AWSimple

A Simple API for Basic AWS Services

James Abel

SF Python Meetup

June 16, 2021

j@abel.co   @jamesabel   www.abel.co

AWSimple

- AWS's boto3 is awesome, but can be overly complex for more straight forward use cases
- AWSimple is a Python package published to PyPI
  - `pip install awsimple`
- Simple Object-Oriented API for AWS S3, DynamoDB, and SNS/SQS
  - S3 : Simple Storage Service
  - DynamoDB : NoSQL database
  - SNS : Simple Notification Service
  - SQS : Simple Queuing Service
- Targets "serverless" services
  - AWS managed
  - On-demand, automatically scales
  - Can be inexpensive or perhaps in free tier (***check your own usage***)
- "blob" storage + NoSQL DB + notification + queueing can be a powerful combination
- Flexible IAM (Identity Access Management)
  - `.test()` methods for access (IAM) debug
- Uses and works with AWS's boto3. Can still utilize boto3 for things AWSimple doesn't provide.
- AWSimple adds:
  - Caching
  - File Hashing
  - Pagination
- In this presentation I'll give a high-level description of the APIs (classes) and touch on some basic examples

S3 Write Example

```
# AWSimple contains a collection of classes for AWS access.
# S3Access is the class to use for AWS S3.
from awsimple import S3Access

s3_access = S3Access(“mybucket")
s3_access.create_bucket()  # OK to already exist
# write “hello world” to S3 object mybucket/helloworld.txt
s3_access.write_string(“hello world”, “helloworld.txt”)
```

S3 Read Example

```
from awsimple import S3Access

s3_access = S3Access("mybucket")
s = s3_access.read_string("helloworld.txt")
print(s)  # hello world
```

S3 Cached Download

```
from pathlib import Path
from awsimple import S3Access

file_name = "helloworld.txt“  # also S3 key
destination = Path(“mylocaldir”, file_name)
s3_access = S3Access("mybucket")
s3_access.download_cached(file_name, destination)
```

True hashing (SHA512) for content-based file caching
(good for big files)

Additional S3Access methods

```
bucket_exists() - test if S3 bucket exists

bucket_list() - list out all buckets in an account

create_bucket() - create S3 bucket

delete_bucket() - delete S3 bucket

delete_object(s3_key: str) - delete an S3 object

dir() - a “directory” of an S3 bucket

download(s3_key: str, dest_path: Union[str, pathlib.Path]) - download an S3 object (no caching)

get_s3_object_metadata(s3_key: str) - get S3 object metadata

get_s3_object_url(s3_key: str) - get S3 object URL

object_exists(s3_key: str) - determine if an s3 object exists

read_lines(s3_key: str) - read contents of an S3 object as a list of strings

set_public_readable(public_readable: bool) – set bucket and object creation as public readable

upload(file_path: Union[str, pathlib.Path], s3_key: str, force=False) - upload a file to an S3
object

write_lines(input_lines: List[str], s3_key: str) - write a list of strings to an S3 bucket
```

DynamoDB

- NoSQL “document” database
- Serverless (can be on-demand)
- Each item in a table requires a unique Primary Key
  - Defined at table creation
  - Partition (hash) Key

    or
  - Partition (hash) + Sort (range) Key combination
- Optional Secondary Indexes can be added (for speed and efficiency)
- Entire table can be “dumped” via a Scan
  - AWSimple provides a cached table scan (for static or slow-changing tables)

DynamoDBAccess

- Simple interface into DynamoDB tables
- Converts Python dicts to/from DynamoDB compatible types
  - Deals with Decimal ⬄ int/float, bytes/bytearray ⬄ str, etc.
    - Not always necessary
  - DynamoDB item → regular JSON
- Simple queries
- Cached table scan (for static or slowly changing tables)
  - Uses table item count (updated in AWS every ~6 hours) in caching protocol

