CAB432 Cloud Computing
Assessable Prac I: Persistence

This prac is concerned with persistence in an application at two different levels. The application is initially very simple, and even when we are finished it will not be too complex. Our main goal is to give you practical experience of working with a cloud based storage service and in managing the interactions between different storage types in the application. Fundamental to this work is understanding the idea of a cache and a cache miss, and actually writing code to manage the process.

We will work with Redis (https://redis.io/) for cache storage, and with S3 for longer term storage. Both of these are object stores. The essential idea is that the app will grab resources from the web, and store them – in both long-term storage and in cache. The details vary, but we can summarise the most important concepts in a few points:

- We search the 'database' based on a particular query or key
- We first check to see if the result is in the cache
- If the result is present:
    - We serve the result from the cache and save some time and bandwidth
- If the result isn't present:
    - We grab the result from long term storage or some external site
    - We serve the result
    - We store the same object in the Redis cache

The Scenario and the Local Cache:

In this exercise, we will work with a basic express application that searches Wikipedia based on a query string and returns the page if there is a match. We will work through two different versions of this app:

- The initial version will use a local installation of Redis on the instance or on your development machine. If there is a cache miss, it will grab the result from Wikipedia. The example is based on the code from https://www.codementor.io/brainyfarm/caching-with-redis-node-js-example-h6o9ii72i
- We will then supplement the application with a higher latency service (S3) and implement an additional endpoint to allow us to store directly to S3 without considering the cache.
- Finally, I will ask you to finish the app based on the logic provided later in the document.

There are a few artificial aspects in this approach, but you will gain experience of the different levels of persistence and you will be able to adapt this code to use in your assignment. We would normally 'prime' the in-memory cache (Redis) with a strategic selection of common requests drawn from the larger and slower storage service (for example S3), and replace the local Redis service with one drawn from the Cloud providers. We will give additional guidance on replacing the local Redis with a cloud service, but we will not require it as part of this prac. There are a lot of configuration choices and so we will keep this more straightforward and explain those once you have completed this task.

The initial version:
The local version of Redis is sufficient for us to understand the basics. There is a reduction in latency and in bandwidth usage. For the purposes of this exercise and to assist you during development, you will need a local Redis service. This is covered in the Appendix at the end of the document, for Windows and Ubuntu 18.04. We assume at this stage that the Redis service is bound to localhost and is listening on port 6379. You may treat the Redis connection code as boilerplate.

The application is very similar to those you have worked with before, though we have stripped away most of the usual express middleware. We shall initially expose only a simple search route and we will not bother with a separate router.

The most obvious differences are at the head of the file:

```
const express = require('express');
const responseTime = require('response-time')
const axios = require('axios');
const redis = require('redis');

const app = express();

// This section will change for Cloud Services
const redisClient = redis.createClient();

redisClient.on('error', (err) => {
  console.log("Error " + err);
});
```

The Redis client will by default talk to the local Redis server on 127.0.0.1:6379. This of course is a poor choice for a cloud application, and ultimately you will need to replace it with proper cloud services. For now, let us explore the rest of the app.

We expose a simple route which takes a query parameter. We strip that away and then we use it to construct a Wikipedia URL and as a key in the Redis cache. If we have seen this key before, we will find the result object in the cache. If we have not, then we will need to use the URL to make a get request to Wikipedia. We serve this JSON, but we then also save it to Redis.

```
// Used for header info later.
app.use(responseTime());
app.get('/api/search', (req, res) => {
  const query = (req.query.query).trim();

  // Construct the wiki URL and key
  const searchUrl = `https://en.wikipedia.org/w/api.php?action=parse&format=json&section=0&page=${query}`;
  const redisKey = `wikipedia:${query}`;

   // Try the cache
  return redisClient.get(redisKey, (err, result) => {

    if (result) {
      // Serve from Cache
      const resultJSON = JSON.parse(result);
      return res.status(200).json(resultJSON);

    } else {
      // Serve from Wikipedia API and store in cache
      return axios.get(searchUrl)
        .then(response => {
          const responseJSON = response.data;
          redisClient.setex(redisKey, 3600, JSON.stringify({ source: 'Redis Cache', ...responseJSON, }));
          return res.status(200).json({ source: 'Wikipedia API', ...responseJSON, });
        })
        .catch(err => {
          return res.json(err);
        });
    }
  });
});

app.listen(3000, () => {
  console.log('Server listening on port: ', 3000);
});
```

As usual, install the necessary modules and then run the app locally or on a cloud instance. There are obvious examples to use. If the query is Redis, then our URL will have the form:

http://localhost:3000/api/search?query=Redis

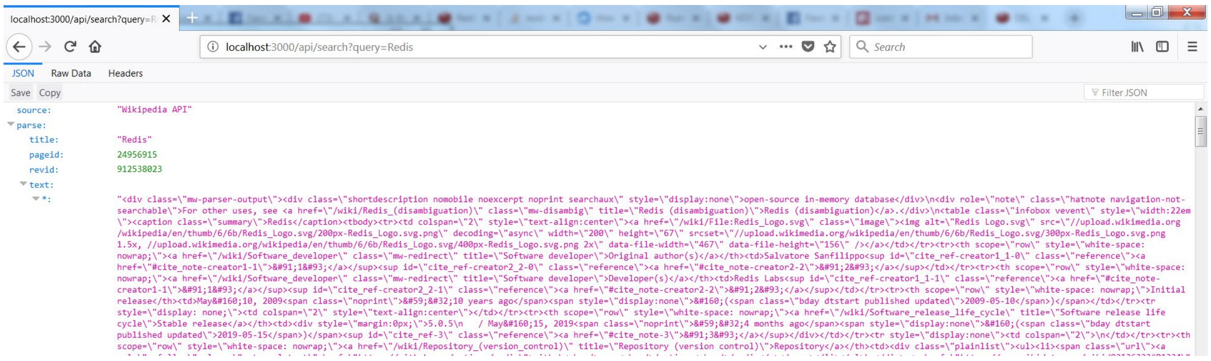
The wiki link we are targeting will have the form:

https://en.m.wikipedia.org/wiki/Redis

And the key will have the form:

