Package: vetiver (via r-universe)

September 17, 2024

Title Version, Share, Deploy, and Monitor Models

Version 0.2.5.9000

Description The goal of 'vetiver' is to provide fluent tooling to version, share, deploy, and monitor a trained model. Functions handle both recording and checking the model's input data prototype, and predicting from a remote API endpoint. The 'vetiver' package is extensible, with generics that can support many kinds of models.

License MIT + file LICENSE

URL https://vetiver.rstudio.com, https://rstudio.github.io/vetiver-r/, https://github.com/rstudio/vetiver-r/

BugReports https://github.com/rstudio/vetiver-r/issues

Depends R (>= 3.6)

Imports bundle, butcher (>= 0.3.1), cereal, cli, fs, generics, glue, hardhat, lifecycle, magrittr (>= 2.0.3), pins (>= 1.3.0), purrr, rapidoc, readr (>= 1.4.0), rlang (>= 1.1.0), tibble, vctrs, withr

Suggests arrow, callr, caret, clustMixType, covr, curl, dplyr, flexdashboard, ggplot2, httpuv, httr, jsonlite, keras, knitr, LiblineaR, luz, mgcv, mlr3 (>= 0.17.0), mlr3data, mlr3learners, mockery, modeldata, parsnip, paws.machine.learning (>= 0.2.0), pingr, plotly, plumber (>= 1.0.0), ranger, recipes (>= 1.1.0), reticulate, rmarkdown, rpart, rsconnect, slider (>= 0.2.2), smdocker (>= 0.1.2), stacks, tensorflow, testthat (>= 3.1.8), tidyselect, torch, vdiffr, workflows, xgboost, yardstick

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Remotes rstudio/reticulate

Repository https://rstudio.r-universe.dev

RemoteUrl https://github.com/rstudio/vetiver-r

RemoteRef HEAD

RemoteSha ff0509c2c60b8d949944095e2d002df55b494220

Contents

api_spec *Update the OpenAPI specification using model metadata*

Description

Update the OpenAPI specification using model metadata

Usage

```
api_spec(spec, vetiver_model, path, all_docs = TRUE)

glue_spec_summary(prototype, return_type)

## Default S3 method:
glue_spec_summary(prototype, return_type = NULL)

## S3 method for class 'data.frame'
glue_spec_summary(prototype, return_type = "predictions")

## S3 method for class 'array'
glue_spec_summary(prototype, return_type = "predictions")
```

Arguments

spec	An OpenAPI Specification formatted list object
vetiver_model	A deployable `vetiver_model()` object
path	The endpoint path
all_docs	Should the interactive visual API documentation be created for *all* POST endpoints in the router `pr`? This defaults to `TRUE`, and assumes that all POST endpoints use the `vetiver_model()` input data prototype.
prototype	An input data prototype from a model
return_type	Character string to describe what endpoint returns, such as "predictions"

Value

`api_spec()` returns the updated OpenAPI Specification object. This function uses `glue_spec_summary()` internally, which returns a `glue` character string.

Examples

```
library(plumber)
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")

glue_spec_summary(v$prototype)
```

```
modify_spec <- function(spec) api_spec(spec, v, "/predict")
pr() %>% pr_set_api_spec(api = modify_spec)
```

`attach_pkgs` *Fully attach or load packages for making model predictions*

Description

These are developer-facing functions, useful for supporting new model types. Some models require one or more R packages to be fully attached to make predictions, and some require only that the namespace of one or more R packages is loaded.

Usage

```
attach_pkgs(pkgs)

load_pkgs(pkgs)
```

Arguments

`pkgs`	A character vector of package names to load or fully attach.

Details

These two functions will attempt either to:

- fully attach or
- load

the namespace of the `pkgs` vector of package names, preserving the current random seed.

To learn more about load vs. attach, read the "Dependencies" chapter of *R Packages*. For deploying a model, it is likely safer to fully attach needed packages but that comes with the risk of naming conflicts between packages.

Value

An invisible `TRUE`.

Examples

```
## succeed
load_pkgs(c("knitr", "readr"))
attach_pkgs(c("knitr", "readr"))

## fail
try(load_pkgs(c("bloopy", "readr")))
try(attach_pkgs(c("bloopy", "readr")))
```

`augment.vetiver_endpoint` *Post new data to a deployed model API endpoint and augment with predictions*

Description

Post new data to a deployed model API endpoint and augment with predictions

Usage

```
## S3 method for class 'vetiver_endpoint'
augment(x, new_data, ...)
```

Arguments

`x`	A model API endpoint object created with `vetiver_endpoint()`.
`new_data`	New data for making predictions, such as a data frame.
`...`	Extra arguments passed to `httr::POST()`

Value

The `new_data` with added prediction column(s).

See Also

`predict.vetiver_endpoint()`

Examples

```
if (FALSE) {
endpoint <- vetiver_endpoint("http://127.0.0.1:8088/predict")
augment(endpoint, mtcars[4:7, -1])
}
```

`augment.vetiver_endpoint_sagemaker` *Post new data to a deployed SageMaker model endpoint and augment with predictions*

Description

Post new data to a deployed SageMaker model endpoint and augment with predictions

Usage

```
## S3 method for class 'vetiver_endpoint_sagemaker'
augment(x, new_data, ...)
```

Arguments

x	A SageMaker model endpoint object created with vetiver_endpoint_sagemaker().
new_data	New data for making predictions, such as a data frame.
...	Extra arguments passed to paws.machine.learning::sagemakerruntime_invoke_endpoint()

Value

The new_data with added prediction column(s).

See Also

predict.vetiver_endpoint_sagemaker()

Examples

```
if (FALSE) {
  endpoint <- vetiver_endpoint_sagemaker("sagemaker-demo-model")
  augment(endpoint, mtcars[4:7, -1])
}
```

---

handler_startup.train *Model handler functions for API endpoint*

Description

These are developer-facing functions, useful for supporting new model types. Each model supported by vetiver_model() uses two handler functions in vetiver_api():

- The handler_startup function executes when the API starts. Use this function for tasks like loading packages. A model can use the default method here, which is NULL (to do nothing at startup).
- The handler_predict function executes at each API call. Use this function for calling predict() and any other tasks that must be executed at each API call.

