2019

Queensland Roads Monitor

CAB432

ASSIGNMENT 3

Jacob Kraut | Nicholas Kress

N10282653 | N9467688

10/27/2019

Contents

Introduction

Purpose & Description

The application purpose will be to allow a user connected to the front end to easily identify motorways and major arterial roads, along with how many cars are currently occupying each road. This will allow the user to ascertain how busy all roads available from the QLD Gov APIs are accurate to the latest available data, which can be refreshed as required. The data will be displayed in an easy to use table that allows sorting, pagination and row selection. This information will only be useful to those who reside in Queensland. The user will be able to make informed choices based on the data presented.

While data is available from the Queensland Government statistics website, this data is only a representation of averages on roads where they have used manual road counters to calculate traffic use. The application we have created counts vehicles live - as images are captured and processed.

Example Use Cases

Use Case 1

As a road user, I want the system to clearly display Queensland roads and the level of traffic currently travelling on them, so I can avoid the roads experiencing heavy traffic.

Use Case 2

As a council member, I want the system to display road usage data in a simple graphic so that we can better allocate funds to relieve congestion on the roads that need it most.

Technical Breakdown

Architecture

Explain how your system operates, making it clear how data flows around the system through requests and responses.

Our system is built utilizing Amazon Web Services. Our application is powered by a single EC2 instance that hosts both the node back-end and the react front-end which are both containerised using Docker. Our short-term and long-term persistence are both powered by AWS. The short-term is a Redis Elasti-Cache instance (REC) and our long-term persistence is a DynamoDB (DDB) instance.

A typical operation would consist as follows. Initially a user would access the webpage which would render the React front-end. Asynchronously, a request is immediately fired to the DDB instance which responds with a collection of the most recently computed data. If a user would like more recent data, they can select the refresh data button. When the button is clicked this sends a request to the node instance which in turn queries the Queensland Government API for more recent data, collates the request and passes the new keys into the REC instance. This creates on average more than 170 images that need vehicle detection processing. This level of processing creates an enormous load on the initial EC2 instance and requires auto-scaling to allow it to be completed in a timely manner with consideration that per image it takes 1 EC2 instance approximately 2 seconds to process. The processing is completed by new instances that are spun up containing a dockerised python application that is responsible for the vehicle detection. Each of the Python instances are stateless, meaning that the key will not be removed from the REC instance until the Python instance has completed computation and therefore the REC acts as short-term computation persistence. As each

image is processed, the front-end maintains a polling to an endpoint hosted by the node back-end which in turn allows the webpage to be rendered with the new data as it's being completed by the scaled-out instances. The webpage displays the data in an easy to use table and an associated doughnut graph. Once all images are finished being processed – and no new load has been added, the instance group will begin to scale back in and terminate unrequired instances.

Libraries, Services and APIs

APIs

Queensland Traffic Webcams API

This API provides a JSON with information about all its available traffic webcams including localities, districts, facing directions – and most importantly - links directly to the most recently captured JPEG. This is our applications only source of external data that is required for it to remain operational.

Front-End

ChartJS

Chart.js is a visualisation npm package using Javascript. It supports an array of different types of responsive charts, allowing the chart to update dynamically and better reflect the current state of the data being input. This package will allow us to heavily visualise the data for users of the application.

ReactJS

React.js is a front-end framework that utilises JSX to build powerful web pages for user consumption. It is a declarative, component-based framework that works well with the Node.js framework.

