Working with Streams & Buffers

In this lab, you will learn to use Node’s Stream & Buffer classes to read, compress & write data.

# Objectives

In this lab, you will learn to

* leverage Node's ability to stream & manipulate data via pipes.

# Compress a file via streams

In the first part of this lab, we're going to compress a file using streams.

## Write the main compression logic

1. Create a file called file.js and in it, require the modules fs and zlib.

## Create a file reading stream

2. Use fs's createReadStream function to create a stream.Readable that reads from our source file, file.js. Make sure to hold on to the reference returned.

## Create a duplex compression stream

3. Next, create a gzip compression stream via zlib's createGzip() function. Note that this is a Duplex stream, meaning it is both Readable and Writable.

## Create a file writing stream

4. One more stream that we'll need is a Writable stream for our output file. Use fs's createWriteStream function to create a writable file stream to a new file, file.js.gz.

## Pipe from the file to the output stream via the compression stream

5. With all of our streams now set up, we can pipe from the read stream to the compression stream, then on to the write stream. Write a chaining pipe expression that begins with our file reading stream, goes through our compression stream, and finishes with our writable stream and execute file.js via node.

## Confirm your results

6. Confirm your results by checking the newly created gzip file file.js.gz. If you have gzip in your path, a simple gzip -t file.js.gz will suffice; if it returns with no error, then the file is a valid gzip archive, otherwise it'll echo an error.

When you've confirmed that your archive is being created properly, move on to the next step.

# Compress URL stream

The next part of this lab is optional and requires Internet access. If you have it, continue on. If not, you're done with this lab!

## Add required modules

7. Open file net.js and require the following modules: fs, zlib, & https.

## Instantiate streams

8. Instantiate two streams: one for compression via zlib.createGzip(), and one for writing to a file (where we'll write the response we'll be getting from the URL).

## Invoke the URL

9. Use the https module's request function, whose first argument is an object with properties hostname, path & method (among others – check out the docs if you're curious), and whose second argument is a callback that takes a response object. Use 'www.google.com' for the hostname, and '/search?q=node.js' for the path, and 'GET' for the method.

Capture the returned object and make sure to call its end() method so that the server knows that the request is complete and it should start responding. Alternatively, you could just call end() on the return value of https.request, without even storing the variable, since we're not going to use it after the end() call.

## Chain the response

10. In the event handler, take the response stream, pipe it to the compression stream, and then pipe that to the writable file stream.

## Run it & test

11. Now that you've got the code written, run net.js via node, extract the newly created google.gz file, and make sure that it's innards look like what Google would return in a query for node.js.

Once you see the results you expect, you're done with this lab!