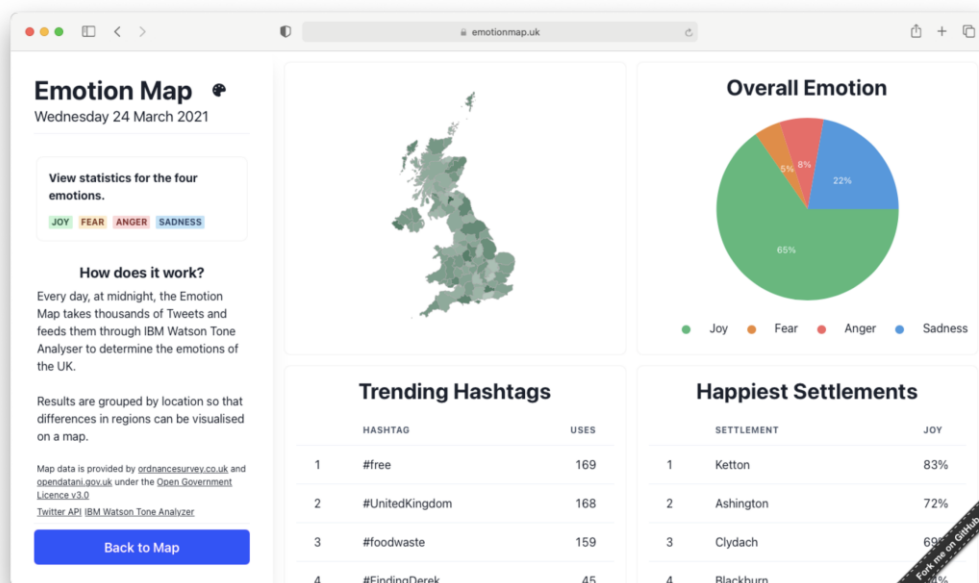
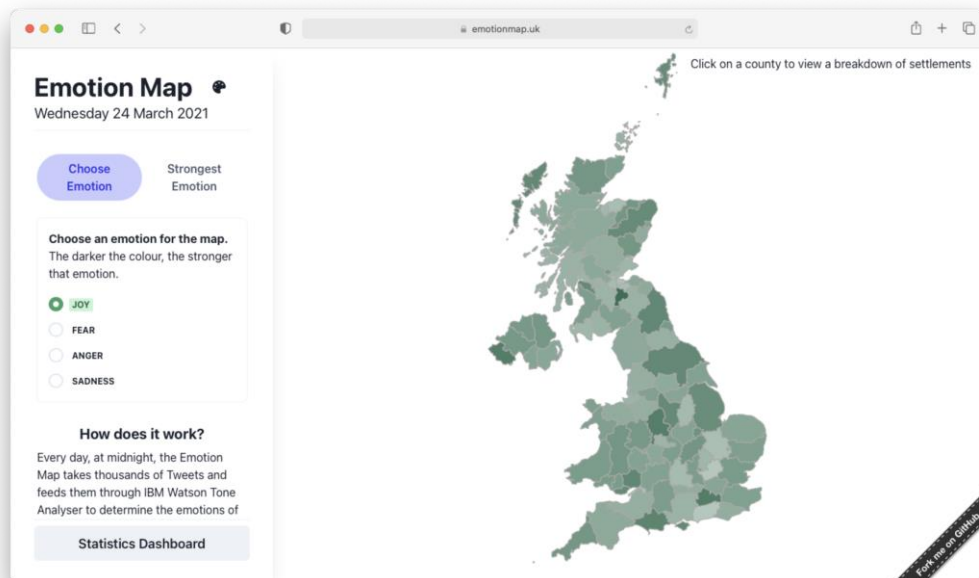


Emotion Map - User Manual

17/03/20



0. Product Overview	3
0.1 Desktop Web Application	4
Emotion Map	4
Statistics Screen	6
1. Functional Aspects of our System	6
1.1 - Geoprocessing	6
City / Town Selection	7
1.2 - Server	7
Express/Node	8
Watson API	8
Twitter API	8
MongoDB	8
1.3 - Web Application	9
React (Javascript Library)	9
2. Deployment	9
2.1 - Local development	9
2.2 - Cloud Deployment (with IBM Cloud)	10
Create IBM Cloud Virtual Server	11
Connect to server	12
Install required tools	13
Clone repository and configure	14
Run the application	14
Connect to web domain	15
3. System Maintenance	16
3.1 - Checking quotas	16
3.2 - Checking application logs	17
3.3 - Website / API not available	17
4. Updating our System in the Future	18
4.1 - Watson API	18
Change Document / Sentence Analysis	18
Change API keys	18
4.2 - Twitter API	19
API Keys & Authorisation	20
4.3 - Changing the Update Frequency	22
Using the Node Scheduler Library	22
4.4 - Updating Stats Pages	23
Adding Another Page using ReactJS	23

Increasing returned results for trending hashtags / leaderboard	24
4.5 - Changing the Map Colour Scaling	24
Changing the Relative Opacities of Your Counties/Regions	24
4.6 - Changing Map Locations	24
Editing the List of Settlements	25
Using a Different Map	25
Changing the Map JSON File on the Client	25
Defining Settlements on the Map	26
Generating counties.json	27
Appendix	29
Installing Git and Docker Compose on CentOS	29

0. Product Overview

Product URL: <https://www.emotionmap.uk>

GitHub Repository (permission from client granted): <https://github.com/IBMIXN/EmotionalMapProject>

The Emotion Map has been developed to provide an example of how our client's Cloud Services, IBM Cloud can be used to demonstrate the benefits and capabilities of Artificial Intelligence (AI).¹

By combining multiple IBM technologies with geospatial data, we have been able to produce a (near) real-time map of the UK that visualises the proportion of four key emotions (joy, fear, anger and sadness) in the countries' Tweets. Twitter was chosen as our data source, since it provides a comprehensive API and benefits from the public and opinionated nature of Tweets. Other social media networks, including Facebook, do not provide enough individual geospatial data for meaningful analysis.

IBM Watson's Tone Analyser is used to process the emotional and language tones and the data is then visualised at settlement granularity. A settlement is a town or city within a county.

The project stems from a business partner of IBM based in Sheffield that was interested in visualising "How happy is Sheffield?". It was noted that although there are available maps that depict the socio-political and economic breakdown of areas, few exist that use Artificial Intelligence to depict emotion.

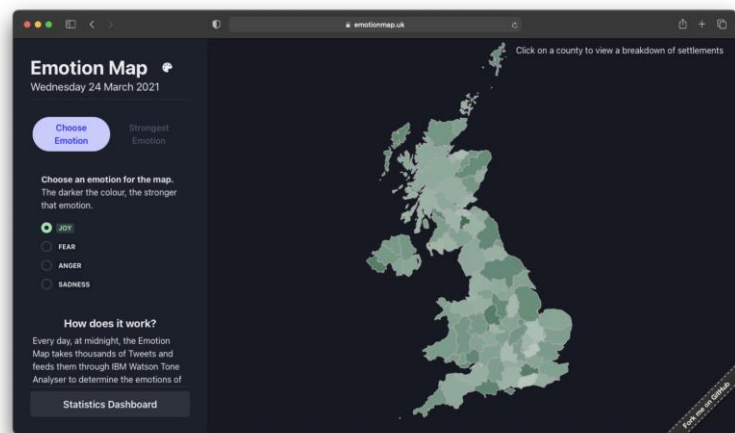
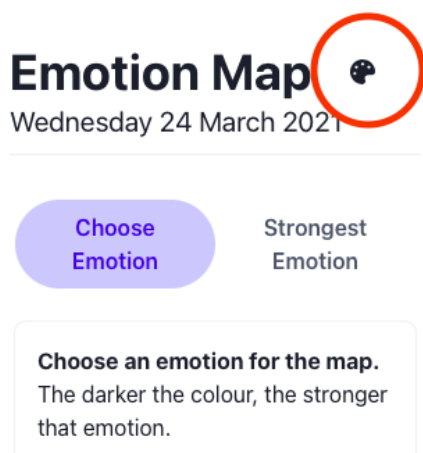
It must be noted that the data presented by our product should not be used to make any decisions and further research is recommended to determine conclusions from our information. This is due to the varying sample sizes used to generate our data and the very real possibility of Artificial Intelligence interpreting a tweet incorrectly. We have tried our best to mitigate the first issue through weighted sampling, however we cannot guarantee the lack of bias in the product. Since our product

¹ This description of the project can also be found in README.md.

purpose is to demonstrate the benefits and capabilities of AI, the struggles of creating a fair AI can also be illustrated with the Emotion Map.

