chalice-http-toolkit

Release 0.0.6

Chris Lapa

Mar 17, 2023

CONTENTS

1 Examples **3**

2 API Reference **5**
- 2.1 Jinja2 API . . . 5
- 2.2 Python API . . . 5

3 Projects.md **11**

4 Installation **13**
- 4.1 Quick setup . . . 13

5 Chalice config.json **15**

6 Project Structure **17**
- 6.1 Architecture . . . 17
- 6.2 ContentManager . . . 17
- 6.3 CloudFront . . . 17
- 6.4 TaskingController . . . 18

7 Considerations **19**
- 7.1 Binary Content . . . 19
- 7.2 Building Responsive Apps . . . 19
- 7.3 Request & Reponse Limits . . . 21
- 7.4 Cold Starts . . . 21

Python Module Index **23**

Index **25**

chalice-http-toolkit enables a Flask like website building experience using Chalice, AWS Lambda, API Gateway & CloudFront. It does this by bolting on Jinja2 templates, and CloudFront cache management layer.

CHAPTER

ONE

EXAMPLES

A simple example project exists here

CHAPTER

TWO

API REFERENCE

2.1 Jinja2 API

static(<filename>) is a function which resolves static contents path. A Chalice route should be set which matches *ContentManager.set_static_handler_prefix()*'s value.

2.2 Python API

2.2.1 chalice_http_toolkit.cloudfront module

class chalice_http_toolkit.cloudfront.**CloudFront**(*content_manager:* chalice_http_toolkit.content.ContentManager, *app: Union[chalice.Chalice, chalice.Blueprint]*, *default_cache_control='no-cache'*)

Bases: object

Supports caching via CloudFront for the following usecases: - Jinja2 Templates by exploring dependancy trees of templates and their arguments. - Static content by extracting modified dates from static content.

Parameters

- **content_manager** – challice_http_toolkit.ContentManager
- **app** – Chalice or Blueprint
- **default_cache_control** – Cache control to use if none specified

Returns CloudFront instance

asset(*get_asset: Callable[], bytes]*, *get_etag: Callable[], str]*, *status_code: int = 200*, *headers: Optional[dict] = None*, *cache_control: Optional[str] = None*) → chalice.Response

Calls get_etag() to generate a tag for the asset, and if it matches the If-None-Match value, we return a 304. Otherwise get_asset() is called and returned.

Parameters

- **get_asset** – Callable to get the asset if ETag doesnt match If-None-Match value
- **get_etag** – Callable to get etag of the asset
- **status_code** – Status code to return content with
- **headers** – Additional headers to return

- **cache_control** – Controls how CloudFront manages re-evaluating the orgin. See https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Expiration.html and https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control.

Returns Chalice Response object

static (*filename: str*, *status_code: int = 200*, *headers: Optional[dict] = None*, *cache_control: Optional[str] = None*) → chalice.Response

If CloudFront supplies If-Modified-Since header in request, then we can check against the modified date of the file to inform CloudFront if its cache needs to be updated or not.

Parameters

- **filename** – Path relative from static_dir to content
- **status_code** – Status code to return content with
- **headers** – Additional headers to return
- **cache_control** – Controls how CloudFront manages re-evaluating the orgin. See https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Expiration.html and https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control.

Returns Chalice Response object

template (*filename: str*, *status_code: int = 200*, *headers: Optional[dict] = None*, *cache_control: Optional[str] = None*, ***kwargs*) → chalice.Response

If CloudFront supplies If-None-Match header in request, then we can check against a hash of the template, its dependancies, and its arguments to inform CloudFront if its cache needs to be updated or not.

