Package 'lambdr'

November 25, 2023

Title Create a Runtime for Serving Containerised R Functions on 'AWS Lambda'

Version 1.2.5

Description Runtime for serving containers that can execute R code on the 'AWS Lambda' serverless compute service <https://aws.amazon.com/lambda/>. Provides the necessary functionality for handling the various endpoints required for accepting new input and sending responses.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Imports httr, jsonlite, logger

Suggests withr, testthat (>= 3.0.0), webmockr, knitr, rmarkdown, lifecycle

Config/testthat/edition 3

URL https://lambdr.mdneuzerling.com/, https://github.com/mdneuzerling/lambdr

BugReports https://github.com/mdneuzerling/lambdr/issues

VignetteBuilder knitr

NeedsCompilation no

Author David Neuzerling [aut, cre], James Goldie [ctb]

Maintainer David Neuzerling <david@neuzerling.com>

Repository CRAN

Date/Publication 2023-11-25 11:00:02 UTC

R topics documented:

lambdr-package 2
as_stringified_json 3

from_base64 4
html_response 4
lambda_config 6
start_lambda 8
stop_html 9

Index **11**

`lambdr-package` *lambdr: Create a Runtime for Serving Containerised R Functions on AWS Lambda*

Description

This package provides an R runtime for the *AWS Lambda* serverless compute service. It is intended to be used to create containers that can run on *AWS Lambda*. `lambdr` provides the necessary functionality for handling the various endpoints required for accepting new input and sending responses.

This package is **unofficial**. Its creators are not affiliated with *Amazon Web Services*, nor is its content endorsed by *Amazon Web Services*. *Lambda*, *API Gateway*, *EventBridge*, *CloudWatch*, and *SNS* are services of *Amazon Web Services*.

To see an example of how to use this package to create a runtime, refer to `vignette("lambda-runtime-in-container", package = "lambdr")`.

The default behaviour is to convert the body of the received event from JSON into arguments for the handler function using the `jsonlite` package. For example, a raw event body of `{"number": 9}` will be converted to `list(number = 9)`. The handler function will then receive the arguments directly after unlisting, eg. `number = 9`. This works for direct invocations, as well as situations where the user wishes to implement behaviour specific to a trigger.

Some invocation types have their own logic for converting the event body into an R object. This is useful for say, using an R function in a Lambda behind an API Gateway, so that the R function does not need to deal with the HTML elements of the invocation. The below invocation types have custom logic implemented. Refer to the vignettes or the package website for more information.

Alternatively, user-defined functions can be provided for parsing event content and serialising results. The user can also use the `identity` function as a deserialiser to pass the raw event content — as a string — to the handler function. Refer to `lambda_config` for more information.

Direct invocations

[Stable]

REST API Gateway invocations

[Experimental] `vignette("api-gateway-invocations", package = "lambdr")`

HTML API Gateway invocations

[Experimental] `vignette("api-gateway-invocations", package = "lambdr")`

EventBridge invocations

[Experimental] `vignette("eventbridge-and-sns-invocations", package = "lambdr")`

SNS invocations

[Experimental] `vignette("eventbridge-and-sns-invocations", package = "lambdr")`

Author(s)

Maintainer: David Neuzerling <`david@neuzerling.com`>

Other contributors:

- James Goldie <`me@jamesgoldie.dev`> [contributor]

See Also

Useful links:

- `https://lambdr.mdneuzerling.com/`
- `https://github.com/mdneuzerling/lambdr`
- Report bugs at `https://github.com/mdneuzerling/lambdr/issues`

`as_stringified_json` *Convert an R object to stringified JSON matching AWS Lambda conventions*

Description

Stringified JSON is a string which can be parsed as a JSON. While a standard JSON interpretation of `list(number = 9)` would be `{"number":9}`, a stringified JSON representation would be `"{\"number\":9}"`.

This function will convert `NULL` values to JSON "nulls", to match the convention used by Lambda event inputs, and values are automatically unboxed.

Usage

```
as_stringified_json(x, ...)
```

Arguments

`x`	R object to be converted to stringified JSON.
`...`	additional arguments (except `auto_unbox` and `null`) passed to `toJSON`

Value

character

Examples

```
as_stringified_json(list(number = 9))
"{\"number\":9}"
```

---

from_base64 *Decode a Base64 encoded value to a string*

Description

Events coming via an API Gateway can have content with bodies encoded as Base64. This is especially true for HTML API Gateways (as opposed to REST API Gateways).

This function propagates `NULL`s. That is, `from_base64(NULL)` returns `NULL`.

Usage

```
from_base64(x)
```

Arguments

`x`	a Base64 string

Value

character

Examples

```
from_base64("eyJudW1iZXIiOjd9")
```

---

html_response *Prepare a HTML response for a Lambda behind an API Gateway*

Description

Lambdas behind API Gateways need to send specially formatted responses that look like this:

```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "application/json"
  },
  "isBase64Encoded": false,
  "body": "{\"best_animal\": \"corgi\"}"
}
```

For basic applications where the handler function is returning a simple result, `lambdr` will do its best to automatically return a result compatible with API Gateways. It will do this whenever an event is detected as having come via an API Gateway. For most purposes this is sufficient, and allows users to focus on the handler function rather than the specifics of how *AWS Lambda* works.