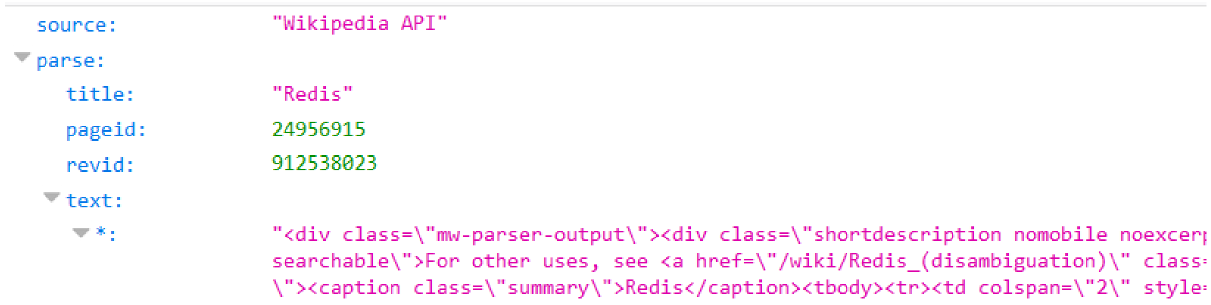
Wikipedia:Redis

Probably the most important thing to note here is the storage of the stringified JSON object in the cache. In this app we do not log this to the console but instead add a source field to the JSON so that the origin of the data is included in the response. Here this is either the “Wikipedia API” or the “Redis Cache”.
The first image shows the Wiki GET for the query ‘Redis’:


localhost:3000/api/search?query=R
localhost:3000/api/search?query=Redis
Search
JSON
Raw Data
Headers
Save
Copy
Filter JSON

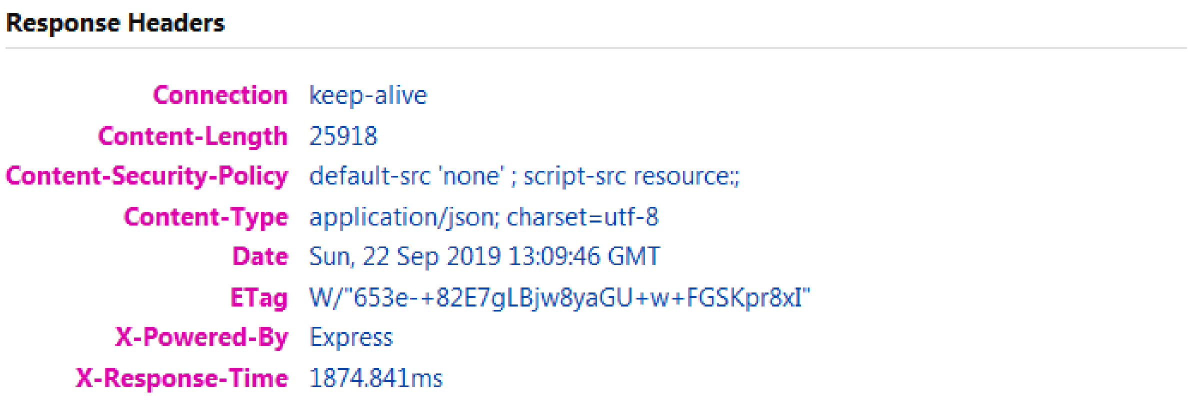
```
source:          "Wikipedia API"
▼ parse:
    title:       "Redis"
    pageid:      24956915
    revid:       912538023
  ▼ text:
    ▼ *:         "<div class=\"mw-parser-output\"><div class=\"shortdescription nomobile noexcerpt noprint searchaux\" style=\"display:none\">open-source in-memory database</div>\n<div role=\"note\" class=\"hatnote navigation-not-
                 searchable\">For other uses, see <a href=\"/wiki/Redis_(disambiguation)\" class=\"mw-disambig\" title=\"Redis (disambiguation)\">Redis (disambiguation)</a>.</div>\n<table class=\"infobox vevent\" style=\"width:22em
                 \"><caption class=\"summary\">Redis</caption><tbody><tr><td colspan=\"2\" style=\"text-align:center\"><a href=\"/wiki/File:Redis_Logo.svg\" class=\"image\"><img alt=\"Redis Logo.svg\" src=\"//upload.wikimedia.org
                 /wikipedia/en/thumb/6/6b/Redis_Logo.svg/200px-Redis_Logo.svg.png\" decoding=\"async\" width=\"200\" height=\"67\" srcset=\"//upload.wikimedia.org/wikipedia/en/thumb/6/6b/Redis_Logo.svg/300px-Redis_Logo.svg.png
                 1.5x, //upload.wikimedia.org/wikipedia/en/thumb/6/6b/Redis_Logo.svg/400px-Redis_Logo.svg.png 2x\" data-file-width=\"467\" data-file-height=\"156\" /></a></td></tr><tr><th scope=\"row\" style=\"white-space:
                 nowrap;\"><a href=\"/wiki/Software_developer\" class=\"mw-redirect\" title=\"Software developer\">Original author(s)</a></th><td>Salvatore Sanfilippo<sup id=\"cite_ref-creator1_1-0\" class=\"reference\"><a
                 href=\"#cite_note-creator1-1\">&#91;1&#93;</a></sup><sup id=\"cite_ref-creator2_2-0\" class=\"reference\"><a href=\"#cite_note-creator2-2\">&#91;2&#93;</a></sup></td></tr><tr><th scope=\"row\" style=\"white-space:
                 nowrap;\"><a href=\"/wiki/Software_developer\" class=\"mw-redirect\" title=\"Software developer\">Developer(s)</a></th><td>Redis Labs<sup id=\"cite_ref-creator1_1-1\" class=\"reference\"><a href=\"#cite_note-
                 creator1-1\">&#91;1&#93;</a></sup><sup id=\"cite_ref-creator2_2-1\" class=\"reference\"><a href=\"#cite_note-creator2-2\">&#91;2&#93;</a></sup></td></tr><tr><th scope=\"row\" style=\"white-space: nowrap;\">Initial
                 release</th><td>May&#160;10, 2009<span class=\"noprint\">&#59;&#32;10 years ago</span><span style=\"display:none\">&#160;(<span class=\"bday dtstart published updated\">2009-05-10</span>)</span></td></tr><tr
                 style=\"display: none;\"><td colspan=\"2\" style=\"text-align:center\"></td></tr><tr><th scope=\"row\" style=\"white-space: nowrap;\"><a href=\"/wiki/Software_release_life_cycle\" title=\"Software release life
                 cycle\">Stable release</a></th><td><div style=\"margin:0px;\">5.0.5\n   / May&#160;15, 2019<span class=\"noprint\">&#59;&#32;4 months ago</span><span style=\"display:none\">&#160;(<span class=\"bday dtstart
                 published updated\">2019-05-15</span>)</span><sup id=\"cite_ref-3\" class=\"reference\"><a href=\"#cite_note-3\">&#91;3&#93;</a></sup></div></td></tr><tr style=\"display:none\"><td colspan=\"2\">\n</td></tr><tr><th
                 scope=\"row\" style=\"white-space: nowrap;\"><a href=\"/wiki/Repository_(version_control)\" title=\"Repository (version control)\">Repository</a></th><td><div class=\"plainlist\"><ul><li><span class=\"url\"><a
```