0.1 Desktop Web Application

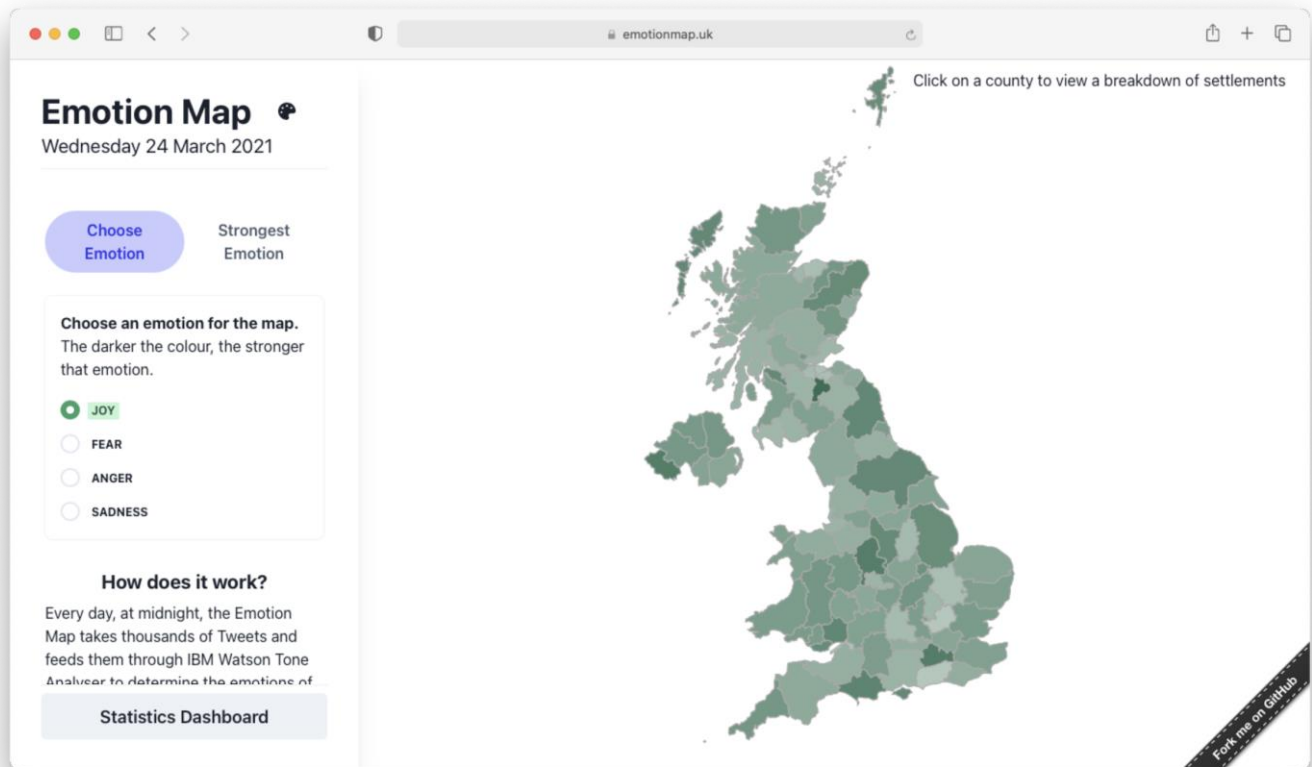
The Emotion Map is accessible via a desktop web application and is currently hosted at <https://emotionmap.uk>. We focused on the desktop web experience for the product since the demonstrative aspect means most usage will be on a large screen. We have built in a responsive UI that will adapt for mobile devices, however, the experience is not optimised and therefore is not recommended. In addition to the two screens, we have also implemented a dark theme in case this is preferred by the user. This can be accessed via the palette icon that is visible on every page.



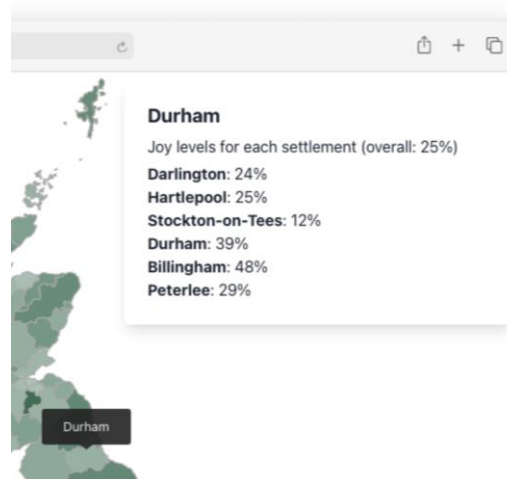
The web application is hosted on a virtual private server and is backed by a range of technologies, including React, Node and MongoDB. It has been designed to be easy to deploy. More about the technologies used can be found later in this document.

Emotion Map

The first screen that the user will view once visiting the web application is the "map" screen. Central to the application is the map, that can be seen in the center of the screen. By default, it is set to show the levels of joy for each county within the country. The darker the colours, the more intense that emotion is within tweets. The user can choose a different emotion to visualise by using the control panel on the left hand side of the screen.



To view more information for a county, the user can hover their mouse over the map to view the county name. Clicking on a county displays an overlay that includes a breakdown of every settlement sampled for this region. Tone analysis for each settlement is visible and will change depending on the selected statistic.



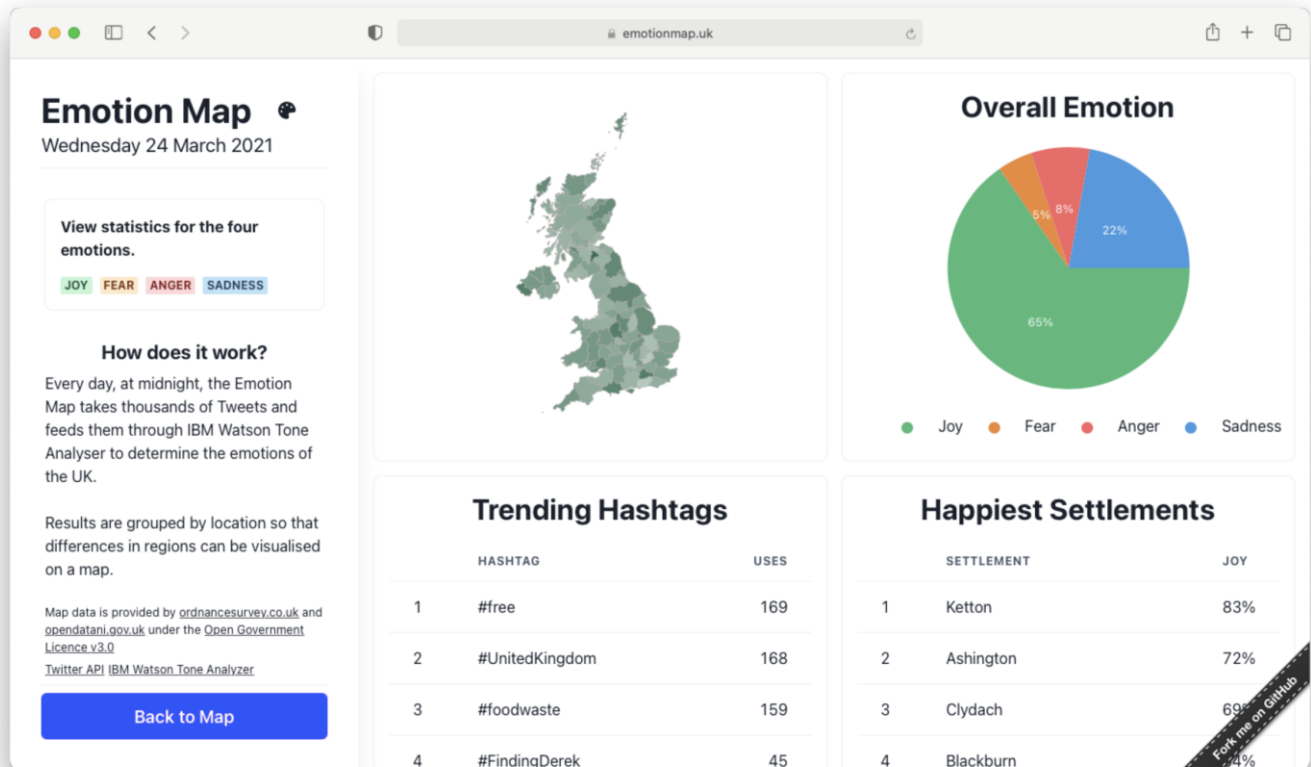
Every county on the map has an equal number of tweets sampled such that it fairly represents the UK.

It is also possible to select "Strongest Emotion" in the control panel which will show the strongest emotion for every county. From our testing, it was found that almost every county displayed more joy within their tweets than any other emotion.

Statistics Screen

The statistics screen is accessible via the /stats path in the URL, or by clicking "View Statistics" on the map screen. This page has four main sections.

The top left part of the page contains a smaller version of the map screen. It will be set to display the same information as was previously viewed on the map screen and can provide some context to the other data that is shown on this screen.



The other three sections are generated from the most recent processing data. The overall emotion pie chart displays the percentage of each emotion in all tweets processed. The trending hashtags list displays the top ten hashtags that we encountered, and the happiest settlements list displays the settlements (not counties) that had the highest joy levels.

1. Functional Aspects of our System

Our product is built upon three main components: geoprocessing, server and client. Each of these components are programmed in Javascript or Typescript and can be found in their respective directories in the code repository.

1.1 - Geoprocessing

In order to retrieve information from the Twitter API, the server needs to know which regions for which we would like to query for tweets. For this, we source county data from

ordnancesurvey.co.uk and opendatani.gov.uk that is licensed under the [Open Government Licence v3.0](https://open.gov.uk/open-government-licence). This allows us to store and modify the region data provided we display attribution on our web application.

Upon consideration of the available quotas for both APIs, it was determined that each county should contain no more than 5 settlements. We created a file with points that correspond to settlements. This file is in the "geojson" format which is a standard used to describe simple geographic features. The tool at <https://geojson.io/> is useful for creating these files. Every point has sample size and radius attributes and the finished file is passed to the code in the geoprocessing subdirectory.

The *process.js* script reads the county data and this Geojson file to produce a *counties.json* output. It assigns all settlements to their correct county so they can be matched on the client, and converts the location data to a *geocode* that can be understood by the Twitter API. An example of the *counties.json* file including London can be seen below.

```
{
  "London": [{
    "name": "Greater London",
    "geocode": "51.513870548723986,-0.1153564453125,20km",
    "sample_size": 300
  }]
}
```

City / Town Selection

The settlements used to represent each county have been selected by members of the team. A combination of Wikipedia and mapping tools were used to determine the largest settlements and the radius of their built up areas. As previously mentioned, up to five settlements on average were selected for each county, however, where appropriate fewer settlements were used to provide maximum coverage. For example, London (above) only contains one settlement with a large radius.

