Hedwig Documentation

Release 3.1.0

Aniruddha Maru

Nov 10, 2018

Contents

Hedwig is a inter-service communication bus that works on AWS SQS/SNS, while keeping things pretty simple and straight forward. It uses json schema draft v4 for schema validation so all incoming and outgoing messages are validated against pre-defined schema.

Hedwig allows separation of concerns between consumers and publishers so your services are loosely coupled, and the contract is enforced by the schema validation. Hedwig may also be used to build asynchronous APIs.

For intra-service messaging, see Taskhawk.

Only Python 3.6+ is supported currently.

This project uses semantic versioning.

CHAPTER 1

Quickstart

1.1 Quickstart

Getting started with Hedwig is easy, but requires a few steps.

1.1.1 Installation

Install the latest hedwig release via *pip*:

```
$ pip install authedwig
```

You may also install a specific version:

```
$ pip install authedwig==1.0.0
```

The latest development version can always be found on Github.

1.1.2 Configuration

Before you can use Hedwig, you need to set up a few settings. For Django projects, simple use Django settings to configure Hedwig, for non-Django projects, you must declare an environment variable called `SETTINGS_MODULE` that points to a module where settings may be found.

Required settings are:

```
AWS_ACCESS_KEY = <YOUR AWS KEY>
AWS_ACCOUNT_ID = <YOUR AWS ACCOUNT ID>
AWS_REGION = <YOUR AWS REGION>
AWS_SECRET_KEY = <YOUR AWS SECRET KEY>

HEDWIG_QUEUE = <YOUR APP HEDWIG QUEUE>
```

(continues on next page)

(continued from previous page)

```
HEDWIG_CALLBACKS = <YOUR CALLBACKS>

HEDWIG_ROUTING = <YOUR INFRA ROUTES>

HEDWIG_SCHEMA_FILE = <PATH TO YOUR SCHEMA FILE>
```

1.1.3 Provisioning

Hedwig works on SQS and SNS as backing queues. Before you can publish/consume messages, you need to provision the required infra. This may be done manually, or, preferably, using Terraform. Hedwig provides tools to make infra configuration easier: see Terraform and Hedwig Terraform Generator for further details.

1.1.4 Fan Out

Hedwig utilizes SNS for fan-out configuration. A publisher publishes messages on a *topic*. This message may be received by zero or more consumers. The publisher needn't be aware of the consuming application at all. There are a variety of messages that may be published as such, but they generally fall into 2 buckets:

1. **Asynchronous API Requests**: Hedwig may be used to call APIs asynchronously. The contract is enforced by your infra-structure by connecting SNS topics to SQS queues, and payload is validated using the schema you define. Response is a delivered using a separate message if required.
2. **Notifications**: The most common use case is to notify other services/apps that may be interested in events. For example, your User Management app can publish a `user.created` message notification to all your apps. As publishers and consumers are loosely coupled, this separation of concerns is very effective in ensuring a stable eco-system.

1.1.5 Using Hedwig

To use hedwig, simply add a message handler like so:

```
def send_email(message: hedwig.models.Message) -> None:
    # send email
```

And then send a message:

```
message = hedwig.models.Message.new(
    hedwig.models.MessageType.send_email,
    StrictVersion('1.0'),
    {
        'to': 'example@email.com',
        'subject': 'Hello!',
    },
)
message.publish()
```

Messages are held in SQS queue until they're successfully executed, or until they fail a configurable number of times. Failed tasks are moved to a Dead Letter Queue, where they're held for 14 days, and may be examined for further debugging.

CHAPTER 2

Usage

2.1 Usage Guide

2.1.1 Callbacks

Callbacks are simple python functions that accept a single argument of type `hedwig.models.Message` -

```
def send_email(message: hedwig.models.Message) -> None:
    # send email
```

You can access the data dict using `message.data` as well as custom headers using `message.headers` and other metadata fields as described in the API docs: *hedwig.models.Message()*.

2.1.2 Publisher

You can run publish messages like so:

```
models.Message.new(MessageType.my_message, StrictVersion('1.0'), data).publish()
```

If you want to include a custom headers with the message (for example, you can include a `request_id` field for cross-application tracing), you can pass in additional parameter `headers`.

