# downtoearth Documentation

*Release undefined*

**ClearDATA**

**Feb 07, 2017**

# Contents

Contents:

CHAPTER 1

# downtoearth

A tool for generating APIs in AWS, powered by Lambda and API Gateway, backed by terraform.

We're just hooking up http verbs to python functions... shouldn't be that tough.

Makes a terraform file for deployment.

CHAPTER 2

# Documentation

Full documentation can be found at https://downtoearth.readthedocs.io/en/latest/.

## 2.1 The api.json file

A json file is used to define your api. From this, downtoearth will generate a terraform document.

```
{
  "Name": "DownToEarthApi",
  "Description": "test API for the downtoearth tool",
  "AccountNumber": "123456789012",
  "LambdaZip": "dist/api-lambda.zip",
  "LambdaHandler": "lambda_handler.lambda_handler",
  "LambdaRuntime": "python2.7",
  "Roles": {
    "MyRole": {
      "PolicyDoc": "GoesHere"
    }
  },
  "Api":{
    "/api/X/{1}": ["GET"],
    "/api/X": ["GET", "POST"],

    "/api/Y": ["GET", "POST"],
    "/api/Y/{1}": ["GET"]
  }
}
```

## 2.2 Writing your lambda

The event dictionary gets filled with a "route" element that contains a string representing the verb and endpoint hit.

```
VERB:/api/my/endpoint/{my_variable}
```

This simple example will show you how to map that name to a python function.

```
def get_y(event, context):
    return dict(oh="yaaaaa!")

function_mapping = {
    "GET:/api/Y": get_y
}


def route_request(event, context):
    if "route" not in event:
        raise ValueError("must have 'route' in event dictionary")

    if event["route"] not in function_mapping:
        raise ValueError("cannot find {0} in function mapping".format(event["route"]))

    func = function_mapping[event["route"]]
    return func(event, context)


def lambda_handler(event, context=None):
    print("event: %s"%event)
    return route_request(event, context)
```

TODO: it'd be awesome if this worked with decorators like flask or chalice.

## 2.3 Router

If your API is straightforward there is no reason to write your own router. We provide one. Your lambda code could be as simple as:

```
from downtoearth.router import Router

ROUTE_MAP = {
    "GET:/v1/book": get_all,
    "POST:/v1/book": post_book,
    "GET:/v1/book/{isbn}": get_book,
    "PUT:/v1/book/{isbn}": update_book,
    "DELETE:/v1/book/{isbn}": remove_book
}


def handle_event(event, context):
    """Route and handle incoming event."""
    router = Router(ROUTE_MAP)
    return router.route_request(event, context)
```

## 2.4 Returning different status codes

The generated API gateway includes a number of common response codes along with their official descriptions. To return a non-200 OK HTTP code, raise an exception with an official description bracketed at the beginning. For example, to return a 404:

```
if not found:
    raise ValueError('[Not Found] Could not find %s' % item_id)
```

Or you can nicely handle responses from DynamoDB:

```
try:
    db.put_item(Item=item,
                ConditionExpression='attribute_not_exists(item_id)')
except ClientError:
    if 'ConditionalCheckFailedException' in e.args[0]:
        raise ValueError('[Conflict] %s already exists' % item['id'])
    else:
        raise Exception('[Internal Server Error] An unknown error occurred.  Info: %s
↪' % e.args[0])
```

The currently supported status codes are defined in rfc7231codes, in api_endpoints.hcl. To add support for a new status code, extend that tuple with a (code, description) pair.

Currently, there is no way to return additional headers or a custom body. All non-200 integration responses just contain the lambda output errorMessage field.

## 2.5 Exceptions

We also provide exceptions helpers for you. If you are using the provided router you won't need this. If you write your own router, use them like this.

```
from downtoearth.exceptions import NotFoundException

if not found:
    raise NotFoundException('Could not find %s' % item_id)
```

## 2.6 Creating the Terraform

```
cli.py INPUT_API_DEFITION_PATH OUTPUT_TERRAFORM_PATH
# or if you have it installed
downtoearth INPUT_API_DEFITION_PATH OUTPUT_TERRAFORM_PATH
```

## 2.7 Stages, Deployment, and You

By default, downtoearth with create a single "production" stage. Create multiple stages by providing an array of names to the Stages key of the config

```
"Stages": ["production", "develop"]
```

Applying the terraform created by downtoearth will create an alias in your lambda for each stage you defined.

Now here's the tricky part: because stages and lambda versions and aliases are so weird, we have to update the lambda that powers a specific stage outside of terraform. This is just easier, I promise. And hopefully, the shape of your API will change much less often than the code that powers it, so you won't have to constantly churn terraform applies just because you fixed a bug in your code.