1.2 - Server

This project is built upon the MERN software stack - MongoDB, Express, React and Node. This means that all the code that we have written is either in Javascript or Typescript.

Our server is built with Javascript, and uses the following technologies:

Express/Node

The server is built using the Express NodeJS library. The server handles connecting to the database, and scheduling the data to be requested and processed every day at midnight. The server has endpoints for all the relevant sections of the web application that require data:

/counties - This returns all the counties, with their corresponding name, emotions and settlements. This is called by the code responsible for creating the map.

```
export interface County {
  name: string;
  emotions: Emotions;
  settlements: Settlement[];
}
```

/hashtags - This returns all the most popular hashtags since the last time the refresh function was called, along with their corresponding number of uses. This is called by the code responsible for creating the stats page.

```
export interface Hashtag {  
  hashtag: string;  
  count: number;  
}
```

/joyfulsettlements - This returns the 10 most joyful settlements since the last time the refresh function was called. For each of these settlements, its corresponding name, emotions, number of sentences successfully processed and returned by the Watson API and number of tweets received for that settlement is returned. This is called by the code responsible for creating the stats page.

```
export interface Settlement {  
  name: string;  
  emotions: Emotions;  
  sentenceCount: number;  
  tweetCount: number;  
}
```

/breakdown - This accesses the full list of settlements and calculates the proportion of the whole of the UK feeling each emotion. This is called by the code responsible for displaying the pie chart on the stats page.

Watson API

Raw tweet data taken from Twitter is sent to the Watson Tone Analyzer API. For each settlement, the tweets from it are concatenated into one long string, which is then analyzed by the Tone Analyzer to get a list of tones for the region. Due to the large amount of data that we are processing, we have had to combine multiple accounts due to the amount of free quota we are allocated per month, though this can be changed (see section 3.1)

Twitter API

For each settlement in every county, the server sends at least one request to the Twitter API for the most recent tweets from that location. The *counties.json* file defines the coordinates of the centre of each settlement, the radius around that point to look for tweets in and the number of tweets desired. These are used as search parameters in the request to the Twitter API. When more than 100 Tweets are desired for a single settlement, the server will send multiple requests as a response can only return 100.

The server extracts from the API response the text body of the tweet, a list of hashtags, and an identifier so that a subsequent request won't return the same tweets.




MongoDB

Processed tone data is stored in the Mongo Database for querying. All data is removed from the database at the start of each day and gradually replaced as tweets are analysed.

In order to keep the application secure, it is not possible to access the database directly in production. This is because of the containerisation used for deployment. If the application is built for

development, the MongoDB Shell or MongoDB Compass can be used to inspect the data stored within the database.

Three collections are used to store county data, individual settlement data, and the usage count of all hashtags found by the Twitter API.

Collections						
CREATE COLLECTION						
Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
counties	0	-	0.0 B	2	24.0 KB	
hashtags	145	66.5 B	9.4 KB	2	48.0 KB	
settlements	0	-	0.0 B	1	12.0 KB	

1.3 - Web Application

React (Javascript Library)

The web application is built using ReactJs and Chakra UI. All of the code is written in TypeScript. The combination of technologies like React and Chakra enables modular components or pages to be added easily to the web application, and a consistent design. Instructions on how to add pages can be found in section 4.4 of this document.

2. Deployment

In order to speed up the deployment process and to provide a local development environment that is easily set up, we decided to use Docker. Docker is a containerisation platform that is used to automate the process of building and running applications - useful since there are many steps required to deploy our application. In this user manual we will not go into depth on how the *docker-compose* configuration files are set up, since these require a lot of technical knowledge. If you would like to learn more about Docker Compose and it's uses, please see docs.docker.com/compose/. In short, Docker Compose provides a simple way of orchestrating a small number of containers and data volumes.

2.1 - Local development

To run the project locally on your system you must first follow the instructions here: <https://docs.docker.com/get-docker/>

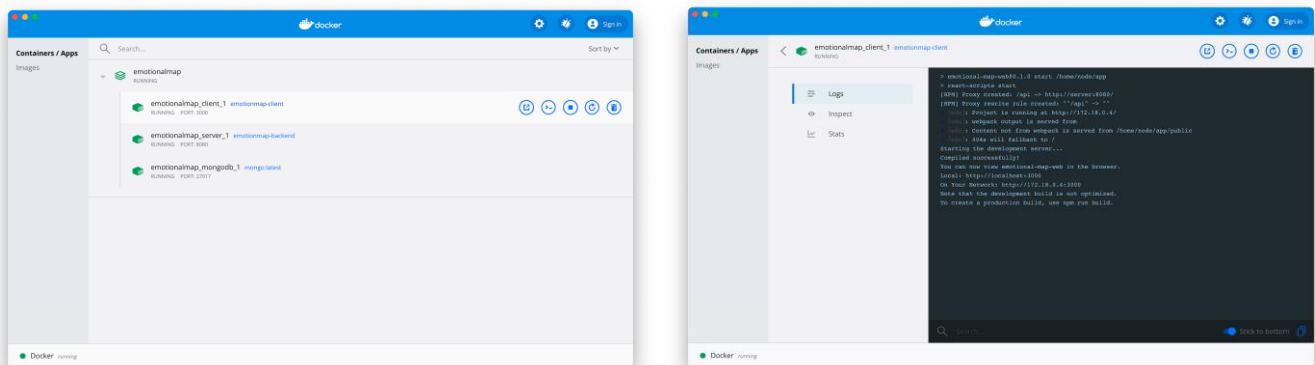
This will install the Docker desktop application and includes tools such as docker-compose. Next, start the docker engine on your system and navigate to your local copy of the application source

files. In the terminal window, run *docker-compose up*. This will install all the required dependencies, and will set up a local version of the MongoDB database that is used to store emotion data. It could take time to build the client files, so wait a few minutes for the application to load. The development environment will be completely separate to the one running on the cloud which can help prevent any outages or damage to the data served by the website, and can be useful when testing new additions to the database schema.

```
danielstone@daniels-mbp EmotionalMap % docker-compose up
Creating network "emotionalmap_default" with the default driver
Creating emotionalmap_mongodb_1 ... done
Creating emotionalmap_server_1 ... done
Creating emotionalmap_client_1 ... done
Attaching to emotionalmap_mongodb_1, emotionalmap_server_1, emotionalmap_client_1
```

The web application can be accessed by entering *http://localhost:3000* into your web browser. All calls to the API are proxied to *http://localhost:8080*. The development build uses the default *docker-compose.yml* file, and is set up to enable hot reloading for both the client and the server. Upon first launch, the server might crash since it cannot access MongoDB (before it has been initialised), therefore, it is recommended to kill the containers with *ctrl-c* before running the command again if the API is unresponsive.

Docker Desktop allows you to monitor containers as they are running and check their logs individually. The following screenshots are from the Docker Desktop application running on macOS. If everything has gone well, your application should look like this.



2.2 - Cloud Deployment (with IBM Cloud)

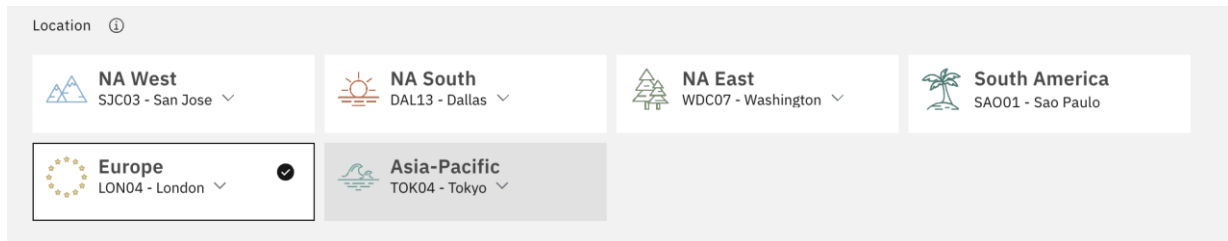
For this project we have decided to use Docker Compose running on a virtual private server, instead of a more complicated Kubernetes setup. (Kubernetes is used for automating deployment, scaling, and management. For this application we do not need to manage scaling).

The instructions here are for IBM Cloud, since IBM is the product's client, however, the same principles can be applied to any cloud provider, such as Digital Ocean, AWS or Google Cloud.

Create IBM Cloud Virtual Server

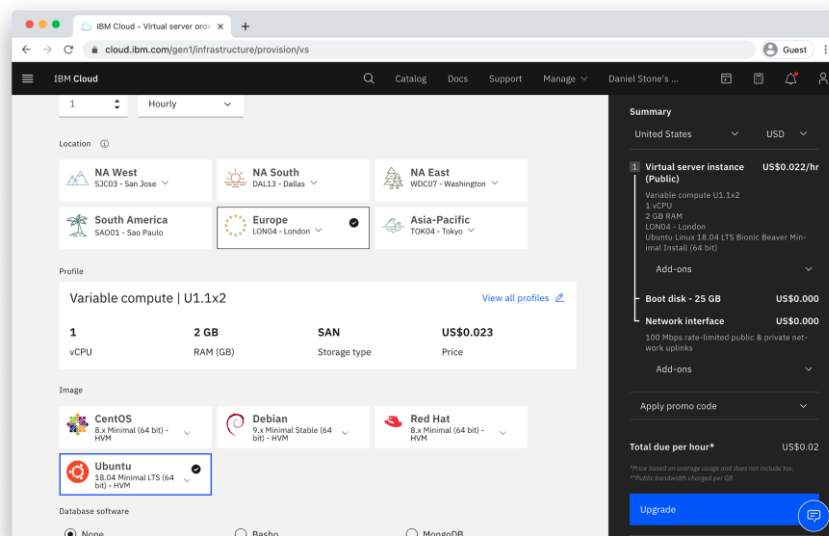
Start by creating an account on the IBM Cloud website (<https://www.ibm.com/uk-en/cloud>). Once your account is created, locate the catalog page and search "Virtual Server for Classic" or follow this URL <https://cloud.ibm.com/gen1/infrastructure/provision/vs> to access the provisioning page for a virtual server (classic).

