2022

IP Locator

CAB432 Assignment 1

Ibrahim Cassim

n10627928

8/12/2022

Contents

Introduction

Mashup Purpose & description

The app/web service that has been developed counts page viewers as well as locating users for a certain IP address. It is provided as a React app through express and has been 'Dockerised' and then deployed on an EC2 instance.

Services used

The API endpoints used for the service are 2 different sites for IP information and the IP's respective locations. The other services include AWS's S3 object storage service, an AWS EC2 instance to run a Docker container with our application.

Ip-api.com

Retrieves information about a requester's IP or queried IP in a JSON format, including information about geolocation. This is the API used for the backend (on the express server).

Endpoint: http://ip-api.com/json

Docs: https://ip-api.com/docs

Ipapi.co

ipapi.co provides a REST API to find the location of an IP address. This is essentially the same functionally as the previous API, the reason this is used additionally for the client side is explained in the "Technical Breakdown" section.

Endpoint: https://ipapi.co/json/

Docs: https://ipapi.co/api /

Amazon S3

Object storage built to retrieve any amount of data from anywhere. Amazon S3 is used for the application to have a bucket which stores an object keeping track of the visitor count persistently.

Docker

The app was wrapped in a docker image and deployed in a docker container. The image can be found https://hub.docker.com/repository/registry-1.docker.io/ibcassim/assignmentone. However this is a private repository.

Mashup Use Cases and Services

IP & Location Information for Users

As an	Internet surfer
I want	To be able to view my IP address and its respective location
So that	I may see what servers and others see as my IP address and where I'm located (or where my VPN is located).

The user will be able to see their IP and their respective location seen from their IP address as well as their server's IP address. This is particularly useful for VPN users who may want to see their specific IP address as well as local servers that AWS can deploy from (assuming the server is deployed in various locations around the world). This is done using ip-api.com for the server and ipapi.co for the client.

Technical breakdown

Architecture and Data Flow

The server is an express server which listens to certain API routes (particularly the /visitorcount route and the route /ipaddress?ip=num.num.num.num) and sends a React application for all other routes, which consists of the root '/' directory. The app is wrapped in an Ubuntu docker image, and a container is deployed on an EC2 instance.

The app uses the aws-sdk library and functions regarding S3 buckets to keep track of the count of visitors. A JSON file (object) that is preconfigured is uploaded to an Amazon S3 Bucket. Every time the /visitorcount route is called, the server sends off the count in the JSON format. It then proceeds to increment the visitor count and sends a new JSON object to the bucket, as seen in the following screenshot:

```
router.get('/visitorcount',function(req, res, next){
    //This requires that you already have set up buckets on S3 and have tokens on ~/.aws/config

    getVisitorCount()
        .then((resp) => {
            //The responseJSON is given in template of the ../counter.json file
            //Body is given as a buffer, so have to convert to string, then JSON
            const responseJson = JSON.parse(resp.Body.toString());
            res.status(200).send(responseJson);

            const updated = responseJson.visitorCount + 1
            updateVisitorCount(updated)
        })
        .catch(function(err) {
            console.error(err, err.stack)
            res.status(400).send({visitorCount: NaN})
        });
})
```

The /ipaddress route requires the IP of the client in the REST format, and then the server retrieves its own IP information and queries the client's IP information to the API given by ip-api.com. The server then sends the IP and location information back to the client in a JSON format.

```
function App() {
  return (
    <div className="App">

      <header className="App-header">
        <p>
          <VisitorCount url="/visitorcount"/>
          <IpInfo ipaddressurl = "/ipaddress"/>
        </p>
      </header>

      <footer>
        <span>
          Contact n10627928@qut.edu.au for any inquiries.
          <br />
          Copyright &copy; 2022. All Rights Reserved.
          <br />
          Credits to ip-api.com and ipapi.co for use of API
        </span>
      </footer>

    </div>
  );
}
```

The server sends a build of the React application to the client side, which contains the necessary files such as the CSS and JS files. Particularly, the application is formatted with 2 main components, one for the count of visitors and another for the IP information. These components are seen in the screenshot.

The count of visitors is easily handled with a hook and a fetch to the API, which is easily retrieved as the API gives a JSON object.

