Australian Social Media Analytics - Using Nectar Cloud Service

Team No. 23

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Vitaly Yakutenko  976504  [vitaly.yakutenko@student.unimelb.edu.au](mailto:vitaly.yakutenko@student.unimelb.edu.au) |  | Himagna Erla  975172  [himagna.erla@student.unimelb.edu.au](mailto:himagna.erla@student.unimelb.edu.au) |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Shireen Hassan  972461  [shireenh@student.unimelb.edu.au](mailto:shireenh@student.unimelb.edu.au) |  | Syed Muhammad Dawer  923859  [sdawer@student.unimelb.edu.au](mailto:sdawer@student.unimelb.edu.au) |  | Areeb Moin  899193  [areeb.moin@student.unimelb.edu.au](mailto:areeb.moin@student.unimelb.edu.au) |

# Abstract

We built a system which is scalable, flexible, and as robust as possible. This system performs data analytics on essentially Twitter data and Financial & Labour Force census data available from AURIN. To deploy our system, and also to automate our services we implemented a range of technologies suggested in the lectures, and also many others.

The main focus of the project was to build a scalable and flexible System Architecture, and make the Front End visualizations flexible, experimental on myriad of scenarios. In other words we are able to work with various scenarios of a topic of our choice without having to tweak our harvesters.

Table of Contents

[Abstract 1](#_Toc513720209)

[1 Introduction 3](#_Toc513720210)

[2 System Design and Architecture 3](#_Toc513720211)

[2.1 Why we choose this Architecture: 4](#_Toc513720212)

[3 Services Running on Nectar Cloud 5](#_Toc513720213)

[3.1 JupyterHub: 5](#_Toc513720214)

[3.2 CouchDB 5](#_Toc513720215)

[3.3 Production Server 6](#_Toc513720216)

[3.3.2 Tweeter Manager 6](#_Toc513720217)

[3.3.3 Geo Analyzer 7](#_Toc513720218)

[3.3.4 Sentiment Analyzer 7](#_Toc513720219)

[4 Technologies Used 8](#_Toc513720220)

[5 Image Creation and Deployment 10](#_Toc513720221)

[6 System Functionalities & Visualizations 11](#_Toc513720222)

[7 Limitation of the Built SYSTEM 14](#_Toc513720223)

[8 Pro and Cons of Nectar Cloud Service 15](#_Toc513720224)

[9 Error handling: Issues and Challenges 16](#_Toc513720225)

[10 user guide 19](#_Toc513720226)

[10.1 Pre-Requisites 19](#_Toc513720227)

[11 Link TO SOURCE Code and Video 22](#_Toc513720228)

[11.1 Link to Source code: 22](#_Toc513720229)

[11.2 Link to Video: 22](#_Toc513720230)

# Introduction

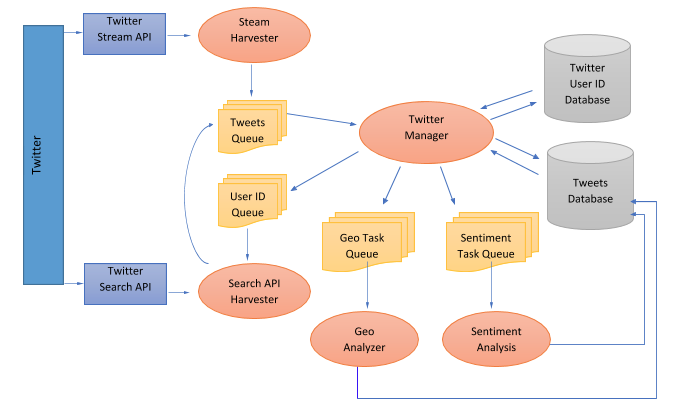
Twitter is a social media platform where people exchange their opinions and views on a myriad of topics, which makes it an ideal source of data for our scenarios and analysis about our topic. We were interested in what people in Australia feel about Cryptocurrencies and related terms (eg. block chain and central ledger), and how does it actually correlate with the financial data on AURIN. Our choice of datasets from AURIN were: median weekly incomes, and labour force data.

We have harvested over 1.3 million Australian tweets till the submission date which were all processed by our services to add sentiment, polarity, and geotags of three levels to each of them and stored them in the Couch DB. This approach allowed us to experiment and evaluate various scenarios, also allowed to specifically filter the tweets to our topic by the tweet text not just the hashtags which can be misleading.

First, in this detailed report we are going to describe the System Design Architecture, elucidate the services running on Nectar cloud, and list out the technologies used and how they were implemented. Second, we will discuss about image creation & deployment, and also, System Functionality & visualizations. Then, we will review the system built, and enumerate the pros and cons of Nectar. Finally, we will highlight the Issues we faced and error handling procedures, and end with a detailed user guide to our system.

# System Design and Architecture

**Description of System Architecture Diagram:**



To make our System scalable we needed a **data pipeline**, wanting one process to do one of the task and move it on to other processes. So in the above diagram you can see that we have Queuing mechanism in place. So the idea is one process will be harvesting all the tweets and putting them in a queue

External system Twitter is used to harvest data for Sentiment Analysis and Geo Analysis. Our **Harvester** is union of two services Stream API and Search API. Stream harvester is to implement twitter stream API, to bring tweets from twitter on real time bases and puts them in tweets queue as separate files in json format.

**Tweets Manager** reads every file from the tweet queue and pushes new tweets to the Tweets database in couchDB. In parallel, it takes attributes required for sentiment analysis (text) along with geo data for reverse geocoding and then put them in sentiment task Queue and Geo Task Queue. Another role of twitter manager is to read unique twitter user ID from Twitter users Database and put these IDs in User ID Queue.

**Twitter Search API harvester,** one by one takes user ID from user ID Queue. For every user ID it searches its history for past seven days, which is the restriction of twitter search API. Then it get tweets of that user and put them file by file in tweets Queue, which are again sent to the tweets manager which writes tweets to the database and same process is repeated.

When every Tweets manager sees a new tweet it creates different task for search API Harvester, sentiment analysis, and geo analysis.