Next, choose a location that is nearest to where you will be using the application. Here, we chose London since this will give the fastest application response times for users in the UK.

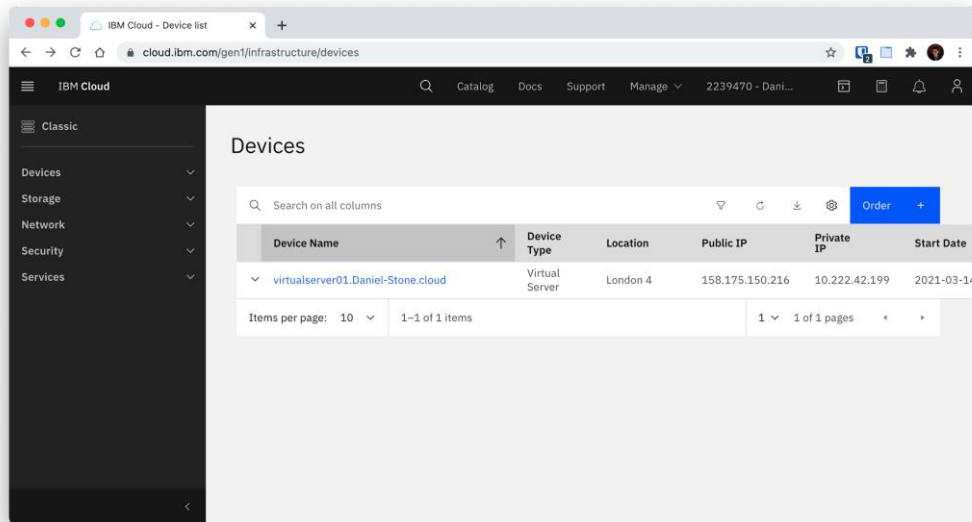


A device profile must be chosen to determine the amount of compute resources dedicated to the machine. This will determine the monthly cost of the hosting and from our testing, it was found that the cheapest (at the time of writing) option can be chosen since the server load will be low. Select the **U1.1x2** machine and then click "Save profile". Next change the operating system to Ubuntu under the Image options - Ubuntu was chosen since it simplifies the installation process for required tools. (instructions for how to install Docker Compose on CentOS can be found in the appendix).

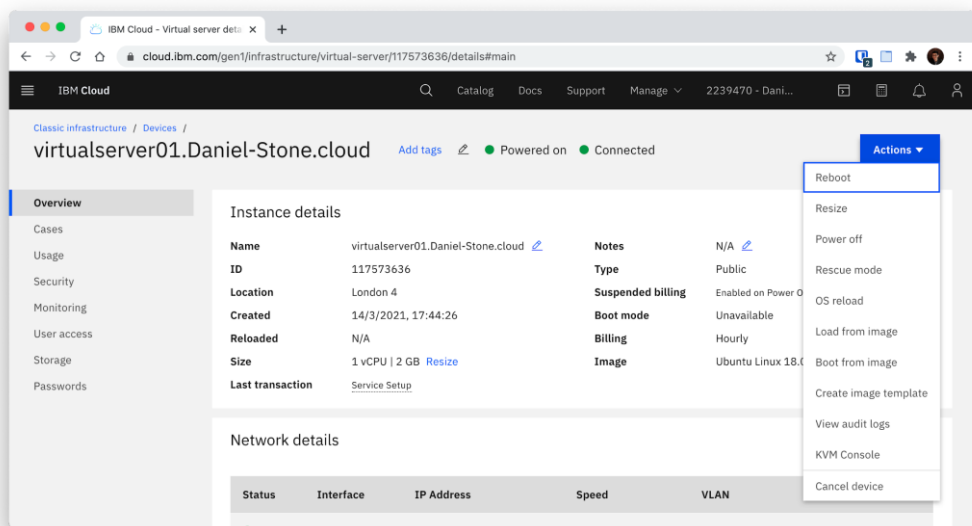
Leave everything else as it is, and then click the "Upgrade" button to add your billing information to your IBM Cloud account. The page should look like this before you click "Upgrade"



Follow the instructions to upgrade your account and check that the server has now been provisioned and is showing at the following URL <https://cloud.ibm.com/gen1/infrastructure/devices>. If the server has not been set up, try setting up the server again now that your account has been upgraded. Wait and refresh the page until the IP addresses and Start Date are displaying in the device list.



Click on the device that has been setup to view the server details. The "Actions" drop-down can be used to turn the server on or off. Note that on IBM Cloud, Virtual Server resources are billed for a minimum of 25% usage even if powered off.



Connect to server

Scroll down to network details and note down the IP address of the public interface. This will be used to SSH into the server and will be required to set up a custom domain.

Network details

[Order IPs](#)

Status	Interface	IP Address	Speed	VLAN	Security Groups
Active	public (eth1)	158.175.150.216/28	100 Mbps	lon04.fcr01a.1445	View
Active	private (eth...	10.222.42.199/26	100 Mbps	lon04.bcr01a.1704	View

Max port speed

100 Mbps public and private network

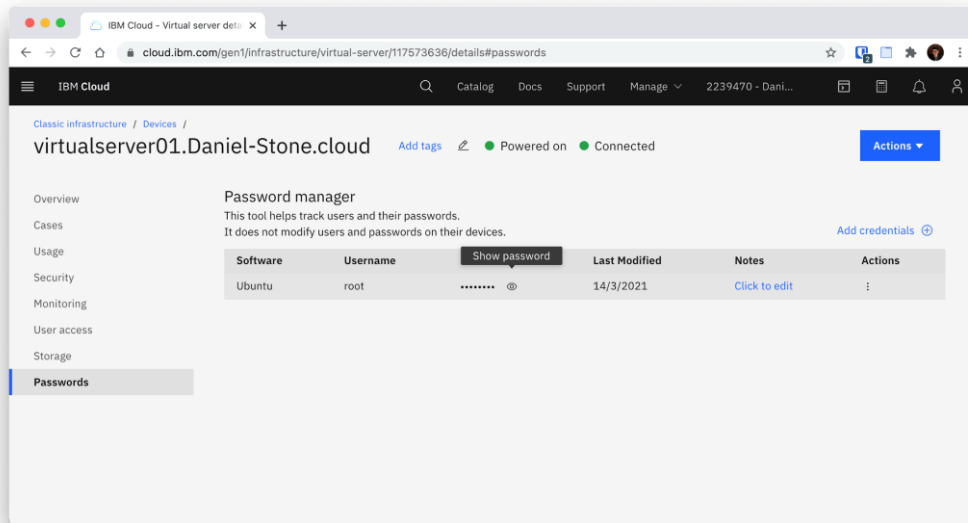
[Modify](#)

Public bandwidth allotment

Current billing cycle

0 GB [Modify](#)

Next, navigate to the "Passwords" tab on the left side of the page. Click the eye icon for the "root" user and copy the password for this account. Keep this private since it provides high privilege access to the web server.



Finally, SSH into the web server - if you're using macOS or Linux run the Terminal application and type `ssh root@[ipaddress]`. For more instructions on how to use SSH to access a server, please refer to this guide: <https://www.digitalocean.com/community/tutorials/ssh-essentials-working-with-ssh-servers-clients-and-keys>. The first time you connect it may look like this:

```
danielstone — root@virtualserver01: ~ — ssh root@158.175.150.216 — 80x24
Last login: Sun Mar 14 17:29:45 on ttys015
[danielstone@daniels-mbp ~ % ssh root@158.175.150.216
The authenticity of host '158.175.150.216 (158.175.150.216)' can't be established.
ECDSA key fingerprint is SHA256:p2JQ8o3pTb7goQl8ArrRAYq9wyxWydpXnqU8yH5X6JM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '158.175.150.216' (ECDSA) to the list of known hosts.
[root@158.175.150.216's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-118-generic x86_64)
```

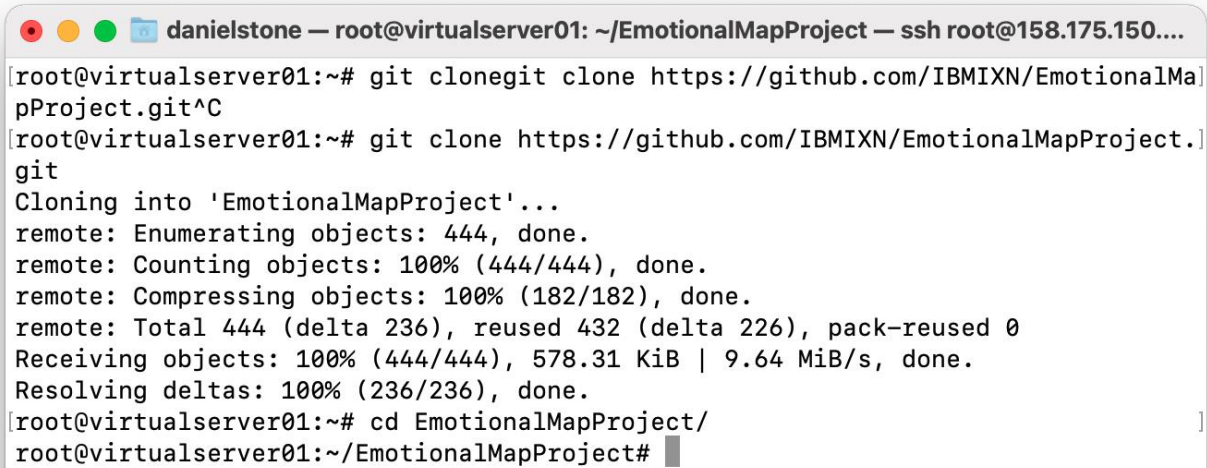
Install required tools

Docker Compose and Git needs to be installed on this machine before we can use it. With Debian, this is easy. Run the following commands and press y when prompted to confirm installation.

```
apt update
apt install docker-compose
```

Clone repository and configure

The code from the GitHub repository must now be cloned to the server. Then run *git clone* <https://github.com/IBMIXN/EmotionalMapProject.git> to download a copy of the application and *cd EmotionalMapProject* to enter the downloaded directory.