For more complicated applications, such as when the Lambda needs to return a specific content type or specific headers, may require a bespoke response. This function will take any R object and format it in style of the above example, allowing for customisation.

When the handler function returns a `html_response` the formatted result will be returned to the API Gateway without further serialisation.

Usage

```
html_response(
  body,
  is_base64 = FALSE,
  status_code = 200L,
  content_type = NULL,
  headers = list()
)
```

Arguments

`body`	the actual result to be delivered. This is not serialised in any way, so if this is a list to be interpreted JSON it should be stringified, that is, it should be a string of a JSON. Consider using the `as_stringified_json` function.
`is_base64`	logical which indicates if `body` is Base64 encoded. Defaults to False.
`status_code`	integer status code of the response. Defaults to `200L` (OK).
`content_type`	MIME type for the content. This will be appended to the headers (as "Content-Type"), unless such a value is already provided to `headers`, in which case this argument is ignored. If not provided then no information on headers will be sent in the response, leaving the beahviour up to the defaults of the API Gateway.
`headers`	additional headers, as a named list, to be included in the response. If this contains a "Content-Type" value then `content_type` is ignored.

Value

A stringified JSON response for an API Gateway, with the "already_serialised" attribute marked as `TRUE`. This will stop `serialise_result` from attempting to serialise the result again.

Examples

```
html_response("abc")
html_response("YWJj", is_base64 = TRUE)
html_response("abc", headers = list(x = "a"))
html_response(
  "<html><body>Hello World!</body></html>",
  content_type = "text/html"
)
```

`lambda_config` *Set up endpoints, variables, and configuration for AWS Lambda*

Description

This function provides a configuration object that can be passed to `start_lambda`. By default it will use the environment variables configured by AWS Lambda and so will often work without arguments.

The most important configuration variable is the handler function which processes invocations of the Lambda. This is configured in any of the three below ways, in order of decreasing priority:

1. configured directly through the AWS Lambda console
2. configured as the `CMD` argument of the Docker container holding the runtime
3. passed as a value to the `handler` argument of `lambda_config`

In the first two options, the handler will be made available to the runtime through the "_HANDLER" environment variable. This function will search for the function in the given `environment`.

If the handler accepts a `context` argument then it will receive a list of suitable event context for every invocation. This argument must be named (`...` will not work), and the configuration may be different for each invocation type. See the section below for more details.

Usage

```
lambda_config(
  handler = NULL,
  runtime_api = NULL,
  task_root = NULL,
  deserialiser = NULL,
  serialiser = NULL,
  decode_base64 = TRUE,
  environ = parent.frame()
)
```

Arguments

`handler`	the function to use for processing inputs from events. The "_HANDLER" environment variable, as configured in AWS, will always override this value if present.
`runtime_api`	character. Used as the host in the various endpoints used by AWS Lambda. This argument is provided for debugging and testing only. The "AWS_LAMBDA_RUNTIME_API" environment variable, as configured by AWS, will always override this value if present.
`task_root`	character. Defines the path to the Lambda function code. This argument is provided for debugging and testing only. The "LAMBDA_TASK_ROOT" environment variable, as configured by AWS, will always override this value if present.

`deserialiser`	function for deserialising the body of the event. By default, will attempt to deserialise the body as JSON, based on whether the input is coming from an API Gateway, scheduled Cloudwatch event, or direct. To use the body as is, pass the `identity` function. To ignore the event content, pass `function(x) list()`. See the vignettes for details on parsing invocations from particular sources.
`serialiser`	function for serialising the result before sending. By default, will attempt to serialise the body as JSON, based on the request type. To send the result as is, pass the `identity` function.
`decode_base64`	logical. Should Base64 input be automatically decoded? This is only used for events coming via an API Gateway. Complicated input (such as images) may be better left as is, so that the handler function can deal with it appropriately. Defaults to `TRUE`. Ignored if a custom `deserialiser` is used.
`environ`	environment in which to search for the function given by the "_HANDLER" environment variable. Defaults to the parent frame.

Details

As a rule of thumb, it takes longer to retrieve a value from an environment variable than it does to retrieve a value from R. This is because retrieving an environment variable requires a system call. Since the environment variables do not change in a Lambda instance, we fetch them once and store them in a configuration object which is passed to the various internal functions.

AWS Lambda variables

The `lambda_config` function obtains the configuration values for the Lambda runtime configures the R session for Lambda based on environment variables made available by Lambda. The following environment variables are available:

- Lambda Runtime API, available as the "AWS_LAMBDA_RUNTIME_API" environment variable, is the host of the various HTTP endpoints through which the runtime interacts with Lambda.
- Lambda Task Root, available as the "LAMBDA_TASK_ROOT" environment variable, defines the path to the Lambda function code. It isn't used in container environments with a custom runtime, as that runtime is responsible for finding and sourcing the function code. Hence, a missing task root is ignored by this package.
- The handler, available as the "_HANDLER" environment variable, is interpreted by R as the function that is executed when the Lambda is called. This value could be anything, as the interpretation is solely up to the runtime, so requiring it to be a function is a standard imposed by this package.