**Sentiment analysis** takes tasks from the queue and extracts the tweet id (which is unique) and text from each task.

**Geo-analyzer** takes tasks from the queue one by one, in which it either gets a bounding box or a pin-pointing coordinate. Check if the point exists in the polygon and then updates the document in the CouchDB with tweet ID, so it adds SA4\_Name, SA4\_ Code, GCCSA\_Name, State and Territory. This is nothing but the geo tagging of every coordinate to AURIN geoSpatial data.

## 2.1 Why we choose this Architecture:

Our architecture is more scalable as tweet manager when sees a new tweets it creates different task for different processes. Each different process is going to take up that tweet and update couchDB instances with functions that they have. This makes our Architecture more scalable, as we can create spawn new instances of processes that fall behind. For example, if Geo analyzer cannot process sufficient amount of tasks, new instances can be added, which can take up that task and distribute them. This is basically like a consumer producer model we have in our Architecture.

# Services Running on Nectar Cloud

## JupyterHub:

We made a Dockerized JupyterHub instance that spawns with authentication through Github. It enables all the team members to log into that instance using their Github profile. We have whitelisted all the users in our group for authentication. At JupyterHub server we did all our initial development. When user gets authenticated through Github, our system creates a dockerize image for them; like a separate instance for each person for development or writing scripts on their own servers and test it out. This was very beneficial, as it enabled our users to do development in isolated environment.

A shared disk was mounted to the JupyterHub, which allowed the team to share notebooks and code as well as test the modules in conjunction with other team member’s code. This speeded up the development of the system, as issues like installing libraries was all handled at JupyterHub itself.

JupyterHub was also used for visualizations, so the data that we got all the visualization were done on that.

## CouchDB

An automated deployment of CouchDB has been implemented to a Nectar Cloud instance through an ansible playbook. We are doing CouchDB installation through its source. The same playbook can be run on many hosts to setup the CouchDB environment.

We then performed clustering and running the CouchDB servers in clustering mode, to achieve this, we have wrote a separate ansible playbook which we run on all the DB hosts to enable the CouchDB clustering. We have setup the two nodes cluster to act as our DB Servers. Due to the memory limitations we are utilizing two medium instances.

The CouchDB clustering is setup as persistent storage that write all the data in /data, which is a secondary volume attached to each nodes. We have several benefits of doing this. We can save our data if we need to or accidentally terminated any of the instances or something unusual happens with the system, we can just detach the volume and attached it to the new CouchDB instance to restore from the same point.

We then enabled the continuous replication between each nodes to replicate all of our Databases. We have used CURL scripts to enable continuous replication.

## Production Server

Once we had the scripts from development phase ready, we could start the production. It is the same process as mentioned above, that we have services which our running on different instances.

Limitation: queues are not shared, so cannot run each task in a different system because they need to have access to /data. So Queues are in structure of folders. Once we have rabbit queue server setup, we can just run one instance per Docker image or one instance per server. That would speed up the operations a bit.

We have other scripts like Nginx installation, which were used for JupyterHub. We had a set of scripts pre-defined for JupyterHub application, so it would just do the basic play book which installs everything, SSL that generates certificates, the Nginx that puts up with proxy pass ports that we have and then the JupyterHub scripts would run which would create the JupyterHub environment. These scripts are interchangeable, so we could use SSL/Nginx in any of the systems we have and it will work out of the box. This is the power of Ansible.

#### Twitter Stream Harvester

#### Twitter Stream Harvester uses tweepy python library to connect to Twitter Streaming API and fetch new data from Twitter in real time. The scalability of the service is achieved by its configuration. Each instance of the service depending on its rank selects subset of geographical rectangles from the config file. When launched, the service starts streaming tweets from the selected set of geo-zones and put them to the disk queue into JSON format. The new tweets are written to the temporary file and then renamed in order to avoid concurrency with Twitter manager.

#### Twitter Search API Harvester

Twitter Search API harvester reads tasks one by one from Twitter Users Queue. The service requests the user’s timeline and downloads up to 1000 latest tweets from the given user. To download tweets it uses tweepy python library that connects to the Twitter Search API. Once user’s tweets history is fetched, the service pushes new tasks into the Tweets Queue that is maintained by Tweeter Manager. When the task is processed, it is deleted from the Twitter Users Queue.

Since the number of new Australian users is not substantial, one instance of the service uses just one instance of Twitter API credentials. However, in order to make the service more scalable and robust in the future, the service should be extended and it should iterate over a list of Twitter credentials in order to process increasing number of new users.

### Tweeter Manager

Tweets Manager as the name suggests is the driving force for the whole system architecture. In a nutshell, its task is to keep the production going continuously.

Tweets Manager consumes documents one by one from the tweets queue, and writes the whole document (the tweet) on to the tweets database in couchDB with unique tweet id as the ‘\_id’ for the document .

In parallel, it extracts the text attribute in tweet, and the unique tweet\_id to make a task/file required for sentiment analysis. Also, it extracts the coordinates or the bounding box for each tweet along with the tweet\_id, and writes this as task for the Geo Analyzer.

In addition, while it writes the documents to tweets DB it generates a list of unique users (user IDs’), and writes this list to the Twitter User ID database, also puts these ID’s in User ID Queue.

### Geo Analyzer

Geo Analyzer tags every coordinate of a tweet to the corresponding Statistical Area level. But, not every tweet contains a pin-pointed coordinate, however to do Geotagging we had to restrict ourselves to pick only the tweets that have pinpointed coordinates. Therefore, we have come up with a clever solution to overcome this and take all the tweets: twitter gives us a bounding box (e.g. bounding box of Melbourne) for tweets without coordinates. So, we take all the tweets, and for tweets with bounding box we take the most probable location of the tweet as the centroid of the bounding box. This centroid is the tagged to its corresponding Statistical Area-4 level polygon.

***Brief description of Geo Analyzer working:***

Geo-analyzer takes tasks from the queue one by one, in which it either gets a bounding box or a pin-pointing coordinate. When Geo analyzer gets a point, using a function it iterates through every SA4 polygon data that is stored on Aurin Database, to find where it lies. If the point exists in the polygon it will return the SA4 name and a corresponding GCCSA (Greater Capital City Statistical Area) name, State and Territory of that polygon of higher level. However, when input is bounding box it calculates the centroid of the bounding box and check if the centroid lies in SA4 polygon using the same procedure used for pin pointing coordinates.