Usage

```
## S3 method for class 'train'
handler_startup(vetiver_model)

## S3 method for class 'train'
handler_predict(vetiver_model, ...)

## S3 method for class 'gam'
handler_startup(vetiver_model)

## S3 method for class 'gam'
handler_predict(vetiver_model, ...)

## S3 method for class 'glm'
handler_predict(vetiver_model, ...)

handler_startup(vetiver_model)

## Default S3 method:
handler_startup(vetiver_model)

handler_predict(vetiver_model, ...)

## Default S3 method:
handler_predict(vetiver_model, ...)

## S3 method for class 'keras.engine.training.Model'
handler_startup(vetiver_model)

## S3 method for class 'keras.engine.training.Model'
handler_predict(vetiver_model, ...)

## S3 method for class 'kproto'
handler_predict(vetiver_model, ...)

## S3 method for class 'lm'
handler_predict(vetiver_model, ...)

## S3 method for class 'luz_module_fitted'
handler_startup(vetiver_model)

## S3 method for class 'luz_module_fitted'
handler_predict(vetiver_model, ...)

## S3 method for class 'Learner'
handler_startup(vetiver_model)

## S3 method for class 'Learner'
```

```
handler_predict(vetiver_model, ...)

## S3 method for class 'ranger'
handler_startup(vetiver_model)

## S3 method for class 'ranger'
handler_predict(vetiver_model, ...)

## S3 method for class 'recipe'
handler_startup(vetiver_model)

## S3 method for class 'recipe'
handler_predict(vetiver_model, ...)

## S3 method for class 'model_stack'
handler_startup(vetiver_model)

## S3 method for class 'model_stack'
handler_predict(vetiver_model, ...)

## S3 method for class 'workflow'
handler_startup(vetiver_model)

## S3 method for class 'workflow'
handler_predict(vetiver_model, ...)

## S3 method for class 'xgb.Booster'
handler_startup(vetiver_model)

## S3 method for class 'xgb.Booster'
handler_predict(vetiver_model, ...)
```

Arguments

`vetiver_model`	A deployable `vetiver_model()` object
`...`	Other arguments passed to `predict()`, such as prediction `type`

Details

These are two generics that use the class of `vetiver_model$model` for dispatch.

Value

A `handler_startup` function should return invisibly, while a `handler_predict` function should return a function with the signature `function(req)`. The request body (`req$body`) consists of the new data at prediction time; this function should return predictions either as a tibble or as a list coercable to a tibble via `tibble::as_tibble()`.

Examples

```
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
handler_startup(v)
handler_predict(v)
```

`map_request_body` *Identify data types for each column in an input data prototype*

Description

The OpenAPI specification of a Plumber API created via `plumber::pr()` can be modified via `plumber::pr_set_api_spec()`, and this helper function will identify data types of predictors and create a list to use in this specification. These are *not* R data types, but instead basic JSON data types. For example, factors in R will be documented as strings in the OpenAPI specification.

Usage

```
map_request_body(prototype)
```

Arguments

`prototype`	An input data prototype from a model

Details

This is a developer-facing function, useful for supporting new model types. It is called by `api_spec()`.

Value

A list to be used within `plumber::pr_set_api_spec()`

Examples

```
map_request_body(vctrs::vec_slice(chickwts, 0))
```

predict.vetiver_endpoint *Post new data to a deployed model API endpoint and return predictions*

Description

Post new data to a deployed model API endpoint and return predictions

Usage

```
## S3 method for class 'vetiver_endpoint'
predict(object, new_data, ...)
```

Arguments

object	A model API endpoint object created with vetiver_endpoint().
new_data	New data for making predictions, such as a data frame.
...	Extra arguments passed to httr::POST()

Value

A tibble of model predictions with as many rows as in new_data.

See Also

augment.vetiver_endpoint()

Examples

```
if (FALSE) {
endpoint <- vetiver_endpoint("http://127.0.0.1:8088/predict")
predict(endpoint, mtcars[4:7, -1])
}
```

predict.vetiver_endpoint_sagemaker *Post new data to a deployed SageMaker model endpoint and return predictions*

Description

Post new data to a deployed SageMaker model endpoint and return predictions

Usage

```
## S3 method for class 'vetiver_endpoint_sagemaker'
predict(object, new_data, ...)
```

Arguments

object	A SageMaker model endpoint object created with vetiver_endpoint_sagemaker().
new_data	New data for making predictions, such as a data frame.
...	Extra arguments passed to paws.machine.learning::sagemakerruntime_invoke_endpoint()

Value

A tibble of model predictions with as many rows as in new_data.

See Also

augment.vetiver_endpoint_sagemaker()

Examples

```
if (FALSE) {
  endpoint <- vetiver_endpoint_sagemaker("sagemaker-demo-model")
  predict(endpoint, mtcars[4:7, -1])
}
```

vetiver_api *Create a Plumber API to predict with a deployable* vetiver_model() *object*

Description

Use vetiver_api() to add a POST endpoint for predictions from a trained vetiver_model() to a Plumber router.

Usage

```
vetiver_api(
  pr,
  vetiver_model,
  path = "/predict",
  debug = is_interactive(),
  ...
)

vetiver_pr_post(
  pr,
  vetiver_model,
  path = "/predict",
  debug = is_interactive(),
  ...,
  check_prototype = TRUE,
  check_ptype = deprecated()
)

vetiver_pr_docs(pr, vetiver_model, path = "/predict", all_docs = TRUE)
```

Arguments

pr	A Plumber router, such as from plumber::pr().
vetiver_model	A deployable vetiver_model() object
path	The endpoint path
debug	TRUE provides more insight into your API errors.
...	Other arguments passed to predict(), such as prediction type
check_prototype	Should the input data prototype stored in vetiver_model (used for visual API documentation) also be used to check new data at prediction time? Defaults to TRUE.
check_ptype	**[Deprecated]**
all_docs	Should the interactive visual API documentation be created for *all* POST endpoints in the router pr? This defaults to TRUE, and assumes that all POST endpoints use the vetiver_model() input data prototype.

Details

You can first store and version your vetiver_model() with vetiver_pin_write(), and then create an API endpoint with vetiver_api().

Setting debug = TRUE may expose any sensitive data from your model in API errors.

Several GET endpoints will also be added to the router pr, depending on the characteristics of the model object:

- a /pin-url endpoint to return the URL of the pinned model

- a `/metadata` endpoint to return any metadata stored with the model
- a `/ping` endpoint for the API health
- a `/prototype` endpoint for the model's input data prototype (use `cereal::cereal_from_json()`) to convert this back to a vctrs ptype

The function `vetiver_api()` uses:

- `vetiver_pr_post()` for endpoint definition and
- `vetiver_pr_docs()` to create visual API documentation

These modular functions are available for more advanced use cases.

Value

A Plumber router with the prediction endpoint added.

Examples

```
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")

library(plumber)
pr() %>% vetiver_api(v)
## is the same as:
pr() %>% vetiver_pr_post(v) %>% vetiver_pr_docs(v)
## for either, next, pipe to `pr_run()`
```

`vetiver_compute_metrics` *Aggregate model metrics over time for monitoring*

Description

These three functions can be used for model monitoring (such as in a monitoring dashboard):

- `vetiver_compute_metrics()` computes metrics (such as accuracy for a classification model or RMSE for a regression model) at a chosen time aggregation `period`
- `vetiver_pin_metrics()` updates an existing pin storing model metrics over time
- `vetiver_plot_metrics()` creates a plot of metrics over time

Usage

```
vetiver_compute_metrics(
  data,
  date_var,
  period,
  truth,
  estimate,
  ...,
  metric_set = yardstick::metrics,
  every = 1L,
  origin = NULL,
  before = 0L,
  after = 0L,
  complete = FALSE
)
```

Arguments

data	A data.frame containing the columns specified by truth, estimate, and ....
date_var	The column in data containing dates or date-times for monitoring, to be aggregated with .period
period	[character(1)] A string defining the period to group by. Valid inputs can be roughly broken into: • "year", "quarter", "month", "week", "day" • "hour", "minute", "second", "millisecond" • "yweek", "mweek" • "yday", "mday"
truth	The column identifier for the true results (that is numeric or factor). This should be an unquoted column name although this argument is passed by expression and support quasiquotation (you can unquote column names).
estimate	The column identifier for the predicted results (that is also numeric or factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name.
...	A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.
metric_set	A yardstick::metric_set() function for computing metrics. Defaults to yardstick::metrics().
every	[positive integer(1)] The number of periods to group together. For example, if the period was set to "year" with an every value of 2, then the years 1970 and 1971 would be placed in the same group.

origin — [Date(1) / POSIXct(1) / POSIXlt(1) / NULL]

The reference date time value. The default when left as NULL is the epoch time of 1970-01-01 00:00:00, *in the time zone of the index*.

