arend

Jose-Maria Vazquez-Jimenez

Oct 02, 2022

CONTENTS:

A simple producer-consumer library for the Beanstalkd queue!

CHAPTER

ONE

INSTALLATION

Install it using *pip*:

```
pip install arend
```

CHAPTER

TWO

BASIC USAGE

Make sure you have the *arend* installed:

```
from arend import arend_task
from arend.backends.mongo import MongoSettings
from arend.brokers import BeanstalkdSettings
from arend.settings import ArendSettings
from arend.worker import consumer

settings = ArendSettings(
    beanstalkd=BeanstalkdSettings(host="beanstalkd", port=11300),
    backend=MongoSettings(
        mongo_connection="mongodb://user:pass@mongo:27017",
        mongo_db="db",
        mongo_collection="Tasks"
    ),
)

@arend_task(queue="my_queue", settings=settings)
def double(num: int):
    return 2 * num

double(2)  # returns 4
task = double.apply_async(args=(4,))  # It is sent to the queue

consumer(queue="my_queue", settings=settings)  # consume tasks from the queue

Task = settings.get_backend()  # you can check your backend for the result
task = Task.get(uuid=task.uuid)
assert task.result == 4
```

CHAPTER

THREE

BACKENDS

The available backends to store logs are **Mongo** and **Redis**.

CHAPTER

FOUR

SETTING YOUR BACKEND WITH ENVIRONMENT VARIABLES

You can set your backend by defining env vars. The *AREND__* prefix indicates that it belongs to the Arend:

```
# Redis
AREND__REDIS_HOST='redis'
AREND__REDIS_DB='1'
AREND__REDIS_PASSWORD='pass'
...

# Mongo
AREND__MONGO_CONNECTION='mongodb://user:pass@mongo:27017'
AREND__MONGO_DB='db'
AREND__MONGO_COLLECTION='logs'
...
```

Now in your code:

```
from arend import arend_task
from arend.worker import consumer


@arend_task(queue="my_queue")
def double(num: int):
    return 2 * num

double.apply_async(args=(4,))  # It is sent to the queue

consumer(queue="my_queue")
```

4.1 Installation

Install arend:

```
pip install arend
```

4.1.1 Advanced: local setup for development (Ubuntu)

These instructions assume that `git`, `docker`, and `docker-compose` are installed on your host machine.

First, clone this repo and make some required directories.:

```
git clone https://github.com/pyprogrammerblog/arend.git
cd arend
```

Then build the docker image:

```
docker-compose run --rm app poetry install
```

Then install dependencies:

```
docker-compose run --rm app poetry install
```

Run the tests:

```
docker-compose run app poetry run pytest
```

Then run the app and access inside the docker with the env activated:

```
docker-compose run --rm app poetry shell
```

Finally you can down the services:

```
docker-compose down
```

4.1.2 Advanced: Jupyter Notebook

Hit the command:

```
docker-compose run --rm -p 8888:8888 app poetry shell
```

Then inside the docker:

```
jupyter notebook --ip 0.0.0.0 --no-browser --allow-root
```

4.2 Arend

arend.**arend_task**(*queue: Optional[str] = None*, *priority: int = 1*, *delay: Union[timedelta, int] = 0*, *max_retries: int = 3*, *retry_backoff_factor: int = 1*, *settings: Optional[*ArendSettings*] = None*)

Register functions as arend task

Usage:

```
>>> from arend import arend_task
>>> from arend.backends.mongo import MongoSettings
>>> from arend.brokers import BeanstalkdSettings
>>> from arend.settings import ArendSettings
>>> from arend.worker import consumer
```

(continues on next page)

(continued from previous page)

```
>>>
>>> settings = ArendSettings(
>>>     beanstalkd=BeanstalkdSettings(host="beanstalkd", port=11300),
>>>     backend=MongoSettings(
>>>         mongo_connection="mongodb://user:pass@mongo:27017",
>>>         mongo_db="db",
>>>         mongo_collection="Tasks"
>>>     ),
>>> )
>>>
>>> @arend_task(queue="my_queue", settings=settings)
>>> def double(num: int):
>>>     return 2 * num
>>>
>>> double(2)  # returns 4
>>> task = double.apply_async(args=(4,))  # It is sent to the queue
>>>
>>> consumer(queue="my_queue", settings=settings)  # consume tasks
>>>
>>> Task = settings.get_backend()  # you can check your result backend
>>> task = Task.get(uuid=task.uuid)
>>> assert task.result == 4
```

By defining the backend in the environment variables:

```
>>> from arend import arend_task
>>> from arend.worker import consumer
>>>
>>>
>>> @arend_task(queue="my_queue")
>>> def double(num: int):
>>>     return 2 * num
>>>
>>> double.apply_async(args=(4,))  # It is sent to the queue
>>>
>>> consumer(queue="my_queue")  # consume tasks from the queue
```

4.3 Settings

class arend.settings.**ArendSettings**(*, *beanstalkd:* BeanstalkdSettings, *backend: Union[*MongoSettings, RedisSettings*]*, *task_max_retries: int = 3*, *task_retry_backoff_factor: int = 1*, *task_priority: int = 1*, *task_delay: int = 0*)

Defines settings for the Arend

Usage:

```
>>> from arend import arend_task
>>> from arend.backends.mongo import MongoSettings
>>> from arend.brokers import BeanstalkdSettings
>>> from arend.settings import ArendSettings
```

(continues on next page)