The IpInfo component requires a bit more work, as the client has to retrieve its own IP, which it does through the ipapi.co API. The reason why this ipapi.co is used specifically for the front-end, is because ip-api.com serves its information via http, but browsers default to https, which for ip-api.com is only reserved for paid members. So the client is unable to retrieve it's own IP as it is met with an error message. The API for ipapi.co does not have this problem thus it is used for the front end. Additionally, ip-api.com may have issues with CORS. Hence, this is

why 2 different APIs are used, as only the API call for the client side had to be changed when the issue arose after testing.

Deployment and the Use of Docker

The app was made into a Docker image, it uses Ubuntu as the base, installs NodeJS, NPM and necessary components. When a container is made, environment variables are set through the docker run command, for ID, keys and token given by AWS to use the S3 bucket.

There may be extra unnecessary installations in the Dockerfile, however these were not tested for and removed, as a local build was taking more than 10 minutes each, so to isolate and remove all unnecessary installations is unrealistic for minor optimisations.

There is also extra configuration required by uploading a JSON file named counter.json to the bucket with only one field “visitorCount” which is set to 1. This is also set for public-read access. This file has also been included in the server file for ease of use.

Test plan

Task	Expected Outcome	Result	Screenshot (Appendix)
Main page (/ directory)	All info visible, showing correct results. With the React app.	PASS	01
Visitor count on main page	Correct information	PASS	02
Visitor Count updating on each visit or refresh of main directory	Visitor count should increment by one	PASS	03
Visitor count directory	JSON object with a field for the count shown correctly	PASS	04
Ipaddress API to the server with a queried IP	Correct information shown for the queried IP and server IP in a JSON obj	PASS	05
Ipaddress route with no query IP	JSON object showing an error	PASS	06
Non existing route	Should show main page with no issues	PASS	07

Difficulties / Exclusions / unresolved & persistent errors /

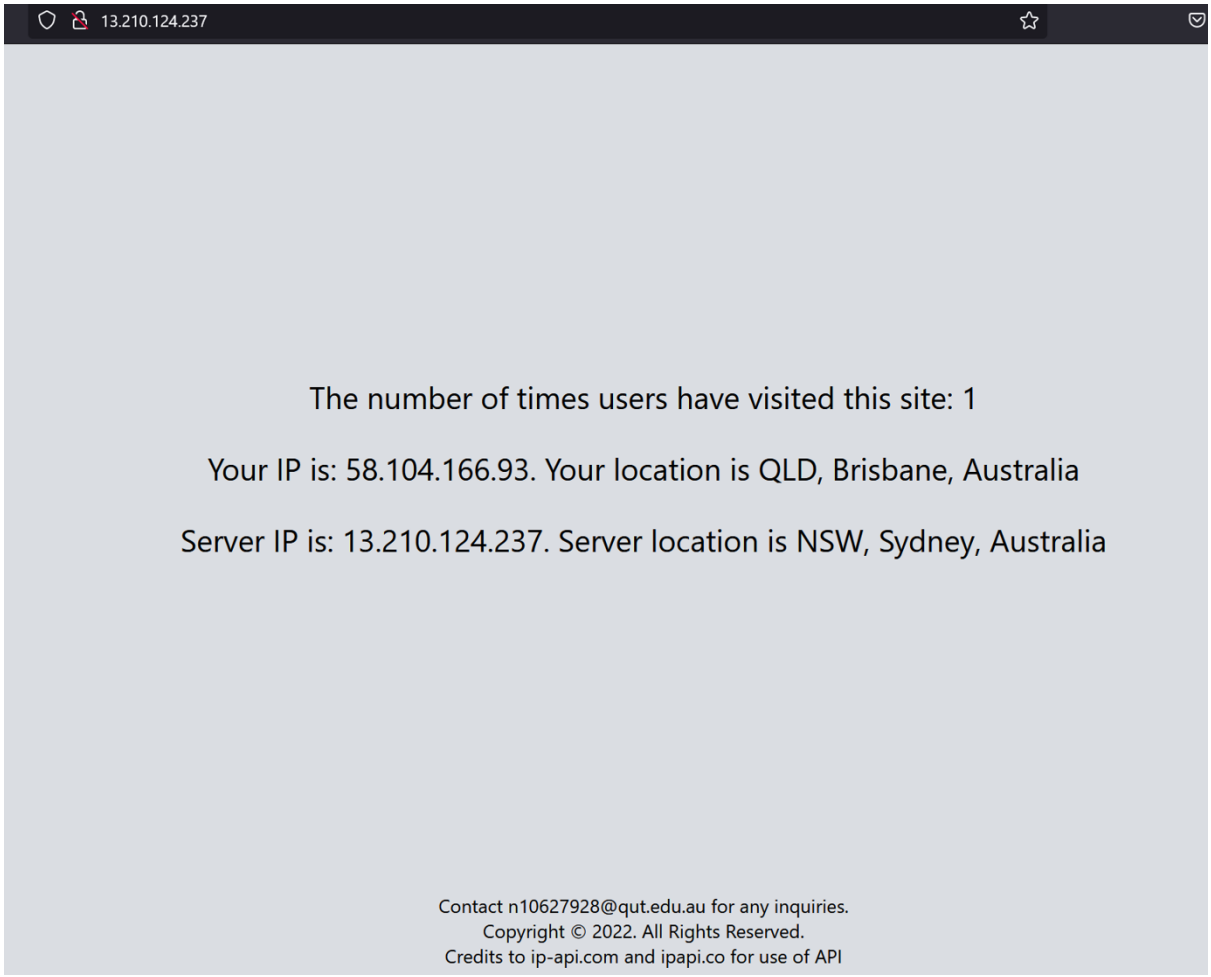
As discussed previously, there were mainly two problems. The first being the issues with the ip-api.com API, as the client could not receive its own API the problem was either due to CORS settings, which was what was shown on the developer tools, or the issue is that browsers default to https, and upon failure they try http. So when the browser for development was being used (Firefox), it may have tried retrieving the client IP from ip-api.com but tried to use the https, and it successfully received a result, however this was the undesired result as there was an error message and not the IP info. It may be that browsers have the https behavior for API retrievals on a page. Or the issue may have been due to CORS. Either way, the solution was using a different API service for the client side.