Note the source in the JSON:



```
source:          "Wikipedia API"
▼ parse:
    title:       "Redis"
    pageid:      24956915
    revid:       912538023
  ▼ text:
    ▼ *:         "<div class=\"mw-parser-output\"><div class=\"shortdescription nomobile noexcer
                 searchable\">For other uses, see <a href=\"/wiki/Redis_(disambiguation)\" class=
                 \"><caption class=\"summary\">Redis</caption><tbody><tr><td colspan=\"2\" style=
```

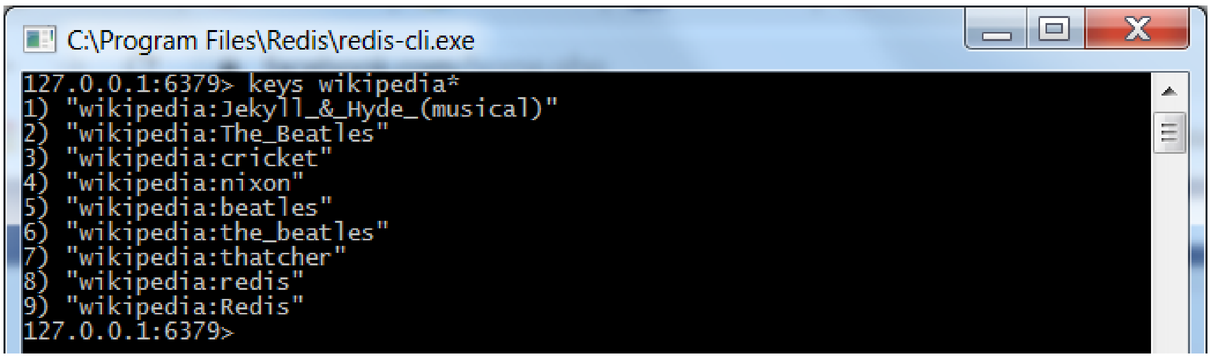
And then when we look at the headers, we see the response has taken nearly two seconds:



Response Headers

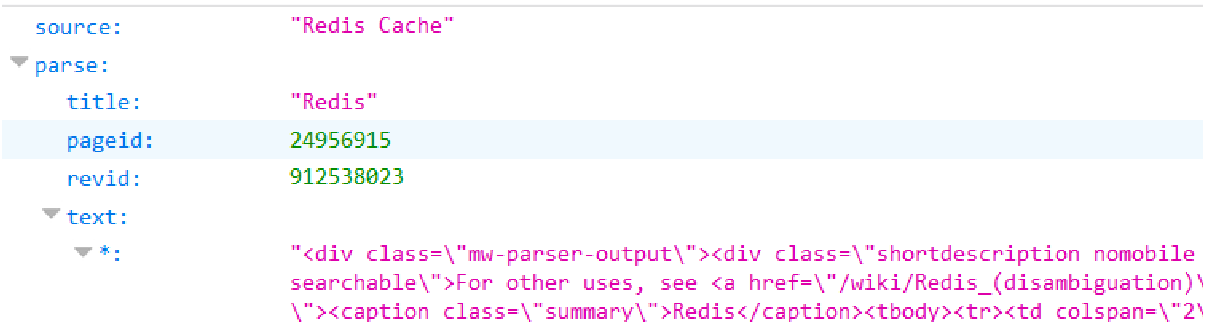
```
             Connection  keep-alive
         Content-Length  25918
Content-Security-Policy  default-src 'none' ; script-src resource:;
           Content-Type  application/json; charset=utf-8
                   Date  Sun, 22 Sep 2019 13:09:46 GMT
                   ETag  W/"653e-+82E7gLBjw8yaGU+w+FGSKpr8xI"
           X-Powered-By  Express
        X-Response-Time  1874.841ms
```

We can see the entries in the cache via the Redis CLI. This is the windows version. Note that the keys are case sensitive. For some queries Wikipedia will give the same response for incorrect casing, and in others it will not.