2.1.3 Consumer

A consumer for SQS based workers can be started as following:

```
consumer.listen_for_messages()
```

This is a blocking function. Don't use threads since this library is **NOT** guaranteed to be thread-safe.

A consumer for Lambda based workers can be started as following:

```
consumer.process_messages_for_lambda_consumer(lambda_event)
```

where `lambda_event` is the event provided by AWS to your Lambda function as described in lambda sns format.

2.1.4 Schema

The schema file must be a JSON-Schema draft v4 schema. There's a few more restrictions in addition to being a valid schema:

- There must be a top-level key called `schemas`. The value must be an object.
- `schemas`: The keys of this object must be message types. The value must be an object.
- `schemas/<message_type>`: The keys of this object must be major version patterns for this message type. The value must be an object.
- `schemas/<message_type>/<major_version>.*`: This object must represent the data schema for given message type, and major version. Any minor version updates must be applied in-place, and must be non-breaking per semantic versioning.

Note that the schema file only contains definitions for major versions. This is by design since minor version MUST be backwards compatible.

Optionally, a key `x-versions` may be used to list full versions under a major version.

For an example, see test hedwig schema.

2.1.5 Testing

Hedwig supports pytest by default and provides pytest testing utilities as part of the *`hedwig.testing.pytest_plugin`* module.

2.2 Configuration

Add appropriate configuration to the app. If not using a Django app, ensure that *SETTINGS_MODULE* is defined to the path of a module where all settings can be found.

AWS_REGION

AWS region

required; string

AWS_ACCOUNT_ID

AWS account id

required; string

AWS_ACCESS_KEY

AWS access key

required; string

AWS_CONNECT_TIMEOUT_S

AWS connection timeout

optional; int; default: 2

AWS_ENDPOINT_SNS

AWS endpoint for SNS. This may be used to customized AWS endpoints to assist with testing, for example, using localstack.

optional; string

AWS_ENDPOINT_SQS

AWS endpoint for SQS. This may be used to customized AWS endpoints to assist with testing, for example, using localstack.

optional; string

AWS_READ_TIMEOUT_S

AWS read timeout

optional; int; default: 2

AWS_SECRET_KEY

AWS secret key

required; string

AWS_SESSION_TOKEN

AWS session token that represents temporary credentials (for example, for Lambda apps)

optional; string

HEDWIG_DATA_VALIDATOR_CLASS

The validator class to use for schema validation. This class must be a sub-class of *hedwig.validator.MessageValidator*, and may add additional validation logic, based on pyjsonschema docs.

For example, to add a new format called `vin`, use this validator:

```
class CustomValidator(hedwig.validator.MessageValidator):
    # simplistic check: 17 alphanumeric characters except i, o, q
    _vin_re = re.compile("^[a-hj-npr-z0-9]{17}$")

    @staticmethod
    @hedwig.models.MessageValidator.checker.checks('vin')
    def check_vin(instance) -> bool:
        if not isinstance(instance, str):
            return True
        return bool(CustomValidator._vin_re.match(instance))
```

optional; fully-qualified class name

HEDWIG_DEFAULT_HEADERS

A function that may be used to inject custom headers into every message, for example, request id. This hook is called right before dispatch, and any headers that are explicitly specified when dispatching may override these headers.

If specified, it's called with the following arguments:

```
default_headers(message=message)
```

where `message` is the outgoing Message object, and its expected to return a dict of strings.

It's recommended that this function be declared with `**kwargs` so it doesn't break on new versions of the library.

optional; fully-qualified function name

HEDWIG_CALLBACKS

A dict of Hedwig callbacks, with values as callables or fully-qualified function names. The key is a tuple of message type and major version pattern of the schema.

required for consumers; `dict[tuple[string, string], string]`

HEDWIG_MESSAGE_ROUTING

A dict of Hedwig message types, with values as topic names. The key is a tuple of message type and major version pattern of the schema. An entry is required for every message type that the app wants to consumer or publish.

