on the primary metric. For example, you may want to maximize accuracy or minimize error.

The primary metric name and goal are specified to `hyperdrive_config()` when you configure a HyperDrive run.

### Usage

```
primary_metric_goal(goal)
```

### Arguments

| | |
|---|---|
| `goal` | A string of the metric goal (either "MAXIMIZE" or "MINIMIZE"). |

### Value

The `PrimaryMetricGoal` object.

## promote_headers_behavior *Defines options for how column headers are processed when reading data from files to create a dataset.*

### Description

Defines options for how column headers are processed when reading data from files to create a dataset. These enumeration values are used in the Dataset class method.

### Usage

```
promote_headers_behavior(option)
```

### Arguments

| | |
|---|---|
| `option` | An integer corresponding to an option for how column headers are to be processed<br>• 0: NO_HEADERS No column headers are read<br>• 1: ONLY_FIRST_FILE_HAS_HEADERS Read headers only from first row of first file, everything else is data. |

- 2: COMBINE_ALL_FILES_HEADERS Read headers from first row of each file, combining identically named columns.
- 3: ALL_FILES_HAVE_SAME_HEADERS Read headers from first row of first file, drops first row from other files.

## Value

The PromoteHeadersBehavior object.

---

`pull_model_package_image` *Pull the Docker image from a* `ModelPackage` *to your local Docker environment*

---

## Description

Pull the Docker image from a created `ModelPackage` to your local Docker environment. The output of this call will display the name of the image. For example: `Status: Downloaded newer image for myworkspacef78fd10`

This can only be used with a Docker image `ModelPackage` (where `package_model()` was called with `generate_dockerfile = FALSE`).

After you've pulled the image, you can start a local container based on this image using Docker commands.

## Usage

```
pull_model_package_image(package)
```

## Arguments

| | |
|---|---|
| `package` | The `ModelPackage` object. |

## Value

None

## See Also

`package_model()`

---

`qlognormal` *Specify a normal distribution of the form* `round(exp(normal(mu, sigma)) / q) * q`

---

## Description

Specify a normal distribution of the form `round(exp(normal(mu, sigma)) / q) * q`.

Suitable for a discrete variable with respect to which the objective is smooth and gets smoother with the size of the variable, which is bounded from one side.

## Usage

```
qlognormal(mu, sigma, q)
```

## Arguments

| | |
|---|---|
| `mu` | A double of the mean of the normal distribution. |
| `sigma` | A double of the standard deviation of the normal distribution. |
| `q` | An integer of the smoothing factor. |

## Value

A list of the stochastic expression.

## See Also

`random_parameter_sampling()`, `grid_parameter_sampling()`, `bayesian_parameter_sampling()`

---

`qloguniform` *Specify a uniform distribution of the form* `round(exp(uniform(min_value, max_value) / q) * q`

---

## Description

Specify a uniform distribution of the form `round(exp(uniform(min_value, max_value) / q) * q`.

This is suitable for a discrete variable with respect to which the objective is "smooth", and gets smoother with the size of the value, but which should be bounded both above and below.

## Usage

```
qloguniform(min_value, max_value, q)
```

### Arguments

| | |
|---|---|
| min_value | A double of the minimum value in the range (inclusive). |
| max_value | A double of the maximum value in the range (inclusive). |
| q | An integer of the smoothing factor. |

### Value

A list of the stochastic expression.

### See Also

random_parameter_sampling(), grid_parameter_sampling(), bayesian_parameter_sampling()

---

## qnormal *Specify a normal distribution of the* form round(normal(mu, sigma) / q) * q

---

### Description

Specify a normal distribution of the form round(normal(mu, sigma) / q) * q.

Suitable for a discrete variable that probably takes a value around mu, but is fundamentally unbounded.

### Usage

```
qnormal(mu, sigma, q)
```

### Arguments

| | |
|---|---|
| mu | A double of the mean of the normal distribution. |
| sigma | A double of the standard deviation of the normal distribution. |
| q | An integer of the smoothing factor. |

### Value

A list of the stochastic expression.

### See Also

random_parameter_sampling(), grid_parameter_sampling(), bayesian_parameter_sampling()

## quniform *Specify a uniform distribution of the form* `round(uniform(min_value, max_value) / q) * q`

### Description

Specify a uniform distribution of the form `round(uniform(min_value, max_value) / q) * q`.

This is suitable for a discrete value with respect to which the objective is still somewhat "smooth", but which should be bounded both above and below.

### Usage

```
quniform(min_value, max_value, q)
```

### Arguments

| | |
|---|---|
| `min_value` | A double of the minimum value in the range (inclusive). |
| `max_value` | A double of the maximum value in the range (inclusive). |
| `q` | An integer of the smoothing factor. |

### Value

A list of the stochastic expression.

### See Also

`random_parameter_sampling()`, `grid_parameter_sampling()`, `bayesian_parameter_sampling()`

## randint *Specify a set of random integers in the range* `[0, upper)`

### Description

Specify a set of random integers in the range `[0, upper)` to sample the hyperparameters from.

The semantics of this distribution is that there is no more correlation in the loss function between nearby integer values, as compared with more distant integer values. This is an appropriate distribution for describing random seeds, for example. If the loss function is probably more correlated for nearby integer values, then you should probably use one of the "quantized" continuous distributions, such as either `quniform()`, `qloguniform()`, `qnormal()`, or `qlognormal()`.

### Usage

```
randint(upper)
```

### Arguments

| | |
|---|---|
| upper | An integer of the upper bound for the range of integers (exclusive). |

### Value

A list of the stochastic expression.

### See Also

random_parameter_sampling(), grid_parameter_sampling(), bayesian_parameter_sampling()

---

## random_parameter_sampling

*Define random sampling over a hyperparameter search space*

---

### Description

In random sampling, hyperparameter values are randomly selected from the defined search space. Random sampling allows the search space to include both discrete and continuous hyperparameters.

### Usage

```
random_parameter_sampling(parameter_space, properties = NULL)
```

### Arguments

| | |
|---|---|
| parameter_space | A named list containing each parameter and its distribution, e.g. list("parameter" = distribution). |
| properties | A named list of additional properties for the algorithm. |

### Value

The RandomParameterSampling object.

### Details

In this sampling algorithm, parameter values are chosen from a set of discrete values or a distribution over a continuous range. Functions you can use include: choice(), randint(), uniform(), quniform(), loguniform(), qloguniform(), normal(), qnormal(), lognormal(), and qlognormal().