Your stage aliases are initially set up to point to the $LATEST version. When you wanna push fresh code to a stage, publish a version of your code, update the alias to point to that version. We will soon provide a downtoearth cli command to help you deploy a zip to a stage, but for now, here's a little `./deploy.sh STAGE` script to help

```
#!/usr/bin/env bash
STAGE=$1
aws lambda update-function-code --function-name MY_FUNCTION_ROOT --zip-file fileb://
↪MY_ZIP.zip
VERSION="$(aws lambda  --region=us-east-1 publish-version --function-name MY_FUNCTION_
↪ROOT | jq -r .Version)"
echo "Created version #$VERSION"
aws lambda update-alias --function-name MY_FUNCTION_ROOT --name $STAGE --function-
↪version $VERSION
```

CHAPTER 3

# Installation

## 3.1 Stable release

To install downtoearth, run this command in your terminal:

```
$ pip install downtoearth
```

This is the preferred method to install, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

## 3.2 From sources

The sources for howsitgoing can be downloaded from the Github repo.

Clone the repository:

```
$ git clone git@github.com:cleardataeng/downtoearth.git
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 4

# Usage

## 4.1 api.json

A json file is used to define your api. From this, downtoearth will generate a terraform document.

```
{
  "Name": "DownToEarthApi",
  "Description": "test API for the downtoearth tool",
  "AccountNumber": "123456789012",
  "LambdaZip": "dist/api-lambda.zip",
  "LambdaHandler": "lambda_handler.lambda_handler",
  "LambdaRuntime": "python2.7",
  "Roles": {
    "MyRole": {
      "PolicyDoc": "GoesHere"
    }
  },
  "Api":{
    "/api/X/{1}": ["GET"],
    "/api/X": ["GET", "POST"],

    "/api/Y": ["GET", "POST"],
    "/api/Y/{1}": ["GET"]
  }
}
```

## 4.2 lambda

The event dictionary gets filled with a "route" element that contains a string representing the verb and endpoint hit.

```
VERB:/api/my/endpoint/{my_variable}
```

This simple example will show you how to map that name to a python function.

```
def get_y(event, context):
    return dict(oh="yaaaaa!")

function_mapping = {
    "GET:/api/Y": get_y
}


def route_request(event, context):
    if "route" not in event:
        raise ValueError("must have 'route' in event dictionary")

    if event["route"] not in function_mapping:
        raise ValueError("cannot find {0} in function mapping".format(event["route"]))

    func = function_mapping[event["route"]]
    return func(event, context)


def lambda_handler(event, context=None):
    print("event: %s"%event)
    return route_request(event, context)
```

TODO: it'd be awesome if this worked with decorators like flask or chalice.

## 4.3 Router

If your API is straightforward there is no reason to write your own router. We provide one. Your lambda code could be as simple as:

```
from downtoearth.router import Router

ROUTE_MAP = {
    "GET:/v1/book": get_all,
    "POST:/v1/book": post_book,
    "GET:/v1/book/{isbn}": get_book,
    "PUT:/v1/book/{isbn}": update_book,
    "DELETE:/v1/book/{isbn}": remove_book
}


def handle_event(event, context):
    """Route and handle incoming event."""
    router = Router(ROUTE_MAP)
    return router.route_request(event, context)
```

## 4.4 Returning different status codes

The generated API gateway includes a number of common response codes along with their official descriptions. To return a non-200 OK HTTP code, raise an exception with an official description bracketed at the beginning. For example, to return a 404:

```
if not found:
    raise ValueError('[Not Found] Could not find %s' % item_id)
```

Or you can nicely handle responses from DynamoDB:

```
try:
    db.put_item(Item=item,
                ConditionExpression='attribute_not_exists(item_id)')
except ClientError:
    if 'ConditionalCheckFailedException' in e.args[0]:
        raise ValueError('[Conflict] %s already exists' % item['id'])
    else:
        raise Exception('[Internal Server Error] An unknown error occurred.  Info: %s
↪' % e.args[0])
```

The currently supported status codes are defined in rfc7231codes, in api_endpoints.hcl. To add support for a new status code, extend that tuple with a (code, description) pair.

Currently, there is no way to return additional headers or a custom body. All non-200 integration responses just contain the lambda output errorMessage field.

## 4.5 Exceptions

We also provide exceptions helpers for you. If you are using the provided router you won't need this. If you write your own router, use them like this.

```
from downtoearth.exceptions import NotFoundException

if not found:
    raise NotFoundException('Could not find %s' % item_id)
```

## 4.6 Creating the Terraform

```
cli.py INPUT_API_DEFITION_PATH OUTPUT_TERRAFORM_PATH
# or if you have it installed
downtoearth INPUT_API_DEFITION_PATH OUTPUT_TERRAFORM_PATH
```

## 4.7 Stages, Deployment, and You

By default, downtoearth with create a single "production" stage. Create multiple stages by providing an array of names to the Stages key of the config

```
"Stages": ["production", "develop"]
```

Applying the terraform created by downtoearth will create an alias in your lambda for each stage you defined.

Now here's the tricky part: because stages and lambda versions and aliases are so weird, we have to update the lambda that powers a specific stage outside of terraform. This is just easier, I promise. And hopefully, the shape of your API will change much less often than the code that powers it, so you won't have to constantly churn terraform applies just because you fixed a bug in your code.

Your stage aliases are initially set up to point to the $LATEST version. When you wanna push fresh code to a stage, publish a version of your code, update the alias to point to that version. We will soon provide a downtoearth cli command to help you deploy a zip to a stage, but for now, here's a little `./deploy.sh STAGE` script to help