Geo Analyzer then updates the document in the CouchDB with tweet ID, so it adds SA4\_Name, SA4\_ Code, GCCSA\_Name, State and Territory. We achieve this mapping of lower level Statistical Areas to higher.

### Sentiment Analyzer

Sentiment Analysis takes jobs from the sentiment task Queue. For every file it retrieves two attributes: the unique Tweet ID and text from every task in the queue. It uses the unique ID of Tweet (not user ID), which is also the ‘\_id’ key for every document, to update the documents on the CouchDB database.

We implemented a tool call *Vader* from the NLTK package of python. Vader is a pre trained model for text analysis; it has a library of Lexicons, which are updated frequently. Also, library of Lexicon contains UTF-8 encoded emoji’s and urban slang words like lmfao, btw, etc.

If we were to develop our own sentiment analysis tool, we would have to train our model on tons of social media data, yet it seems to be difficult to make a library or bag of lexicons that Vader has. So, we opted for Vador as it is a quite flexible and a splendid tool.

**Brief description Sentiment Analyzer working:**

Vader takes text as input and calls function for sentiment analyzer; this tokenizes every word in the text, and for each word it has a rating in its library of Lexicons, hence for whole text it calculates a probability of text being positive, negative or neutral. The pos, neu, and neg scores are ratios for proportions of text that fall in each category (so these should all add up to be 1)

Using these three probabilities it will calculate a compound score, which is in the scale from -1 to 1. Compound score are used to tell the sentiment of text. So, Sentiment Analyzer gives, and updates the following attributes in our document in couchDB using the Unique tweet id.

'neu' : score ∈ [0,1]

'pos' : score ∈ [0,1]

'neg' : score ∈ [0,1]

‘Compound’: score ∈ [-1, 1]

'Sentiment ': Positive / Negative / Neutral

'Intensity': Moderate / Strong

The following table is how sentiment and intensity is calculated by our Sentiment Analyzer using the Vader’s compound scores.

|  |  |  |
| --- | --- | --- |
| Case | Compound Score (x) | Text sentiment |
| 1 | 0.5 > x > 0 | Positive |
| 2 | 1.0 > x > 0.5 | Strong positive |
| 3 | 0 > x > -0.5 | Negative |
| 4 | -0.5 > x > -1.0 | Strong Negative |
| 5 | x= 1 | Neutral |

# Technologies Used

**JupyterHub:**

A development & testing environment with a workspace, and a shared-directory for every team member was developed. It also hosts the workspace where we build our Frontend visualization using Ploty & Jupyter.

**Visualization**

* **Plotly:**

An extensive library in Python used along with Jupyter, which is capable of creating a myriad varieties of visualizations and dashboards. They are embedded into our WebApp.

* **Pandas**

Library to join different datasets (AURIN and CouchDB data) and transform/aggregate data for visualization

* **Seaborn**

Library for creating statistical plots in Jupyter

**Ansible:**

Ansible was used for system configurations. All the servers were setup using ansible playbooks. The playbooks were written in a modular way which makes it easy to reuse them in servers which required similar setup. For example, SSL certificates and Nginx web server are used in couchDB as well as JupyterHub deployments. A single playbook were used to cater to these needs.

**Boto:**

Boto python library was used to spawn new instances over the Nectar cloud. These boto scripts were used to spawn instances according to the flavors & availability zone specified, it was also used to connect to AWS Route53 service to create hostnames for the systems.

**AWS Route53:**

AWS provides a DNS management API named Route53. We used this to manage our domain name. This API provides us with a means to create, update or delete records or subdomains within our domain programmatically. We set all of our servers with subdomains so that it is easy for development purposes and also for server maintenance.

**Docker:**

We used docker containers and images while deploying JupyterHub. This was done to isolate development environment for each of the developers in our team. Using docker, we were able to fully utilize the limited compute instances available and also making development efforts and code sharing faster.

**Shapely Lib:**

Shapely library of Python was used in our Geo Analyzer service, which allowed us to merge Statistical Area polygons (geoJSON), calculate centroids, check distance to a polygon, and check if a point lies in/on polygon.

**CouchDB:**

We used Cloudant Python library to write/update documents in CouchDB and fetch data from it.

We used CouchDB’s built-in MapReduce capabilities to aggregate data (count tweets in different SA4 areas as well as GCCSA areas and count positive/negative/neutral tweets).

We created 3 databases in CouchDB: Tweets, Twitter Users, Aurin Datasets.

# Image Creation and Deployment

For Deployment we needed one system up to run all the processes Twitter Harvester, Stream and Search API, Sentiment Analyzer, Geo Analyzer and Topic Analyzer. But to make it more scalable, we are utilising multiple instances for these processes and implemented ranking based workload division.

**Ranking Based Mechanism**

Each of our harvester was developed to scale, each of them would have a rank associated to them and they would then divide their workload by only harvesting tweets from their assigned location. We also made our ansible script such that it would generate configuration files for each of the hosts present under “production” group of our hosts file, and in each of the configuration files, we assign a unique rank and different set of twitter credentials.

For Example, if we have two production instances, in configuration it is going to assign a rank, e.g. instance one rank#0, instance two rank#1, and so on. It will automatically create those configurations for both the servers and our analyzer use that rank to separate out the locations from where we are getting the tweets. We have config.json to hold all the configurations for the total number of instances.

Each server will just process the part of Australia, which it is assigned. This way we can scale it up to how many systems we want. We also use different twitter API’s for different servers, so that we don’t reach the limitations of getting a certain number of API calls.

**Deployment Architecture**

For our deployments, we removed the need to use hard to remember and hard to manage IP addresses by using AWS Route53 which is a Cloud Managed DNS. Our deployment is based on domain names. This also makes it easy for us to manage our hosts file in Ansible. Since we are only entering domain names to the host file, we don’t really need to change it again and again.

Our current deployment architecture is such that we use Boto library to create connections with Nectar cloud and AWS Route53 service and we use Ansible for installing applications and setting up our instances. Through Boto, we first check whether the required subdomains already exists or not, If no entry is found for our required machines, we spawn the required instance, else if subdomains already exist and the IP associated to that domain is also up and running, we do not spawn instances and move ahead with calling a sub-process to invoke the relevant Ansible scripts. In our boto script, we can provide the flavor, availability zone and volume size required for that instance. However, we did not give the flexibility to define security groups or key pairs for the sake of simplicity.