A terminal window titled 'danielstone — root@virtualserver01: ~/EmotionalMapProject — ssh root@158.175.150....'. The terminal shows the following commands and output:

```
[root@virtualserver01:~# git clone https://github.com/IBMIXN/EmotionalMapProject.git^C
[root@virtualserver01:~# git clone https://github.com/IBMIXN/EmotionalMapProject.git
git
Cloning into 'EmotionalMapProject'...
remote: Enumerating objects: 444, done.
remote: Counting objects: 100% (444/444), done.
remote: Compressing objects: 100% (182/182), done.
remote: Total 444 (delta 236), reused 432 (delta 226), pack-reused 0
Receiving objects: 100% (444/444), 578.31 KiB | 9.64 MiB/s, done.
Resolving deltas: 100% (236/236), done.
[root@virtualserver01:~# cd EmotionalMapProject/
root@virtualserver01:~/EmotionalMapProject#
```

Finally, API and database credentials need to be copied into this web server. Use nano to write these files: Run *nano api-credentials.env* to open the editor. Paste in the API credentials in the format below and then save the file by pressing *ctrl-x* and then *y* and *enter*. The same must be done for *mongo-variables.env*.

The format for these files is as follows - versions of these files with our API tokens have been used to run the application and can be found along with this submission.

api-credentials.env

```
TWITTER_API_BEARER_TOKEN=[twitter bearer token]
TONE_ANALYZER_URLS=[url1],[url2]
TONE_ANALYZER_KEYS=[key1],[key2]
```

mongo-variables.env

```
MONGO_INITDB_ROOT_USERNAME=emotionalmaproot
MONGO_INITDB_ROOT_PASSWORD=[randompassword]
```

Run the application

Finally, run:

docker-compose -f docker-compose.prod.yml up -d --build

This will download all required files, start the API server, database and initialise the Caddy web server. The website should now be accessible at the server's IP address, however it is inaccessible due to our SSL requirement.

```
danielstone — root@virtualserver01: ~/EmotionalMapProject — ssh root@158.175.150...
Step 3/8 : COPY server/package*.json ./
----> dcc410132ff5
Step 4/8 : RUN npm ci --production
----> Running in 9a22d90670d6

> bufferutil@4.0.3 install /usr/app/server/node_modules/bufferutil
> node-gyp-build

> utf-8-validate@5.0.4 install /usr/app/server/node_modules/utf-8-validate
> node-gyp-build

added 209 packages in 3.394s
Removing intermediate container 9a22d90670d6
----> 425525d74a6d
Step 5/8 : COPY server/ ./
----> cff641bed1f3
Step 6/8 : ENV PORT 8080
----> Running in 5eef4afa9a61
Removing intermediate container 5eef4afa9a61
----> ad3a54f4bfe6
Step 7/8 : EXPOSE 8080
----> Running in de36042e1a39
```

Example output as the application is being deployed.

Check the application logs at any point by running:

docker-compose logs -f

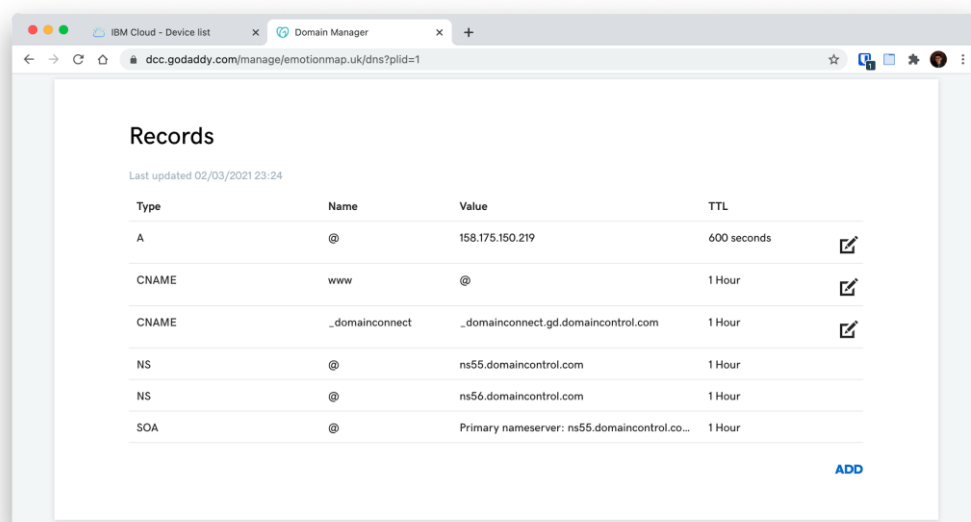
Stop the application by running:

docker-compose -f docker-compose.prod.yml down

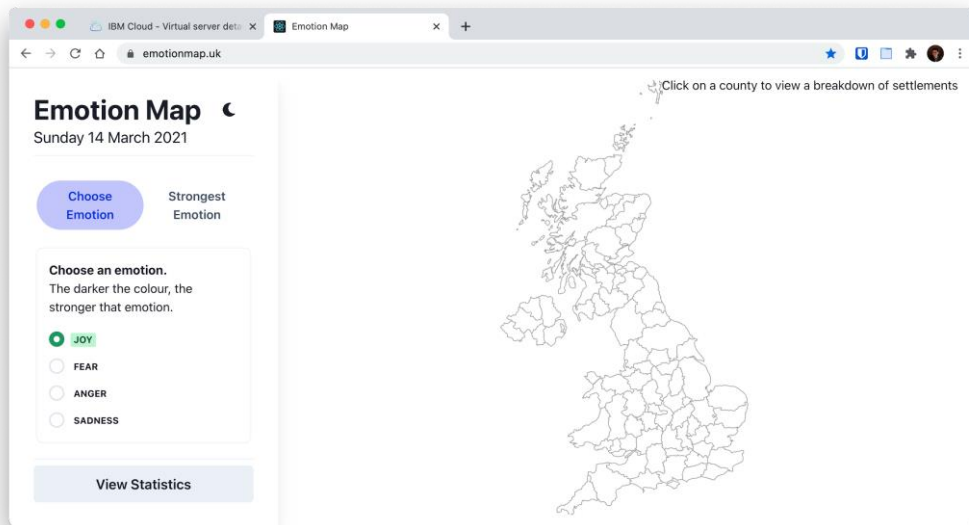
Connect to web domain

To connect the web page (hosted at the IP address) to the custom domain, first navigate to the DNS settings of your domain registrar and add a type A record with the value pointing to the IP address. Once the DNS update has propagated, restart the server to generate SSL certificates.

If the domain to be used is different to *emotionmap.uk*, the Caddyfile in the repository should be changed to match.



Once the server is running and the domain is set up, the website should be accessible. The map will be populated for the first time at midnight.



3. System Maintenance

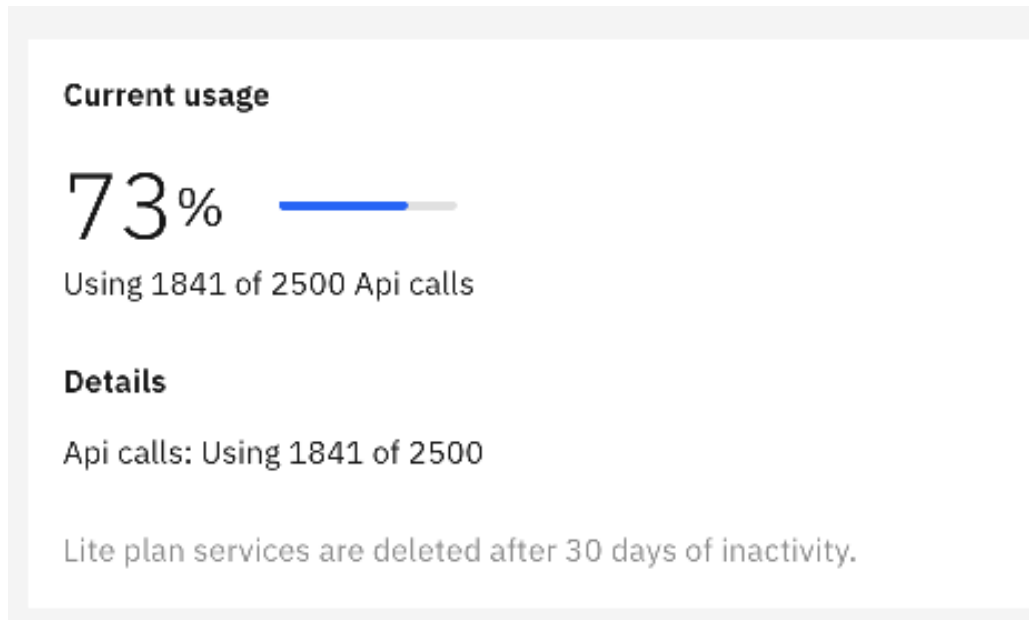
3.1 - Checking quotas

For the Tone Analyzer, we have provided 5 sets of API keys, which should be enough to run the Emotion Map as-is perpetually. However, if you decide to expand the system (such as analyzing more tweets), then you may want to view how much available quota your account has left. At <https://cloud.ibm.com/resources>, you can scroll down to "Tone Analyzer". Then, navigating to "Plan",

you can view your current usage. A useful heuristic is that the website uses approximately 400-500 API calls per day.