This is generally used to define the anchor time to count from, which is relevant when the every value is > 1.

before, after — [integer(1) / Inf]

The number of values before or after the current element to include in the sliding window. Set to Inf to select all elements before or after the current element. Negative values are allowed, which allows you to "look forward" from the current element if used as the .before value, or "look backwards" if used as .after.

complete — [logical(1)]

Should the function be evaluated on complete windows only? If FALSE, the default, then partial computations will be allowed.

Details

For arguments used more than once in your monitoring dashboard, such as date_var, consider using R Markdown parameters to reduce repetition and/or errors.

Value

A dataframe of metrics.

See Also

vetiver_pin_metrics(), vetiver_plot_metrics()

Examples

```
library(dplyr)
library(parsnip)
data(Chicago, package = "modeldata")
Chicago <- Chicago %>% select(ridership, date, all_of(stations))
training_data <- Chicago %>% filter(date < "2009-01-01")
testing_data <- Chicago %>% filter(date >= "2009-01-01", date < "2011-01-01")
monitoring <- Chicago %>% filter(date >= "2011-01-01", date < "2012-12-31")
lm_fit <- linear_reg() %>% fit(ridership ~ ., data = training_data)

library(pins)
b <- board_temp()

original_metrics <-
    augment(lm_fit, new_data = testing_data) %>%
    vetiver_compute_metrics(date, "week", ridership, .pred, every = 4L)

new_metrics <-
    augment(lm_fit, new_data = monitoring) %>%
    vetiver_compute_metrics(date, "week", ridership, .pred, every = 4L)
```

`vetiver_create_description.train` *Model constructor methods*

Description

These are developer-facing functions, useful for supporting new model types. Each model supported by `vetiver_model()` uses up to four methods when the deployable object is created:

- The `vetiver_create_description()` function generates a helpful description of the model based on its characteristics. This method is required.
- The `vetiver_create_meta()` function creates the correct `vetiver_meta()` for the model. This is especially helpful for specifying which packages are needed for prediction. A model can use the default method here, which is to have no special metadata.
- The `vetiver_ptype()` function finds an input data prototype from the training data (a zero-row slice) to use for checking at prediction time. This method is required.
- The `vetiver_prepare_model()` function executes last. Use this function for tasks like checking if the model is trained and reducing the size of the model via `butcher::butcher()`. A model can use the default method here, which is to return the model without changes.

Usage

```
## S3 method for class 'train'
vetiver_create_description(model)

## S3 method for class 'train'
vetiver_prepare_model(model)

## S3 method for class 'gam'
vetiver_create_description(model)

## S3 method for class 'gam'
vetiver_prepare_model(model)

## S3 method for class 'glm'
vetiver_create_description(model)

## S3 method for class 'glm'
vetiver_prepare_model(model)

## S3 method for class 'keras.engine.training.Model'
vetiver_create_description(model)

## S3 method for class 'keras.engine.training.Model'
vetiver_prepare_model(model)
```

```
## S3 method for class 'kproto'
vetiver_create_description(model)

## S3 method for class 'kproto'
vetiver_prepare_model(model)

## S3 method for class 'lm'
vetiver_create_description(model)

## S3 method for class 'lm'
vetiver_prepare_model(model)

## S3 method for class 'luz_module_fitted'
vetiver_create_description(model)

## S3 method for class 'luz_module_fitted'
vetiver_prepare_model(model)

## S3 method for class 'Learner'
vetiver_create_description(model)

## S3 method for class 'Learner'
vetiver_prepare_model(model)

vetiver_create_description(model)

## Default S3 method:
vetiver_create_description(model)

vetiver_prepare_model(model)

## Default S3 method:
vetiver_prepare_model(model)

## S3 method for class 'ranger'
vetiver_create_description(model)

## S3 method for class 'ranger'
vetiver_prepare_model(model)

## S3 method for class 'recipe'
vetiver_create_description(model)

## S3 method for class 'recipe'
vetiver_prepare_model(model)

## S3 method for class 'model_stack'
vetiver_create_description(model)
```

```
## S3 method for class 'model_stack'
vetiver_prepare_model(model)

## S3 method for class 'workflow'
vetiver_create_description(model)

## S3 method for class 'workflow'
vetiver_prepare_model(model)

## S3 method for class 'xgb.Booster'
vetiver_create_description(model)

## S3 method for class 'xgb.Booster'
vetiver_prepare_model(model)
```

Arguments

model	A trained model, such as an `lm()` model or a tidymodels `workflows::workflow()`.

Details

These are four generics that use the class of `model` for dispatch.

Examples

```
cars_lm <- lm(mpg ~ ., data = mtcars)
vetiver_create_description(cars_lm)
vetiver_prepare_model(cars_lm)
```

---

vetiver_create_meta.train

Metadata constructors for `vetiver_model()` *object*

Description

These are developer-facing functions, useful for supporting new model types. The metadata stored in a `vetiver_model()` object has four elements:

- `$user`, the metadata supplied by the user
- `$version`, the version of the pin (which can be `NULL` before pinning)
- `$url`, the URL where the pin is located, if any
- `$required_pkgs`, a character string of R packages required for prediction

Usage

```
## S3 method for class 'train'
vetiver_create_meta(model, metadata)

## S3 method for class 'gam'
vetiver_create_meta(model, metadata)

## S3 method for class 'keras.engine.training.Model'
vetiver_create_meta(model, metadata)

## S3 method for class 'kproto'
vetiver_create_meta(model, metadata)

## S3 method for class 'luz_module_fitted'
vetiver_create_meta(model, metadata)

vetiver_meta(user = list(), version = NULL, url = NULL, required_pkgs = NULL)

vetiver_create_meta(model, metadata)

## Default S3 method:
vetiver_create_meta(model, metadata)

## S3 method for class 'Learner'
vetiver_create_meta(model, metadata)

## S3 method for class 'ranger'
vetiver_create_meta(model, metadata)

## S3 method for class 'recipe'
vetiver_create_meta(model, metadata)

## S3 method for class 'model_stack'
vetiver_create_meta(model, metadata)

## S3 method for class 'workflow'
vetiver_create_meta(model, metadata)

## S3 method for class 'xgb.Booster'
vetiver_create_meta(model, metadata)
```

Arguments

model	A trained model, such as an `lm()` model or a tidymodels `workflows::workflow()`.
metadata	A list containing additional metadata to store with the pin. When retrieving the pin, this will be stored in the `user` key, to avoid potential clashes with the metadata that pins itself uses.
user	Metadata supplied by the user

version	Version of the pin
url	URL for the pin, if any
required_pkgs	Character string of R packages required for prediction

Value

The vetiver_meta() constructor returns a list. The vetiver_create_meta function returns a vetiver_meta() list.