Back-End

ReactJS/NodeJS EC2 instance

- Serves frontend and backend using NGINX proxy_pass
- Frontend and backend containerised separately
- On request, Retrieves most recent data from DynamoDB
- On request, if recent data not up to date, send URLs to store in redis

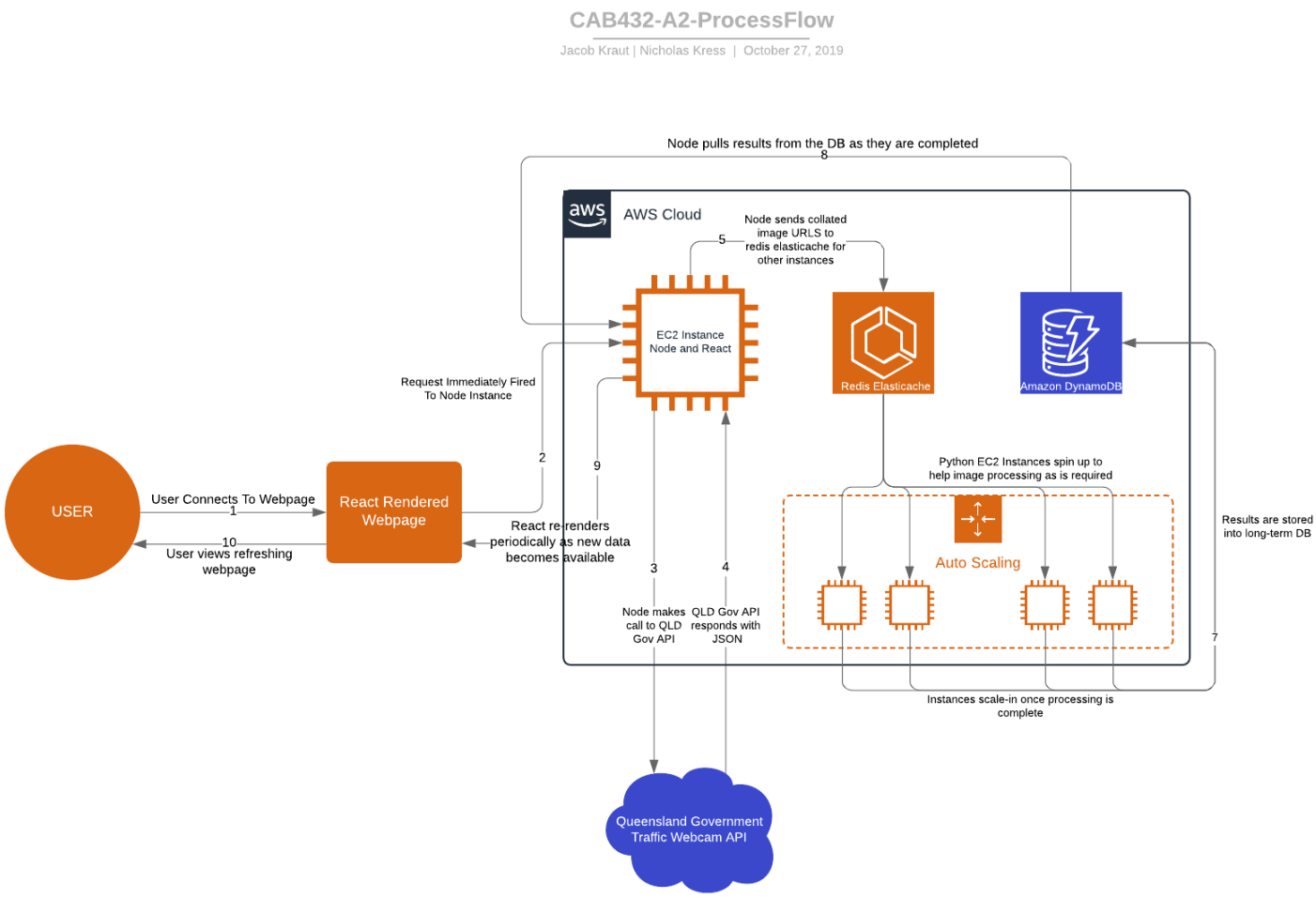
Python EC2 compute instances

- Uses python specific libraries including NumPy and openCV
- Setup to auto-scale based on load
- Continuously waits for unresolved values in redis cache to become available
- Process values from redis cache and store in DynamoDB