There may a be problem with the extension Ublock Origin and the displaying of information, thus it needs to be disabled for optimal results.

The second major issue is with bloat in the docker image. Each build was taking more than 10 minutes, thus trying to single out the best result was unrealistic. There was also a problem also trying to use Ubuntu 18.04 as the base, as it installs NodeJS and NPM with certain versions that fail when running `npm install`, thus a later version was required.

User guide

The application is entirely served on the / root directory as seen in the screenshot. It displays the number of visitors to the site and the user's and server's respective IP and geolocation information.



Analysis

Question 1: Vendor Lock-in

There are 4 services being relied upon, first ip-api.com, secondly ipapi.co, thirdly Amazon S3, and lastly Amazon EC2. Regarding the first two, these can be very easily replaced as a simple Google search returns many, many different APIs for obtaining relevant information for IP addresses. Even relying on an API is unnecessary as the information can be developed by itself and then a database can be manually used to retrieve the necessary information. Regarding the latter two, particularly Amazon's services, it requires slightly more effort, however reliance solely on them is certainly not the case. In the case that Amazon cannot be used, all that is required is using any other cloud vendor such as GCP or Azure for persistence, which would change the buckets.js into code relevant for their respective cloud vendor. Deployment on a virtual machine that is not Amazon EC2 can be dealt with, with the same approach by deploying on another cloud vendor.

For commercial scale, reliability is not much of a worry as Amazon guarantees 99.99% availability time for EC2, 99.9% uptime for S3 bucket and 99.999999999% guarantee of data durability. Any more and they can be legally responsible. (Quinn, 2021)

Some examples of relevant competitors for IP APIs are: https://ipapi.com/, https://www.ipify.org/, https://ipgeolocation.io/. Only the JSON format and fetch URL would need to be changed for these.

For persistence: https://azure.microsoft.com/en-au/products/storage/blobs/, https://cloud.google.com/storage. There are guides on migration from S3 buckets, such as https://cloud.google.com/storage/docs/migrating however migrating will still not be difficult as only buckets.js will change & relevant environment variables.

For Virtual Machines: https://azure.microsoft.com/en-us/products/virtual-machines/, https://cloud.google.com/compute. Changing to these is simple as using a different VM and doesn't require much change.

Question 2: Container Deployment

Containers require making images, for large applications, building a new image for every change can be time consuming. Additionally, debugging and viewing errors during production can be difficult for containers. Problems with configuration can also be a large problem as that might require man time for an IT team to see if the issue is with the code or application or VM or container or configurations. Deploying to a VM can make configurations easier, with auto scaling services however can be more resource consuming.

Persistence cannot exist at the container level and applications have to be stateless if it wants to balance load. Thus persistence would have to be specific to a shared environment such as a cloud SQL service or something like S3.