Examples

```
vetiver_meta()

cars_lm <- lm(mpg ~ ., data = mtcars)
vetiver_create_meta(cars_lm, list())
```

---

vetiver_create_rsconnect_bundle

Create an Posit Connect bundle for a vetiver model API

Description

Use vetiver_create_rsconnect_bundle() to create a Posit Connect model API bundle for a vetiver_model() that has been versioned and stored via vetiver_pin_write().

Usage

```
vetiver_create_rsconnect_bundle(
  board,
  name,
  version = NULL,
  predict_args = list(),
  filename = fs::file_temp(pattern = "bundle", ext = ".tar.gz"),
  additional_pkgs = character(0)
)
```

Arguments

board	A pin board, created by board_folder(), board_connect(), board_url() or another board_ function.
name	Pin name.
version	Retrieve a specific version of a pin. Use pin_versions() to find out which versions are available and when they were created.
predict_args	A list of optional arguments passed to vetiver_api() such as the prediction type.

`filename` The path for the model API bundle to be created (can be used as the argument to `connectapi::bundle_path()`)

`additional_pkgs` Any additional R packages that need to be **attached** via `library()` to run your API, as a character vector.

Details

This function creates a deployable bundle. See Posit Connect docs for how to deploy this bundle, as well as the connectapi R package for how to integrate with Connect's API from R.

The two functions `vetiver_create_rsconnect_bundle()` and `vetiver_deploy_rsconnect()` are alternatives to each other, providing different strategies for deploying a vetiver model API to Posit Connect.

Value

The location of the model API bundle `filename`, invisibly.

See Also

`vetiver_write_plumber()`, `vetiver_deploy_rsconnect()`

Examples

```
library(pins)
b <- board_temp(versioned = TRUE)
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
vetiver_pin_write(b, v)

## when you pin to Posit Connect, your pin name will be typically be like:
## "user.name/cars_linear"
vetiver_create_rsconnect_bundle(
    b,
    "cars_linear",
    predict_args = list(debug = TRUE)
)
```

---

vetiver_dashboard *R Markdown format for model monitoring dashboards*

Description

R Markdown format for model monitoring dashboards

Usage

```
vetiver_dashboard(pins = list(), display_pins = TRUE, ...)

get_vetiver_dashboard_pins()

pin_example_kc_housing_model(board = pins::board_local(), name = "seattle_rf")
```

Arguments

pins	A list containing board, name, and version, as in pins::pin_read()
display_pins	Should the dashboard display a link to the pin(s)? Defaults to TRUE, but only creates a link if the pin contains a URL in its metadata.
...	Arguments passed to flexdashboard::flex_dashboard()
board	A pin board, created by board_folder(), board_connect(), board_url() or another board_ function.
name	Pin name.

Details

The vetiver_dashboard() function is a specialized type of **flexdashboard**. See the flexdashboard website for additional documentation: https://pkgs.rstudio.com/flexdashboard/

Before knitting the example vetiver_dashboard() template, execute the helper function pin_example_kc_housing_model to set up demonstration model and metrics pins needed for the monitoring demo. This function will:

- fit an example model to training data
- pin the vetiver model to your own pins::board_local()
- compute metrics from testing data
- pin these metrics to the same local board

These are the steps you need to complete before setting up monitoring your real model.

---

vetiver_deploy_rsconnect

Deploy a vetiver model API to Posit Connect

Description

Use vetiver_deploy_rsconnect() to deploy a vetiver_model() that has been versioned and stored via vetiver_pin_write() as a Plumber API on Posit Connect.

Usage

```
vetiver_deploy_rsconnect(
  board,
  name,
  version = NULL,
  predict_args = list(),
  appTitle = glue::glue("{name} model API"),
  ...,
  additional_pkgs = character(0)
)
```

Arguments

board	A pin board, created by board_folder(), board_connect(), board_url() or another board_ function.
name	Pin name.
version	Retrieve a specific version of a pin. Use pin_versions() to find out which versions are available and when they were created.
predict_args	A list of optional arguments passed to vetiver_api() such as the prediction type.
appTitle	The API title on Posit Connect. Use the default based on name, or pass in your own title.
...	Other arguments passed to rsconnect::deployApp() such as appName, account, or launch.browser.
additional_pkgs	Any additional R packages that need to be **attached** via library() to run your API, as a character vector.

Details

The two functions vetiver_deploy_rsconnect() and vetiver_create_rsconnect_bundle() are alternatives to each other, providing different strategies for deploying a vetiver model API to Posit Connect.

When you first deploy to Connect, your API will only be accessible to you. You can change the access settings so others can also access the API. For all access settings other than "Anyone - no login required", anyone querying your API (including you) will need to pass authentication details with your API call, as shown in the Connect documentation.

Value

The deployment success (TRUE or FALSE), invisibly.

See Also

vetiver_write_plumber(), vetiver_create_rsconnect_bundle()

Examples

```
library(pins)
b <- board_temp(versioned = TRUE)
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
vetiver_pin_write(b, v)

if (FALSE) {
## pass args for predicting:
vetiver_deploy_rsconnect(
    b,
    "user.name/cars_linear",
    predict_args = list(debug = TRUE)
)

## specify an account name through `...`:
vetiver_deploy_rsconnect(
    b,
    "user.name/cars_linear",
    account = "user.name"
)
}
```

`vetiver_deploy_sagemaker` *Deploy a vetiver model API to Amazon SageMaker*

Description

Use `vetiver_deploy_sagemaker()` to deploy a `vetiver_model()` that has been versioned and stored via `vetiver_pin_write()` as a Plumber API on Amazon SageMaker.

Usage

```
vetiver_deploy_sagemaker(
  board,
  name,
  instance_type,
  ...,
  predict_args = list(),
  docker_args = list(),
  build_args = list(),
  endpoint_args = list(),
  repo_name = glue("vetiver-sagemaker-{name}")
)
```

Arguments

board	An AWS S3 board created with pins::board_s3(). This board must be in the correct region for your SageMaker instance.
name	Pin name.
instance_type	Type of EC2 instance to use; see Amazon SageMaker pricing.
...	Not currently used.
predict_args	A list of optional arguments passed to vetiver_api() such as the prediction type.
docker_args	A list of optional arguments passed to vetiver_write_docker() such as the lockfile name or whether to use rspm. Do not pass additional_pkgs here, as this function uses additional_pkgs = required_pkgs(board).
build_args	A list of optional arguments passed to vetiver_sm_build() such as the model version or the compute_type.
endpoint_args	A list of optional arguments passed to vetiver_sm_endpoint() such as accelerator_type or data_capture_config.
repo_name	The name for the AWS ECR repository to store the model.

Details

This function stores your model deployment image in the same bucket used by board.