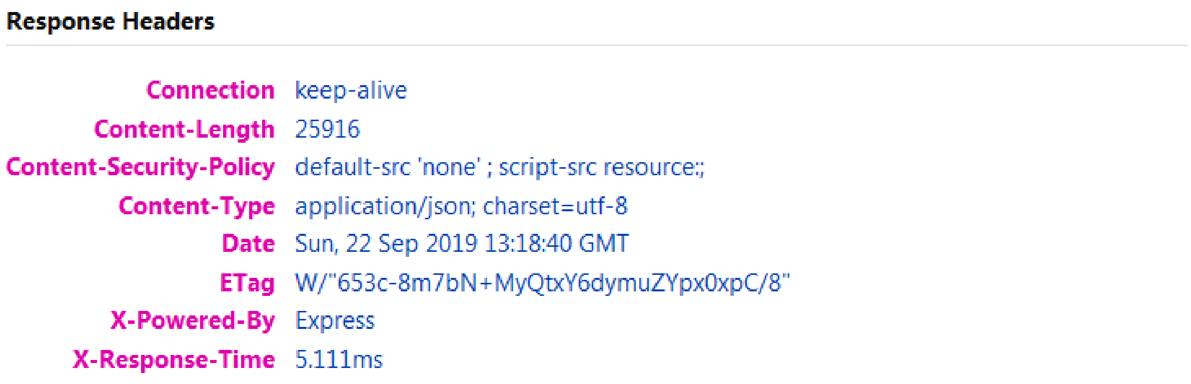
```
127.0.0.1:6379> keys wikipedia*
1) "wikipedia:Jekyll_&_Hyde_(musical)"
2) "wikipedia:The_Beatles"
3) "wikipedia:cricket"
4) "wikipedia:nixon"
5) "wikipedia:beatles"
6) "wikipedia:the_beatles"
7) "wikipedia:thatcher"
8) "wikipedia:redis"
9) "wikipedia:Redis"
127.0.0.1:6379>
```

If we repeat the same query, then we successfully hit the cache:



```
source:          "Redis Cache"
parse:
  title:         "Redis"
  pageid:        24956915
  revid:         912538023
  text:
    *:           "<div class=\"mw-parser-output\"><div class=\"shortdescription nomobile
                 searchable\">For other uses, see <a href=\"/wiki/Redis_(disambiguation)\
                 \"><caption class=\"summary\">Redis</caption><tbody><tr><td colspan=\"2\
```

And then we again check the headers to see that the response took only 5 milliseconds:



Response Headers

Connection	keep-alive
Content-Length	25916
Content-Security-Policy	default-src 'none' ; script-src resource:;
Content-Type	application/json; charset=utf-8
Date	Sun, 22 Sep 2019 13:18:40 GMT
ETag	W/"653c-8m7bN+MyQtxY6dymuZYpx0xpC/8"
X-Powered-By	Express
X-Response-Time	5.111ms

So far so good. As noted, our application is hardly stateless in its present form, and you will ultimately use a caching service from the cloud vendors. Here we will get experience of programmatic access to cloud services through S3. We will work with the node API, and while we are using AWS, it could just as easily be Azure or Google CP.

Working in the Cloud:

S3 – the Simple Storage Service – was one of the original AWS services. As you know, it allows anyone with an AWS account to store data in 'buckets'. We have seen how this works in the lectures, and here we will follow a process similar to that seen in an earlier node video, accessing it programmatically using the AWS node API. Our approach will be a little unrealistic as we would normally set this up with a substantial amount of data. Our learning objective in this exercise is to allow you to consider data at three levels:

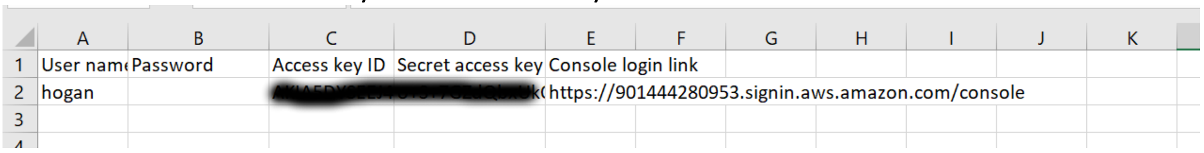
- Data which is not stored by the application and must be retrieved from the web
- Data which is held in long-term storage (S3) but not in cache
- Data which is in the cache and in long-term storage.

So we will introduce a second route on the application called `/api/store` – this will take the parameter 'key' and you should use exactly the same text as for the query. The effect of this route will be to create the wiki URL and retrieve the data as before, and to store the result only in S3. We will then modify the search code to check in sequence:

- Do we have the data in the cache? If so, serve it from cache.
- If not, do we have the data in S3? If so, serve it from S3 and store it in the cache.
- If not, we retrieve from the web and store it in *both* S3 and the cache.