It's recommended that major versions of a message be published on separate topics.

required; `dict[tuple[string, string], string]`

HEDWIG_PRE_PROCESS_HOOK

A function which can used to plug into the message processing pipeline *before* any processing happens. This hook may be used to perform initializations such as set up a global request id based on message headers. If specified, this will be called with the following arguments for SQS apps:

```
pre_process_hook(sqs_queue_message=sqs_queue_message)
```

where `sqs_queue_message` is of type `boto3.sqs.Message`. And for Lambda apps as so:

```
pre_process_hook(sns_record=record)
```

where `sns_record` is a `dict` of a single record with format as described in lambda sns format.

It's recommended that this function be declared with `**kwargs` so it doesn't break on new versions of the library.

optional; fully-qualified function name

HEDWIG_POST_DESERIALIZE_HOOK

A function which can used to plug into the message processing pipeline *after* serializing from JSON succeeds. This hook may be used to modify the format over the wire. If specified, this will be called with the following arguments:

```
post_deserialize_hook(message_data=message_data)
```

where `message_data` is of type `dict`.

It's recommended that this function be declared with `**kwargs` so it doesn't break on new versions of the library.

optional; fully-qualified function name

HEDWIG_PRE_SERIALIZE_HOOK

A function which can used to plug into the message processing pipeline *before* serializing to JSON. This hook may be used to modify the format over the wire. If specified, this will be called with the following arguments:

```
pre_serialize_hook(message_data=message_data)
```

where `message_data` is of type `dict`.

It's recommended that this function be declared with `**kwargs` so it doesn't break on new versions of the library.

optional; fully-qualified function name

HEDWIG_PUBLISHER

Name of the publisher

required for publishers; string

HEDWIG_QUEUE

The name of the hedwig queue (exclude the `HEDWIG-` prefix).

required; string

HEDWIG_SCHEMA_FILE

The filepath to a JSON-Schema file representing the Hedwig schema. This json-schema must contain all messages under a top-level key `schemas`. Each message's schema must include all valid versions for that message.

required; string; filepath

HEDWIG_SYNC

Flag indicating if Hedwig should work synchronously. If set to `True` a published message will be dispatched immediately using `HEDWIG_CALLBACKS` without calling any SQS APIs. This is similar to Celery's Eager mode and is helpful for integration testing. It's assumed that your service handles the message you're dispatching in sync mode.

optional; bool; default False

2.3 API reference

`hedwig.consumer.`**`listen_for_messages`**(*num_messages=10*, *visibility_timeout_s=None*, *loop_count=None*, *shutdown_event=None*)

Starts a Hedwig listener for message types provided and calls the callback handlers like so:

callback_fn(message).

The message is explicitly deleted only if callback function ran successfully. In case of an exception the message is kept on queue and processed again. If the callback keeps failing, SQS dead letter queue mechanism kicks in and the message is moved to the dead-letter queue.

This function is blocking by default. It may be run for specific number of loops by passing *loop_count*. It may also be stopped by passing a shut down event object which can be set to stop the function.

Parameters

- **`num_messages`** (`int`) – Maximum number of messages to fetch in one SQS API call. Defaults to 10
- **`visibility_timeout_s`** (`Optional[int]`) – The number of seconds the message should remain invisible to other queue readers. Defaults to None, which is queue default
- **`loop_count`** (`Optional[int]`) – How many times to fetch messages from SQS. Default to None, which means loop forever.
- **`shutdown_event`** (`Optional[Event]`) – An event to signal that the process should shut down. This prevents more messages from being de-queued and function exits after the current messages have been processed.

Return type `None`

`hedwig.consumer.`**`process_messages_for_lambda_consumer`**(*lambda_event*)

Return type `None`

`class` `hedwig.models.`**`Message`**(*data*)

Model for Hedwig messages. All properties of a message should be considered immutable. A Message object will always have known message format schema and message format schema version even if the data _may_ not be valid.

validate()
Validates a message using JSON schema.

Raise `hedwig.ValidationError` if validation fails.

Return type `None`

classmethod new(*msg_type*, *data_schema_version*, *data*, *msg_id=None*, *headers=None*)
Creates Message object given type, data schema version and data. This is typically used by the publisher code.