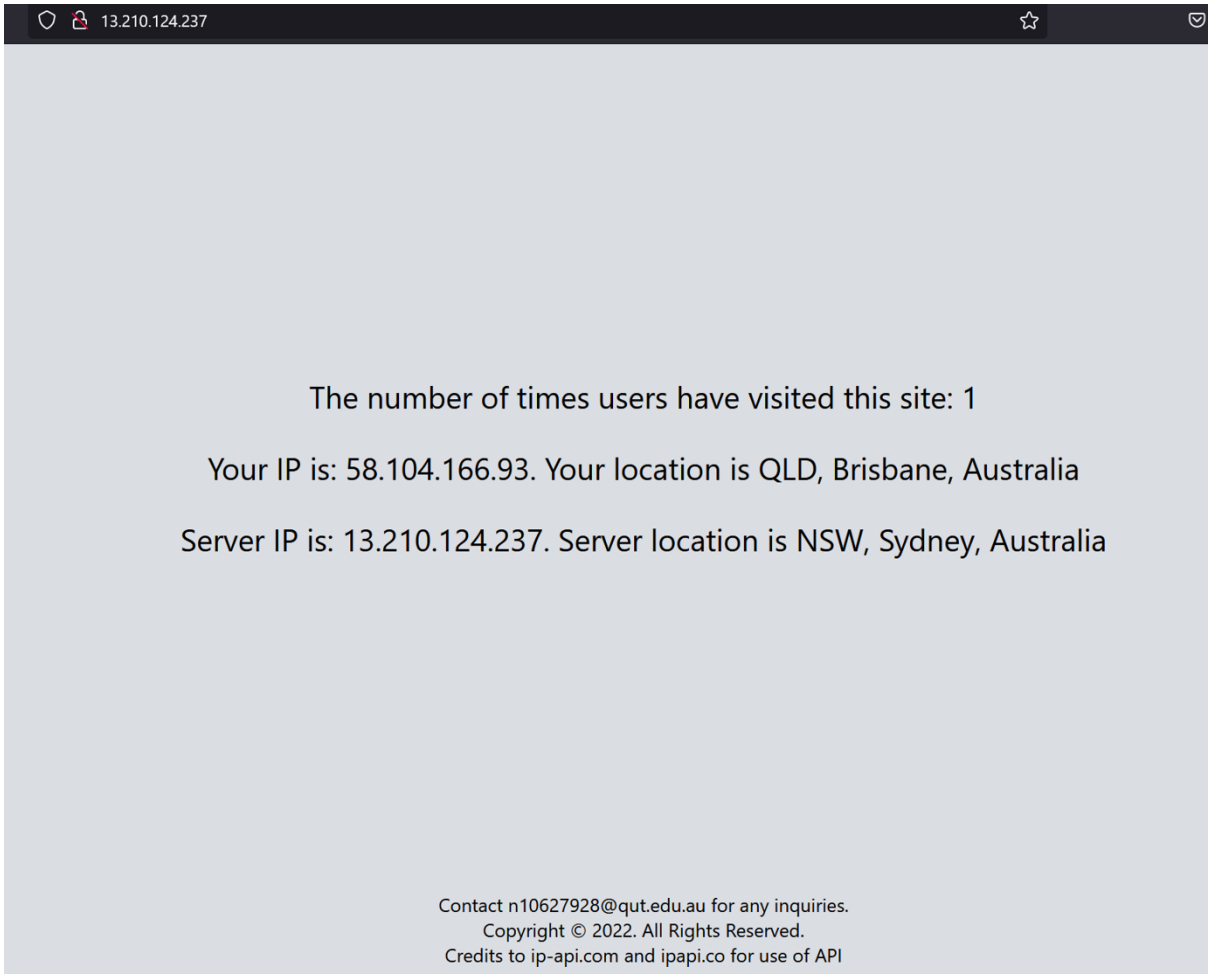
References

Corey Quinn https://www.lastweekinaws.com/blog/s3s-durability-guarantees-arent-what-you-think/ (04/21/2021)

