Framework

# JobsRedis

Redis implementation for Hummingbird jobs framework

## Overview

Hummingbird Jobs Queue driver using RediStack.

## Setup

Currently `RediStack` is not setup to use `ServiceLifecycle`. So to ensure clean shutdown of `RediStack` you either need to use the `RedisConnectionPoolService` that is part of HummingbirdRedis or write your own `Service` type that will manage the shutdown of a `RedisConnectionPool`.

### Using HummingbirdRedis

If you choose to use `HummingbirdRedis` you can setup a JobQueue using `RediStack` as follows

```
let redisService = try RedisConnectionPoolService(
    .init(hostname: redisHost, port: 6379),
    logger: logger
)
let jobQueue = JobQueue(
    .redis(
        redisService.pool,
        configuration: .init(
            queueKey: "MyJobQueue",
            pollTime: .milliseconds(50)
```

```
        )
    ),
    numWorkers: 10,
    logger: logger
)
let serviceGroup = ServiceGroup(
    configuration: .init(
        services: [redisService, jobQueue],
        gracefulShutdownSignals: [.sigterm, .sigint],
        logger: logger
    )
)
try await serviceGroup.run()
```

The Redis job queue configuration includes two values.

- `queueKey`: Prefix to all the Redis keys used to store queues.
- `pollTime`: This is the amount of time between the last time the queue was empty and the next time the driver starts looking for pending jobs.

## Write RedisConnectionPool Service

Alternatively you can write your own `Service` to manage the lifecycle of the `RedisConnectionPool`. This basically keeps a reference to the `RedisConnectionPool` and waits for graceful shutdown. At graceful shutdown it will close the connection pool. Unfortunately `RedisConnectionPool` is not `Sendable` so we either have to add an `@unchecked Sendable` to `RedisConnectionPoolService` or import `RediStack` using `@preconcurrency`.

```
struct RedisConnectionPoolService: Service, @unchecked Sendable {
    let pool: RedisConnectionPool

    public func run() async throws {
        // Wait for graceful shutdown and ignore cancellation error
        try? await gracefulShutdown()
        // close connection pool
        let promise = self.pool.eventLoop.makePromise(of: Void.self)
        self.pool.close(promise: promise)
```

```
            return try await promise.futureResult.get()
        }
    }
```

# Additional Features

There are features specific to the Redis Job Queue implementation.

## Push Options

When pushing a job to the queue there are a number of options you can provide.

### Delaying jobs

As with all queue drivers you can add a delay before a job is processed. The job will sit in the pending queue and will not be available for processing until time has passed its delay until time.

```
// Add TestJob to the queue, but don't process it for 2 minutes
try await jobQueue.push(TestJob(), options: .init(delayUntil: .now + 120))
```

## Cancellation

The `RedisJobQueue` conforms to protocol `CancellableJobQueue`. This requires support for cancelling jobs that are in the pending queue. It adds one new function `cancel(jobID:)`. If you supply this function with the `JobID` returned by `push(_:options:)` it will remove it from the pending queue.

```
// Add TestJob to the queue and immediately cancel it
let jobID = try await jobQueue.push(TestJob(), options: .init(delayUntil: .n
try await jobQueue.cancel(jobID: jobID)
```

## Pause and Resume

The `RedisJobQueue` conforms to protocol `ResumableJobQueue`. This requires support for pausing and resuming jobs that are in the pending queue. It adds two new functions `pause(jobID:)` and `resume(jobID:)`. If you supply these function with the `JobID` returned by `push(_:options:)` you can remove from the pending queue and add them back in at a later date.

```
// Add TestJob to the queue and immediately remove it and then add it back t
let jobID = try await jobQueue.push(TestJob(), options: .init(delayUntil: .n
try await jobQueue.pause(jobID: jobID)
try await jobQueue.resume(jobID: jobID)
```

# Topics

## Job Queue

`class` **RedisJobQueue**

Redis implementation of job queue driver

## Enumerations

`enum` **RedisScriptFlush**

Script flush mode

---

# See Also

## Related Documentation

- Jobs

  Offload work your server would be doing to another server.

- JobsPostgres

  Postgres implementation for Hummingbird jobs framework

- Hummingbird

  Lightweight, modern, flexible server framework written in Swift.

## Reference

- JobsPostgres

  Postgres implementation for Hummingbird jobs framework