We created some shared roles in Ansible for general use for more than one type of Instances. For example we created roles for generating SSL certificates, deploying Nginx, Mounting and formatting volumes and installing common packages like python, pip, etc. We use Jinja templating to generate configurations for shared roles like nginx.

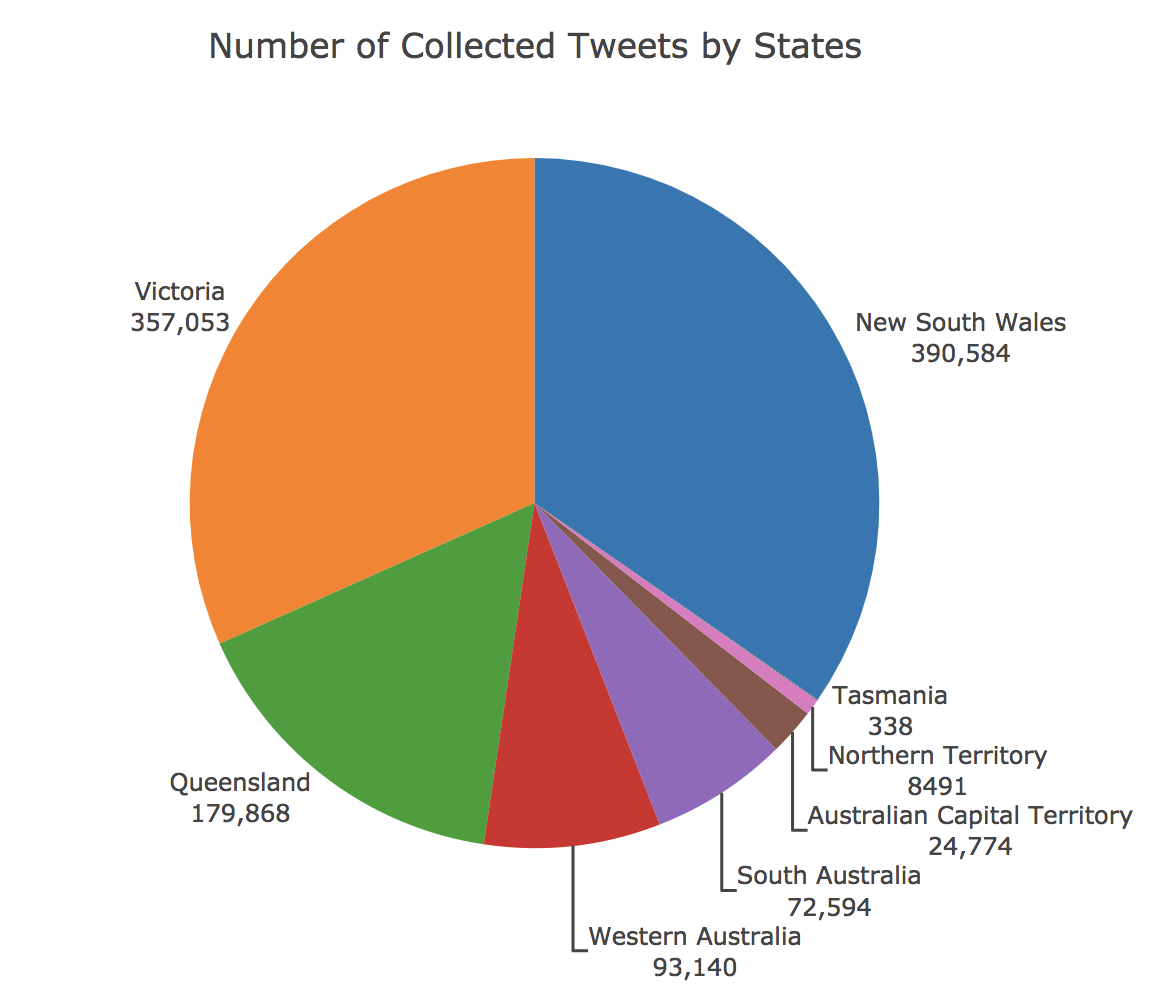
For mounting our volumes, we created a standard path “/data”, where all volumes will be attached. In mount playbook, we create this directory of /data, format the device with required file system and mounting it to /data, we also change the ownership of this folder to the user of the system so that it can be accessed without using root privileges and all of the services installed can read and write to it. All application are configured to use /data as their primary storage which ensures that the systems can handle reboots and crashes.