3.2 - Checking application logs

If an error has occurred, it may be possible to diagnose the error by using the application logs.



Provided the deployment has been set up with a virtual private server as described in part 2 of this document, this is very easy. Simply SSH into the VPS by using the credentials from your cloud provider, and move into the directory with the application files. Run the following command in order to "follow" the logs in realtime. Logs for all components of the system - web server, API and database - can be inspected by this command.

```
docker-compose logs -f
```

3.3 - Website / API not available

If the website is down, there are a number of checks that can be done to determine the cause:

- Check the device being used to access the website is connected to the internet.
- Restart the device being used.
- Clear browser data.

4. Updating our System in the Future

4.1 - Watson API

Change Document / Sentence Analysis

The Watson Tone Analyzer has two modes of operation: either sentence-level analysis, or document-level analysis. Sentence level analysis means that every sentence will be individually analyzed, which will improve the granularity of the returned data. We will then aggregate the data from each of the tweets into a combined tone. However, this uses up a lot of requests, and is not ideal in circumstances where there isn't a lot of quota available. Document-level analysis analyzes the document as a whole. This is more efficient when it comes to requests, but it comes at the cost of the returned data being much more homogenous (due to the wide-ranging emotions found in most tweets)

To change between document and sentence level analysis, you will need to change the server/processing/index.js file.

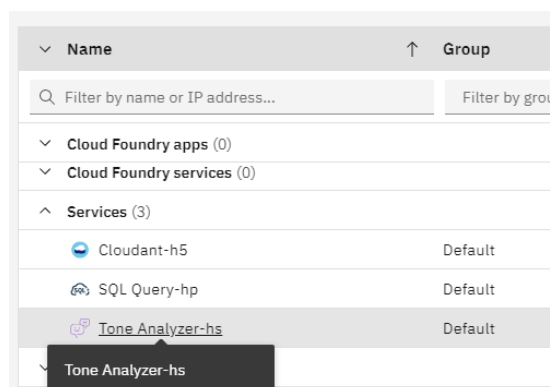
- Line 49: change `analyseTweets(tweets, true)` to `analyseTweets(tweets, false)`. The second argument to `analyseTweets(tweetdata, useSentences)` controls whether or not to do sentence-level analysis.
- Line 63: change `weights.push(sentenceCount)` to `weights.push(1)`. This controls the weighting, which helps in averaging out the data - but since you're using document level analysis, there are no weights.

Change API keys

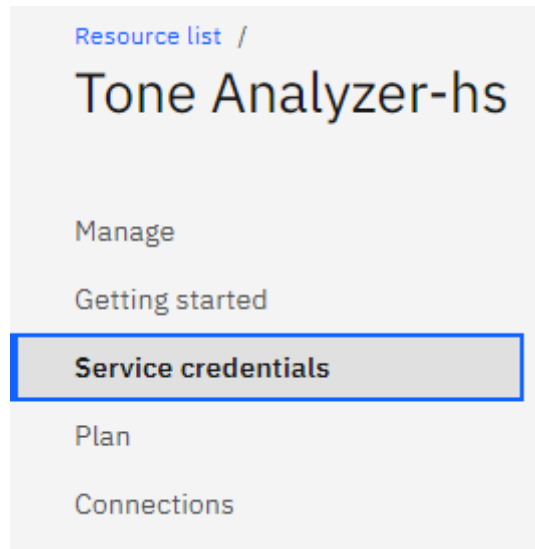
API credentials are stored in *api-credentials.env*. This file should contain 3 lines:

- TWITTER_API_BEARER_TOKEN
- TONE_ANALYZER_URLS
- TONE_ANALYZER_KEYS

To add an additional API key to this file, first navigate to <https://cloud.ibm.com/resources>. Scroll down to "Tone Analyzer":



Then select "Service Credentials".



You can then generate a key which should produce a json file similar to this:

```
^ ☐ Auto-generated service credentials 2020-11-24 9:46 AM 
```

```
{
  "apikey": "[REDACTED]",
  "iam_apikey_description": "Auto-generated for key [REDACTED]",
  "iam_apikey_name": "Auto-generated service credentials",
  "iam_role_crn": "[REDACTED]",
  "iam_serviceid_crn": "[REDACTED]",
  "url": "https://api.eu-gb.tone-analyzer.watson.cloud.ibm.com/instances/[REDACTED]"
}
```

To add your credentials to the .env file, copy the API key (the first entry in this json document) to the *TONE_ANALYZER_KEYS* line, separating it from the last entry by a comma. Also, copy the full url (the last entry in this json document) to the *TONE_ANALYZER_URLS* file, separating it from the last entry by a comma.

4.2 - Twitter API

We use version 1.1 of the Twitter API because version 2 does not allow us to search for tweets based on location. There are two offerings for this API: standard and premium. The standard API is free and provides enough functionality and a large enough request quota for the Emotion Map.

Specifically, the Search Tweets endpoint is used. Guides and the full reference are available at developer.twitter.com/en/docs/twitter-api/v1/tweets/search/. However, this section of the user manual should provide the information necessary to make minor updates to the server without needing further research.

API Keys & Authorisation

Authentication is required on all requests to the Twitter API. The Search Tweets API accepts both OAuth 1.0a and OAuth 2.0 Bearer Token, but we strongly recommend you use the latter: the quota on the number of Tweets is halved when using the former.

The first step is to set up a Twitter developer account. You must apply at developer.twitter.com/en/apply-for-access with precise details on how you plan to use the API.

Once your account has been approved, you will have access to your dashboard (developer.twitter.com/en/portal/dashboard). Here you will see a “New Project” button. Follow the steps to detail the project name, use case and description. You can now create a new App within this project, from the project overview.

The screenshot shows the 'Academic' project overview page. At the top, it says 'ACADEMIC PROJECT' and 'Academic'. Below this are tabs for 'Overview' (selected) and 'Settings'. The main content area is divided into two columns. The left column has a 'Usage' section with a bar chart icon and a 'MONTHLY TWEET CAP USAGE' section showing a progress bar at 0% with 11,765 Tweets pulled of 10,000,000. The right column has a 'Helpful docs' section with links to 'How to use projects', 'Authentication overview', and 'Authentication best practices'. At the bottom of the left column is an 'Apps' section with a '+ Add App' button. A note at the bottom right explains that Apps are where you get access keys & tokens, and set permissions, and that you're allowed one App per Project.

ACADEMIC PROJECT

Academic

Overview Settings

Usage

MONTHLY TWEET CAP USAGE ⓘ

11,765 Tweets pulled of 10,000,000 0% Resets on February 11 at 00:00 UTC

Apps

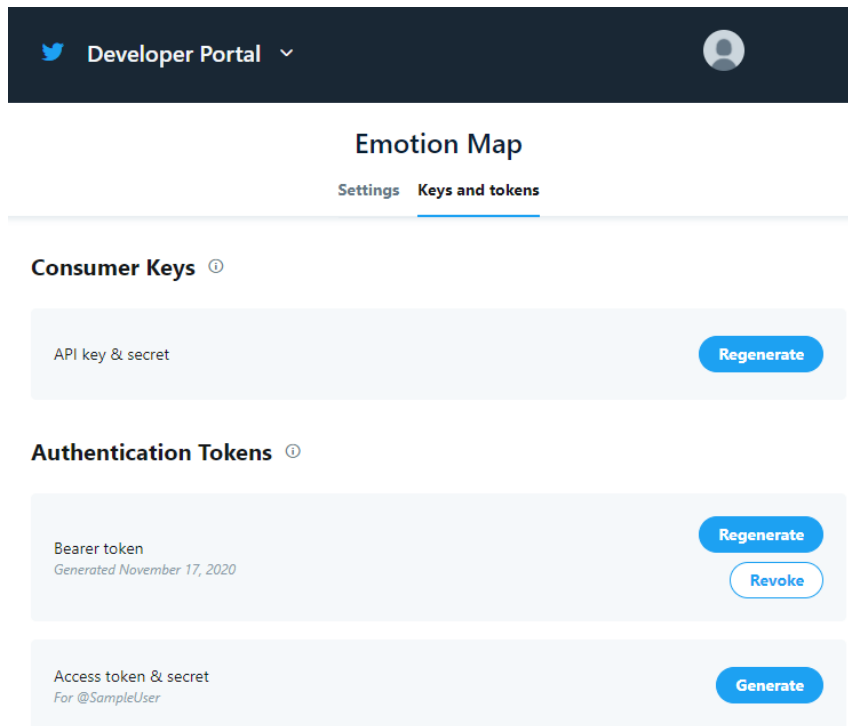
NO APPS HERE + Add App

Helpful docs

- [How to use projects](#)
- [Authentication overview](#)
- [Authentication best practices](#)

Apps are where you get your access keys & tokens, and set permissions. Right now, you're allowed one App per Project. We'll let you know when you can add more.