Parameters

- **filename** – Path to template in which to be rendered or cached
- **status_code** – Status code to return content with
- **headers** – Additional headers to return
- **cache_control** – Controls how CloudFront manages re-evaluating the orgin. See https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Expiration.html and https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control.
- **kwargs** – Keyword arguments to be passed to template in the event that it is rendered, must be pickleable

Returns Chalice Response object

2.2.2 chalice_http_toolkit.content module

class chalice_http_toolkit.content.**ContentManager**(*templates_dir: str*, *static_dir: str*, *app: Union[chalice.Chalice, chalice.Blueprint]*, *magic: Optional[magic.Magic] = None*)

Bases: object

ACCEPTS2PIL = {'image/gif': 'GIF', 'image/jpeg': 'JPEG', 'image/jpg': 'JPEG', 'imag

Creates a ContentManager instance which allows rendering templates, returning static content and dynamic assets plus more. Generally after a ContentManager is called, set_static_handler_prefix() should be called directlty after to setup your static() handler in Jinja templates.

Parameters

- **templates_dir** – Base path for templates

- **static_dir** – Base path for static content
- **app** – Chalice app
- **magic** – Optional python-magic instance, often required given Lambda hosts dont have libmagic installed.

Returns ContentManager instance

asset(*body: bytes*, *status_code: int = 200*, *headers: Optional[dict] = None*) → *chalice_http_toolkit.response_with_binary.ResponseWithBinary*
Useful for returning dynamic static content such as images loaded from a database.

Parameters

- **body** – Bytes representing content to return
- **status_code** – Status code to return content with
- **headers** – Additional headers to return

Returns Chalice Response object

convert_by_accepts(*img: bytes*, *accepts: str*, *default='WEBP'*) → bytes
Converts an images type based on the accepts format (ie image/jpeg) using Pillow. If default=None, will throw an exception if no conversion found.

Parameters

- **img** – Bytes of image
- **accepts** – Mimetype format, current supports: image/jpeg, image/jpg, image/webp, image/png, image/gif
- **default** – Default format if cant convert, prevents an exception being thrown if cant convert.

Returns Converted image

Raises Exception if cant find matching type to convert to

html(*body: str*, *status_code: int = 200*, *headers: Optional[dict] = None*) → chalice.Response
Useful for returning html content that has been rendered already using render_template().

Parameters

- **body** – String containing rendered template
- **status_code** – Status code to return content with
- **headers** – Additional headers to return

Returns Chalice Response object

json(*body: str*, *status_code: int = 200*, *headers: Optional[dict] = None*) → chalice.Response
Useful for returning json content.

Parameters

- **body** – String containing json
- **status_code** – Status code to return content with
- **headers** – Additional headers to return

Returns Chalice Response object

redirect(*url: str*, *status_code: int = 301*) → chalice.Response
By default generates a 301 redirect Response.

Parameters **url** – URL to redirect to

Returns Chalice Response object

render_template(*filename: str*, ***kwargs*) → str
Renders a template using the existing Jinja2 environment.

Parameters

- **filename** – Path to template
- **kwargs** – Additional kwargs to be passed into template render

Returns Rendered template as a string

set_static_handler_prefix(*path: str*)
Defines a function static() to allow dynamic resolution of static content in templates. The path argument should correlate with a Chalice endpoint to fetch the static content.

Parameters **path** – Path to match chalice endpoint in which static() calls will match in templates.

Returns None

static(*filename: str*, *status_code: int = 200*, *headers: Optional[dict] = None*) → *chalice_http_toolkit.response_with_binary.ResponseWithBinary*
Useful for returning static content in the configured static directory.

Parameters

- **filename** – Path relative from static_dir to content
- **status_code** – Status code to return content with
- **headers** – Additional headers to return

Returns Chalice Response object

xml(*body: str*, *status_code: int = 200*, *headers: Optional[dict] = None*) → chalice.Response
Useful for returning xml content that has been rendered already using render_template().