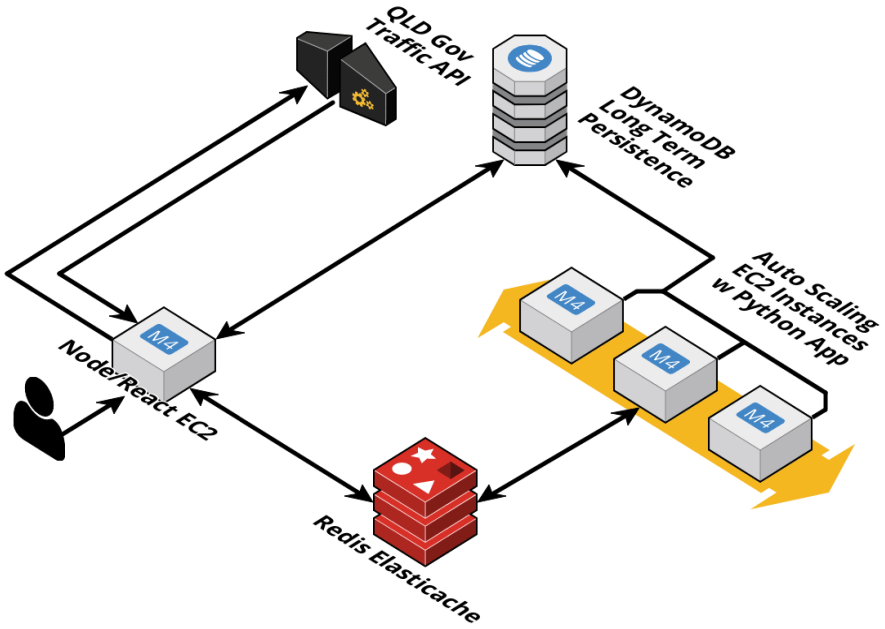
DynamoDB JSON Store

- Long term persistence
- Stores results of computed images for as long as is required
- Serves data on request

Process Flow Diagram



Cloud Network Architecture Diagram



Test Plan

Test	Expected Outcome	Actual Outcome
Normal Operation		
Load webpage	Page with navbar appears and initially a spinner shows loading.	PASS
Table and graph load	Loading spinner disappears and content loads into frame	PASS
Navbar updates with last fetched time	Shows "Loading…" until loading is complete and then shows most recently fetched time.	PASS
Click fetch button on navbar	Small delay and then new data should start to come through, as well as the timestamp updating.	PASS
Graph updates to reflect changes	As data comes through graph should dynamically update to match new incoming data.	PASS
Table updates to reflect changes	As data comes through table should dynamically update to match new incoming data.	PASS
Hover mouse over 1 arm on polar graph	Tooltip should show up showing location and its car count, arm should also change colour to a different colour from remaining arms.	PASS
Page navigation on table - Next	After the table has been populated with enough entries, clicking next should change to next page.	PASS
Page navigation on table - Previous	After the table has been populated with enough entries, clicking next and then previous should take you from one page back to original page.	PASS
Extending displayed rows in table	Selecting dropdown menu and changing to additional rows should extend the table down the page and bring up a scroll bar.	PASS
Clicking Location header in table	Should organise the results by alphabetical order and clicking again should inverse it.	PASS
Clicking Car Count header in table	Should organise results by lowest first and clicking again should organise by highest first.	PASS
Change page number in table entry box	Either using the up and down arrow or by entering a number in the page box and pressing enter should navigate to that page number.	PASS
Error Scenarios		
Page cannot access the /list endpoint.	Spinner disappears and error message displays in the middle of the page alerting user to failure.	PASS
Page cannot access the /latest endpoint.	Spinner disappears and error message displays in the middle of the page alerting user to failure.	PASS
Click fetch data button	If server is unavailable or there is an error, button should turn red with error message and return to normal once operation is successful again.	PASS

Difficulties / Exclusions / Unresolved & Persistent Errors

This assignment was executed well overall and we were surprised at how smooth the whole process turned out to be. There was some initial difficulty in getting all the gears working together but once we had the hang of it, the rest was straightforward.