Parameters

- **msg_type** (*MessageType*) – MessageType instance
- **data_schema_version** (`StrictVersion`) – StrictVersion representing data schema
- **data** (`dict`) – The dict to pass in *data* field of Message.
- **msg_id** (`Optional[str]`) – Custom message identifier. If not passed, a randomly generated uuid will be used.
- **headers** (`Optional[dict]`) – Custom headers

Return type *Message*

publish()
Publish this message on Hedwig infra

extend_visibility_timeout(*visibility_timeout_s*)
Extends visibility timeout of a message for long running tasks.

Return type `None`

data_schema_version
StrictVersion object representing data schema version. May be *None* if message can't be validated.

Return type `StrictVersion`

id
Message identifier

Return type `str`

schema
Message schema

Return type `str`

type
MessageType. May be none if message is invalid

Return type *MessageType*

format_version
Message format version (this is different from data schema version)

Return type `StrictVersion`

metadata
Message metadata

Return type *Metadata*

timestamp
Timestamp of message creation in epoch milliseconds

Return type int

headers
Custom headers sent with the message

Return type dict

receipt
SQS receipt for the task. This may be used to extend message visibility if the task is running longer than expected using *Message.extend_visibility_timeout()*

Return type Optional[str]

publisher
Publisher of message

Return type Optional[str]

data
Message data

Return type dict

topic
The SNS topic name for routing the message

Return type str

class hedwig.models.**Metadata**(*data*)

timestamp
Timestamp of message creation in epoch milliseconds

Return type int

publisher
Publisher of message

Return type str

receipt
SQS receipt for the task. This may be used to extend message visibility if the task is running longer than expected using *Message.extend_visibility_timeout()*

Return type Optional[str]

headers
Custom headers sent with the message

Return type dict

class hedwig.models.**MessageType**
Enumeration representing the message types supported for this service. This is automatically created based on setting *HEDWIG_MESSAGE_ROUTING*

class hedwig.validator.**MessageValidator**(*schema=None*)

checker = <jsonschema._format.FormatChecker object>
FormatChecker that checks for *format* JSON-schema field. This may be customized by an app by overriding setting *HEDWIG_DATA_VALIDATOR_CLASS* and defining more format checkers.

validate(*message*)
Validates a message using JSON Schema

Return type None

hedwig.commands.**requeue_dead_letter**(*num_messages=10*, *visibility_timeout=None*)
Re-queues everything in the Hedwig DLQ back into the Hedwig queue.

Parameters

- **num_messages** (int) – Maximum number of messages to fetch in one SQS call. Defaults to 10.
- **visibility_timeout** (Optional[int]) – The number of seconds the message should remain invisible to other queue readers. Defaults to None, which is queue default

Return type None

2.3.1 Testing

hedwig.testing.pytest_plugin.**mock_hedwig_publish**()
A pytest fixture that mocks Hedwig publisher and lets you verify that your test publishes appropriate messages.

Return type Generator[*HedwigPublishMock*, None, None]

class hedwig.testing.pytest_plugin.**HedwigPublishMock**(**args*, ***kw*)
Custom mock class used by *hedwig.testing.pytest_plugin.mock_hedwig_publish()* to mock the publisher.

assert_message_not_published(*msg_type*, *data=None*, *version=StrictVersion ('1.0')*)
Helper function to check that a Hedwig message of given type, data and schema was NOT sent.

Return type None

assert_message_published(*msg_type*, *data=None*, *version=StrictVersion ('1.0')*)
Helper function to check if a Hedwig message with given type, data and schema version was sent.

Return type None

2.3.2 Exceptions

class hedwig.exceptions.**RetryException**(**args*, ***kwargs*)
Special exception that does not log an exception when it is received. This is a retryable error.

class hedwig.exceptions.**IgnoreException**
Indicates that this task should be ignored.

class hedwig.exceptions.**ValidationError**
Message failed JSON schema validation

class hedwig.exceptions.**ConfigurationError**
There was some problem with settings

class hedwig.exceptions.**CallbackNotFound**
No callback found that can handle the given message. Check your *CALLBACKS* settings.

2.4 Release Notes

Current version: v3.1.0

2.4.1 v1.0

- Initial version

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

V