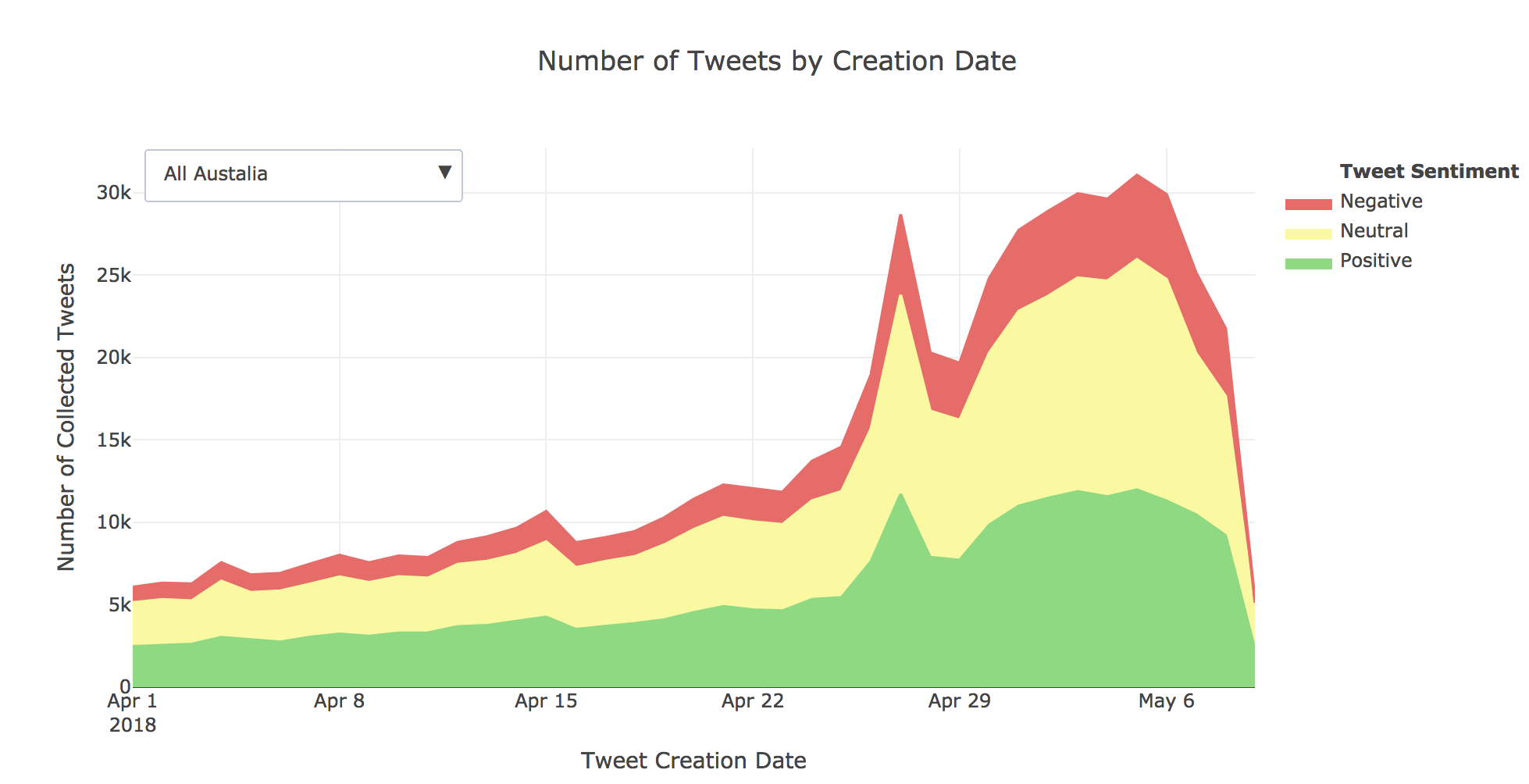
# System Functionalities & Visualizations

Frontend application was implemented as a Jupyter notebook that fetches documents with AURIN data and data from MapReduce views from Tweets database on CouchDB Server, joins the data using pandas library and outputs it with plotly and seaborn libraries.

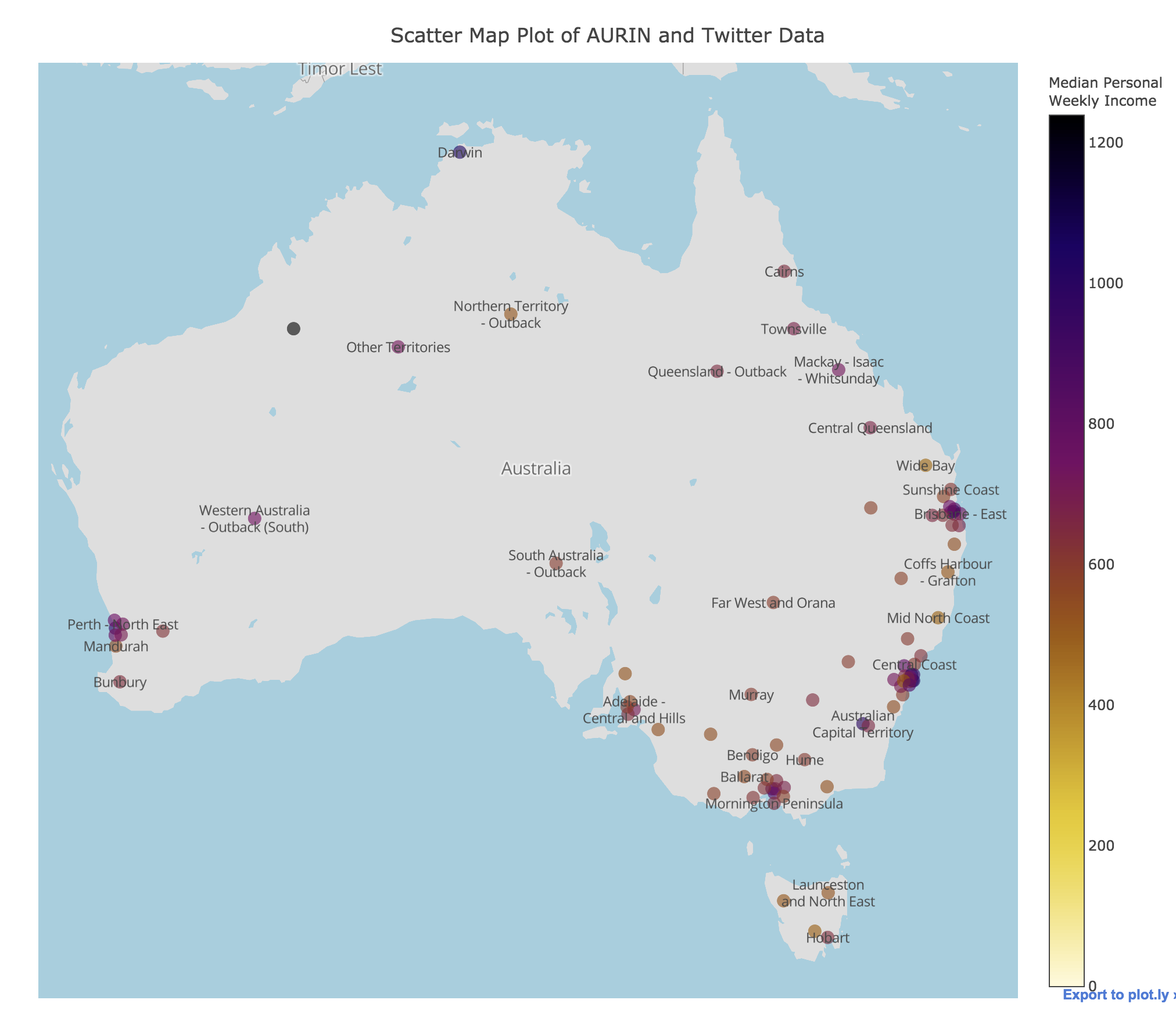
Frontend is available on <http://jupyter2.cloudprojectnectar.co/results/>

A set of different scenarios was developed to provide user interaction with the system.