Extensions

Currently our application only services roads that are within the state of Queensland, in part because we reside there and also because they have such an open and available API to access. A reasonable extension to this application could be the addition of traffic counting Australia wide. This scenario would only be possible with available webcam APIs in their respective state. Another potential extension could be making further use of long-term storage and steadily collate and display data over longer periods of time. Historical data could be used for various purposes but ideally would begin to show a story about which roads – major arterial or not – are the busiest over certain times of day/month/year and any other applicable time periods for historical access. This extension would only become useful the longer the application remains live.

User Guide

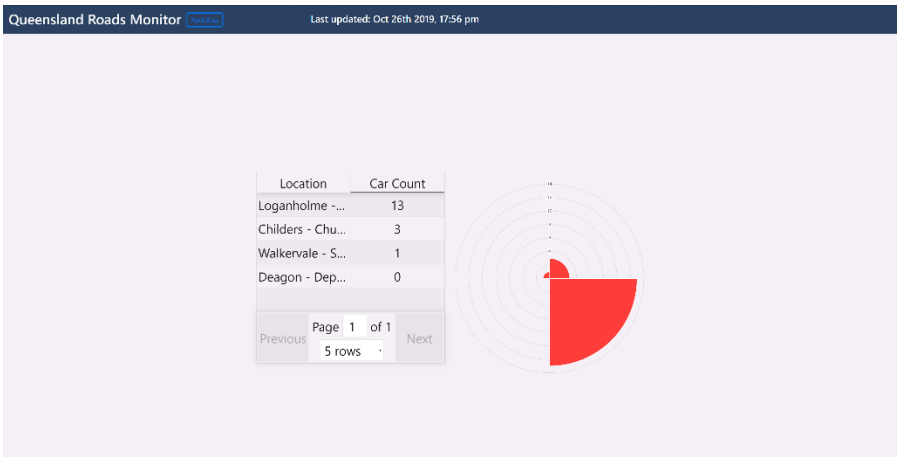
This application was built to be very self-intuitive and simplistic and implementation in this aspect was very successful. The user guide is as follows.

- Access the webpage – This will bring you to a screen similar to the one below.

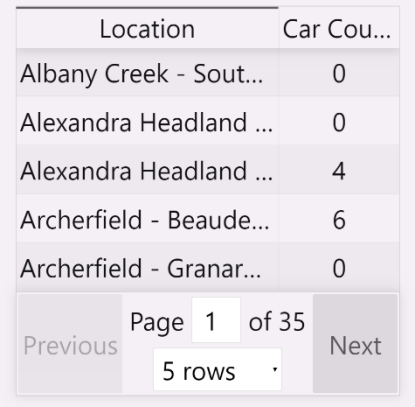
- Don't be concerned about the lack of information – it's just loading and once it's complete you will see a screen similar to below. The data and time on the navbar indicate the most recently processed data that's available. Below the navbar is the data displayed in an interactive table and a polar graph.

- To refresh the data to the current minute, go to the navbar and click on the fetch data button. This will tell the server you'd like new information processed.

- After a few seconds your webpage will now look similar to below. This data will progressively update as the server processes more images and sends the data through. This includes the table and the graph which will also update dynamically.

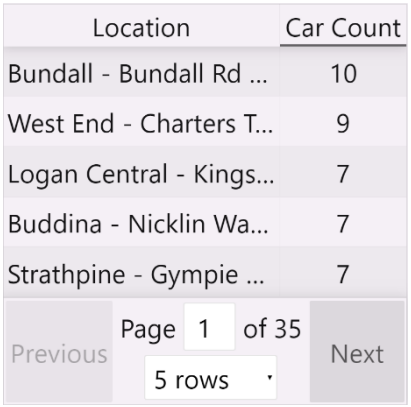


- At any time – including while the data is updating – you can use the table to organise the results by total car count or by alphabetical location using the tabs at the top of the table. Simply click on either one to arrange the data how you like, for count its lowest to highest and reversible and for location it's alphabetical starting from A and reversible. Below are 2 images showing sorting by location and by car count.