From the app settings page, navigate to the “Keys and tokens” tab. Here you can view the Bearer Token. Copy this and store it safely and securely; you will not be able to view it again from the Developer Portal.



The “Keys and tokens” tab after generating the bearer token.

You now need to store the bearer token in the file `api-credentials.example.env` in the server.

The server will now use this bearer token in a header in all requests to the Twitter API.

```
TWITTER_API_BEARER_TOKEN=AAAAAAAAAAAAAAAAAANGUnfPVqh6ERrUNn3fKNJlN7zQMFg0rGIInb2fdGR0ew3IJSS
TONE_ANALYZER_URLS=
TONE_ANALYZER_KEYS=
```

Changing Query Parameters

The query parameters currently used can be seen in the server file `tweets.js`.

Full reference can be found at developer.twitter.com/en/docs/twitter-api/v1/tweets/search/api-reference/get-search-tweets, but changing some parameters may break parts of the code.

```
let url = 'https://api.twitter.com/1.1/search/tweets.json';
let query = `?result_type=recent&geocode=${settlements[key].geocode} +
&count=${page_size}&include_entities=true&tweet_mode=extended&lang=en`;
```

The parameter `tweet_mode` should be set to `extended`, otherwise all tweets returned will be cut short to 140 characters (when some are as long as 240 characters).

`include_entities` is set to `true` so that a list of hashtags are included in the response. However, much more unused information is returned too, for example about URLs and attached media. If hashtag analysis was either done by the server or removed as a feature of the website, `include_entities` could be set to `false`.

`result_type` can have one of three values: `recent`, `mixed` or `popular`. `Recent` has been chosen as it always returns the most recent tweets in reverse-chronological order. `Popular` returns only the most

popular results, meaning tweets one week old could be returned. Mixed includes both, retaining a real-time element but also giving greater recognition to more significant tweets.

lang filters tweets for a specified language. English is chosen because IBM Watson Tone Analyzer cannot understand other languages. Omit this parameter to remove the language filter.

The *q* parameter is not currently used. It limits tweets to those matching the specified search query. This could for example be used to display on the map solely emotions on a certain topic, person or event. However, the number of returned recent tweets might be very low when also restricting to a geocode.

The value for the parameter *geocode* changes for every settlement. They are defined in *counties.json*. We discuss how to change the contents of this file in section 4.6.

The value for *count* also changes depending on the settlement, as different numbers of tweets can be requested in *counties.json*. This also changes when over 100 tweets are desired for a settlement. The Twitter API returns a maximum of 100 tweets, so to receive a higher number the server breaks it down into multiple requests, the first asking for 100, the last likely fewer.

4.3 - Changing the Update Frequency

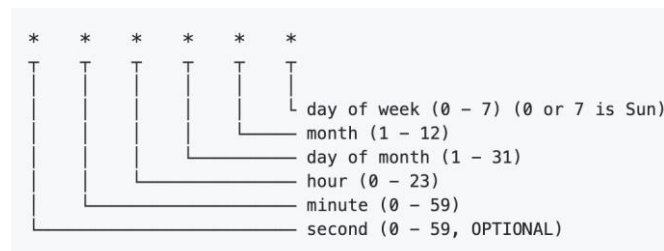
Using the Node Scheduler Library

Currently, the refresh function is scheduled to run at midnight every day using the **node-schedule** NodeJS library (See: <https://github.com/node-schedule/node-schedule#readme>). The refresh function updates our data in the database:

This code fragment can be found in server/**server.js**

```
schedule.scheduleJob('0 0 * * *', () => {  
  refresh();  
});
```

By changing the values represented by 0s and the stars in the above screenshot, the time of the day at which the refresh function is called can be changed. The schema below shows which values correspond to changes in the time:



Thus, for example, if you wanted the refresh function to be run at 7.30am each day, you would replace "0 0 * * *" with "* 15 7 * *". Of course, the function can also be called at a specific date or day

of the week or a particular month of the year, if you so please. The text used to describe this schedule is called a cron schedule expression.

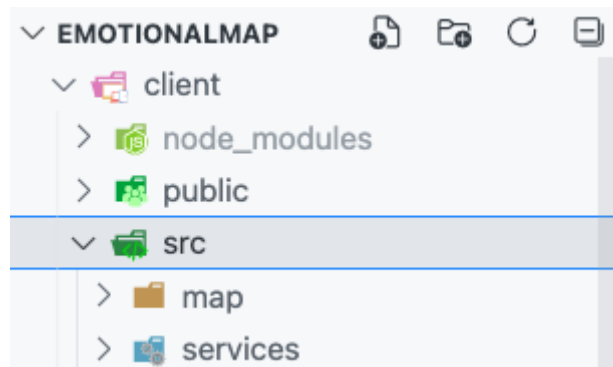
4.4 - Updating Stats Pages

Adding Another Page using ReactJS

There are currently two pages on the web application, the map and the statistics dashboard.

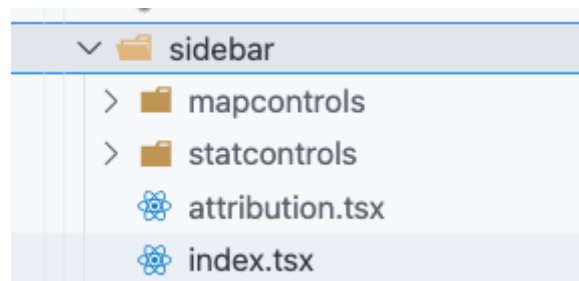
1. Create page

Navigate to the 'src' folder and create a folder for the page to be created. Inside the folder, save the page as 'index.tsx'. Making sure to export the new page.



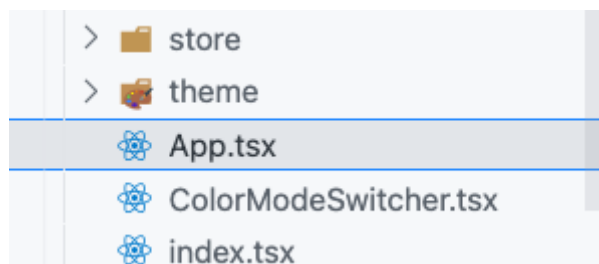
2. Add button

Add a button to the 'index.tsx' file located in the sidebar folder to redirect the user to the new page.



3. Add route

In the file 'App.tsx' located in the src folder, import the new page at the top. Add the route of the new page in the switch.



Increasing returned results for trending hashtags / leaderboard

By default, the web client displays all of the trending hashtags and most joyful settlements that the API returns. Therefore, it is easy to change the number of results displayed by changing the database query in the Express server file. First, navigate to `server/server.js` and locate the *hashtags* or *joyfulsettlements* endpoints. Change the integer in `limit()` from 10 to another suitable number. The modified code below has been changed to return 20 tweets.

```
app.get('/hashtags', async (req, res) => {
  console.log("Request: Fetching hashtags")
  const hashtags = await Hashtag.find().sort({ count: -1 }).limit(20)
  res.json(hashtags);
});
```

4.5 - Changing the Map Colour Scaling

Changing the Relative Opacities of Your Counties/Regions

The web application accepts emotion scores within the range of 0 and 1. These are returned by the *counties* endpoint on the server. In order to change how this data is represented on the map components (in both the map and stats pages), the *client/src/map/index.tsx* file will need to be modified.

The following line within the Map component code (around line 110) is responsible for the colour scale.

```
109 const colourScale = scaleLinear<string>().domain([0, 0.5]).range(["white", colour]);
```

Currently, it is set to be a linear scale with values of 0 displayed as white, and 0.5-1.0 displayed as the darkest emotion colour. The domain of 0 to 0.5 was decided since very few counties showed emotion levels greater than 0.5, and by reducing the range, the differences in emotion can be represented more clearly on the map.

4.6 - Changing Map Locations

Before making any changes, it is important to understand how data relating to the map is distributed about the system. The graphical map that the user sees is defined by one or more JSON files in the *client/src/map* directory. These determine the shapes of the regions (in this case counties) that make up the map. The server does not need its own geographical representation of the map. It does however need information on each of the settlements and a mapping from county to settlement, all provided in the file *server/data/counties.json*. The client can request data on a county (and the settlements within it) by specifying the county name; that is the link between server and client. The *geoprocessing* directory in the repository contains tools for building the server's representation from a new JSON map file. This would be useful for replacing the provided map of the UK.

Editing the List of Settlements

If changes need to be made to a small number of settlements, it is easiest to manually edit the *counties.json* file located at *server/data/counties.json*.

```
"Armagh": [  
  {  
    "name": "Craigavon",  
    "geocode": "54.452476553950206,-6.395416259765625,6km",  
    "sample_size": 105  
  },  
  {  
    "name": "Armagh",  
    "geocode": "54.34675177664337,-6.656513214111328,3km",  
    "sample_size": 94  
  }  
],
```

Removing a settlement is as simple as deleting the corresponding object enclosed in curly brackets. Adding a settlement within a county can be achieved by copying and pasting the declaration of another settlement then editing the fields. Be sure to add or remove commas between settlements as necessary.

The geocode must be in the format as shown in the above example: latitude, longitude, radius. Radii can be given in kilometers or miles, denoted by *km* and *mi* respectively. There is no limit on the sample size.

Using a Different Map

If a different map is to be used, it is likely that the list of settlements will also need to be updated. This section covers all changes that need to be made and how to make use of the included geoprocessing tools to make this process easier.