### See Also

choice(), randint(), uniform(), quniform(), loguniform(), qloguniform(), normal(), qnormal(), lognormal(), qlognormal()

## Examples

```
## Not run:
param_sampling <- random_parameter_sampling(list("learning_rate" = normal(10, 3),
                                                 "keep_probability" = uniform(0.05, 0.1),
                                                "batch_size" = choice(c(16, 32, 64, 128))))

## End(Not run)
```

---

`random_split_dataset` *Split file streams in the dataset into two parts randomly and approximately by the percentage specified.*

---

## Description

Split file streams in the dataset into two parts randomly and approximately by the percentage specified.

## Usage

```
random_split_dataset(dataset, percentage, seed = NULL)
```

## Arguments

| | |
|---|---|
| `dataset` | The Dataset object. |
| `percentage` | The approximate percentage to split the Dataset by. This must be a number between 0.0 and 1.0. |
| `seed` | An optional seed to use for the random generator. |

## Value

A new Dataset object representing the two datasets after the split.

---

`register_azure_blob_container_datastore`
*Register an Azure blob container as a datastore*

---

## Description

Register an Azure blob container as a datastore. You can choose to use either the SAS token or the storage account key.

## Usage

```
register_azure_blob_container_datastore(
  workspace,
  datastore_name,
  container_name,
  account_name,
  sas_token = NULL,
  account_key = NULL,
  protocol = NULL,
  endpoint = NULL,
  overwrite = FALSE,
  create_if_not_exists = FALSE,
  skip_validation = FALSE,
  blob_cache_timeout = NULL,
  grant_workspace_access = FALSE,
  subscription_id = NULL,
  resource_group = NULL
)
```

## Arguments

| | |
|---|---|
| workspace | The Workspace object. |
| datastore_name | A string of the name of the datastore. The name must be case insensitive and can only contain alphanumeric characters and underscores. |
| container_name | A string of the name of the Azure blob container. |
| account_name | A string of the storage account name. |
| sas_token | A string of the account SAS token. |
| account_key | A string of the storage account key. |
| protocol | A string of the protocol to use to connect to the blob container. If NULL, defaults to 'https'. |
| endpoint | A string of the endpoint of the blob container. If NULL, defaults to 'core.windows.net'. |
| overwrite | If TRUE, overwrites an existing datastore. If the datastore does not exist, it will create one. |
| create_if_not_exists | If TRUE, creates the blob container if it does not exists. |
| skip_validation | If TRUE, skips validation of storage keys. |
| blob_cache_timeout | An integer of the cache timeout in seconds when this blob is mounted. If NULL, defaults to no timeout (i.e. blobs will be cached for the duration of the job when read). |
| grant_workspace_access | If TRUE, grants workspace Managed Identities (MSI) access to the user storage account. This should be set to TRUE if the storage account is in VNET. If TRUE, Azure ML will use the workspace MSI token to grant access to the user storage account. It may take a while for the granted access to reflect. |

`subscription_id`
: A string of the subscription id of the storage account.

`resource_group`
: A string of the resource group of the storage account.

## Value

The `AzureBlobDatastore` object.

## Details

In general we recommend Azure Blob storage over Azure File storage. Both standard and premium storage are available for blobs. Although more expensive, we suggest premium storage due to faster throughput speeds that may improve the speed of your training runs, particularly if you train against a large dataset.

## Examples

```
## Not run:
ws <- load_workspace_from_config()
ds <- register_azure_blob_container_datastore(ws,
                                              datastore_name = 'mydatastore',
                                            container_name = 'myazureblobcontainername',
                                              account_name = 'mystorageaccoutname',
                                              account_key = 'mystorageaccountkey')

## End(Not run)
```

---

# register_azure_data_lake_gen2_datastore

*Initialize a new Azure Data Lake Gen2 Datastore.*

---

## Description

Initialize a new Azure Data Lake Gen2 Datastore.

## Usage

```
register_azure_data_lake_gen2_datastore(
  workspace,
  datastore_name,
  filesystem,
  account_name,
  tenant_id,
  client_id,
  client_secret,
  resource_url = NULL,
  authority_url = NULL,
  protocol = NULL,
```

```
  endpoint = NULL,
  overwrite = FALSE
)
```

## Arguments

| | |
|---|---|
| workspace | The workspace this datastore belongs to. |
| datastore_name | The datastore name. |
| filesystem | The name of the Data Lake Gen2 filesystem. |
| account_name | The storage account name. |
| tenant_id | The Directory ID/Tenant ID of the service principal. |
| client_id | The Client ID/Application ID of the service principal. |
| client_secret | The secret of the service principal. |
| resource_url | The resource URL, which determines what operations will be performed on the data lake store, defaults to https://storage.azure.com/ which allows us to perform filesystem operations. |
| authority_url | The authority URL used to authenticate the user, defaults to "https://login.microsoftonline.com". |
| protocol | Protocol to use to connect to the blob container. If None, defaults to "https". |
| endpoint | The endpoint of the blob container. If None, defaults to "core.windows.net". |
| overwrite | Whether to overwrite an existing datastore. If the datastore does not exist, it will create one. The default is FALSE. |

## Value

The `azureml.data.azure_data_lake_datastore.AzureDataLakeGen2Datastore` object.

## Examples

```
# Create and register an Azure Data Lake Gen2 Datastore to a workspace.

my_adlsgen2_ds <- register_azure_data_lake_gen2_datastore(workspace = your_workspace,
                                       datastore_name = <name for this datastore>,
                                                        filesystem = 'test',
                                       tenant_id = your_workspace$auth$tenant_id,
                                       client_id = your_workspace$auth$service_principal_id,
                                       client_secret = your_workspace$auth$service_principal_passwor
```

## See Also

unregister_datastore(), get_datastore()

`register_azure_file_share_datastore` *Register an Azure file share as a datastore*

## Description

Register an Azure file share as a datastore. You can choose to use either the SAS token or the storage account key.

## Usage

```
register_azure_file_share_datastore(
  workspace,
  datastore_name,
  file_share_name,
  account_name,
  sas_token = NULL,
  account_key = NULL,
  protocol = NULL,
  endpoint = NULL,
  overwrite = FALSE,
  create_if_not_exists = FALSE,
  skip_validation = FALSE
)
```

## Arguments

| | |
|---|---|
| `workspace` | The `Workspace` object. |
| `datastore_name` | A string of the name of the datastore. The name must be case insensitive and can only contain alphanumeric characters and underscores. |
| `file_share_name` | A string of the name of the Azure file share. |
| `account_name` | A string of the storage account name. |
| `sas_token` | A string of the account SAS token. |
| `account_key` | A string of the storage account key. |
| `protocol` | A string of the protocol to use to connect to the file store. If `NULL`, defaults to `'https'`. |
| `endpoint` | A string of the endpoint of the file store. If `NULL`, defaults to `'core.windows.net'`. |
| `overwrite` | If `TRUE`, overwrites an existing datastore. If the datastore does not exist, it will create one. |
| `create_if_not_exists` | If `TRUE`, creates the file share if it does not exists. |
| `skip_validation` | If `TRUE`, skips validation of storage keys. |

## Value

The AzureFileDatastore object.