Parameters

- **body** – String containing rendered template
- **status_code** – Status code to return content with
- **headers** – Additional headers to return

Returns Chalice Response object

2.2.3 chalice_http_toolkit.multipart module

chalice_http_toolkit.multipart.**ex_multipart**(*app: Union[chalice.Chalice, chalice.Blueprint]*) → dict
Decodes content_type='multipart/form-data' and returns a dictionary of parts in the format: {

"<name>": {"data": <content>, "filename": <filename>}

} :returns: Dictionary of decoded content

2.2.4 chalice_http_toolkit.response_with_binary module

class chalice_http_toolkit.response_with_binary.**ResponseWithBinary**(**args: Any*, ***kwargs: Any*)

Bases: chalice.

Wrapper class to override isBase64Encoded behaviour in the Reponse class. Setting isBase64Encoded is usually done to encode Binary content for API Gateway so that it knows to decode it.

isBase64Encoded = False

isLocal = True

to_dict(*binary_types=None*) → dict

2.2.5 chalice_http_toolkit.tasking module

CHAPTER

THREE

PROJECTS.MD

Projects using chalice-http-toolkit so far:

- isitmouldy

CHAPTER

FOUR

INSTALLATION

Base installation via pip:

```
pip install chalice-http-toolkit
```

The above command only installs the base dependancies, there is also an additional extra_requires directive shown below that installs packages which would normally push your Lambda package over the 250MB limit.

For local testing the rest of the dependancies can be installed via:

```
pip install chalice-http-toolkit[layered]
```

For deployments to AWS Lambda the following layers should be used:

```
arn:aws:lambda:us-east-2:770693421928:layer:Klayers-python38-jinja2:2
arn:aws:lambda:us-east-2:770693421928:layer:Klayers-python38-Pillow:10
```

OR run

```
$ chalice-http-toolkit layers -r us-east-1 # Returns the latest compatible layer␣
↪versions
arn:aws:lambda:us-east-1:770693421928:layer:Klayers-python38-jinja2:6
arn:aws:lambda:us-east-1:770693421928:layer:Klayers-python38-Pillow:15
```

4.1 Quick setup

challice-http-toolkit now comes with a CLI utility to quicky create a project, usage is as follows:

```
chalice-http-toolkit setup -n myapp -p . -e magic -r us-east-1
```

The above script does the following:

1. Creates your app.py, with handlers for index, static content, 404 and a event to keep the Lambda function warm.
2. Creates directory structure for templates, static content, chalice libraries.
3. Sets up Chalice layers (Pillow and Jinja2) which are required chalice-http-toolkit.
4. If `-e magic` is provided, then magic binaries are included in `chalicelib/libs` directory. This is required because Lambda instances dont include these binaries so magic wont work without it.

The latest layers can be fetched using the following command:

```
chalice-http-toolkit layers -r us-east-1
```

Currently only Python 3.8 is supported for chalice-http-toolkit, this is because we would need to extract binaries for magic from different AWS Linux versions with varying Python versions and also build Pillow and Jinja2 layers for different Python versions.

CHAPTER

FIVE

CHALICE CONFIG.JSON

A basic Chalice `config.json` is defined below which has two stages, `dev` is meant for local testing and `prod` is the stage which gets deployed to AWS Lambda. The `STAGE` environment variable must be defined for local testing to work around some differences between AWS API Gateway behaviour vs the way locally served API works.

```
{
  "version": "2.0",
  "app_name": "example",
  "stages": {
    "dev": {
      "api_gateway_stage": "dev",
      "lambda_timeout": 60,
      "lambda_memory_size": 64,
      "environment_variables": {
        "STAGE": "dev"
      },
    },
    "prod": {
      "api_gateway_stage": "prod",
      "layers": ["arn:aws:lambda:us-east-2:770693421928:layer:Klayers-python38-
↪jinja2:2",
                 "arn:aws:lambda:us-east-2:770693421928:layer:Klayers-python38-
↪Pillow:10"],
      "lambda_timeout": 60,
      "lambda_memory_size": 64,
      "environment_variables": {
        "STAGE": "prod"
      }
    }
  }
}
```

CHAPTER

SIX

PROJECT STRUCTURE

Templates and static content needs to be placed in the `chalicelib` directory because that is where additional content is packaged into the deployment by Chalice.