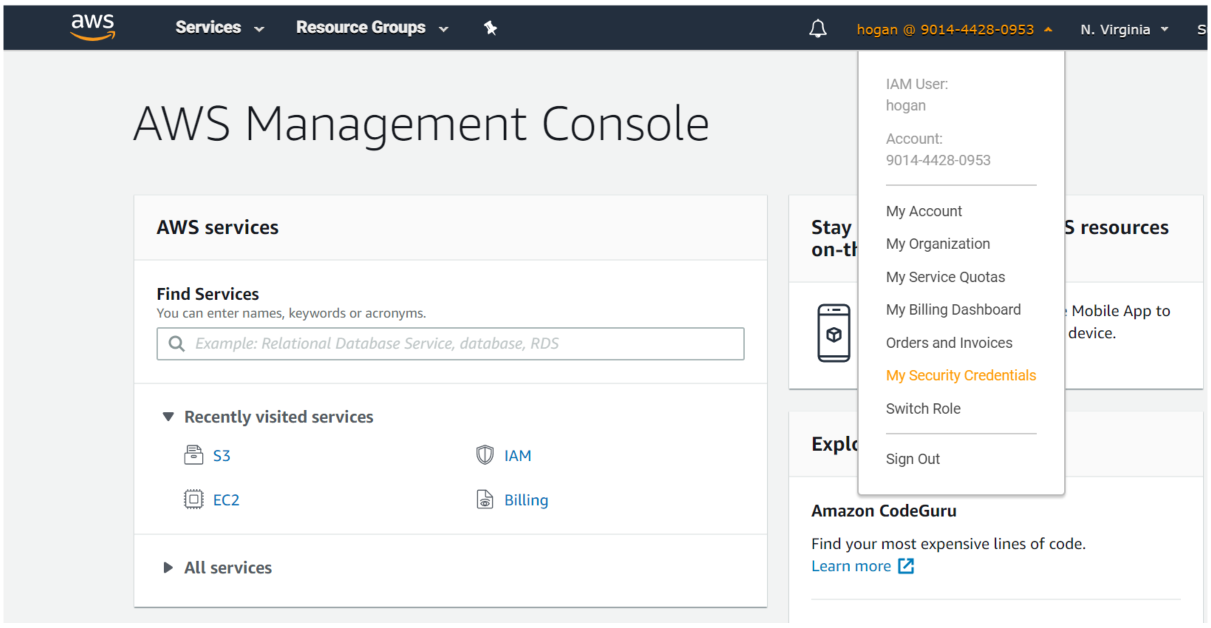
We will modify the code as needed below. It is very straightforward and we will tolerate a small (to medium) amount of duplicated code.

Before we manage any of this we need to access your credentials. We have seen this previously in Lecture 3-5. You need the Access Key ID and the Secret Access Key that you downloaded as a csv when you first received your account:

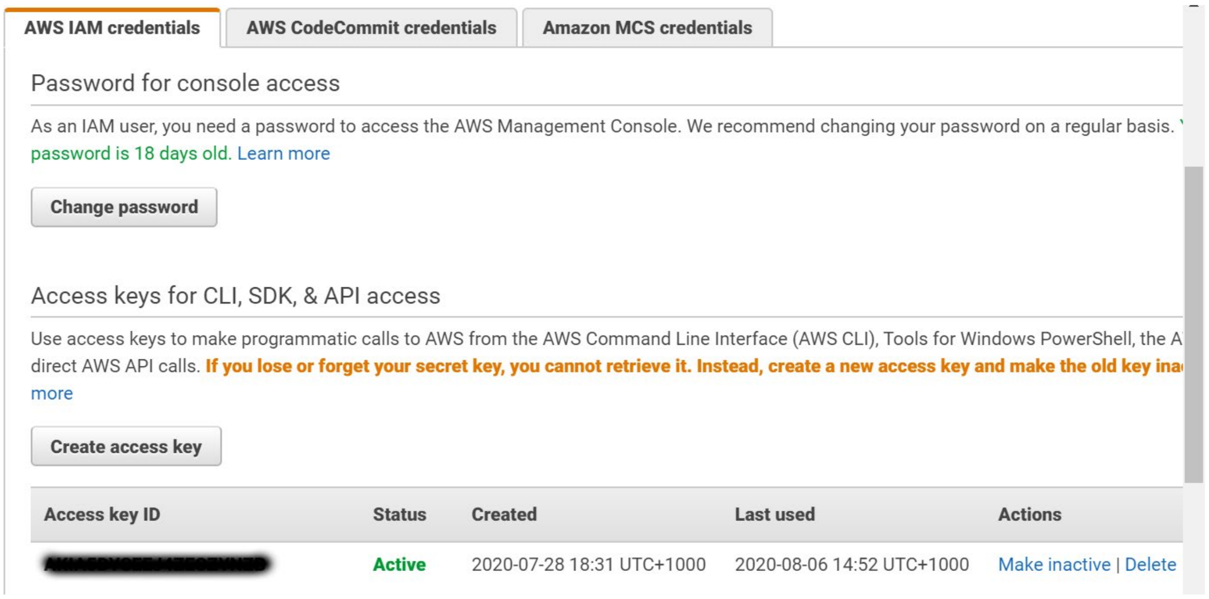


	A	B	C	D	E	F	G	H	I	J	K
1	User nam	Password	Access key ID	Secret access key	Console login link						
2	hogan		[redacted]	[redacted]	https://901444280953.signin.aws.amazon.com/console						
3											

If you have lost these, login and navigate to your credentials from the account menu:



When you get to the IAM page you should scroll down as shown below, and follow the instructions to create new credentials as usual:



Password for console access

As an IAM user, you need a password to access the AWS Management Console. We recommend changing your password on a regular basis. password is 18 days old. Learn more

Access keys for CLI, SDK, & API access

Use access keys to make programmatic calls to AWS from the AWS Command Line Interface (AWS CLI), Tools for Windows PowerShell, the A direct AWS API calls. **If you lose or forget your secret key, you cannot retrieve it. Instead, create a new access key and make the old key ina** more

Access key ID	Status	Created	Last used	Actions
[redacted]	Active	2020-07-28 18:31 UTC+1000	2020-08-06 14:52 UTC+1000	Make inactive | Delete

The instructions below will mimic those from lecture. Create a credentials file that can be used to support programmatic access. The details are well laid out in the guide linked here:

https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/loading-node-credentials-shared.html