## Details

In general we recommend Azure Blob storage over Azure File storage. Both standard and premium storage are available for blobs. Although more expensive, we suggest premium storage due to faster throughput speeds that may improve the speed of your training runs, particularly if you train against a large dataset.

## Examples

```
## Not run:
ws <- load_workspace_from_config()
ds <- register_azure_file_share_datastore(ws,
                                          datastore_name = 'mydatastore',
                                          file_share_name = 'myazurefilesharename',
                                          account_name = 'mystorageaccoutname',
                                          account_key = 'mystorageaccountkey')

## End(Not run)
```

---

# register_azure_postgre_sql_datastore

*Initialize a new Azure PostgreSQL Datastore.*

---

## Description

Initialize a new Azure PostgreSQL Datastore.

## Usage

```
register_azure_postgre_sql_datastore(
  workspace,
  datastore_name,
  server_name,
  database_name,
  user_id,
  user_password,
  port_number = NULL,
  endpoint = NULL,
  overwrite = FALSE
)
```

### Arguments

| | |
|---|---|
| workspace | The workspace this datastore belongs to. |
| datastore_name | The datastore name. |
| server_name | The PostgreSQL server name. |
| database_name | The PostgreSQL database name. |
| user_id | The User ID of the PostgreSQL server. |
| user_password | The User Password of the PostgreSQL server. |
| port_number | The Port Number of the PostgreSQL server. |
| endpoint | The endpoint of the PostgreSQL server. If NULL, defaults to postgres.database.azure.com. |
| overwrite | Whether to overwrite an existing datastore. If the datastore does not exist, it will create one. The default is FALSE. |

### Value

The `azureml.data.azure_postgre_sql_datastore.AzurePostgreSqlDatastore` object.

---

## register_azure_sql_database_datastore *Initialize a new Azure SQL database Datastore.*

---

### Description

Initialize a new Azure SQL database Datastore.

### Usage

```
register_azure_sql_database_datastore(
  workspace,
  datastore_name,
  server_name,
  database_name,
  tenant_id,
  client_id,
  client_secret,
  resource_url = NULL,
  authority_url = NULL,
  endpoint = NULL,
  overwrite = FALSE,
  username = NULL,
  password = NULL
)
```

### Arguments

| | |
|---|---|
| workspace | The workspace this datastore belongs to. |
| datastore_name | The datastore name. |
| server_name | The SQL server name. |
| database_name | The SQL database name. |
| tenant_id | The Directory ID/Tenant ID of the service principal. |
| client_id | The Client ID/Application ID of the service principal. |
| client_secret | The secret of the service principal. |
| resource_url | The resource URL, which determines what operations will be performed on the SQL database store, if NULL, defaults to https://database.windows.net/. |
| authority_url | The authority URL used to authenticate the user, defaults to https://login.microsoftonline.com. |
| endpoint | The endpoint of the SQL server. If NULL, defaults to database.windows.net. |
| overwrite | Whether to overwrite an existing datastore. If the datastore does not exist, it will create one. The default is FALSE. |
| username | The username of the database user to access the database. |
| password | The password of the database user to access the database. |

### Value

The `azureml.data.azure_sql_database_datastore.AzureSqlDatabaseDatastore` object.

---

## register_dataset *Register a Dataset in the workspace*

---

### Description

Register the Dataset in the workspace, making it available to other users of the workspace.

### Usage

```
register_dataset(
  workspace,
  dataset,
  name,
  description = NULL,
  tags = NULL,
  create_new_version = FALSE
)
```

### Arguments

| | |
|---|---|
| workspace | The AzureML workspace in which the Dataset is to be registered. |
| dataset | The dataset to be registered. |
| name | The name of the Dataset in the workspace. |
| description | A description of the Dataset. |
| tags | Named list of tags to give the Dataset. Defaults to NULL. |
| create_new_version | Boolean to register the dataset as a new version under the specified name. |

### Value

The registered Dataset object.

---

## register_do_azureml_parallel *Registers AMLCompute as a parallel backend with the foreach package.*

---

### Description

Registers AMLCompute as a parallel backend with the foreach package.

### Usage

```
register_do_azureml_parallel(workspace, compute_target)
```

### Arguments

| | |
|---|---|
| workspace | The Workspace object which has the compute_target. |
| compute_target | The AMLCompute target to use for parallelization. |

---

## register_environment *Register an environment in the workspace*

---

### Description

The environment is automatically registered with your workspace when you submit an experiment or deploy a web service. You can also manually register the environment with `register_environment()`. This operation makes the environment into an entity that is tracked and versioned in the cloud, and can be shared between workspace users.

Whe used for the first time in training or deployment, the environment is registered with the workspace, built, and deployed on the compute target. The environments are cached by the service. Reusing a cached environment takes much less time than using a new service or one that has bee updated.

### Usage

```
register_environment(environment, workspace)
```

### Arguments

| | |
|---|---|
| environment | The Environment object. |
| workspace | The Workspace object. |

### Value

The Environment object.

---

## register_model *Register a model to a given workspace*

---

### Description

Register a model to the given workspace. A registered model is a logical container for one or more files that make up your model. For example, if you have a model that's stored in multiple files, you can register them as a single model in your workspace. After registration, you can then download or deploy the registered model and receive all the files that were registered.

Models are identified by name and version. Each time you register a model with the same name as an existing one, your workspace's model registry assumes that it's a new version. The version is incremented, and the new model is registered under the same name.

### Usage

```
register_model(
  workspace,
  model_path,
  model_name,
  datasets = NULL,
  tags = NULL,
  properties = NULL,
  description = NULL,
  child_paths = NULL,
  sample_input_dataset = NULL,
  sample_output_dataset = NULL,
  resource_configuration = NULL
)
```

## Arguments

| | |
|---|---|
| workspace | The Workspace object. |
| model_path | A string of the path on the local file system where the model assets are located. This can be a direct pointer to a single file or folder. If pointing to a folder, the child_paths parameter can be used to specify individual files to bundle together as the Model object, as opposed to using the entire contents of the folder. |
| model_name | A string of the name to register the model with. |
| datasets | A list of two-element lists where the first element is the dataset-model relationship and the second is the corresponding dataset, e.g. list(list("training", train_ds), list("inferencing", infer_ds)). Valid values for the data-model relationship are 'training', 'validation', and 'inferencing'. |
| tags | A named list of key-value tags to give the model, e.g. list("key" = "value") |
| properties | A named list of key-value properties to give the model, e.g. list("key" = "value"). |
| description | A string of the text description of the model. |
| child_paths | A list of strings of child paths of a folder specified by model_name. Must be provided in conjunction with a model_path pointing to a folder; only the specified files will be bundled into the Model object. |
| sample_input_dataset | Sample input dataset for the registered model. |
| sample_output_dataset | Sample output dataset for the registered model. |
| resource_configuration | 'ResourceConfiguration" object to run the registered model. |

## Value

The Model object.