The function vetiver_deploy_sagemaker() uses:

- vetiver_sm_build() to build and push a Docker image to ECR,
- vetiver_sm_model() to create a SageMaker model, and
- vetiver_sm_endpoint() to deploy a SageMaker model endpoint.

These modular functions are available for more advanced use cases.

If you are working locally, you will likely need to explicitly set up your execution role to work correctly. Check out "Execution role requirements" in the smdocker documentation, and especially note that the bucket containing your vetiver model needs to be added as a resource in your IAM role policy.

Value

The deployed vetiver_endpoint_sagemaker().

See Also

vetiver_sm_build(), vetiver_sm_model(), vetiver_sm_endpoint()

Examples

```
if (FALSE) {
library(pins)
b <- board_s3(bucket = "my-existing-bucket")
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
vetiver_pin_write(b, v)

endpoint <- vetiver_deploy_sagemaker(
    board = b,
    name = "cars_linear",
    instance_type = "ml.t2.medium",
    predict_args = list(type = "class", debug = TRUE)
)
}
```

`vetiver_endpoint` *Create a model API endpoint object for prediction*

Description

This function creates a model API endpoint for prediction from a URL. No HTTP calls are made until you actually `predict()` with your endpoint.

Usage

```
vetiver_endpoint(url)
```

Arguments

`url`	An API endpoint URL

Value

A new `vetiver_endpoint` object

Examples

```
vetiver_endpoint("https://colorado.rstudio.com/rsc/seattle-housing/predict")
```

vetiver_endpoint_sagemaker *Create a SageMaker model API endpoint object for prediction*

Description

This function creates a model API endpoint for prediction from a Sagemaker Model. No HTTP calls are made until you actually `predict()` with your endpoint.

Usage

```
vetiver_endpoint_sagemaker(model_endpoint)
```

Arguments

`model_endpoint`	The name of the Amazon SageMaker model endpoint.

Value

A new `vetiver_endpoint_sagemaker` object

Examples

```
vetiver_endpoint_sagemaker("vetiver-sagemaker-demo-model")
```

vetiver_model *Create a vetiver object for deployment of a trained model*

Description

A `vetiver_model()` object collects the information needed to store, version, and deploy a trained model. Once your `vetiver_model()` object has been created, you can:

- store and version it as a pin with `vetiver_pin_write()`
- create an API endpoint for it with `vetiver_api()`

Usage

```
vetiver_model(
  model,
  model_name,
  ...,
  description = NULL,
  metadata = list(),
  save_prototype = TRUE,
```

```
  save_ptype = deprecated(),
  versioned = NULL
)

new_vetiver_model(
  model,
  model_name,
  description,
  metadata,
  prototype,
  versioned
)
```

Arguments

model	A trained model, such as an `lm()` model or a tidymodels `workflows::workflow()`.
model_name	Model name or ID.
...	Other method-specific arguments passed to `vetiver_ptype()` to compute an input data prototype, such as `prototype_data` (a sample of training features).
description	A detailed description of the model. If omitted, a brief description of the model will be generated.
metadata	A list containing additional metadata to store with the pin. When retrieving the pin, this will be stored in the `user` key, to avoid potential clashes with the metadata that pins itself uses.
save_prototype	Should an input data prototype be stored with the model? The options are `TRUE` (the default, which stores a zero-row slice of the training data), `FALSE` (no input data prototype for visual documentation or checking), or a dataframe to be used for both checking at prediction time *and* examples in API visual documentation.
save_ptype	**[Deprecated]**
versioned	Should the model object be versioned when stored with `vetiver_pin_write()`? The default, `NULL`, will use the default for the `board` where you store the model.
prototype	An input data prototype. If `NULL`, there is no checking of new data at prediction time.

Details

You can provide your own data to `save_prototype` to use as examples in the visual documentation created by `vetiver_api()`. If you do this, consider checking that your input data prototype has the same structure as your training data (perhaps with `hardhat::scream()`) and/or simulating data to avoid leaking PII via your deployed model.

Some models, like `ranger::ranger()`, keras, and luz (torch), *require* that you pass in example training data as `prototype_data` or else explicitly set `save_prototype = FALSE`. For non-rectangular data input to models, such as image input for a keras or torch model, we currently recommend that you turn off prototype checking via `save_prototype = FALSE`.

Value

A new `vetiver_model` object.

Examples

```
cars_lm <- lm(mpg ~ ., data = mtcars)
vetiver_model(cars_lm, "cars-linear")
```

`vetiver_pin_metrics` *Update model metrics over time for monitoring*

Description

These three functions can be used for model monitoring (such as in a monitoring dashboard):

- `vetiver_compute_metrics()` computes metrics (such as accuracy for a classification model or RMSE for a regression model) at a chosen time aggregation `period`
- `vetiver_pin_metrics()` updates an existing pin storing model metrics over time
- `vetiver_plot_metrics()` creates a plot of metrics over time

Usage

```
vetiver_pin_metrics(
  board,
  df_metrics,
  metrics_pin_name,
  .index = .index,
  overwrite = FALSE,
  type = NULL,
  ...
)
```

Arguments

`board`	A pin board, created by `board_folder()`, `board_connect()`, `board_url()` or another `board_` function.
`df_metrics`	A tidy dataframe of metrics over time, such as created by `vetiver_compute_metrics()`.
`metrics_pin_name`	Pin name for where the *metrics* are stored (as opposed to where the model object is stored with `vetiver_pin_write()`).
`.index`	The variable in `df_metrics` containing the aggregated dates or date-times (from `time_var` in `data`). Defaults to `.index`.
`overwrite`	If `FALSE` (the default), error when the new metrics contain overlapping dates with the existing pin.If `TRUE`, overwrite any metrics for dates that exist both in the existing pin and new metrics with the *new* values.

type	File type used to save metrics to disk. With the default NULL, uses the type of the existing pin. Options are "rds" and "arrow".
...	Additional arguments passed on to methods for a specific board.

Details

Sometimes when you monitor a model at a given time aggregation, you may end up with dates in your new metrics (like new_metrics in the example) that are the same as dates in your existing aggregated metrics (like original_metrics in the example). This can happen if you need to re-run a monitoring report because something failed. With overwrite = FALSE (the default), vetiver_pin_metrics() will error when there are overlapping dates. With overwrite = TRUE, vetiver_pin_metrics() will replace such metrics with the new values. You probably want FALSE for interactive use and TRUE for dashboards or reports that run on a schedule.

You can initially create your pin with type = "arrow" or the default (type = "rds"). vetiver_pin_metrics() will update the pin using the same type by default.

Value

A dataframe of metrics.