* Distribution of tweets by Australian states represented as interactive pie charts.
  + Distribution of all tweets
  + Distribution of topic specific tweets  
    
* Distribution of tweets over the time by Greater Capital City Statistical Areas in Australia and tweets sentiment represented as stacked area charts providing option to select specific Capital City Areas from a dropdown menu.
  + Distribution of all tweets
  + Distribution of topic specific tweets



* Map plot with AURIN data and tweets sentiment count for each Statistical Area-4 level scattered on the map. The color of the markers represent the income level.



* **Regression plots**:

We obtained only a small sample of tweets strictly related to CryptoCurrencies from the 1.30 Million that we harvested, so using concepts of Statistical Inference we built a least square regression model to estimate the correlation between the Sentiment and the income/unemployment rate for the whole population of Australia.

Linear Regression Graphs:

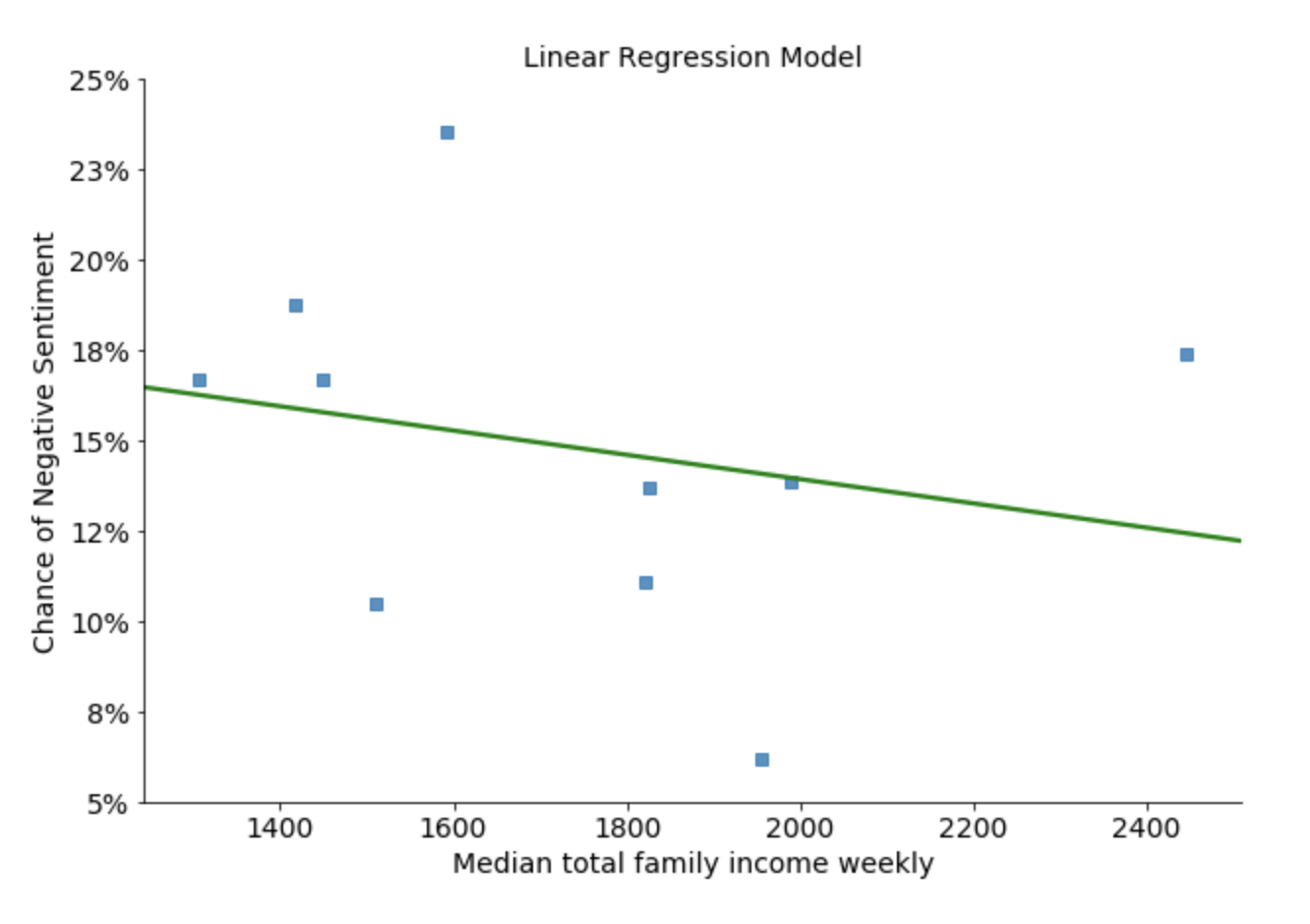
**Y-Axis:** The chance (in %) of a tweet about Crypto being Negative/Positive/Neutral.

**X-Axis:** The corresponding median family income that the each point (Urban area) has, which is obtained from AURIN.

Every point on the Linear Regression Graph represents the proportion of the particular sentiment to the total tweets about Cryptocurrencies and its corresponding median family income for each urban area. This line will allow us see the correlation of the parameters for the whole population of Australia by large.

Quantile Graphs:

Every point in the QQPlot (Quantile Quantile) represents the relation between the sample residual quantiles and Normal Quantiles. If all circles are close, or lie on the line then our assumption that data is Normally Distributed is accurate, which will validate the Linear Regression graph (the best fit line) above.



# Limitation of the Built SYSTEM

**(i)Limitation for Queue Mechanism:**

The application uses local disk queues rather than distributed queues. Each deployment on each server spawns their own queues. If distributed queuing mechanism was used, that would make the system more scalable, and all processor ping that server to get tasks. We implemented queuing in folder perspective, in which the tweets manager creates different folders for different processes and just writes a single tweet in a file for each tasks. When each process start up from the configuration, they are provided with a task queue from which they can read files, process it, update couchDB instance and later remove that file. Hence they have a number of files that they need to process.

We could do an ls (Linux command) to check tasks pending for that process. This way, If we there are a lot of files in pending state we can start up a new process so that the workload is divide between two processes.