## Examples

Registering a model from a single file:

```
ws <- load_workspace_from_config()
model <- register_model(ws,
                        model_path = "my_model.rds",
                        model_name = "my_model",
                        datasets = list(list("training", train_dataset)))
```

## See Also

resource_configuration()

`register_model_from_run` *Register a model for operationalization.*

## Description

Register a model for operationalization.

## Usage

```
register_model_from_run(
  run,
  model_name,
  model_path = NULL,
  tags = NULL,
  properties = NULL,
  description = NULL,
  datasets = NULL,
  sample_input_dataset = NULL,
  sample_output_dataset = NULL,
  resource_configuration = NULL
)
```

## Arguments

| | |
|---|---|
| `run` | The Run object. |
| `model_name` | The name of the model. |
| `model_path` | The relative cloud path to the model, for example, "outputs/modelname". When not specified, `model_name` is used as the path. |
| `tags` | A dictionary of key value tags to assign to the model. |
| `properties` | A dictionary of key value properties to assign to the model. These properties cannot be changed after model creation, however new key-value pairs can be added. |
| `description` | An optional description of the model. |
| `datasets` | A list of two-element lists where the first element is the dataset-model relationship and the second is the corresponding dataset, e.g. `list(list("training", train_ds), list("inferencing", infer_ds))`. Valid values for the data-model relationship are 'training', 'validation', and 'inferencing'. |
| `sample_input_dataset` | Sample input dataset for the registered model. |
| `sample_output_dataset` | Sample output dataset for the registered model. |
| `resource_configuration` | 'ResourceConfiguration" object to run the registered model. |

## Value

The registered Model.

## Examples

```
registered_model <- register_model_from_run(run = run,
                                            model_name = "my model",
                                            model_path = 'outputs/model.rds',
                                            tags = list("version" = "0"),
                                    datasets = list(list("training", train_dataset),
                                                    list("validation", validation_dataset)),
                                  resource_configuration = resource_configuration(2, 2, 0))
```

## See Also

`resource_configuration()`

---

# reload_local_webservice_assets

*Reload a local web service's entry script and dependencies*

---

## Description

This restarts the service's container with copies of updated assets, including the entry script and local dependencies, but it does not rebuild the underlying image. Accordingly, changes to the environment will not be reflected in the reloaded local web service. To handle those changes call `update_local_webservice()` instead.

## Usage

```
reload_local_webservice_assets(webservice, wait = FALSE)
```

## Arguments

| | |
|---|---|
| `webservice` | The `LocalWebservice` object. |
| `wait` | If `TRUE`, wait for the service's container to reach a healthy state. Defaults to `FALSE`. |

## Value

None

resource_configuration *Initialize the ResourceConfiguration.*

## Description

Initialize the ResourceConfiguration.

## Usage

```
resource_configuration(cpu = NULL, memory_in_gb = NULL, gpu = NULL)
```

## Arguments

| | |
|---|---|
| cpu | The number of CPU cores to allocate for this resource. Can be a decimal. |
| memory_in_gb | The amount of memory (in GB) to allocate for this resource. Can be a decimal If TRUE, decode the raw log bytes to a string. |
| gpu | The number of GPUs to allocate for this resource. |

## Value

The ResourceConfiguration object.

## See Also

register_model_from_run

## Examples

```
## Not run:
rc <- resource_configuration(2, 2, 0)

registered_model <- register_model_from_run(run, "my_model_name",
                                            "path_to_my_model",
                                            resource_configuration = rc)

## End(Not run)
```

r_environment *Create an environment*

## Description

Configure the R environment to be used for training or web service deployments. When you submit a run or deploy a model, Azure ML builds a Docker image and creates a conda environment with your specifications from your `Environment` object within that Docker container.

If the `custom_docker_image` parameter is not set, Azure ML will build a predefined base image (CPU or GPU depending on the `use_gpu` flag) and install any R packages specified in the `cran_packages`, `github_packages`, or `custom_url_packages` parameters.

## Usage

```
r_environment(
  name,
  version = NULL,
  environment_variables = NULL,
  r_version = NULL,
  rscript_path = NULL,
  snapshot_date = NULL,
  cran_packages = NULL,
  github_packages = NULL,
  custom_url_packages = NULL,
  bioconductor_packages = NULL,
  custom_docker_image = NULL,
  image_registry_details = NULL,
  use_gpu = FALSE,
  shm_size = NULL
)
```

## Arguments

| | |
|---|---|
| `name` | A string of the name of the environment. |
| `version` | A string of the version of the environment. |
| `environment_variables` | A named list of environment variables names and values. These environment variables are set on the process where the user script is being executed. |
| `r_version` | The version of R to be installed. |
| `rscript_path` | The Rscript path to use if an environment build is not required. The path specified gets used to call the user script. |
| `snapshot_date` | Date of MRAN snapshot to use. |
| `cran_packages` | A list of `cran_package` objects to be installed. |
| `github_packages` | A list of `github_package` objects to be installed. |

`custom_url_packages`
: A character vector of packages to be installed from local directory or custom URL.

`bioconductor_packages`
: A character vector of packages to be installed from Bioconductor.

`custom_docker_image`
: A string of the name of the Docker image from which the image to use for training or deployment will be built. If not set, a predefined Docker image will be used. To use an image from a private Docker repository, you will also have to specify the `image_registry_details` parameter.

`image_registry_details`
: A `ContainerRegistry` object of the details of the Docker image registry for the custom Docker image.

`use_gpu`
: Indicates whether the environment should support GPUs. If `TRUE`, a predefined GPU-based Docker image will be used in the environment. If `FALSE`, a predefined CPU-based image will be used. Predefined Docker images (CPU or GPU) will only be used if the `custom_docker_image` parameter is not set.

`shm_size`
: A string for the size of the Docker container's shared memory block. For more information, see Docker run reference If not set, a default value of `'2g'` is used.

## Value

The `Environment` object.

## Details

Once built, the Docker image appears in the Azure Container Registry associated with your workspace, by default. The repository name has the form *azureml/azureml_<uuid>*. The unique identifier (*uuid*) part corresponds to a hash computed from the environment configuration. This allows the service to determine whether an image corresponding to the given environment already exists for reuse.

If you make changes to an existing environment, such as adding an R package, a new version of the environment is created when you either submit a run, deploy a model, or manually register the environment. The versioning allows you to view changes to the environment over time.