(continued from previous page)

```
>>> from arend.worker import consumer
>>>
>>> settings = ArendSettings(
>>>     beanstalkd=BeanstalkdSettings(host="beanstalkd", port=11300),
>>>     backend=MongoSettings(
>>>         mongo_connection="mongodb://user:pass@mongo:27017",
>>>         mongo_db="db",
>>>         mongo_collection="Tasks"
>>>     ),
>>>     task_max_retries = 3
>>>     task_retry_backoff_factor = 1
>>>     task_priority = 0
>>>     task_delay = 1
>>> )
```

get_backend() → Type[Union[*MongoTask*, *RedisTask*]]

Return a Task Backend with configuration already set

class arend.settings.**BeanstalkdSettings**(*, *host: str*, *port: int*)

Defines settings for the Beanstalkd Queue

4.4 Mongo Backend

class arend.backends.mongo.**MongoSettings**(*, *mongo_connection: str*, *mongo_db: str*, *mongo_collection: str*)

Mongo Settings. Defines settings for Mongo Backend

get_backend() → Type[*MongoTask*]

Returns a MongoTask class with settings already set

class arend.backends.mongo.**MongoTask**(*, *uuid: UUID = None*, *name: str*, *description: str = None*, *location: str = 'tasks'*, *status: str = 'SCHEDULED'*, *result: str = None*, *detail: Optional[str] = None*, *start_time: Optional[datetime] = None*, *end_time: Optional[datetime] = None*, *args: tuple = None*, *kwargs: dict = None*, *queue: str*, *delay: int = 0*, *priority: int = 1*, *count_retries: int = 0*, *max_retries: int = 3*, *retry_backoff_factor: int = 1*, *created: datetime = None*, *updated: datetime = None*)

MongoTask class. Defines the Task Model for Mongo Backend

Usage:

```
>>> from arend.backends.mongo import MongoSettings
>>>
>>> settings = MongoSettings(
>>>     mongo_connection="mongodb://user:pass@mongo:27017",
>>>     mongo_db="db",
>>>     mongo_collection="Tasks"
>>> )
>>> MongoTask = MongoSettings.get_backend()  # type: Type[MongoTask]
>>> task = MongoTask(name="My task", ...)
>>> task.save()
```

(continues on next page)

(continued from previous page)

```
>>>
>>> assert task.dict() == {"name": "My task", ...}
>>> assert task.json() == '{"name": "My task", ...}'
>>>
>>> task = MongoTask.get(uuid=UUID("<your-uuid>"))
>>> assert task.description == "A cool task"
>>>
>>> assert task.delete() == 1
>>> assert task.delete() == 0
```

delete() → int

Deletes object in DataBase

Usage:

```
>>> assert task.delete() == 1  # count deleted 1
>>> assert task.delete() == 0  # count deleted 0
```

classmethod get(*uuid: UUID*) → Optional[*MongoTask*]

Get object from DataBase

Usage:

```
>>> task = MongoTask.get(uuid=UUID("<your-uuid>"))
>>> assert task.uuid == UUID("<your-uuid>")
```

save() → *MongoTask*

Updates/Creates object in DataBase

Usage:

```
>>> task = MongoTask(name="My Task")
>>> task.save()
>>> task.description = "A new description"
>>> task.save()
```

class arend.backends.mongo.**MongoTasks**(*, *tasks: List[*MongoTask*] = None*, *count: int = 0*)

Defines the MongoTasks collection

4.5 Redis Backend

class arend.backends.redis.**RedisSettings**(*, *redis_host: str*, *redis_port: int = 6379*, *redis_db: int = 1*, *redis_password: str*, *redis_extras: Dict = None*)

Defines settings for Redis Backend

get_backend() → Type[*RedisTask*]

Returns a RedisTask class with settings already set

class arend.backends.redis.**RedisTask**(*, *uuid: UUID = None*, *name: str*, *description: str = None*, *location: str = 'tasks'*, *status: str = 'SCHEDULED'*, *result: str = None*, *detail: Optional[str] = None*, *start_time: Optional[datetime] = None*, *end_time: Optional[datetime] = None*, *args: tuple = None*, *kwargs: dict = None*, *queue: str*, *delay: int = 0*, *priority: int = 1*, *count_retries: int = 0*, *max_retries: int = 3*, *retry_backoff_factor: int = 1*, *created: datetime = None*, *updated: datetime = None*)

RedisLog class. Defines the Log for Redis Backend

Usage:

```
>>> from arend.backends.redis import RedisSettings
>>>
>>> settings = RedisSettings(
>>>     redis_host="redis",
>>>     redis_port=6379,
>>>     redis_db="logs",
>>>     redis_password="pass"
>>> )
>>> RedisTask = RedisSettings.get_backend()  # type: Type[RedisLog]
>>> task = RedisTask(name="My task", description="A cool task")
>>> task.save()
>>>
>>> assert task.dict() == {"name": "My task", ...}
>>> assert task.json() == '{"name": "My task", ...}'
>>>
>>> task = RedisTask.get(uuid=UUID("<your-uuid>"))
>>> assert task.description == "A cool task"
>>>
>>> assert task.delete() == 1
```

delete() → int

Deletes object in DataBase

Usage:

```
>>> assert task.delete() == 1  # count deleted 1
>>> assert task.delete() == 0  # count deleted 0
```

classmethod get(*uuid: UUID*) → Optional[*RedisTask*]

Get object from DataBase

Usage:

```
>>> log = RedisTask.get(uuid=UUID("<your-uuid>"))
>>> assert log.uuid == UUID("<your-uuid>")
```

save() → *RedisTask*

Updates/Creates object in DataBase

Usage:

```
>>> task = RedisTask(name="My Task")
>>> task.save()
>>> task.description = "A new description"
>>> task.save()
```

class arend.backends.redis.**RedisTasks**(*, *tasks: List[RedisTask] = None*, *count: int = 0*)

Defines the RedisTasks collection

4.6 License

4.7 Any help

Everyone is encouraged to file bug reports, feature requests, and pull requests through GitHub.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX