# pydantic-sqs

Andrew Herrington

Nov 07, 2022

# CONTENTS

Convert your pydantic models to and from AWS SQS messages. Send and receive AWS SQS messages directly to pydantic objects.

CHAPTER

# ONE

# DEPENDENCIES

- Python +3.7
- pydantic
- aiobotocore

## 1.1 Quick Start

### 1.1.1 Examples

Examples are in the examples/ directory of this repo.

### 1.1.2 Installation

Install the package

```
pip install pydantic-sqs
```

### 1.1.3 Quick Usage

Import the `Store`, the `RedisConfig` and the `Model` classes.

Store and RedisConfig let you configure and customize the connection to your redis instance. Model is the base class for your ORM models.

```python
from pydantic_sqs import RedisConfig, Model, Store

# Create models as you would create pydantic models i.e. using typings
class Book(Model):
    _primary_key_field: str = 'title'
    title: str
    author: str
    published_on: date
    in_stock: bool = True

# Do note that there is no concept of relationships here
class Library(Model):
    # the _primary_key_field is mandatory
```

(continues on next page)

(continued from previous page)

```
    _primary_key_field: str = 'name'
    name: str
    address: str

# Create the store and register your models
store = Store(name='some_name', redis_config=RedisConfig(db=5, host='localhost',␣
↪port=6379),life_span_in_seconds=3600)
store.register_model(Book)
store.register_model(Library)

# Sample books. You can create as many as you wish anywhere in the code
books = [
    Book(title="Oliver Twist", author='Charles Dickens', published_on=date(year=1215,␣
↪month=4, day=4),
        in_stock=False),
    Book(title="Great Expectations", author='Charles Dickens', published_
↪on=date(year=1220, month=4, day=4)),
    Book(title="Jane Eyre", author='Charles Dickens', published_on=date(year=1225,␣
↪month=6, day=4), in_stock=False),
    Book(title="Wuthering Heights", author='Jane Austen', published_on=date(year=1600,␣
↪month=4, day=4)),
]
# Some library objects
libraries = [
    Library(name="The Grand Library", address="Kinogozi, Hoima, Uganda"),
    Library(name="Christian Library", address="Buhimba, Hoima, Uganda")
]

async def work_with_orm():
  # Insert them into redis
  await Book.insert(books)
  await Library.insert(libraries)

  # Select all books to view them. A list of Model instances will be returned
  all_books = await Book.select()
  print(all_books) # Will print [Book(title="Oliver Twist", author="Charles Dickens",␣
↪published_on=date(year=1215, month=4, day=4), in_stock=False), Book(...]

  # Or select some of the books
  some_books = await Book.select(ids=["Oliver Twist", "Jane Eyre"])
  print(some_books) # Will return only those two books

  # Or select some of the columns. THIS RETURNS DICTIONARIES not MODEL Instances
  # The Dictionaries have values in string form so you might need to do some extra work
  books_with_few_fields = await Book.select(columns=["author", "in_stock"])
  print(books_with_few_fields) # Will print [{"author": "'Charles Dickens", "in_stock":
↪"True"},...]

  # Update any book or library
  await Book.update(_id="Oliver Twist", data={"author": "John Doe"})

  # Delete any number of items
```

(continues on next page)

(continued from previous page)

```
  await Library.delete(ids=["The Grand Library"])
```

## 1.2 Serialization

### 1.2.1 Data in Redis

pydantic-sqs uses Redis Hashes to store data. The `` `_primary_key_field` `` of each Model is used as the key of the hash.

Because Redis only supports string values as the fields of a hash, data types have to be serialized.

### 1.2.2 Simple data types

Simple python datatypes that can be represented as a string and natively converted by pydantic are converted to strings and stored. Examples are ints, floats, strs, bools, and Nonetypes.

### 1.2.3 Complex data types

Complex data types are dumped to json with json.dumps().

Custom serialization is possible using json_default and json_object_hook.

These methods are part of the abstract model and can be overridden in your model to dump custom objects to json and then back to objects. An example is available in examples

## 1.3 Development

The Makefile has useful targets to help setup your development encironment. We suggest using pyenv to have access to multiple python versions easily.

### 1.3.1 Environment Setup

- Clone the repo and enter its root folder

```
git clone https://github.com/andrewthetechie/pydantic-sqs.git && cd pydantic-sqs
```

- Create a python 3.9 virtual environment and activate it. We suggest using pyenv to easily setup multiple python environments on multiple versions.

```
# We use the extra python version (3.6, 3.7, 3.8) for tox testing
pyenv install 3.9.7 3.6.15 3.7.12 3.8.12
pyenv virtualenv 3.9.7 python-aioredis
pyenv local python-aioredis 3.6.15 3.7.12 3.8.12
```

- Install the dependencies

  ```
  make setup
  ```

### 1.3.2 How to Run Tests

- Run the test command to run tests on only python 3.9

```
pytest
```

- Run the tox command to run all python version tests

```
tox
```

### 1.3.3 Test Requirements

Prs should always have tests to cover the change being made. Code coverage goals for this project are 100% coverage.

### 1.3.4 Code Linting

All code should pass Flake8 and be blackened. If you install and setup pre-commit (done automatically by environment setup), pre-commit will lint your code for you.

You can run the linting manually with make

```
make lint
```

#### CI

CI is run via Github Actions on all PRs and pushes to the main branch.

Releases are automatically released by Github Actions to Pypi.

## 1.4 Generated Module Documentation

This documentation is automatically generated from python docstrings.

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search