## Predefined Docker images

When submitting a training job or deploying a model, Azure ML runs your training script or scoring script within a Docker container. If no custom Docker image is specified with the `custom_docker_image` parameter, Azure ML will build a predefined CPU or GPU Docker image. The predefine images extend the Ubuntu 16.04 Azure ML base images and include the following dependencies:

| Dependencies | Version | |
|---|---|---|
| azuremlsdk | latest | |
| R | 3.6.0 | |
| Commonly used R packages | - | 80+ of the most popular R packages for |
| Python | 3.7.0 | |
| azureml-defaults | latest | `azureml-defaults` contains the `azureml-core` and `applicationinsights` packag |

| | |
|---|---|
| rpy2 | latest |
| CUDA (GPU image only) | 10.0 |

## See Also

`estimator()`, `inference_config()`

## Examples

```
# The following example defines an environment that will build the default
# base CPU image.
## Not run:
r_env <- r_environment(name = 'myr_env',
                       version = '1')

## End(Not run)
```

---

`save_model_package_files`

*Save a Dockerfile and dependencies from a* `ModelPackage` *to your local file system*

---

## Description

Download the Dockerfile, model, and other assets needed to build an image locally from a created `ModelPackage`.

This can only be used with a Dockerfile `ModelPackage` (where `package_model()` was called with `generate_dockerfile = TRUE` to indicated that you wanted only the files and not a fully built image).

`save_model_package_files()` downloads the files needed to build the image to the `output_directory`. The Dockerfile included in the saved files references a base image stored in an Azure container registry. When you build the image on your local Docker installation, you will need the address, username, and password to authenticate to the registry. You can get this information using `get_model_package_container_registry()`.

## Usage

```
save_model_package_files(package, output_directory)
```

## Arguments

| | |
|---|---|
| `package` | The `ModelPackage` object. |
| `output_directory` | A string of the local directory that will be created to contain the contents of the package. |

## Value

None

## See Also

`package_model()`, `get_model_package_container_registry()`

---

`service_principal_authentication` *Manages authentication using a service principle instead of a user identity.*

---

## Description

Service Principal authentication is suitable for automated workflows like for CI/CD scenarios. This type of authentication decouples the authentication process from any specific user login, and allows for managed access control.

## Usage

```
service_principal_authentication(
  tenant_id,
  service_principal_id,
  service_principal_password,
  cloud = "AzureCloud"
)
```

## Arguments

| | |
|---|---|
| `tenant_id` | The string id of the active directory tenant that the service identity belongs to. |
| `service_principal_id` | The service principal ID string. |
| `service_principal_password` | The service principal password/key string. |
| `cloud` | The name of the target cloud. Can be one of "AzureCloud", "AzureChinaCloud", or "AzureUSGovernment". If no cloud is specified, "AzureCloud" is used. |

## Value

`ServicePrincipalAuthentication` object

## Examples

Service principal authentication involves creating an App Registration in Azure Active Directory. First, you generate a client secret, and then you grant your service principal role access to your machine learning workspace. Then, you use the `ServicePrincipalAuthentication` object to manage your authentication flow.

```
svc_pr_password <- Sys.getenv("AZUREML_PASSWORD")
svc_pr <- service_principal_authentication(tenant_id="my-tenant-id",
                                           service_principal_id="my-application-id",
                                          service_principal_password=svc_pr_password)

ws <- get_workspace("<your workspace name>",
                    "<your subscription ID>",
                    "<your resource group>",
                    auth = svc_pr)
```

## See Also

`get_workspace()` `interactive_login_authentication()`

---

`set_default_datastore` *Set the default datastore for a workspace*

---

## Description

Set the default datastore associated with the workspace.

## Usage

```
set_default_datastore(workspace, datastore_name)
```

## Arguments

| | |
|---|---|
| `workspace` | The `Workspace` object. |
| `datastore_name` | The name of the datastore to be set as default. |

## Value

None

## See Also

`get_default_datastore()`

## set_secrets *Add secrets to a keyvault*

### Description

Add a named list of secrets into the keyvault associated with the workspace.

### Usage

```
set_secrets(keyvault, secrets)
```

### Arguments

| | |
|---|---|
| keyvault | The Keyvault object. |
| secrets | The named list of secrets to be added to the keyvault, where element name corresponds to the secret name. |

### Value

None

### Examples

```
## Not run:
ws <- load_workspace_from_config()
my_secret <- Sys.getenv("MY_SECRET")
keyvault <- get_default_keyvault(ws)
set_secrets(list("mysecret" = my_secret))

## End(Not run)
```

## skip_from_dataset *Skip file streams from the top of the dataset by the specified count.*

### Description

Skip file streams from the top of the dataset by the specified count.

### Usage

```
skip_from_dataset(dataset, count)
```

### Arguments

| | |
|---|---|
| dataset | The Dataset object. |
| count | The number of file streams to skip. |

### Value

A new Dataset object representing the dataset with file streams skipped.

---

## split_tasks *Splits the job into parallel tasks.*

---

### Description

Splits the job into parallel tasks.

### Usage

```
split_tasks(args_list, node_count, process_count_per_node)
```

### Arguments

| | |
|---|---|
| args_list | The list of arguments which are distributed across all the processes. |
| node_count | Number of nodes in the AmlCompute cluster. |
| process_count_per_node | Number of processes per node. |

---

## start_logging_run *Create an interactive logging run*

---

### Description

Create an interactive run that allows the user to log metrics and artifacts to a run locally.

Any metrics that are logged during the interactive run session are added to the run record in the experiment. If an output directory is specified, the contents of that directory is uploaded as run artifacts upon run completion.

This method is useful if you would like to add experiment tracking and artifact logging to the corresponding run record in Azure ML for local runs without have to submit an experiment run to a compute target with `submit_experiment()`.

### Usage

```
start_logging_run(experiment, outputs = NULL, snapshot_directory = NULL)
```

### Arguments

| | |
|---|---|
| experiment | The `Experiment` object. |
| outputs | (Optional) A string of the local path to an outputs directory to track. |
| snapshot_directory | (Optional) Directory to take snapshot of. Setting to `NULL` will take no snapshot. |

## Value

The Run object of the started run.

## See Also

complete_run()

## Examples

```
## Not run:
ws <- load_workspace_from_config()
exp <- experiment(ws, name = 'myexperiment')
run <- start_logging_run(exp)
log_metric_to_run("Accuracy", 0.9)
complete_run(run)

## End(Not run)
```

---

# submit_child_run *Submit an experiment and return the active child run*

---

## Description

Submit an experiment and return the active child run.