Location	Car Cou...
Albany Creek - Sout...	0
Alexandra Headland ...	0
Alexandra Headland ...	4
Archerfield - Beaude...	6
Archerfield - Granar...	0

Previous Page 1 of 35 Next
5 rows



Location	Car Count
Bundall - Bundall Rd ...	10
West End - Charters T...	9
Logan Central - Kings...	7
Buddina - Nicklin Wa...	7
Strathpine - Gympie ...	7

Previous Page 1 of 35 Next
5 rows

- Navigating between pages is easy, simply click the next or previous buttons to move between pages – noting that previous or next will be disabled if there doesn't exist a previous or next page respectively.

Location	Car Cou...
Archerfield - Ipswich...	1
Ashmore - Currumb...	1
Bald Hills - Gateway ...	0
Bargara - Bargara Rd...	1
Beenleigh - George ...	1

Previous Page 2 of 35 Next

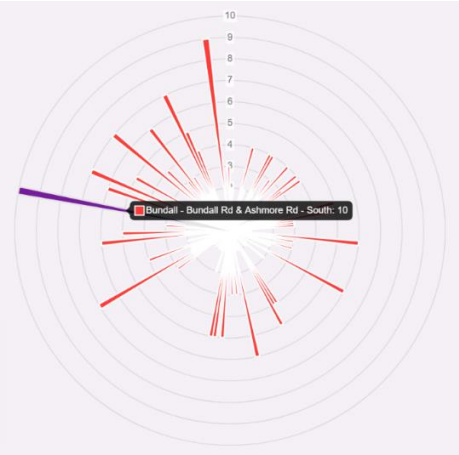
5 rows

Location	Car Cou...
Archerfield - Ipswich...	1
Ashmore - Currumb...	1
Bald Hills - Gateway ...	0
Bargara - Bargara Rd...	1
Beenleigh - George ...	1

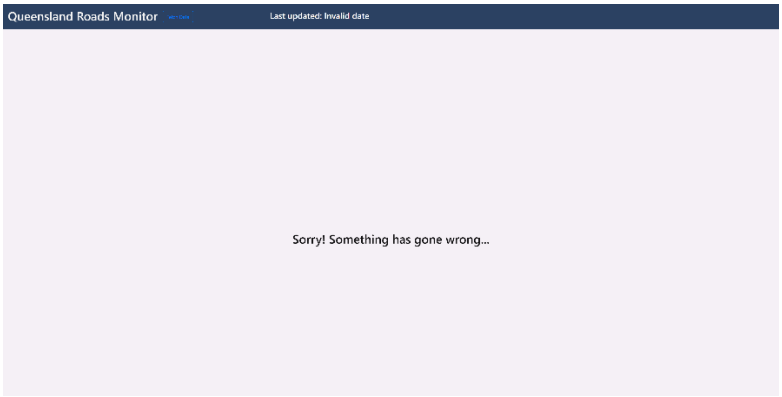
Previous Page 2 of 35 Next

5 rows

- The polar graph that accompanies the table makes it easy to see the busiest roads by extending their graph arm out further on the polar. To see what location each arm is representing, hover the mouse cursor over the arm and it will show a floating tooltip with location and count. To make it easier to see which arm you're hovering over, the arm will change to a different colour to the remaining arms.



- Lastly, on occasion things can fail on the server-side and this is represented when you see the image below. When this happens, we recommend waiting a while and then trying again to see if the issue has been resolved.



References

We have not directly quoted any sources in our report. Instead I've provided links to all of the libraries and API's we have used.

Queensland Traffic API Specification

https://www.data.qld.gov.au/dataset/131940-traffic-and-travel-information-geojson-api/resource/0948e11f-b250-4e1d-9395-376ebcb1a66b

React Bootstrap

https://react-bootstrap.github.io/

Amazon EC2 Instance

https://aws.amazon.com/ec2/

Amazon DynamoDB

https://aws.amazon.com/dynamodb/

Python

https://www.python.org/

NodeJS

https://nodejs.org/en/

ReactJS

https://reactjs.org/

ChartJS

https://www.chartjs.org/