See Also

vetiver_compute_metrics(), vetiver_plot_metrics()

Examples

```
library(dplyr)
library(parsnip)
data(Chicago, package = "modeldata")
Chicago <- Chicago %>% select(ridership, date, all_of(stations))
training_data <- Chicago %>% filter(date < "2009-01-01")
testing_data <- Chicago %>% filter(date >= "2009-01-01", date < "2011-01-01")
monitoring <- Chicago %>% filter(date >= "2011-01-01", date < "2012-12-31")
lm_fit <- linear_reg() %>% fit(ridership ~ ., data = training_data)

library(pins)
b <- board_temp()

## before starting monitoring, initiate the metrics and pin
## (for example, with the testing data):
original_metrics <-
    augment(lm_fit, new_data = testing_data) %>%
    vetiver_compute_metrics(date, "week", ridership, .pred, every = 4L)
pin_write(b, original_metrics, "lm_fit_metrics", type = "arrow")

## to continue monitoring with new data, compute metrics and update pin:
new_metrics <-
    augment(lm_fit, new_data = monitoring) %>%
    vetiver_compute_metrics(date, "week", ridership, .pred, every = 4L)
vetiver_pin_metrics(b, new_metrics, "lm_fit_metrics")
```

vetiver_pin_write *Read and write a trained model to a board of models*

Description

Use vetiver_pin_write() to pin a trained model to a board of models, along with an input prototype for new data and other model metadata. Use vetiver_pin_read() to retrieve that pinned object.

Usage

```
vetiver_pin_write(board, vetiver_model, ..., check_renv = FALSE)

vetiver_pin_read(board, name, version = NULL, check_renv = FALSE)
```

Arguments

board	A pin board, created by board_folder(), board_connect(), board_url() or another board_ function.
vetiver_model	A deployable vetiver_model() object
...	Additional arguments passed on to methods for a specific board.
check_renv	Use renv to record the packages used at training time with vetiver_pin_write() and check for differences with vetiver_pin_read(). Defaults to FALSE.
name	Pin name.
version	Retrieve a specific version of a pin. Use pin_versions() to find out which versions are available and when they were created.

Details

These functions read and write a vetiver_model() pin on the specified board containing the model object itself and other elements needed for prediction, such as the model's input data prototype or which packages are needed at prediction time. You may use pins::pin_read() or pins::pin_meta() to handle the pin, but vetiver_pin_read() returns a vetiver_model() object ready for deployment.

Value

vetiver_pin_read() returns a vetiver_model(); vetiver_pin_write() returns the name of the new pin, invisibly.

Examples

```
library(pins)
model_board <- board_temp()

cars_lm <- lm(mpg ~ ., data = mtcars)
```

```
v <- vetiver_model(cars_lm, "cars_linear")
vetiver_pin_write(model_board, v)
model_board

vetiver_pin_read(model_board, "cars_linear")

# can use `version` argument to read a specific version:
pin_versions(model_board, "cars_linear")

# can store an renv lockfile as part of the pin:
vetiver_pin_write(model_board, v, check_renv = TRUE)
```

`vetiver_plot_metrics` *Plot model metrics over time for monitoring*

Description

These three functions can be used for model monitoring (such as in a monitoring dashboard):

- `vetiver_compute_metrics()` computes metrics (such as accuracy for a classification model or RMSE for a regression model) at a chosen time aggregation `period`
- `vetiver_pin_metrics()` updates an existing pin storing model metrics over time
- `vetiver_plot_metrics()` creates a plot of metrics over time

Usage

```
vetiver_plot_metrics(
  df_metrics,
  .index = .index,
  .estimate = .estimate,
  .metric = .metric,
  .n = .n
)
```

Arguments

`df_metrics`	A tidy dataframe of metrics over time, such as created by `vetiver_compute_metrics()`.
`.index`	The variable in `df_metrics` containing the aggregated dates or date-times (from `time_var` in `data`). Defaults to `.index`.
`.estimate`	The variable in `df_metrics` containing the metric estimate. Defaults to `.estimate`.
`.metric`	The variable in `df_metrics` containing the metric type. Defaults to `.metric`.
`.n`	The variable in `df_metrics` containing the number of observations used for estimating the metric.

Value

A `ggplot2` object.

See Also

vetiver_compute_metrics(), vetiver_pin_metrics()

Examples

```
library(dplyr)
library(parsnip)
data(Chicago, package = "modeldata")
Chicago <- Chicago %>% select(ridership, date, all_of(stations))
training_data <- Chicago %>% filter(date < "2009-01-01")
testing_data <- Chicago %>% filter(date >= "2009-01-01", date < "2011-01-01")
monitoring <- Chicago %>% filter(date >= "2011-01-01", date < "2012-12-31")
lm_fit <- linear_reg() %>% fit(ridership ~ ., data = training_data)

library(pins)
b <- board_temp()

## before starting monitoring, initiate the metrics and pin
## (for example, with the testing data):
original_metrics <-
    augment(lm_fit, new_data = testing_data) %>%
    vetiver_compute_metrics(date, "week", ridership, .pred, every = 4L)
pin_write(b, original_metrics, "lm_fit_metrics", type = "arrow")

## to continue monitoring with new data, compute metrics and update pin:
new_metrics <-
    augment(lm_fit, new_data = monitoring) %>%
    vetiver_compute_metrics(date, "week", ridership, .pred, every = 4L)
vetiver_pin_metrics(b, new_metrics, "lm_fit_metrics")

library(ggplot2)
vetiver_plot_metrics(new_metrics) +
    scale_size(range = c(2, 4))
```

---

vetiver_prepare_docker

Generate files necessary to build a Docker container for a vetiver model

---

Description

Deploying a vetiver model via Docker requires several files. Use this function to create these needed files in the directory located at `path`.

Usage

```
vetiver_prepare_docker(
  board,
```

```
  name,
  version = NULL,
  path = ".",
  predict_args = list(),
  docker_args = list()
)
```

Arguments

board	A pin board, created by board_folder(), board_connect(), board_url() or another board_ function.
name	Pin name.
version	Retrieve a specific version of a pin. Use pin_versions() to find out which versions are available and when they were created.
path	A path to write the Plumber file, Dockerfile, and lockfile, capturing the model's dependencies.
predict_args	A list of optional arguments passed to vetiver_api() such as the prediction type.
docker_args	A list of optional arguments passed to vetiver_write_docker() such as the lockfile name or whether to use rspm. Do not pass additional_pkgs here, as this function uses additional_pkgs = required_pkgs(board).

Details

The function vetiver_prepare_docker() uses:

- vetiver_write_plumber() to create a Plumber file and
- vetiver_write_docker() to create a Dockerfile and renv lockfile

These modular functions are available for more advanced use cases. For models such as keras and torch, you will need to edit the generated Dockerfile to, for example, COPY requirements.txt requirements.txt or similar.

Value

An invisible TRUE.

Examples

```
library(pins)
b <- board_temp(versioned = TRUE)
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
vetiver_pin_write(b, v)

vetiver_prepare_docker(b, "cars_linear", path = tempdir())
```

vetiver_ptype.train *Create a vetiver input data prototype*

Description

Optionally find and return an input data prototype for a model.