```
#!/usr/bin/env bash
STAGE=$1
aws lambda update-function-code --function-name MY_FUNCTION_ROOT --zip-file fileb://
↪MY_ZIP.zip
VERSION="$(aws lambda  --region=us-east-1 publish-version --function-name MY_FUNCTION_
↪ROOT | jq -r .Version)"
echo "Created version #$VERSION"
aws lambda update-alias --function-name MY_FUNCTION_ROOT --name $STAGE --function-
↪version $VERSION
```

CHAPTER 5

# Contributing

Contribution are welcome. You can contribute in the following ways.

- report bugs
- fix bugs
- implement features
- write documentation
- submit feedback

## 5.1 Get Started

## 5.2 Pull / Merge Request Guidelines

CHAPTER 6

# Credits

## 6.1 Core Developers

- Manoli Yiannakakis manoli.yiannakakis@cleardata.com

## 6.2 Contributors

- John Bloom john.bloom@cleardata.com
- Jeffrey DeFond jeff.defond@cleardata.com
- Brian Hammons brian.hammons@cleardata.com
- Herkermer Sherwood herkermer.sherwood@cleardata.com

CHAPTER 7

# downtoearth

## 7.1 downtoearth package

### 7.1.1 Submodules

### 7.1.2 downtoearth.cli module

downtoearth creates terraform files from api configuration definitions.

downtoearth.cli.**parse_args**()
Parse arguments.

downtoearth.cli.**main**()
Build template and output to file.

### 7.1.3 downtoearth.default module

Define defaults used in downtoearth.

### 7.1.4 downtoearth.exceptions module

Exceptions for downtoearth.

These are helpers provided so that you can raise proper HTTP code errors from your API.

**Usage:** from downtoearth.exceptions import NotFoundException raise NotFoundException('your princess is in another castle')

**exception** downtoearth.exceptions.**BadRequestException**(*msg*)
Bases: exceptions.Exception

**exception** downtoearth.exceptions.**ConflictException**(*msg*)
Bases: exceptions.Exception

**exception** downtoearth.exceptions.**InternalServerErrorException**(*msg*)
Bases: exceptions.Exception

**exception** downtoearth.exceptions.**NotFoundException**(*msg*)
Bases: exceptions.Exception

**exception** downtoearth.exceptions.**NotImplementedException**(*msg*)
Bases: exceptions.Exception

### 7.1.5 downtoearth.model module

downtoearth API model.

**class** downtoearth.model.**ApiModel**(*args*)
Bases: object

downtoearth data model.

**get_endpoints**()
Get all paths that contain methods.

**get_api_template_variables**()
Get API template variables.

**render_terraform**()
Return a rendered terraform template.

**run_stage_deployments**()

**run_terraform**()
Return a apply terraform after template rendered.

**class** downtoearth.model.**UrlTree**
Bases: object

**process_url**(*url*, *methods*)

**traverse_tree**(*node*, *depth=0*)

**get_endpoints**()

**class** downtoearth.model.**UrlNode**(*url*, *methods=None*, *parent=None*)
Bases: object

**prefix**

**url_name**

**full_url**

**get_endpoint_info**(*api_name*)

**append_methods**(*methods*)

**add_child**(*url*, *methods=None*)

**is_leaf**()

**is_root**()

**get_child**(*value*)

**has_child**(*value*)

**is_variable**()

### 7.1.6 downtoearth.router module

Router helper for API's using downtoearth.

**class** downtoearth.router.**Router**(*route_map=None*, *param_order=None*)
Bases: object

Routing object for given route_map.

**Parameters**

- **route_map** (*optional[dict]*) – map of route to delegate ex.

  **map = {** "GET:/v1/book": get_all, "POST:/v1/book": post_book, "GET:/v1/book/{isbn}": get_book, "PUT:/v1/book/{isbn}": update_book, "DELETE:/v1/book/{isbn}": remove_book

  }

- **param_order** (*optional[list]*) – order of precedence for parameters This should include all three parameter types. ex. ['path', 'querystring', 'body'] defaults to ['path', 'body', 'querystring']

**DEFAULTS = {'param_order': ['path', 'body', 'querystring']}**

**add_full_route**(*route*, *delegate*)
Add route given route and delegate function.

**Parameters**

- **route** (*str*) – route in format "VERB:route/{variables}"
- **delegate** (*func*) – function to call

**add_route**(*verb*, *path*, *delegate*)
Add route given verb, path, delegate function.

**Parameters**

- **verb** (*str*) – HTTP verb ex. GET, POST
- **path** (*str*) – path
- **delegate** (*func*) – function to call

**route_request**(*event*, *context*)
Route incoming request.

### 7.1.7 downtoearth.test module

### 7.1.8 downtoearth.version module

Package version will be written by setup.py.

### 7.1.9 Module contents

CHAPTER 8

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## d

# Index