## Usage

```
submit_child_run(parent_run, config = NULL, tags = NULL)
```

## Arguments

| | |
|---|---|
| parent_run | The parent Run object. |
| config | The RunConfig object |
| tags | Tags to be added to the submitted run, e.g., "tag": "value". |

## Value

A Run object.

submit_experiment *Submit an experiment and return the active created run*

## Description

submit_experiment() is an asynchronous call to Azure Machine Learning service to execute a trial on local or remote compute. Depending on the configuration, submit_experiment() will automatically prepare your execution environments, execute your code, and capture your source code and results in the experiment's run history.

To submit an experiment you first need to create a configuration object describing how the experiment is to be run. The configuration depends on the type of trial required. For a script run, provide an Estimator object to the config parameter. For a HyperDrive run for hyperparameter tuning, provide a HyperDriveConfig to config.

## Usage

```
submit_experiment(experiment, config, tags = NULL)
```

## Arguments

| | |
|---|---|
| experiment | The Experiment object. |
| config | The Estimator or HyperDriveConfig object. |
| tags | A named list of tags for the submitted run, e.g. list("tag" = "value"). |

## Value

The ScriptRun or HyperDriveRun object.

## See Also

estimator(), hyperdrive_config()

## Examples

```
# This example submits an Estimator experiment
## Not run:
ws <- load_workspace_from_config()
compute_target <- get_compute(ws, cluster_name = 'mycluster')
exp <- experiment(ws, name = 'myexperiment')
est <- estimator(source_directory = '.',
                 entry_script = 'train.R',
                 compute_target = compute_target)
run <- submit_experiment(exp, est)

## End(Not run)
```

---

`take_from_dataset` *Take a sample of file streams from top of the dataset by the specified count.*

---

## Description

Take a sample of file streams from top of the dataset by the specified count.

## Usage

```
take_from_dataset(dataset, count)
```

## Arguments

| | |
|---|---|
| `dataset` | The Dataset object. |
| `count` | The number of file streams to take. |

## Value

A new Dataset object representing the sampled dataset.

---

`take_sample_from_dataset` *Take a random sample of file streams in the dataset approximately by the probability specified.*

---

## Description

Take a random sample of file streams in the dataset approximately by the probability specified.

## Usage

```
take_sample_from_dataset(dataset, probability, seed = NULL)
```

## Arguments

| | |
|---|---|
| `dataset` | The Dataset object. |
| `probability` | The probability of a file stream being included in the sample. |
| `seed` | An optional seed to use for the random generator. |

## Value

A new Dataset object representing the sampled dataset.

`truncation_selection_policy` *Define a truncation selection policy for early termination of HyperDrive runs*

## Description

Truncation selection cancels a given percentage of lowest performing runs at each evaluation interval. Runs are compared based on their performance on the primary metric and the lowest X% are terminated.

## Usage

```
truncation_selection_policy(
  truncation_percentage,
  evaluation_interval = 1L,
  delay_evaluation = 0L
)
```

## Arguments

`truncation_percentage`
: An integer of the percentage of lowest performing runs to terminate at each interval.

`evaluation_interval`
: An integer of the frequency for applying policy.

`delay_evaluation`
: An integer of the number of intervals for which to delay the first evaluation.

## Value

The `TruncationSelectionPolicy` object.

## Details

This policy periodically cancels the given percentage of runs that rank the lowest for their performance on the primary metric. The policy strives for fairness in ranking the runs by accounting for improving model performance with training time. When ranking a relatively young run, the policy uses the corresponding (and earlier) performance of older runs for comparison. Therefore, runs aren't terminated for having a lower performance because they have run for less time than other runs.

The truncation selection policy takes the following configuration parameters:

- `truncation_percentage`: An integer of the percentage of lowest performing runs to terminate at each evaluation interval.
- `evaluation_interval`: Optional. The frequency for applying the policy. Each time the training script logs the primary metric counts as one interval.

- delay_evaluation: Optional. The number of intervals to delay the policy evaluation. Use this parameter to avoid premature termination of training runs. If specified, the policy applies every multiple of evaluation_interval that is greater than or equal to delay_evaluation.

For example, when evaluating a run at a interval N, its performance is only compared with the performance of other runs up to interval N even if they reported metrics for intervals greater than N.

## Examples

```
# In this example, the early termination policy is applied at every interval
# starting at evaluation interval 5. A run will be terminated at interval 5
# if its performance at interval 5 is in the lowest 20% of performance of all
# runs at interval 5
## Not run:
early_termination_policy = truncation_selection_policy(
                                                    truncation_percentage = 20L,
                                                    evaluation_interval = 1L,
                                                    delay_evaluation = 5L)

## End(Not run)
```

---

# uniform *Specify a uniform distribution of options to sample from*

---

## Description

Specify a uniform distribution of options to sample the hyperparameters from.

## Usage

```
uniform(min_value, max_value)
```

## Arguments

| | |
|---|---|
| min_value | A double of the minimum value in the range (inclusive). |
| max_value | A double of the maximum value in the range (inclusive). |

## Value

A list of the stochastic expression.

## See Also

random_parameter_sampling(), grid_parameter_sampling(), bayesian_parameter_sampling()

## unregister_all_dataset_versions

*Unregister all versions under the registration name of this dataset from the workspace.*

### Description

Unregister all versions under the registration name of this dataset from the workspace.

### Usage

```
unregister_all_dataset_versions(dataset)
```

### Arguments

| | |
|---|---|
| dataset | The dataset to be unregistered. |

### Value

None

## unregister_datastore *Unregister a datastore from its associated workspace*

### Description

Unregister the datastore from its associated workspace. The underlying Azure storage will not be deleted.

### Usage

```
unregister_datastore(datastore)
```

### Arguments

| | |
|---|---|
| datastore | The AzureBlobDatastore or AzureFileDatastore object. |

### Value

None

update_aci_webservice *Update a deployed ACI web service*

## Description

Update an ACI web service with the provided properties. You can update the web service to use a new model, a new entry script, or new dependencies that can be specified in an inference configuration.

Values left as NULL will remain unchanged in the web service.

## Usage

```
update_aci_webservice(
  webservice,
  tags = NULL,
  properties = NULL,
  description = NULL,
  auth_enabled = NULL,
  ssl_enabled = NULL,
  ssl_cert_pem_file = NULL,
  ssl_key_pem_file = NULL,
  ssl_cname = NULL,
  enable_app_insights = NULL,
  models = NULL,
  inference_config = NULL
)
```

## Arguments

| | |
|---|---|
| webservice | The AciWebservice object. |
| tags | A named list of key-value tags for the web service, e.g. list("key" = "value"). Will replace existing tags. |
| properties | A named list of key-value properties to add for the web service, e.g. list("key" = "value"). |
| description | A string of the description to give the web service. |
| auth_enabled | If TRUE enable key-based authentication for the web service. |
| ssl_enabled | Whether or not to enable SSL for this Webservice. |
| ssl_cert_pem_file | A string of the cert file needed if SSL is enabled. |
| ssl_key_pem_file | A string of the key file needed if SSL is enabled. |
| ssl_cname | A string of the cname if SSL is enabled. |
| enable_app_insights | If TRUE enable AppInsights for the web service. |

`models` A list of `Model` objects to package into the updated service.