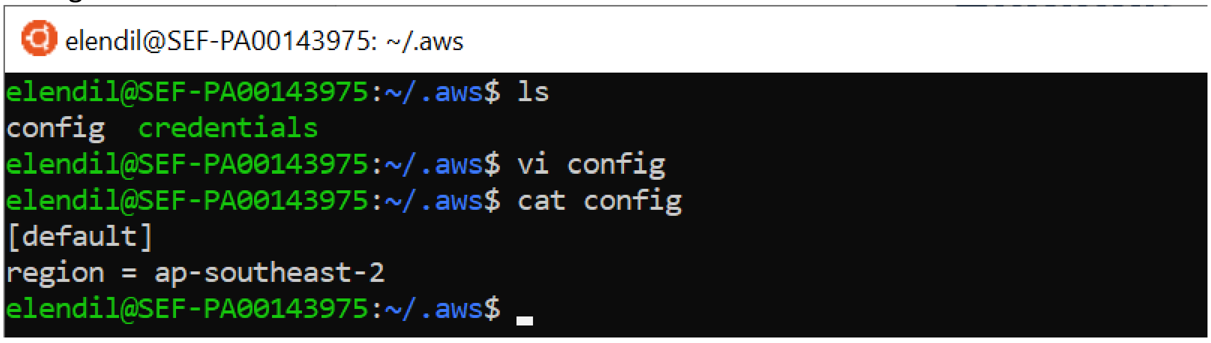
We can have a number of alternative profiles but we will use the default:

```
[default]
aws_access_key_id = <YOUR_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_SECRET_ACCESS_KEY>
```

The guide also tells you where to store the credentials file. If using Windows make sure that the environment variable `USER_NAME` is set. The locations are as follows:

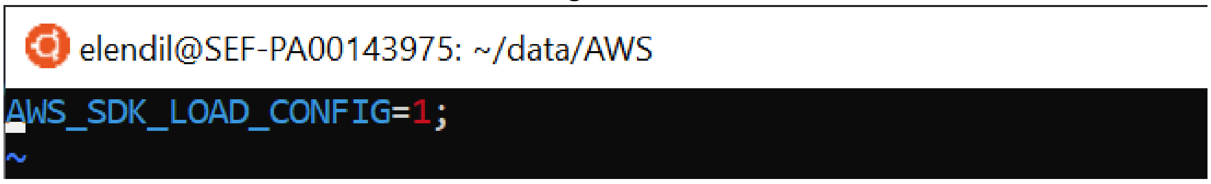
- For Linux, Unix, and macOS: `~/.aws/credentials`
- On Windows: `C:\Users\USER_NAME\.aws\credentials`

Given the restrictions on the accounts we will also ensure that we set the region in a configuration file as shown below:



```
elendil@SEF-PA00143975: ~/.aws
elendil@SEF-PA00143975:~/.aws$ ls
config  credentials
elendil@SEF-PA00143975:~/.aws$ vi config
elendil@SEF-PA00143975:~/.aws$ cat config
[default]
region = ap-southeast-2
elendil@SEF-PA00143975:~/.aws$
```

To ensure that the config file is read properly, we will need to set an environment variable. As discussed in the lectures, we will be using a local `.env` file:



elendil@SEF-PA00143975: ~/data/AWS

```
AWS_SDK_LOAD_CONFIG=1;
~
```

The `credTest.js` file supplied is based on the code found in the getting started guide here:

https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/getting-started-nodejs.html

The only real difference is the use of the `dotenv` module as seen below. Use this in your node sdk code to ensure that the `.env` file is read and thus the credentials file is loaded:

```
require('dotenv').config();
var AWS = require("aws-sdk");

AWS.config.getCredentials(function(err) {
  if (err) console.log(err.stack);
  // credentials not loaded
  else {
    console.log("Access key:", AWS.config.credentials.accessKeyId);
    console.log("Secret access key:", AWS.config.credentials.secretAccessKey);
  }
});
```

As always, install the necessary modules and then run the application using node as usual. If it works, your credentials should appear on the screen.

In the next steps we will adapt the standard S3 example from the AWS Node docs to use some of the logic from our Wikipedia and Redis example above. You will then take this code and adapt it for use in the `/api/store` route in the main application. When you finally use this route, the URL you type into the browser will look something like this:

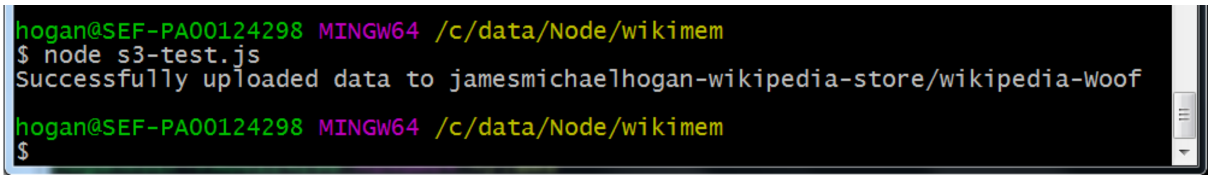
http://xxxxx.xxx.xxx:3000/api/store?key=Redis

We will take it in stages to make sure that everything is working. Copy the code below and replace the xxxxx with some variation on your name or some other identifier that will be sufficient to make the bucket name globally unique. Note that the bucket name ***cannot*** contain upper case characters. In my case, I used my full name as lower case and that worked fine. The actual usage of buckets will of course depend on the scale and organisation of the data. Here we will just keep everything simple.

```
require('dotenv').config();
const AWS = require('aws-sdk');

// Create unique bucket name
const bucketName = 'xxxxxxxxxxx-wikipedia-store';
```