Usage

```
## S3 method for class 'train'
vetiver_ptype(model, ...)

## S3 method for class 'gam'
vetiver_ptype(model, ...)

## S3 method for class 'glm'
vetiver_ptype(model, ...)

## S3 method for class 'keras.engine.training.Model'
vetiver_ptype(model, ...)

## S3 method for class 'kproto'
vetiver_ptype(model, ...)

## S3 method for class 'lm'
vetiver_ptype(model, ...)

## S3 method for class 'luz_module_fitted'
vetiver_ptype(model, ...)

## S3 method for class 'Learner'
vetiver_ptype(model, ...)

vetiver_ptype(model, ...)

## Default S3 method:
vetiver_ptype(model, ...)

vetiver_create_ptype(model, save_prototype, ...)

## S3 method for class 'ranger'
vetiver_ptype(model, ...)

## S3 method for class 'recipe'
vetiver_ptype(model, ...)

## S3 method for class 'model_stack'
```

```
vetiver_ptype(model, ...)

## S3 method for class 'workflow'
vetiver_ptype(model, ...)

## S3 method for class 'xgb.Booster'
vetiver_ptype(model, ...)
```

Arguments

model	A trained model, such as an lm() model or a tidymodels workflows::workflow().
...	Other method-specific arguments passed to vetiver_ptype() to compute an input data prototype, such as prototype_data (a sample of training features).
save_prototype	Should an input data prototype be stored with the model? The options are TRUE (the default, which stores a zero-row slice of the training data), FALSE (no input data prototype for visual documentation or checking), or a dataframe to be used for both checking at prediction time *and* examples in API visual documentation.

Details

These are developer-facing functions, useful for supporting new model types. A vetiver_model() object optionally stores an input data prototype for checking at prediction time.

- The default for save_prototype, TRUE, finds an input data prototype (a zero-row slice of the training data) via vetiver_ptype().
- save_prototype = FALSE opts out of storing any input data prototype.
- You may pass your own data to save_prototype, but be sure to check that it has the same structure as your training data, perhaps with hardhat::scream().

Value

A vetiver_ptype method returns a zero-row dataframe, and vetiver_create_ptype() returns either such a zero-row dataframe, NULL, or the dataframe passed to save_prototype.

Examples

```
cars_lm <- lm(mpg ~ cyl + disp, data = mtcars)

vetiver_create_ptype(cars_lm, TRUE)

## calls the right method for `model` via:
vetiver_ptype(cars_lm)

## can also turn off prototype
vetiver_create_ptype(cars_lm, FALSE)

## some models require that you pass in training features
cars_rf <- ranger::ranger(mpg ~ ., data = mtcars)
vetiver_ptype(cars_rf, prototype_data = mtcars[,-1])
```

`vetiver_sm_build` *Deploy a vetiver model API to Amazon SageMaker with modular functions*

Description

Use the function `vetiver_deploy_sagemaker()` for basic deployment on SageMaker, or these three functions together for more advanced use cases:

- `vetiver_sm_build()` generates and builds a Docker image on SageMaker for a vetiver model
- `vetiver_sm_model()` creates an Amazon SageMaker model
- `vetiver_sm_endpoint()` deploys an Amazon SageMaker model endpoint

Usage

```
vetiver_sm_build(
  board,
  name,
  version = NULL,
  path = fs::dir_create(tempdir(), "vetiver"),
  predict_args = list(),
  docker_args = list(),
  repository = NULL,
  compute_type = c("BUILD_GENERAL1_SMALL", "BUILD_GENERAL1_MEDIUM",
    "BUILD_GENERAL1_LARGE", "BUILD_GENERAL1_2XLARGE"),
  role = NULL,
  bucket = NULL,
  vpc_id = NULL,
  subnet_ids = list(),
  security_group_ids = list(),
  log = TRUE,
  ...
)

vetiver_sm_model(
  image_uri,
  model_name,
  role = NULL,
  vpc_config = list(),
  enable_network_isolation = FALSE,
  tags = list()
)

vetiver_sm_endpoint(
  model_name,
  instance_type,
  endpoint_name = NULL,
```

```
  initial_instance_count = 1,
  accelerator_type = NULL,
  tags = list(),
  kms_key = NULL,
  data_capture_config = list(),
  volume_size = NULL,
  model_data_download_timeout = NULL,
  wait = TRUE
)
```

Arguments

board	An AWS S3 board created with `pins::board_s3()`. This board must be in the correct region for your SageMaker instance.
name	Pin name.
version	Retrieve a specific version of a pin. Use `pin_versions()` to find out which versions are available and when they were created.
path	A path to write the Plumber file, Dockerfile, and lockfile, capturing the model's dependencies.
predict_args	A list of optional arguments passed to `vetiver_api()` such as the prediction `type`.
docker_args	A list of optional arguments passed to `vetiver_write_docker()` such as the `lockfile` name or whether to use `rspm`. Do not pass `additional_pkgs` here, as this function uses `additional_pkgs = required_pkgs(board)`.
repository	The ECR repository and tag for the image as a character. Defaults to `sagemaker-studio-${domain_id}:`
compute_type	The CodeBuild compute type as a character. Defaults to `BUILD_GENERAL1_SMALL`.
role	The ARN IAM role name (as a character) to be used with: • CodeBuild for `vetiver_sm_build()` • the SageMaker model for `vetiver_sm_model()` Defaults to the SageMaker Studio execution role.
bucket	The S3 bucket to use for sending data to CodeBuild as a character. Defaults to the SageMaker SDK default bucket.
vpc_id	ID of the VPC that will host the CodeBuild project such as `"vpc-05c09f91d48831c8c"`.
subnet_ids	List of subnet IDs for the CodeBuild project, such as `list("subnet-0b31f1863e9d31a67")`.
security_group_ids	List of security group IDs for the CodeBuild project, such as `list("sg-0ce4ec0d0414d2ddc")`.
log	A logical to show the logs of the running CodeBuild build. Defaults to `TRUE`.
...	Docker build parameters (Use "_" instead of "-"; for example, Docker optional parameter `build-arg` becomes `build_arg`).
image_uri	The AWS ECR image URI for the Amazon SageMaker Model to be created (for example, as returned by `vetiver_sm_build()`).
model_name	The Amazon SageMaker model name to be deployed.

vpc_config
: A list containing the VPC configuration for the Amazon SageMaker model API VpcConfig (optional).
  - Subnets: List of subnet ids
  - SecurityGroupIds: List of security group ids

enable_network_isolation
: A logical to specify whether the container will run in network isolation mode. Defaults to FALSE.

tags
: A named list of tags for labeling the Amazon SageMaker model or model endpoint to be created.

instance_type
: Type of EC2 instance to use; see Amazon SageMaker pricing.

endpoint_name
: The name to use for the Amazon SageMaker model endpoint to be created, if to be different from model_name.

initial_instance_count
: The initial number of instances to run in the endpoint.

accelerator_type
: Type of Elastic Inference accelerator to attach to an endpoint for model loading and inference, for example, "ml.eia1.medium".

kms_key
: The ARN of the KMS key used to encrypt the data on the storage volume attached to the instance hosting the endpoint.

data_capture_config
: A list for configuration to control how Amazon SageMaker captures inference data.

volume_size
: The size, in GB, of the ML storage volume attached to the individual inference instance associated with the production variant. Currently only Amazon EBS gp2 storage volumes are supported.

model_data_download_timeout
: The timeout value, in seconds, to download and extract model data from Amazon S3.

wait
: A logical for whether to wait for the endpoint to be deployed. Defaults to TRUE.

