**Bulk Loading Postgres with Node.js**

The fastest way to bulk load data into Postgres is to call [Copy](https://www.postgresql.org/docs/9.5/static/sql-copy.html), which is a SQL command to load data into a table from a flat file. To connect to Postgres with Node.js, we can use the [node-postgres](https://node-postgres.com/features/connecting) module (pg). To use the copy function, we can use the [pg-copy-streams](https://github.com/brianc/node-pg-copy-streams) module, which enables you to execute the copy function from a file readable stream.

We will first check out how to load the table and then create the code that does truncate & load.

**Setup**

Let’s set it up by creating a project and installing required modules.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

mkdir node-pg-test

cd node-pg-test

npm init

npm i pg -ES

npm i pg-copy-streams -ES

[/cc]

**config.json**

For the connection details, let’s use config.json and import the connection from the config file.

[cc lang="text" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

{

"host": "your host",

"user": "your username",

"pw": "your password",

"db": "your database name",

"port": "port, 5432 is default"

}

[/cc]

**Import modules and define global variables**

We import the required modules. Then, we define the input file path and table name. We can import the connection details from config.json file located in the same folder.

**Load Table**

Let’s create the readable stream and pipe the copy function.

**Truncate Table**

Before doing the classic truncate and load ingestion pattern, let’s review the code for truncating table.

**Truncate & Load**

Let’s put them all together to do truncate & load. Remember that the functions we are using are asynchronous. To ensure sequential execution of the code, we are using callback. With callback, we can make sure that the copy function only gets executed after the truncate statement was successfully executed.