6.1 Architecture

Placing CloudFront in front of a chalice-http-toolkit website is not required, but if your website is popular it probably starts to make sense given that every invocation ends up costing more then it would for CloudFront to server the website.

6.2 ContentManager

class chalice_http_toolkit.content.**ContentManager**

This class is designed to support:

- serving static content (such as Javascript/CSS). Although these might be better served by S3 depending on scale.
- rendering Jinja2 templates
- support generating 304 redirects
- support returning JSON Responses
- serving assets
- converting images based on the accepts header in a request
- cleaner handling of binary content

Every chalice-http-toolkit app requires a ContentManager be created, usually right after the Chalice app is created.

6.3 CloudFront

class chalice_http_toolkit.cloudfront.**CloudFront**

This class is designed to support:

- caching static content by modifed dates
- caching Jinja2 templates by hashing dependancies (ie arguments and child templates) without rendering.
- caching assets

Using this module is optional, but it is best practice.

6.4 TaskingController

class chalice_http_toolkit.tasking.**TaskingController**

This class is designed to support:

- processing of asynchronous tasking via a FIFO SQS Queue.

CHAPTER

SEVEN

CONSIDERATIONS

There are a few differences between locally testing & a deployed instance of a Chalice website. They are detailed below.

7.1 Binary Content

Due to the way API Gateway handles binary content, the binary content is Base64 encoded in the absence of an Accepts header in the request. This restriction does not apply for local testing. In practice this means if you make a request to your Lambda function, API gateway will decode the Base64 data based on the Accepts header for us. Therefore if its not supplied, no decoding happens and a Base64 blob is returned.

7.2 Building Responsive Apps

Building a webapp with chalice-http-toolkit is a little bit different, the responsiveness of your application depends on having the shortess execution time possible Lambda functions. In practice if you start to make requests to other services like error/user tracking between returning data from your function, the users experience will get worse. A better approach is to put these requests onto a SQS queue and process them elsewhere given that SQS is usually pretty speedy.

To achieve this design, there is a function called `TaskingControllerFactory` which enables asynchronous processing of messages through pushing to a FIFO SQS queue, and receiving in another Lambda function for processing. Below is an example of deferring requests to posthog.com.

```
from chalice import Chalice
from chalicelib.tracking import tasking

app = Chalice(app_name='testapp')

# ALL handler registration MUST happen in app.py as its the execution entrypoint when␣
↪AWS
# executes our functions.
tasking.register_handler(app)
```

```
import traceback
import json
import logger
import os
from app import app
from chalice_http_toolkit.tasking import TaskingControllerFactory

```

(continues on next page)

(continued from previous page)

```
tasking = TaskingControllerFactory(os.environ.get('TASKING_QUEUE_NAME'), "1")

def _on_new_message(record):
    try:
        message = json.loads(record.body)
    except Exception:
        traceback.print_exc()

    t = message.get('type')
    logger.debug(f'// [{t}] Processing Start')
    if t == 'posthog_track':
        user_id = message.get('user_id')
        event_type = message.get('event_type')
        event_attributes = message.get('event_attributes')
        posthog.capture(user_id, event_type, properties=event_attributes)
        posthog.flush()
    elif t == 'posthog_register':
        user_id = message.get('user_id')
        event_attributes = message.get('event_attributes')
        posthog.identify(user_id, properties=event_attributes)
        posthog.flush()
    elif t == 'posthog_link':
        user_id = message.get('user_id')
        distinct_id = message.get('distinct_id')
        posthog.alias(user_id, distinct_id)
        posthog.flush()

    logger.debug(f'// [{t}] Processing End')

tasking.set_message_handler(_on_new_message)


def register(user_id, event_attributes=None):
    if not posthog.disabled:
        if event_attributes:
            event_attributes = {}
        s = json.dumps({'type': 'posthog_register',
                        'user_id': user_id,
                        'event_attributes': event_attributes})
        tasking.post(s)

def track(user_id, event_type, event_attributes=None):
    if not posthog.disabled:
        if event_attributes:
            event_attributes = {}
        s = json.dumps({'type': 'posthog_track',
                        'user_id': user_id,
                        'event_type': event_type,
                        'event_attributes': event_attributes})
        tasking.post(s)

def link(user_id, distinct_id):
    if not posthog.disabled:
        s = json.dumps({'type': 'posthog_link',
                        'user_id': user_id,
                        'distinct_id': distinct_id})
        tasking.post(s)
```