Changing the Map JSON File on the Client

Once you have found suitable data to describe the regions of your map, we recommend you download both GeoJSON and TopoJSON formats. GeoJSON will be useful for generating the *counties.json* file, whilst the compressed TopoJSON can be served to the client, meaning less data is sent over the network. There exist online tools to convert GeoJSON files to TopoJSON. Move any TopoJSON files to *client/src/map*. We have used multiple files because we used different sources for British and Northern Irish regions. The file *index.tsx* will need to be edited to use different files. For example, you may need to change the below URLs to match your files' names.

```
import britain from "../britishcountiestopojson.json";  
import northernireland from "../northernirelandtopojson.json";
```

The following line must be changed if a different number of source files are used; just add or remove variables from the array.

```
{[britain, northernireland].map((g, index) => {
```

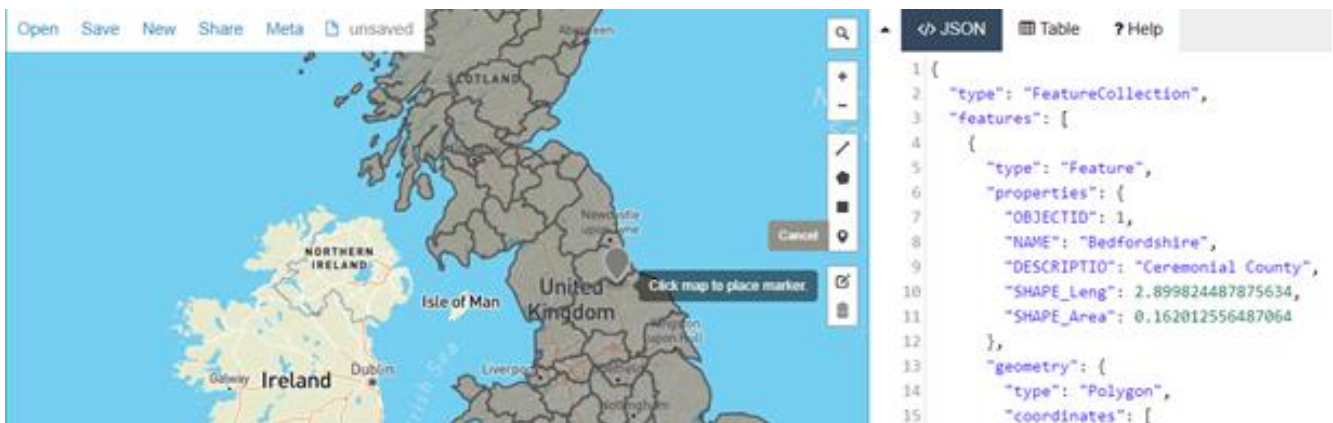
Finally, minor changes may be necessary depending on the format of your files. For example, the British and Northern Irish files use different keys for the name property: *NAME* and *CountyName*. Code like that below may be used to work with multiple conventions.

```
const name = geo.properties.NAME || geo.properties.CountyName;
```

Defining Settlements on the Map

Only the server needs to hold data on settlements - the client does not display them on its map and can get emotion data on settlements by specifying their county in its request to the server. Settlement data is therefore only held in *server/data/counties.json*. Manually specifying each of the fields for every settlement would be an inefficient process, so we have devised a method to reduce the workload.

Open the website <https://geojson.io/> and open however many GeoJSON files you use to define the regions of your map. You will see these regions outlined and shaded. This makes it easier to see which settlements will be in which regions. To add a new settlement, click on the marker tool and then click again on the centre of the settlement on the map.



This will add a new object to the bottom of the "JSON" panel. Scroll down to see the new text, and type out the properties in the format illustrated below.

```

{
  "type": "Feature",
  "properties": {
    "radius": "5km",
    "name": "Durham",
    "sample_size": 60
  },
  "geometry": {
    "type": "Point",
    "coordinates": [
      -1.5754222869873047,
      54.7755933873935
    ]
  }
},

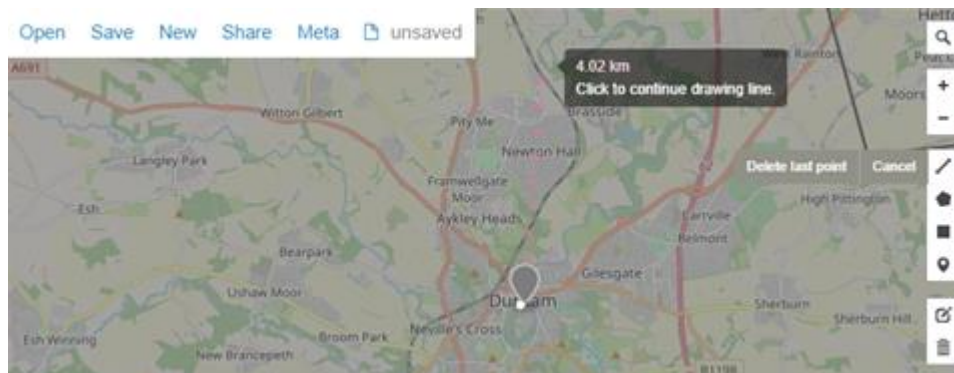
```

When adding subsequent settlements, after placing a new marker, navigate to the “Table” panel and scroll to the bottom-right. You will see an empty row has been added. Filling in the attributes in the columns “radius”, “name” and “sample_size” should be much easier than typing JSON by hand.



5km	Durham	60
7km	Newcastle	

To more easily gauge the radius of a settlement, we recommend you use the polyline tool by clicking once in its centre, then hovering around the circumference to read the distance displayed.



Make sure to cancel drawing the polyline by hitting Esc or clicking “cancel”.

Once you have added all the settlements to the map, save it as a GeoJSON file and move it to the *geoprocessing/input* directory. Manually select and delete all features that are not settlements (all the regions).

Generating counties.json

The script *geoprocessing/process.js* produces the file *counties.json* that the server uses. It takes as input the json file of settlements created in the last step as well as the GeoJSON files making up the

regions of the new map. It refactors the settlement properties (radius, name and sample size) and coordinates and organises settlements into their regions (e.g. counties) through a geometric lookup.

In the *geoprocessing/data* directory, copy in all GeoJSON files that make up your new map. These should represent the same regions as those in the TopoJSON files in *client/src/map*. Check that you have moved the json file of settlements to *geoprocessing/input* as instructed in the last section.

It may be necessary to modify the code of *process.js* to work with the new map data. In particular, like with *index.tsx* for the client, the number and names of map GeoJSON files may need to be changed when importing, and the region name property may need to be accessed differently. The code allows for another file to contribute settlement data, in addition to the one produced with *geojson.io*. See the provided file *geoprocessing/input/json.json*. This is in the same format as the *counties.json* file we aim to create. You may therefore use this to combine ready-formatted settlement data with unformatted data to produce *counties.json*. Otherwise, the code below should be removed from *process.js*.

```
// In case some of the data is provided in this format already and needs normalising
const json = require("./input/json.json");
result = {...result, ...json}
```

One further modification to consider is the exclusion of the following code block:

```
// Scale tweets down
const tweetTotalPerCounty = 200
for (const [county, settlements] of Object.entries(result)) {
  for (let settlement of settlements) {
    settlement.sample_size = Math.floor((settlement.sample_size / countyTotals[county]) * tweetTotalPerCounty)
  }
}
```

This presently looks at the “sample_size” attributes of all settlements in a region and scales them proportionally to add up to the variable “tweetTotalPerCounty”. It means that every region will request the same number of tweets, but the individual settlements within retain their relative sample sizes. Removing this code will mean the “sample_size” of settlements in the output file *counties.json* will equal those specified in the input files.

Before executing *process.js*, run the command ‘npm install’ in the *geoprocessing* directory to install the required modules. Then run *process.js* with the command ‘node process.js’. Navigate to the directory *geoprocessing/output* to see two newly generated files. You may ignore *counties_list.json*; it may be helpful for recording which settlements are in which regions, but it is not used in code anywhere. Move *counties.json* to *server/data*. The server will now use this representation of the new map.

To reiterate, it is not necessary to follow the above steps to generate *counties.json*. One may instead write the file by hand, however for many settlements it should be quicker to follow our method.

Appendix

Installing Git and Docker Compose on CentOS

To install docker on CentOS, follow the instructions at the following url:

<https://docs.docker.com/engine/install/centos/#install-using-the-convenience-script>. This script is used to install docker in the simplest way possible.

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

Once this is complete, run the following commands to enable docker on boot, and reboot. Wait 5 minutes and then SSH back into the server to continue to install docker-compose.

```
sudo systemctl enable docker.service
sudo systemctl enable containerd.service
reboot
```

To install docker-compose (version 1.28.5) run the following commands (these can also be found at <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-centos-7>, with the version number changed to 1.28.5):

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.28.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version
```

Example output:

```
[[root@emotionalmapproject ~]# sudo curl -L "https://github.com/docker/compose/releases/download/1.28.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100   633  100   633    0     0   8554      0 --:--:-- --:--:-- --:--:--   8554
100 11.6M  100 11.6M    0     0 10.0M      0 0:00:01 0:00:01 --:--:-- 11.1M
[[root@emotionalmapproject ~]# sudo chmod +x /usr/local/bin/docker-compose
[[root@emotionalmapproject ~]# docker-compose --version
docker-compose version 1.28.5, build c4eb3a1f
[[root@emotionalmapproject ~]#
```

The version number printed out should match the requested version. Install Git with *sudo yum install git* and enter y when prompted to confirm the installation. Now continue with the steps described earlier.