Details

The function vetiver_sm_build() generates the files necessary to build a Docker container to deploy a vetiver model in SageMaker and then builds the image on AWS CodeBuild. The resulting image is stored in AWS ECR. This function creates a Plumber file and Dockerfile appropriate for SageMaker, for example, with path = "/invocations" and port = 8080.

If you run into problems with Docker rate limits, then either

- authenticate to Docker from SageMaker, or
- use a public ECR base image, passed through docker_args

Value

vetiver_sm_build() returns the AWS ECR image URI and vetiver_sm_model() returns the model name (both as characters). vetiver_sm_endpoint() returns a new vetiver_endpoint_sagemaker() object.

See Also

vetiver_prepare_docker(), vetiver_deploy_sagemaker(), vetiver_endpoint_sagemaker()

Examples

```
if (FALSE) {
library(pins)
b <- board_s3(bucket = "my-existing-bucket")
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
vetiver_pin_write(b, v)

new_image_uri <- vetiver_sm_build(
    board = b,
    name = "cars_linear",
    predict_args = list(type = "class", debug = TRUE),
    docker_args = list(
        base_image = "FROM public.ecr.aws/docker/library/r-base:4.2.2"
    )
)

model_name <- vetiver_sm_model(
    new_image_uri,
    tags = list("my_custom_tag" = "fuel_efficiency")
)

vetiver_sm_endpoint(model_name, "ml.t2.medium")
}
```

---

vetiver_sm_delete *Delete Amazon SageMaker model, endpoint, and endpoint configuration*

Description

Use this function to delete the Amazon SageMaker components used in a vetiver_endpoint_sagemaker() object. This function does *not* delete any pinned model object in S3.

Usage

```
vetiver_sm_delete(object, delete_model = TRUE, delete_endpoint = TRUE)
```

Arguments

object	The model API endpoint object to be deleted, created with vetiver_endpoint_sagemaker().
delete_model	Delete the SageMaker model? Defaults to TRUE.
delete_endpoint	Delete both the endpoint and endpoint configuration? Defaults to TRUE.

Value

TRUE, invisibly

See Also

vetiver_deploy_sagemaker(), vetiver_sm_build(), vetiver_endpoint_sagemaker()

---

vetiver_type_convert *Convert new data at prediction time using input data prototype*

Description

This is a developer-facing function, useful for supporting new model types. At prediction time, new observations typically must be checked and sometimes converted to the data types from training time.

Usage

```
vetiver_type_convert(new_data, ptype)
```

Arguments

new_data	New data for making predictions, such as a data frame.
ptype	An input data prototype, such as a 0-row slice of the training data

Value

A converted dataframe

Examples

```
library(tibble)
training_df <- tibble(x = as.Date("2021-01-01") + 0:9,
                      y = LETTERS[1:10], z = letters[11:20])
training_df

prototype <- vctrs::vec_slice(training_df, 0)
vetiver_type_convert(tibble(x = "2021-02-01", y = "J", z = "k"), prototype)

## unsuccessful conversion generates an error:
try(vetiver_type_convert(tibble(x = "potato", y = "J", z = "k"), prototype))

## error for missing column:
try(vetiver_type_convert(tibble(x = "potato", y = "J"), prototype))
```

`vetiver_write_docker` *Write a Dockerfile for a vetiver model*

Description

After creating a Plumber file with `vetiver_write_plumber()`, use `vetiver_write_docker()` to create a Dockerfile plus a `vetiver_renv.lock` file for a pinned `vetiver_model()`.

Usage

```
vetiver_write_docker(
  vetiver_model,
  plumber_file = "plumber.R",
  path = ".",
  ...,
  lockfile = "vetiver_renv.lock",
  rspm = TRUE,
  base_image = glue::glue("FROM rocker/r-ver:{getRversion()}"),
  port = 8000,
  expose = TRUE,
  additional_pkgs = character(0)
)
```

Arguments

`vetiver_model`	A deployable `vetiver_model()` object
`plumber_file`	A path for your Plumber file, created via `vetiver_write_plumber()`. Defaults to `plumber.R` in the working directory.
`path`	A path to write the Dockerfile and `lockfile`, capturing the model's package dependencies. Defaults to the working directory.
`...`	Not currently used.
`lockfile`	The generated lockfile in `path`. Defaults to `"vetiver_renv.lock"`.
`rspm`	A logical to use the RStudio Public Package Manager for `renv::restore()` in the Docker image. Defaults to `TRUE`.
`base_image`	The base Docker image to start with. Defaults to `rocker/r-ver` for the version of R you are working with, but models like keras will require a different base image.
`port`	The server port for listening: a number such as 8080 or an expression like `'as.numeric(Sys.getenv("PORT"))'` when the port is injected as an environment variable.
`expose`	Add `EXPOSE` to the Dockerfile? This is helpful for using Docker Desktop but does not work with an expression for `port`.
`additional_pkgs`	A character vector of additional package names to add to the Docker image. For example, some boards like `pins::board_s3()` require additional software; you can use `required_pkgs(board)` here.

Value

The content of the Dockerfile, invisibly.

Examples

```
library(pins)
tmp_plumber <- tempfile()
b <- board_temp(versioned = TRUE)
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
vetiver_pin_write(b, v)
vetiver_write_plumber(b, "cars_linear", file = tmp_plumber)

## default port
vetiver_write_docker(v, tmp_plumber, tempdir())
## install more pkgs, like those required to access board
vetiver_write_docker(v, tmp_plumber, tempdir(),
                     additional_pkgs = required_pkgs(b))
## port from env variable
vetiver_write_docker(v, tmp_plumber, tempdir(),
                     port = 'as.numeric(Sys.getenv("PORT"))',
                     expose = FALSE)
```

---

vetiver_write_plumber *Write a deployable Plumber file for a vetiver model*

Description

Use `vetiver_write_plumber()` to create a `plumber.R` file for a `vetiver_model()` that has been versioned and stored via `vetiver_pin_write()`.

Usage

```
vetiver_write_plumber(
  board,
  name,
  version = NULL,
  ...,
  file = "plumber.R",
  rsconnect = TRUE,
  additional_pkgs = character(0)
)
```

Arguments

board	A pin board, created by board_folder(), board_connect(), board_url() or another board_ function.
name	Pin name.
version	Retrieve a specific version of a pin. Use pin_versions() to find out which versions are available and when they were created.
...	Other arguments passed to vetiver_api() such as the endpoint path or prediction type.
file	A path to write the Plumber file. Defaults to plumber.R in the working directory. See plumber::plumb() for naming precedence rules.
rsconnect	Create a Plumber file with features needed for Posit Connect? Defaults to TRUE.
additional_pkgs	Any additional R packages that need to be **attached** via library() to run your API, as a character vector.

Details

By default, this function will find and use the latest version of your vetiver model; the model API (when deployed) will be linked to that specific version. You can override this default behavior by choosing a specific version.

Value

The content of the plumber.R file, invisibly.

Examples

```
library(pins)
tmp <- tempfile()
b <- board_temp(versioned = TRUE)
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
vetiver_pin_write(b, v)

vetiver_write_plumber(b, "cars_linear", file = tmp)
```

Index