DynamoDB Example

```
from awsimple import DynamoDBAccess

dynamodb_access = DynamoDBAccess("users_example")
# use email as a partition key in our primary key (no sort key)
dynamodb_access.create_table("email")

dynamodb_access.put_item({"email": "victor@victorwooten.com", "first_name": "Victor", "last_name": "Wooten"})

# Add "middle_name" in a new key/value pair. This is a feature of NoSQL - no database migration needed.
dynamodb_access.put_item({"email": "john@ledzeppelin.com", "first_name": "John", "middle_name": "Paul",
"last_name": "Jones"})

# look up user info for one of our users
# this is a "get" since we're using a key and will always get back exactly one item
user_info = dynamodb_access.get_item("email", "john@ledzeppelin.com")
```

Additional DynamoDBAccess methods

```
delete_all_items() - delete all the items in a table.

delete_item(partition_key: str, partition_value: Union[str, int], sort_key: Optional[str] = None, sort_value: Optional[Union[str, int]] = None) -
delete table item

delete_table() - deletes the current table (e.g. “drop table”)

get_item(partition_key: str, partition_value: Union[str, int], sort_key: Optional[str] = None, sort_value: Optional[Union[str, int]] = None) - get a
DB item

get_primary_keys() - get the table’s primary keys

get_table_names() - get all DynamoDB tables for this AWS account

put_item(item: dict) - put (write) a DynamoDB table item

query(*args) - query exact match

query_begins_with(*args) - query if begins with

query_one(partition_key: str, partition_value, direction: awsimple.dynamodb.QuerySelection, secondary_index_name: str = None) - query and return one
or none items

scan_table() → returns entire table

scan_table_cached(invalidate_cache: bool = False) - read data table(s) from AWS with caching

table_exists() - test if table exists

upsert_item(partition_key: str, partition_value: Union[str, int], sort_key: Optional[str] = None, sort_value: Optional[Union[str, int]] = None,
item: Optional[dict] = None) - upsert (update or insert) table item


dict_to_dynamodb(input_value: Any, convert_images: bool = True, raise_exception: bool = True) – returns a dictionary that follows AWS boto3 item
standards

dynamodb.dynamodb_to_dict(item) - convert a DynamoDB item to a serializable dict

dynamodb.dynamodb_to_json(item, indent=None) - convert a DynamoDB item to JSON
```

SNS/SQS - Serverless Notification and Queuing

- SNS Notifications
  - Topics
  - e.g. email, SMS
  - send to SQS queue
- SQS - Simple Queuing Service
  - Send, store, and receive messages between software components
    - Microservice communication
    - Task management
    - Workload distribution
    - Scheduling
    - … etc.
  - Short poll when expecting a message
  - Long poll for “waiters”

SNS/SQS example

```
from awsimple import SNSAccess, SQSPollAccess

# creation
sqs_access = SQSPollAccess("myqueue")
sqs_access.create_queue()
sns_access = SNSAccess("mytopic")
sns_access.create_topic()
sns_access.subscribe(sqs_access)  # subscribe the SQS queue to the SNS topic

# usage
sns_access.publish(“my message", “my subject")  # will end up in SQS queue
message = json.loads(sqs_access.receive_message().message)
print(message["Message"])  # “my message"
```

AWSimple and AWS’s IAM

- AWSimple supports profile and access key/secret access key pair
  - `S3Access(“mybucket”, profile_name=“myprofile”)`

    or
  - `S3Access(“mybucket”, aws_access_key_id=“myaccesskey”, aws_secret_access_key=“mysecretaccesskey”)`
- ***<u>Key management is the responsibility of the user</u>***
- Profiles are kept in files at ~/.aws
  - credentials - keys
  - config – e.g. AWS region
- If passing access keys in directly to AWSimple classes, use a ***<u>secure</u>*** key access method
- The AWS `region` can also be specified when instantiating an AWSimple access class
- You may want to sub-class the AWSimple access classes to support your IAM method

```
class MyS3Access(S3Access):
    def __init__(self, bucket: str, **kwargs):
        super().__init__(bucket, profile_name=“myprofile”, **kwargs)
```

- Application Usage
  - BUP (Windows local backup for S3, DynamoDB and github)
    - https://github.com/jamesabel/bup
  - PyShip (Python freezer/installer)
    - https://github.com/jamesabel/pyship
  - Proprietary applications/libraries
  - … could be you!
- Featured on PythonBytes Podcast Episode #224
- Repo: https://github.com/jamesabel/awsimple
- Docs: https://awsimple.readthedocs.io/

BACKUP