Redis with Node.js (node_redis)

To use Redis with Node.js, you need to install a Node.js Redis client. The following sections explain how to use node_redis, a community-recommended Redis client for Node.js.

Another community-recommended client for Node.js developers is ioredis. You can find additional Node.js clients for Redis in the Node.js section of the Redis Clients page.

Install node_redis

See the node_redis README file for installation instructions.

To install node_redis, run:

```bash
npm install redis
```

Connect to Redis

There are several ways that you can connect to Redis, each with different security considerations.

Default password

The following code creates a connection to Redis:

```javascript
const redis = require('redis');
const client = redis.createClient({
  host: '<hostname>',
  port: <port>,
  password: '<password>'
});
client.on('error', err => {
  console.log('Error ' + err);
});
```

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 password you use to access Redis, if necessary.

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

TLS

The following example demonstrates how to make a connection to Redis using TLS:
const redis = require('redis');
const fs = require('fs');

const client = redis.createClient({
  host: '<hostname>',
  port: <port>,
  tls: {
    key: fs.readFileSync('path_to_keyfile', encoding='ascii'),
    cert: fs.readFileSync('path_to_certfile', encoding='ascii'),
    ca: [ fs.readFileSync('path_to_ca_certfile', encoding='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 (also known as ACLs). ACLs allow you to create named user accounts, each having its own password.

The node_redis client doesn't currently support ACL commands or the AUTH command with a username and password. Therefore, you will need to disable the client's built in auth function and use the generic send_command function, which allows you to send any arbitrary command to Redis, to authenticate.

Example:

```javascript
const redis = require('redis');
const client = redis.createClient({
  host: '127.0.0.1',
  port: <port>
});

// Disable client's AUTH command.
client['auth'] = null;

// send_command expects a command name and array of parameters.
client.send_command('AUTH', ['<username>', '<password>']);
```

Replace the <port>, <username>, and <password> with the values for the ACL user that you are connecting as.

Example code for Redis commands

After your application connects to Redis, you can read and write data.

The following code snippet assigns the value bar to the Redis key foo, reads it back, and prints it:
client.set('foo', 'bar', (err, reply) => {
  if (err) throw err;
  console.log(reply);
});

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

Example output:

$ node example_node_redis.js
OK
bar

The node_redis client exposes a function named for each Redis command.

These functions take strings as arguments, the first of which is usually the Redis key to run the command against. You can also add an optional error first callback function after the other arguments.

Promises and async/await

To use promises and async/await with node_redis, wrap it using Bluebird's promisifyAll, as shown here:

```javascript
const redis = require('redis');
const { promisifyAll } = require('bluebird');
promisifyAll(redis);

const runApplication = async () => {
  // Connect to redis at 127.0.0.1 port 6379 no password.
  const client = redis.createClient();

  await client.setAsync('foo', 'bar');
  const fooValue = await client.getAsync('foo');
  console.log(fooValue);
};

runApplication();
```

This creates an async equivalent of each function, adding Async as a suffix.

Alternatively, you can promisify a subset of node_redis functions one at a time using native Node.js promises and util.promisify:
const redis = require('redis');
const { promisify } = require('util');

const runApplication = async () => {
    // Connect to redis at 127.0.0.1 port 6379 no password.
    const client = redis.createClient();

    const setAsync = promisify(client.set).bind(client);
    const getAsync = promisify(client.get).bind(client);

    await setAsync('foo', 'bar');
    const fooValue = await getAsync('foo');
    console.log(fooValue);
};

runApplication();

Updated: November 2, 2021