**(ii)Limitation for starting new Process:**

The Volume that we have can be attached to one instance at a time. This leads us to run all of these processes on a single server. We could use FTP client, in one system we can save all of the information and every single process is getting there task from an external FTP server but that would have created much overhead for processing.

Other option was to use object store but it also had an overhead. Object store is Amazon S3 service, where you can store and retrieve file using curl command or by rest API, but it has network latency.

The perfect solution to use could be Rabbit Queue, which would be an external server. It would also have latency involved but that would be distributed.

# Pro and Cons of Nectar Cloud Service

**Pros:**

Using the Nectar cloud, we can automate our whole infrastructure, we can create and delete instances on the go. We can take backup snapshots of our volumes, transfer volumes between projects. Volumes can be detached; incase if an instance is not being use, we could spawn new instance and attach that volume to it.

It provides pre-configured AMIs for quick launch. We have several choices to launch any Linux based distros, and can create our own AMI’s or use AMI’s from other researches, which could decrease our time to setup the environment.

Nectar also has extra features and controls such as Network Management, Orchestration using Heat, Object Store which works like AWS S3 storage, etc. to make our tasks easier. Since it is based on Openstack, Nectar can grow in its own space given the need from researchers.

All over it's a great asset for researchers as they can utilize this power of distributed computing to solve their complex research problems.

**Cons:**

Instance provided by Nectar cloud are slow in comparison to other commercial services like AWS, Azure, etc. We had a quota on number of volumes and instances that we can utilize, which made it difficult for us to make the system distributed.

Nectar cloud is not error free, as with any other service provider, you are dependent on the support in case any problems arise. In this case, since Nectar is not a commercial system, its support is not as fast as you would expect. For our project we had to discard couchDB instances and replace it with a new one. On terminating the existing machine it went to deleting state but got stuck for two days. We had to create a nectar ticket and the person from support took a lot of time to fix it which caused delay in our development as that Quota could not be used. Commercial cloud providers like AWS, guarantee you uptime and has faster customer support service.

We did not see any Windows based AMIs to run a Windows Server instance on Nectar. There were only Linux based AMIs.

The bandwidth in Nectar instances is very limited, While fetching tweets from the CouchDB source, we observed that the data transfer rate was too slow because of which we had to run our script for more than 36 hours to fetch only few 100K tweets.

# Error handling: Issues and Challenges

**(i) Issue with Volume transfer in Nectar Research Cloud:**

A quota of four instances per team was assigned, which we felt was less. As we were using jupyter hub, for each user we required a different isolated server. Therefore we used a dockerised image for JupyterHub, giving everyone their own docker container maintained by one JupyterHub container. This way, we could provide service to every group member with just one server.

For couchDB instances we did not want to put it into one instance using docker because couchDB has much more load as compare to JupyterHub, which is why we created two instance for that. We were left with only one of the instance to run our continuous processing scripts.

For processing, we needed at least two servers, so we tried to transfer the volume from our team quota to one of the team mates. As there is an option on Nectar Cloud to accept volume from different groups/projects but this did not work as Nectar did not even allow to receive volume on personal project.

So we ran one of the production systems without persistent storage; for sending task on queues, which was are run on one of our team mate personal project and other production system was run on Nectar group project.

**(ii) Instance stuck on Delete:**

During our development phase, we created and tore down a lot of instances as per the need by the development team, and it was crucial, given the time constraint that we provide the development team with whatever tools they needed to get the job done. We had a problem while deleting one of the instances that were running on Nectar, which we were to replace with an instance of CouchDB to be utilized by our development team. The instance did not provide us with any way to restore the state of the machine, and all control options were gone. Nectar did not allow us to provision any more instances as our quota was up. We had to open a support ticket for this issue and it took two days to resolve this matter. These two days were a waste for our development team.

**(iii)Installation on fresh instances:**

Whenever we used to create an instance using boto, the spawned ubuntu server would not allow us to install anything on it using apt-get. It would give us the error: “Unable to lock the administration directory” which usually occurs when some other user is trying to install something. This error used to occur for every newly spawned instance, as a workaround, we added a restart command in our provisioning script which would take care of this problem.

**(iv)Handling of Duplication and twitter restriction quota:**

Duplication is handled by design; when a document is created in couchDB, tweet ID’s are used to name the document. CouchDB does not allow to have several document with same name, hence there can be no duplicate tweets.

We have a separate database in couchDB for the list of users called “Twitter User ID database”. Before putting new users in queue for Harvester to process, tweets manager requests couchDB to check for user ID in the database, if document with such user ID exists; it doesn’t create task for API search harvester. However if user ID doesn’t exist in database; then it creates a task for API harvester and put a file with task in API search queue.

Tweet manager processes tweets one by one, if it gets an error while processing, it will simply skip that tweet and process next one. To write new task on disk, a temporary file is created. When task is fully written file is renamed on the disk, after which the analyzer processors will access file. When tweets manager is writing to couchDB, an error can occurs; to resolve it will try to re-establish the connection and write the tweet again.

Twitter Quota was not one of the issues that we faced. As the number of active Australian users is not so high. So we can harvest tweets from their timelines without bumping into the quota. During peak times, when too many new users get into the queue and we reach the limit, we just re-attempt to query API in a loop with delay of 1 second until Twitter allows us to fetch the data from particular user id. Also we fetch data for particular user only once and we do not add them to queue the second time. To achieve this result we created a separate database with Twitter Users.

However if in future we face such issue with Quota, our ideal approach to handle this problem would be to create additional Twitter API keys and iterate them in code when quota is reached.

**(v) Geo Analyzer returning Null value for many tweets:**

Geo Analyzer was taking many tasks and was giving null output for many of the tasks. Which made us realize not every corner or Australia is covered by these SA4 polygons. So, we added a clever algorithm to Geo Analyzer; if the SA4 polygon function returns a null it will go to another function which will check the distance of point to every polygon, and will return nearest polygon that has the minimum distance with point.

This improved our accuracy of geo tag for every point or bounding box. Also, made our Geo Analysis service reliable for any kind of point in Australia.