Appendices

Test suite screenshots

01

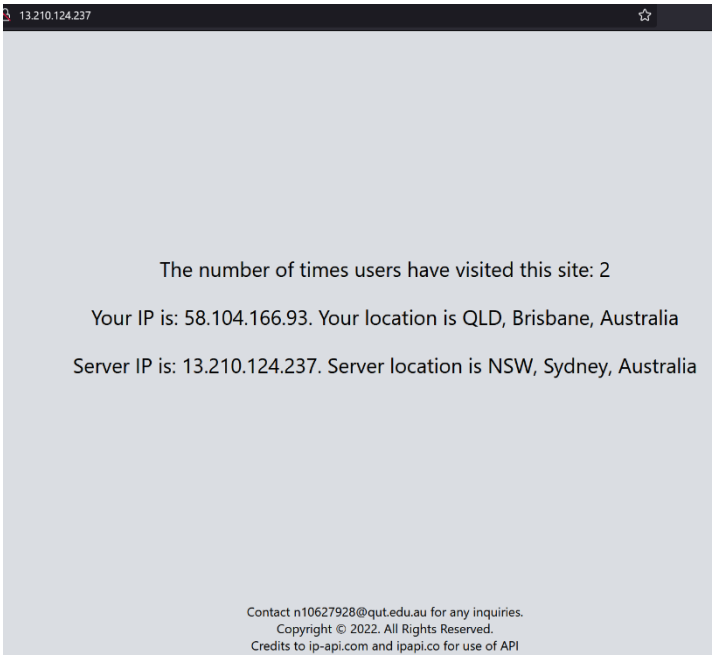


02

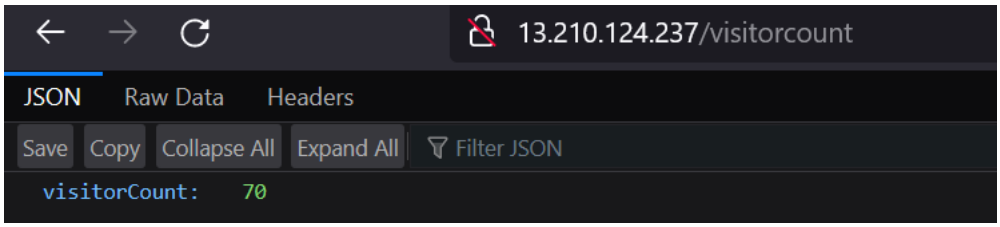
See above

03

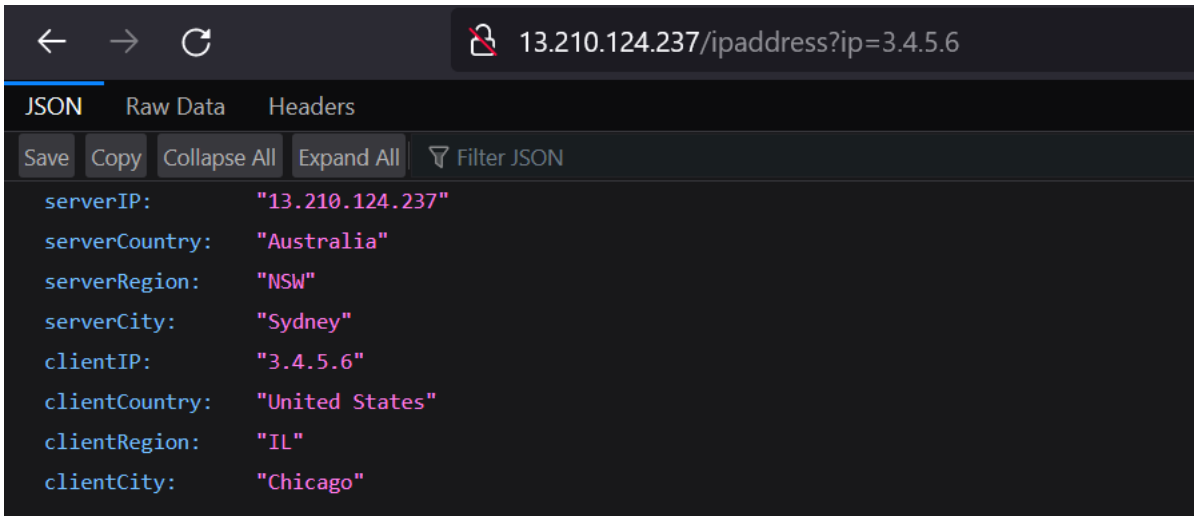
After refresh:



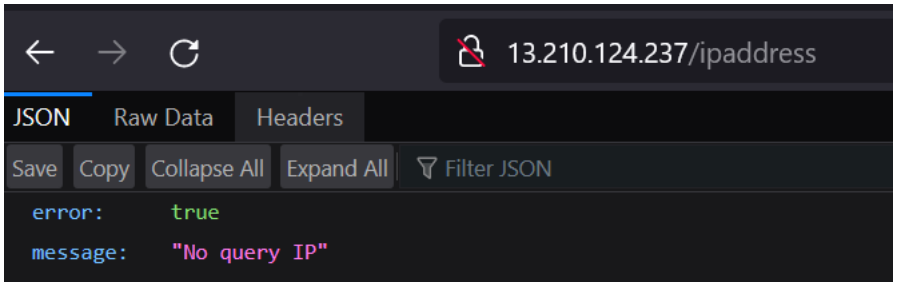
04


13.210.124.237/visitorcount
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
visitorCount: 70

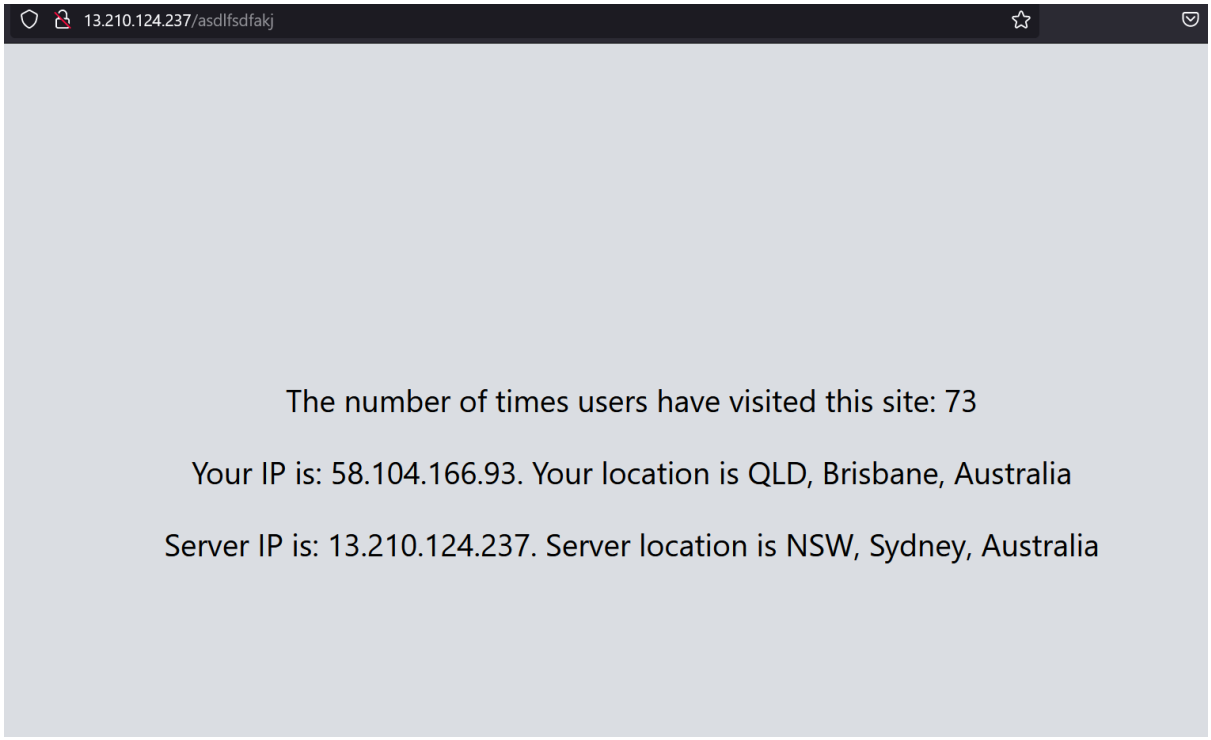
05


13.210.124.237/ipaddress?ip=3.4.5.6
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
serverIP: "13.210.124.237"
serverCountry: "Australia"
serverRegion: "NSW"
serverCity: "Sydney"
clientIP: "3.4.5.6"
clientCountry: "United States"
clientRegion: "IL"
clientCity: "Chicago"

06


13.210.124.237/ipaddress
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
error: true
message: "No query IP"

07


13.210.124.237/asdlfsdfakj
The number of times users have visited this site: 73
Your IP is: 58.104.166.93. Your location is QLD, Brisbane, Australia
Server IP is: 13.210.124.237. Server location is NSW, Sydney, Australia