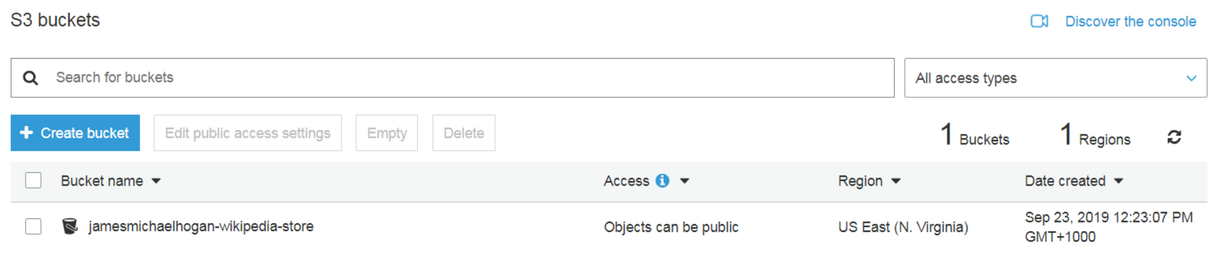
```
//Basic key/key - fixed here, modify for the route code
const key = 'Woof';

const s3Key = `wikipedia-${key}`;

// Create a promise on S3 service object
const bucketPromise = new AWS.S3({apiVersion: '2006-03-
01'}).createBucket({Bucket: bucketName}).promise();

// Handle promise fulfilled/rejected states
bucketPromise.then(function(data) {
    // Create params for putObject call
    const objectParams = {Bucket: bucketName, Key: s3Key, Body: 'Sam Wonder Dog'};
    // Create object upload promise
    const uploadPromise = new AWS.S3({apiVersion: '2006-03-
01'}).putObject(objectParams).promise();
    uploadPromise.then(function(data) {
        console.log("Successfully uploaded data to " + bucketName + "/" + s3Key);
    });
}).catch(function(err) {
    console.error(err, err.stack);
});
```

Some of the style is old fashioned, but it works, and we get the message that we want:



```
hogan@SEF-PA00124298 MINGW64 /c/data/Node/wikimem
$ node s3-test.js
Successfully uploaded data to jamesmichaelhogan-wikipedia-store/wikipedia-Woof

hogan@SEF-PA00124298 MINGW64 /c/data/Node/wikimem
$
```

You can manage remaining buckets using the S3 dashboard (note incorrect region):



Bucket name	Access	Region	Date created
jamesmichaelhogan-wikipedia-store	Objects can be public	US East (N. Virginia)	Sep 23, 2019 12:23:07 PM GMT+1000

We will now migrate some of this code into the main application. First, at the top of the app, we need to set up the bucket that we are going to use:

```
require('dotenv').config();
const AWS = require('aws-sdk');

// Cloud Services Set-up
// Create unique bucket name
const bucketName = 'xxxxxxxxxxxx-wikipedia-store';
```

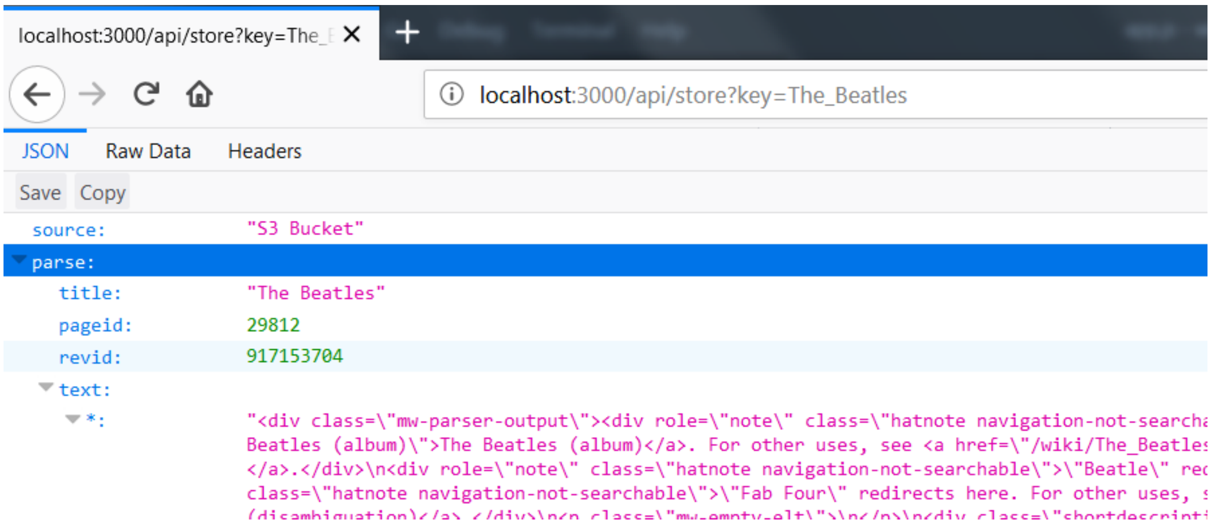
```
// Create a promise on S3 service object
const bucketPromise = new AWS.S3({apiVersion: '2006-03-
01'}).createBucket({Bucket: bucketName}).promise();
bucketPromise.then(function(data) {
    console.log("Successfully created " + bucketName);
})
.catch(function(err) {
    console.error(err, err.stack);
});
```

As before, choose something fairly unique and lower case to replace the xxxx pattern above. Further down, we introduce the new `/api/store` route, modelled on the earlier search involving Wikipedia and the Redis cache. I have used a very small fontsize due to the length of some of the lines. Note that S3 returns JSON by default, but we want the `Body` (a string):

```
app.get('/api/store', (req, res) => {
    const key = (req.query.key).trim();

    // Construct the wiki URL and S3 key
    const searchUrl = `https://en.wikipedia.org/w/api.php?action=parse&format=json&section=
0&page=${key}`;
    const s3Key = `wikipedia-${key}`;

    // Check S3
    const params = { Bucket: bucketName, Key: s3Key};

    return new AWS.S3({apiVersion: '2006-03-01'}).getObject(params, (err, result) => {
      if (result) {
        // Serve from S3
        console.log(result);
        const resultJSON = JSON.parse(result.Body);
        return res.status(200).json(resultJSON);

      } else {
        // Serve from Wikipedia API and store in S3
        return axios.get(searchUrl)

        // See next page…

      }
    });
});
```

```
      } else {
        // Serve from Wikipedia API and store in S3
        return axios.get(searchUrl)
          .then(response => {
            const responseJSON = response.data;
            const body = JSON.stringify({ source: 'S3 Bucket', ...responseJSON});
            const objectParams = {Bucket: bucketName, Key: s3Key, Body: body};
            const uploadPromise = new AWS.S3({apiVersion: '2006-03-
01'}).putObject(objectParams).promise();
            uploadPromise.then(function(data) {
                console.log("Successfully uploaded data to " + bucketName + "/" + s3Key);
            });
            return res.status(200).json({ source: 'Wikipedia API', ...responseJSON, });
          })
        .catch(err => {
          return res.json(err);
        });
      }
    });
  });
```