`inference_config` An `InferenceConfig` object.

## Value

None

---

`update_aks_webservice` *Update a deployed AKS web service*

---

## Description

Update an AKS web service with the provided properties. You can update the web service to use a new model, a new entry script, or new dependencies that can be specified in an inference configuration.

Values left as `NULL` will remain unchanged in the web service.

## Usage

```
update_aks_webservice(
  webservice,
  autoscale_enabled = NULL,
  autoscale_min_replicas = NULL,
  autoscale_max_replicas = NULL,
  autoscale_refresh_seconds = NULL,
  autoscale_target_utilization = NULL,
  auth_enabled = NULL,
  cpu_cores = NULL,
  memory_gb = NULL,
  enable_app_insights = NULL,
  scoring_timeout_ms = NULL,
  replica_max_concurrent_requests = NULL,
  max_request_wait_time = NULL,
  num_replicas = NULL,
  tags = NULL,
  properties = NULL,
  description = NULL,
  models = NULL,
  inference_config = NULL,
  gpu_cores = NULL,
  period_seconds = NULL,
  initial_delay_seconds = NULL,
  timeout_seconds = NULL,
  success_threshold = NULL,
  failure_threshold = NULL,
  namespace = NULL,
```

```
  token_auth_enabled = NULL
)
```

## Arguments

| | |
|---|---|
| webservice | The AksWebservice object. |
| autoscale_enabled | If TRUE enable autoscaling for the web service. |
| autoscale_min_replicas | An int of the minimum number of containers to use when autoscaling the web service. |
| autoscale_max_replicas | An int of the maximum number of containers to use when autoscaling the web service. |
| autoscale_refresh_seconds | An int of how often in seconds the autoscaler should attempt to scale the web service. |
| autoscale_target_utilization | An int of the target utilization (in percent out of 100) the autoscaler should attempt to maintain for the web service. |
| auth_enabled | If TRUE enable key-based authentication for the web service. Defaults to TRUE. |
| cpu_cores | The number of cpu cores to allocate for the web service. Can be a decimal. Defaults to 0.1. |
| memory_gb | The amount of memory (in GB) to allocate for the web service. Can be a decimal. Defaults to 0.5. |
| enable_app_insights | If TRUE enable AppInsights for the web service. Defaults to FALSE. |
| scoring_timeout_ms | An int of the timeout (in milliseconds) to enforce for scoring calls to the web service. |
| replica_max_concurrent_requests | An int of the number of maximum concurrent requests per node to allow for the web service. |
| max_request_wait_time | An int of the maximum amount of time a request will stay in the queue (in milliseconds) before returning a 503 error. |
| num_replicas | An int of the number of containers to allocate for the web service. If this parameter is not set then the autoscaler is enabled by default. |
| tags | A named list of key-value tags for the web service, e.g. list("key" = "value"). Will replace existing tags. |
| properties | A named list of key-value properties to add for the web service, e.g. list("key" = "value"). |
| description | A string of the description to give the web service. |
| models | A list of Model objects to package into the updated service. |
| inference_config | An InferenceConfig object. |

gpu_cores An int of the number of gpu cores to allocate for the web service.

period_seconds An int of how often in seconds to perform the liveness probe. Minimum value is 1.

initial_delay_seconds
An int of the number of seconds after the container has started before liveness probes are initiated.

timeout_seconds
An int of the number of seconds after which the liveness probe times out. Minimum value is 1.

success_threshold
An int of the minimum consecutive successes for the liveness probe to be considered successful after having failed. Minimum value is 1.

failure_threshold
An int of the number of times Kubernetes will try the liveness probe when a Pod starts and the probe fails, before giving up. Minimum value is 1.

namespace A string of the Kubernetes namespace in which to deploy the web service: up to 63 lowercase alphanumeric ('a'-'z', '0'-'9') and hyphen ('-') characters. The first last characters cannot be hyphens.

token_auth_enabled
If TRUE, enable token-based authentication for the web service. If enabled, users can access the web service by fetching an access token using their Azure Active Directory credentials. Both token_auth_enabled and auth_enabled cannot be set to TRUE.

## Value

None

---

# update_aml_compute *Update scale settings for an AmlCompute cluster*

---

## Description

Update the scale settings for an existing AmlCompute cluster.

## Usage

```
update_aml_compute(
  cluster,
  min_nodes = NULL,
  max_nodes = NULL,
  idle_seconds_before_scaledown = NULL
)
```

### Arguments

| | |
|---|---|
| cluster | The AmlCompute cluster. |
| min_nodes | An integer of the minimum number of nodes to use on the cluster. |
| max_nodes | An integer of the maximum number of nodes to use on the cluster. |
| idle_seconds_before_scaledown | An integer of the node idle time in seconds before scaling down the cluster. |

### Value

None

---

## update_local_webservice *Update a local web service*

---

### Description

Update a local web service with the provided properties. You can update the web service to use a new model, a new entry script, or new dependencies that can be specified in an inference configuration.

Values left as NULL will remain unchanged in the service.

### Usage

```
update_local_webservice(
  webservice,
  models = NULL,
  deployment_config = NULL,
  wait = FALSE,
  inference_config = NULL
)
```

### Arguments

| | |
|---|---|
| webservice | The LocalWebservice object. |
| models | A list of Model objects to package into the updated service. |
| deployment_config | A LocalWebserviceDeploymentConfiguration to apply to the web service. |
| wait | If TRUE, wait for the service's container to reach a healthy state. Defaults to FALSE. |
| inference_config | An InferenceConfig object. |

### Value

None

upload_files_to_datastore *Upload files to the Azure storage a datastore points to*

## Description

Upload the data from the local file system to the Azure storage that the datastore points to.

## Usage

```
upload_files_to_datastore(
  datastore,
  files,
  relative_root = NULL,
  target_path = NULL,
  overwrite = FALSE,
  show_progress = TRUE
)
```

## Arguments

| | |
|---|---|
| datastore | The AzureBlobDatastore or AzureFileDatastore object. |
| files | A character vector of the absolute path to files to upload. |
| relative_root | A string of the base path from which is used to determine the path of the files in the Azure storage. For example, if we upload /path/to/file.txt, and we define the base path to be /path, when file.txt is uploaded to the blob storage or file share, it will have the path of /to/file.txt. If target_path is also given, then it will be used as the prefix for the derived path from above. The base path must be a common path of all of the files, otherwise an exception will be thrown. |
| target_path | A string of the location in the blob container or file share to upload the data to. Defaults to NULL, in which case the data is uploaded to the root. |
| overwrite | If TRUE, overwrites any existing data at target_path. |
| show_progress | If TRUE, show progress of upload in the console. |

## Value

The DataReference object for the target path uploaded.

upload_files_to_run *Upload files to a run*

## Description

Upload files to the run record.

