Redis with Node.js (ioredis)

In order to use Redis with Node.js, you need to install a Node.js Redis client. The following sections demonstrate the use of ioredis, a community-recommended Redis client for Node.js with build-in support for promises.

Another community-recommended client for Node.js developers is node_redis. Additional Node.js clients for Redis can be found under the Node.js section of the Redis Clients page.

Install ioredis

See the ioredis README file for installation instructions.

To install ioredis, run:

```bash	npm install ioredis
```

Connect to Redis

This example code creates a connection to Redis:

```javascript
const Redis = require('ioredis');
const redis = new Redis({
  host: '<hostname>',
  port: <port>,
  password: '<password>'
});
```

Replace the values in the example with the values for your Redis instance:

- `<hostname>` - The name of the host your database runs on
- `<port>` - The port that the database is running on (default: 6379)
- `<password>` - The default Redis password, if configured

⚠️ **Note:** Remember to always store passwords outside of your code, for example in environment variables.

TLS

This example shows how to configure ioredis to make a connection to Redis using TLS:
const Redis = require('ioredis');
const fs = require('fs');

const redis = new Redis({
    host: 'hostname',
    port: <port>,
    tls: {
        key: fs.readFileSync('path_to_keyfile', 'ascii'),
        cert: fs.readFileSync('path_to_certfile', 'ascii'),
        ca: [ fs.readFileSync('path_to_ca_certfile', 'ascii') ]
    }
});

Where you must provide:

- `<hostname>` - The name of the host your database runs on
- `<port>` - The port that the database is running on (default: 6379)

**ACL user and password**

Redis 6 introduced **Access Control Lists**. ACLs provide the capability to create named user accounts, each having its own password.

To connect to Redis as an ACL user, provide the user’s username and password when creating the client:

```javascript
const Redis = require('ioredis');
const redis = new Redis({
    host: 'hostname',
    port: <port>,
    username: 'username',
    password: 'password'
});
```

Make sure to replace the values in the example with the values for your Redis instance:

- `<hostname>` - The name of the host your database runs on
- `<port>` - The port that the database is running on (default: 6379)
- `<username>` - The username of the ACL user
- `<password>` - The password of the ACL user

**Note:**

`ioredis` uses the Redis `INFO` command when it creates a new connection, so your ACL user must have permission to run `INFO`.

If your ACL user does not have the required permissions, connection attempts will fail with a `NOPERM` error message:

```
[ioredis] Unhandled error event: ReplyError: NOPERM this user has no permissions to run the 'info' command or its subcommand
```

**Example code for Redis commands**

Once connected to Redis, you can read and write data with Redis command functions.

The following code snippet reads the value stored at the key `foo` and prints it:
// Open a connection to Redis
// 'foo' has been set to 'bar'.
...

client.get('foo', (err, reply) => {
    if (err) throw err;
    console.log(reply);
});

Example output:

$ node example_ioredis.js
bar

The ioredis client exposes a function for each Redis command.

The first argument is usually the Redis key to run the command against. You can also add an optional error first callback function after the other arguments.

If you do not provide a callback function, the ioredis function returns a promise:

// Open a connection to Redis
// 'foo' has been set to 'bar'.

const redisDemo = async () => {
    const reply = await redis.get('foo');
    console.log(reply);
};

redisDemo();

Example output:

$ node example_ioredis.js
bar

Updated: November 2, 2021