7.3 Request & Reponse Limits

API Gateway has a request & response filesize limit of 10MB. This restriction does not apply for local testing.

7.4 Cold Starts

Chalice based websites are vulnerable to the Lambda Coldstart problem. A cold start happens when you execute an inactive function. The delay comes from AWS provisioning your selected runtime container and then running your function. Functions stay 'warm' for approximately 5 minutes, which means they respond to requests much quicker. After this period of time, the contianer is dropped by AWS and a cold start needs to happen.

PYTHON MODULE INDEX

C

chalice_http_toolkit.cloudfront, 17
chalice_http_toolkit.content, 17
chalice_http_toolkit.multipart, 8
chalice_http_toolkit.response_with_binary, 9
chalice_http_toolkit.tasking, 18

INDEX

A

ACCEPTS2PIL (*chalice_http_toolkit.content.ContentManager attribute*), 6
asset() (*chalice_http_toolkit.cloudfront.CloudFront method*), 5
asset() (*chalice_http_toolkit.content.ContentManager method*), 7

C

chalice_http_toolkit.cloudfront module, 5, 17
chalice_http_toolkit.content module, 6, 17
chalice_http_toolkit.multipart module, 8
chalice_http_toolkit.response_with_binary module, 9
chalice_http_toolkit.tasking module, 18
CloudFront (*class in chalice_http_toolkit.cloudfront*), 5, 17
ContentManager (*class in chalice_http_toolkit.content*), 6, 17
convert_by_accepts() (*chalice_http_toolkit.content.ContentManager method*), 7

E

ex_multipart() (*in module chalice_http_toolkit.multipart*), 8

H

html() (*chalice_http_toolkit.content.ContentManager method*), 7

I

isBase64Encoded (*chalice_http_toolkit.response_with_binary.ResponseWithBinary attribute*), 9
isLocal (*chalice_http_toolkit.response_with_binary.ResponseWithBinary attribute*), 9

J

json() (*chalice_http_toolkit.content.ContentManager method*), 7

M

module
chalice_http_toolkit.cloudfront, 5, 17
chalice_http_toolkit.content, 6, 17
chalice_http_toolkit.multipart, 8
chalice_http_toolkit.response_with_binary, 9
chalice_http_toolkit.tasking, 18

R

redirect() (*chalice_http_toolkit.content.ContentManager method*), 7
render_template() (*chalice_http_toolkit.content.ContentManager method*), 8
ResponseWithBinary (*class in chalice_http_toolkit.response_with_binary*), 9

S

set_static_handler_prefix() (*chalice_http_toolkit.content.ContentManager method*), 8
static() (*chalice_http_toolkit.cloudfront.CloudFront method*), 6
static() (*chalice_http_toolkit.content.ContentManager method*), 8

T

TaskingController (*class in chalice_http_toolkit.tasking*), 18
template() (*chalice_http_toolkit.cloudfront.CloudFront method*), 6
text() (*chalice_http_toolkit.response_with_binary.ResponseWithBinary method*), 9

X

xml() (*chalice_http_toolkit.content.ContentManager method*), 8