# user guide

## 10.1 Pre-Requisites

**(I) Set Nectar Cloud Key & Secret**

To run any of the scripts, first set your Nectar keys (EC2 Key and Secret) as an environment variable. This is important; as this sensitive information cannot be given over public git repository

To get your access key and secret, follow the steps below:

* Login to Nectar Cloud at: <https://dashboard.rc.nectar.org.au/project/>
* Navigate to "Access & Security" from the left-sidebar
* At the top, navigate to "API Access"
* Then select the "View Credentials" from the top-left buttons
* Copy the EC2 Access Key and Secret safely somewhere (NEVER share this over public internet as anyone can access our machines using these credentials

Once you have the key and secret, you need to set it as your environment variables:

If you are using Ubuntu, just run the following commands on your terminal

*`****``bash***

***export ENV\_KEY='your\_ec2\_key'***

***export ENV\_SECRET='your\_ec2\_secret'***

***```***

These two commands will set your environment variables but will only be valid for that terminal session.

You can add these two commands in your *`~/.bashrc`* file if you want them to be loaded every time the terminal opens up.

**(II) Set up default Security Group and Key Pairs**

Our application uses the default security group and a shared key for all of our instances. Follow the steps below to allow the ports required by us:

* Login to Nectar Cloud
* Navigate to “Access & Security” from the left-sidebar
* At the top, navigate to “Security Groups”
* For the default security group, select “Manage Rules”
* One by one, add ports: 80 (HTTP), (443) HTTPS, 22 (SSH), 4369, 5984, 5986, 6984 and 8000
* Now from the Access & Security page, navigate to Key Pairs
* Click on import key
* From our downloaded code, copy the contents of nectar-project/Cloud.key and paste it in the public key section, name this key as “Cloud”

**(III) Set Route53 Key & Secret**

We are using Route53 to manage our DNS entry. We purchased a domain name for this project: ***`cloudprojectnectar.co****`*. We

Are managing all out instances using domain names for the sake of simplicity and also for management of instances, in

Case we want to swap instances without the users feeling any difference.

To set the keys, run the following commands:

***```bash***

***export AWS\_ENV\_KEY='AKIAIMMBKKRHGSD7NHQA'***

***export AWS\_ENV\_SECRET='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'***

***```***

NOTE: We are providing these credentials for the examiner to check our work. The keys will be deleted after grading is done.

**(IV) Running deployment scripts:**

*Important Note:* Since the servers are already up, the ansible scripts will only update the existing machines with the code that is present in the repository. To provision these again from scratch on a different set of Nectar keys, you will first need to delete all the registered subdomains so that the provisioning scripts creates new instances and assign them the subdomains. We have provided an untested script for that inside provisioning script folder named: “delete\_all\_domains.py”. Please do not run this script if you don’t have the intention to spawn new instances from scratch.

To run our services, you need to follow the below mentioned steps:

* Take git clone or download our project

```

git clone https://github.com/Vitaly-Yakutenko/nectar-project.git

```

* Add your github user as admin inside the JupyterHub userfile

```

cd nectar-project/ansible/roles/JupyterHub/files/

nano userlist

```

* We should also have a look at the “creds.json” file under “nectar-project/ansible/roles”. This contains all the twitter keys for each of the production instances.
* Navigate to our provisioning scripts folder

```

cd nectar-project/provisioning-scripts/

```

* Install dependencies

```

pip install -r requirements.txt

```

* Start our services one by one.

```

python start\_services.py jupyter

python start\_services.py production

python start\_services.py couchdb

```

NOTE: The start\_services.py script checks first if the domain is already registered. If it is registered and the system is alive, it will not try to create a new instance, and would just run the ansible script associated to that particular service.

Once we run all the services, we now have our instances up and running, we can access the services using the following urls:

1. JupyterHub: <https://jupyter2.cloudprojectnectar.co>
2. CouchDB:
   1. <https://couch1.cloudprojectnectar.co/_utils>
   2. <https://couch2.cloudprojectnectar.co/_utils>/

Our production servers will start processing twitter feeds, all our services including harvester, sentiment analyzer, geo analyzer will be up and running at *prod1.cloudprojectnectar.co* and *prod2.cloudprojectnectar.co*

***Scaling Instances:***

Scalability is an important part of our project. To scale our production servers, you need to follow the steps below:

* You should ensure that you have the configurations for each of the total number of instances required in our “nectar-project/ansible/roles/creds.json” file
* All the subdomains required should be present under the production group of our ansible hosts file which is located at “nectar-project/ansible/hosts”
* You would need to update the “nectar-project/provisioning-scripts/start-services.py” script, under “start\_production()” function, to invoke the create\_machine function for newly added host

Similarly to scale couchDB instances we need to follow the same above mentioned steps but this time we will add the newly added host under “start\_couchdb()” function in our start\_services script. It will automatically make all the hosts part of one cluster.

**(V) Running Visualisation notebook:**

Frontend web application can be accessed on <http://jupyter2.cloudprojectnectar.co/results/> In order to update results on this page the corresponding Jupyter notebook (Final\_Visualisation-cutted) should be executed in Jupyter Hub and then exported as a HTML page to the folder configured as a home directory on nginx server.

**(VI) Building Views for visualisation:**

We have used the built-in views provided by CouchDD, so to replicate the view data that we produced regarding the topic (CryptoCurrencies), or sentiments please view these sample map functions available at.

<https://github.com/Vitaly-Yakutenko/nectar-project/blob/master/couchDB/views.txt>

**(VII) Description of Directories in our GIT Repository**:

Our GitHub Repository is structured in the following way:

**Nectar project ~ Directories**

* ansible: contains all the ansible scripts for our deployment
* App: contains all our processes, harvesters, analyzers, etc.
* couchDB: contains some internal resources regarding couchdb
* notebooks: contains jupyter notebooks created while testing
* provisioning-scripts: contains boto scripts to manage instances

# 11 Link TO SOURCE Code and Video

## 11.1 Link to Source code:

<https://github.com/Vitaly-Yakutenko/nectar-project>

## 11.2 Link to Video:

<https://www.youtube.com/channel/UC3kjQ9MuBR2la9ZPRgSf3fA?view_as=subscriber>

Note: The above URL is for our YouTube channel, and there would be only one video which is our project video. Since, our video is very large, and is still uploading we had to give the url of our channel.