These `handler`, `runtime_api` and `task_root` arguments to the `lambda_config` function can also provide values to these configuration options, although the environment variables will always be used if available. While it may be sensible to provide the `handler` function directly, the other two configuration options are only provided for debugging and testing purposes.

Event context

Context is metadata associated with each invocation. If the handler function accepts a `context` argument then it will automatically receive at runtime a named list consisting of these values along with the arguments in the body (if any). For example, a function such as `my_func(x, context)` will receive the context argument automatically. The `context` argument must be named (`...` will not work).

Refer to `vignette("lambda-runtime-in-container", package = "lambdr")` for details.

`start_lambda` *Start the Lambda runtime*

Description

This is the main function of the package, responsible for starting the infinite loop of listening for new invocations. It relies on configuration provided to the `config` argument and produced by the `lambda_config` function.

Usage

```
start_lambda(
  config = lambda_config(environ = parent.frame()),
  timeout_seconds = NULL
)
```

Arguments

`config`	A list of configuration values as created by the `lambda_config` function.
`timeout_seconds`	If set, the function will stop listening for events after this timeout. The timeout is checked between events, so this won't interrupt the function while it is waiting for a new event. This argument is provided for testing purposes, and shouldn't otherwise need to be set: AWS should handle the shutdown of idle Lambda instances.

Details

See `vignette("lambda-runtime-in-container", package = "lambdr")` for an example of how to use this function to place an R Lambda Runtime in a container.

This package uses the `logger` package for logging. Debug log entries can be enabled with `logger::log_threshold(logger`
This will log additional information such as raw event bodies.

Event context

Context is metadata associated with each invocation. If the handler function accepts a `context` argument then it will automatically receive at runtime a named list consisting of these values along with the arguments in the body (if any). For example, a function such as `my_func(x, context)` will receive the context argument automatically. The `context` argument must be named (`...` will not work).

Refer to `vignette("lambda-runtime-in-container", package = "lambdr")` for details.

AWS Lambda variables

The lambda_config function obtains the configuration values for the Lambda runtime configures the R session for Lambda based on environment variables made available by Lambda. The following environment variables are available:

- Lambda Runtime API, available as the "AWS_LAMBDA_RUNTIME_API" environment variable, is the host of the various HTTP endpoints through which the runtime interacts with Lambda.
- Lambda Task Root, available as the "LAMBDA_TASK_ROOT" environment variable, defines the path to the Lambda function code. It isn't used in container environments with a custom runtime, as that runtime is responsible for finding and sourcing the function code. Hence, a missing task root is ignored by this package.
- The handler, available as the "_HANDLER" environment variable, is interpreted by R as the function that is executed when the Lambda is called. This value could be anything, as the interpretation is solely up to the runtime, so requiring it to be a function is a standard imposed by this package.

These `handler`, `runtime_api` and `task_root` arguments to the lambda_config function can also provide values to these configuration options, although the environment variables will always be used if available. While it may be sensible to provide the `handler` function directly, the other two configuration options are only provided for debugging and testing purposes.

Examples

```
## Not run:
# A general usage pattern involves sourcing necessary functions and running
# this `start_lambda` in a `runtime.R` file which is then executed to start
# the runtime. In the following example, the function handler can be set to
# "lambda" either as the container `CMD`, or configured through AWS Lambda.

parity <- function(number) {
  list(parity = if (as.integer(number) %% 2 == 0) "even" else "odd")
}

start_lambda()

# Alternatively, it can be passed as an argument `handler = parity` to
# the lambda configuration. If the handler is configured through other means
# then this will be ignored:

start_lambda(config = lambda_config(handler = parity))

## End(Not run)
```

`stop_html` *Raise an error with an optional HTML status code for API Gateways*

Description

This variation of `stop` can be used to raise an error with a specific error code. This is provided to the API Gateway to return an appropriate response. It had no use outside of invocations via an API Gateway.

If a status code is not provided, a generic "500" internal server error will be used.

Usage

```
stop_html(..., code = 500L)
```

Arguments

`...`	zero or more objects which can be coerced to character (and which are pasted together with no separator). This forms the error message.
`code`	HTTP status code to return (if applicable). Defaults to `500`, which is a generic "Internal Server Error". This is used when errors are to be returned to an API Gateway.

Examples

```
## Not run:
stop_html("Resource doesn't exist", code = 404L)

## End(Not run)
```

Index

as_stringified_json, 3, *5*

from_base64, 4

html_response, 4

lambda_config, *2*, 6, *7–9*
lambdr (lambdr-package), 2
lambdr-package, 2
logger, *8*

start_lambda, *6*, 8
stop_html, 9

toJSON, *3*