Note: Runs automatically capture files in the specified output directory, which defaults to "./outputs". Use `upload_files_to_run()` only when additional files need to be uploaded or an output directory is not specified.

## Usage

```
upload_files_to_run(names, paths, timeout_seconds = NULL, run = NULL)
```

## Arguments

| | |
|---|---|
| `names` | A character vector of the names of the files to upload. |
| `paths` | A character vector of relative local paths to the files to be upload. |
| `timeout_seconds` | An int of the timeout in seconds for uploading the files. |
| `run` | The Run object. |

## Value

None

## Examples

```
ws <- load_workspace_from_config()
exp <- experiment(ws, name = 'myexperiment')

# Start an interactive logging run
run <- start_logging_run(exp)

# Upload files to the run record
filename1 <- "important_file_1"
filename2 <- "important_file_2"
upload_files_to_run(names = c(filename1, filename2),
             paths = c("path/on/disk/file_1.txt", "other/path/on/disk/file_2.txt"))

# Download a file from the run record
download_file_from_run(filename1, "file_1.txt")
```

## See Also

`upload_folder_to_run()` `download_file_from_run()` `download_files_from_run()`

upload_folder_to_run *Upload a folder to a run*

## Description

Upload the specified folder to the given prefix name to the run record.

Note: Runs automatically capture files in the specified output directory, which defaults to "./outputs". Use `upload_folder_to_run()` only when additional files need to be uploaded or an output directory is not specified.

## Usage

```
upload_folder_to_run(name, path, run = NULL)
```

## Arguments

| | |
|---|---|
| name | A string of the name of the folder of files to upload. |
| path | A string of the relative local path to the folder to upload. |
| run | The Run object. |

## Value

None

## Examples

```
ws <- load_workspace_from_config()
exp <- experiment(ws, name = 'myexperiment')

# Start an interactive logging run
run <- start_logging_run(exp)

# Upload folder to the run record
upload_folder_to_run(name = "important_files",
                     path = "path/on/disk")

# Download a file from the run record
download_file_from_run("important_files/existing_file.txt", "local_file.txt")
```

## See Also

`upload_files_to_run()` `download_file_from_run()` `download_files_from_run()`

## upload_to_datastore *Upload a local directory to the Azure storage a datastore points to*

### Description

Upload a local directory to the Azure storage the datastore points to.

### Usage

```
upload_to_datastore(
  datastore,
  src_dir,
  target_path = NULL,
  overwrite = FALSE,
  show_progress = TRUE
)
```

### Arguments

| | |
|---|---|
| datastore | The AzureBlobDatastore or AzureFileDatastore object. |
| src_dir | A string of the local directory to upload. |
| target_path | A string of the location in the blob container or file share to upload the data to. Defaults to NULL, in which case the data is uploaded to the root. |
| overwrite | If TRUE, overwrites any existing data at target_path. |
| show_progress | If TRUE, show progress of upload in the console. |

### Value

The DataReference object for the target path uploaded.

## view_run_details *Initialize run details widget*

### Description

Initializes a ShinyApp in RStudio Viewer (or the default browser if Viewer is unavailable) showing details of the submitted run. If using RStudio, the plot will auto-update with information collected from the server. For more details about the run, click the web view link. The widget will stop running once the run has reached a terminal state: "Failed", "Completed", or "Canceled".

If you are running this method from an RMarkdown file, the run details table will show up in the code chunk output instead of the Viewer.

### Usage

```
view_run_details(run, auto_refresh = TRUE)
```

### Arguments

| | |
|---|---|
| run | Run object |
| auto_refresh | Boolean indicating whether or not widget should update run details automatically. The default is TRUE when using RStudio. |

---

## wait_for_deployment *Wait for a web service to finish deploying*

---

### Description

Automatically poll on the running web service deployment and wait for the web service to reach a terminal state. Will throw an exception if it reaches a non-successful terminal state.

Typically called after running `deploy_model()`.

### Usage

```
wait_for_deployment(webservice, show_output = FALSE)
```

### Arguments

| | |
|---|---|
| webservice | The `LocalWebservice`, `AciWebservice`, or `AksWebservice` object. |
| show_output | If `TRUE`, print more verbose output. Defaults to `FALSE`. |

### Value

None

### See Also

`deploy_model()`

---

## wait_for_model_package_creation *Wait for a model package to finish creating*

---

### Description

Wait for a model package creation to reach a terminal state.

### Usage

```
wait_for_model_package_creation(package, show_output = FALSE)
```

### Arguments

| | |
|---|---|
| package | The ModelPackage object. |
| show_output | If TRUE, print more verbose output. Defaults to FALSE. |

### Value

None

---

## wait_for_provisioning_completion
*Wait for a cluster to finish provisioning*

---

### Description

Wait for a cluster to finish provisioning. Typically invoked after a create_aml_compute() or create_aks_compute() call.

### Usage

```
wait_for_provisioning_completion(cluster, show_output = FALSE)
```

### Arguments

| | |
|---|---|
| cluster | The AmlCompute or AksCompute object. |
| show_output | If TRUE, more verbose output will be provided. |

### Value

None

### See Also

create_aml_compute(), create_aks_compute()

## wait_for_run_completion

*Wait for the completion of a run*

### Description

Wait for the run to reach a terminal state. Typically called after submitting an experiment run with `submit_experiment()`.

### Usage

```
wait_for_run_completion(run, show_output = TRUE)
```

### Arguments

| | |
|---|---|
| `run` | The Run object. |
| `show_output` | If `TRUE`, print verbose output to console. |

### Value

None

### See Also

`submit_experiment()`

## write_workspace_config

*Write out the workspace configuration details to a config file*

### Description

Write out the workspace ARM properties to a config file. Workspace ARM properties can be loaded later using `load_workspace_from_config()`. The method provides a simple way of reusing the same workspace across multiple files or projects. Users can save the workspace ARM properties using this function, and use `load_workspace_from_config()` to load the same workspace in different files or projects without retyping the workspace ARM properties.

### Usage

```
write_workspace_config(workspace, path = NULL, file_name = NULL)
```

## Arguments

| | |
|---|---|
| workspace | The Workspace object whose config has to be written down. |
| path | A string of the location to write the config.json file. The config file will be located in a directory called '.azureml'. The parameter defaults to the current working directory, so by default config.json will be located at '.azureml/'. |
| file_name | A string of the name to use for the config file. The parameter defaults to 'config.json'. |

## Value

None

## See Also

load_workspace_from_config()

# Index