Note that we have labelled the object with the source “S3 Bucket” and when we try to ‘store’ The Beatles a second time, we see that the application serves the JSON from S3 as confirmation that it is already there.


localhost:3000/api/store?key=The_
localhost:3000/api/store?key=The_Beatles
JSON
Raw Data
Headers
Save
Copy

```
source:        "S3 Bucket"
parse:
  title:       "The Beatles"
  pageid:      29812
  revid:       917153704
  text:
    *:         "<div class=\"mw-parser-output\"><div role=\"note\" class=\"hatnote navigation-not-searcha
               Beatles (album)\">The Beatles (album)</a>. For other uses, see <a href=\"/wiki/The_Beatles
               </a>.</div>\n<div role=\"note\" class=\"hatnote navigation-not-searchable\">\"Beatle\" rec
               class=\"hatnote navigation-not-searchable\">\"Fab Four\" redirects here. For other uses, s
               (disambiguation)</a>.</div>\n<p class=\"mw-empty-elt\">\n</p>\n<div class=\"shortdescripti
```

In order to complete this prac for assessment, you must now do one more thing. You must insert a check for S3 based on the code above into the `/api/search` route. As discussed above, the sequence must be: the Redis cache, then S3, then the web. The `/api/store` route will remain untouched. Your task is to extend the search route as follows:

- You check the cache as before. If it is there, serve it.
- If it isn’t there, you check to see if it is in S3. If it is in S3, serve it, but also store it in the cache.

- If it isn’t in S3 either, go and get it from Wikipedia and store it in S3 *and* in the Cache
- [Note the minor differences in the key, for example “wikipedia:Redis” vs “wikipedia-Redis”, and make sure you change the `source` field in the JSON.]

You have everything you need to deal with this. The tutors will help you, but the pre-requisite code has already been given to you. Ultimately you will need to replace the local Redis service with one from AWS, but you won’t need to do that for this assessment item.

Once you have this working, you can demo quickly to a tutor. It is not necessary to deploy to the cloud for this exercise. Localhost is fine. We will advise later on virtual alternatives to face to face demos for those who can’t attend a face to face session.

Appendix: Installing Redis.

There is an extremely good guide to installing Redis under Windows at the links below. The approach for Windows 10 is based on the Linux subsystem and the approach is covered in detail here:

https://redislabs.com/blog/redis-on-windows-10/

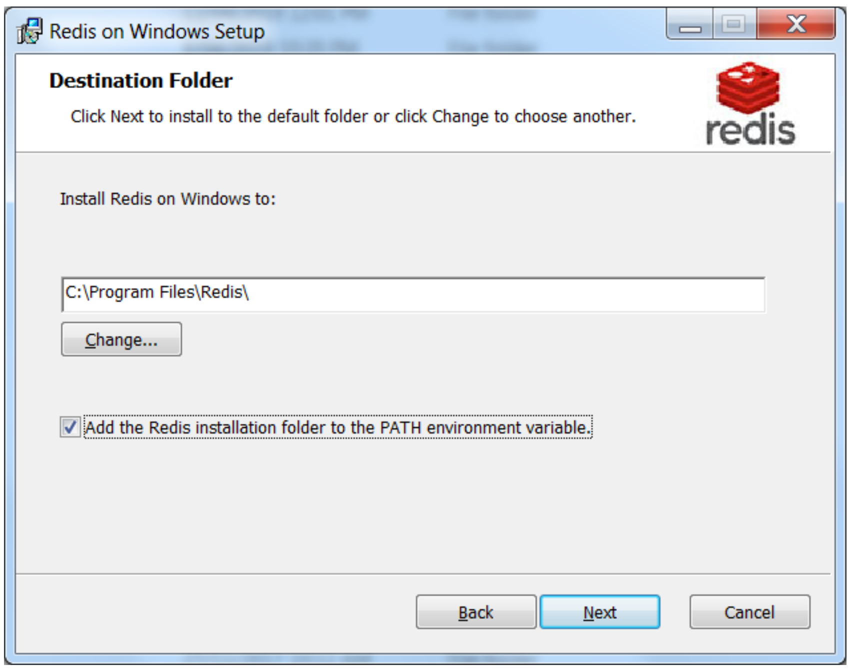
My old laptop was Windows 7 and so I used the link here:

https://redislabs.com/blog/redis-on-windows-8-1-and-previous-versions/

and then the msi download as shown here:

https://github.com/microsoftarchive/redis/releases/tag/win-3.2.100

The only thing to worry about is the checkbox shown here to make sure that the path is updated.



Redis on Windows Setup

Destination Folder

Click Next to install to the default folder or click Change to choose another.

Install Redis on Windows to:

C:\Program Files\Redis\

☑ Add the Redis installation folder to the PATH environment variable.

If you can run the msi, try to ignore most of the rest of the text as it is not really needed. You should have a look at the section *Connect to Redis for Windows* in order to work with the command line.

For those running Linux, as always our friends at Digital Ocean have it covered:

https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-redis-on-ubuntu-18-04