Automation Logic

DevOps Assessment
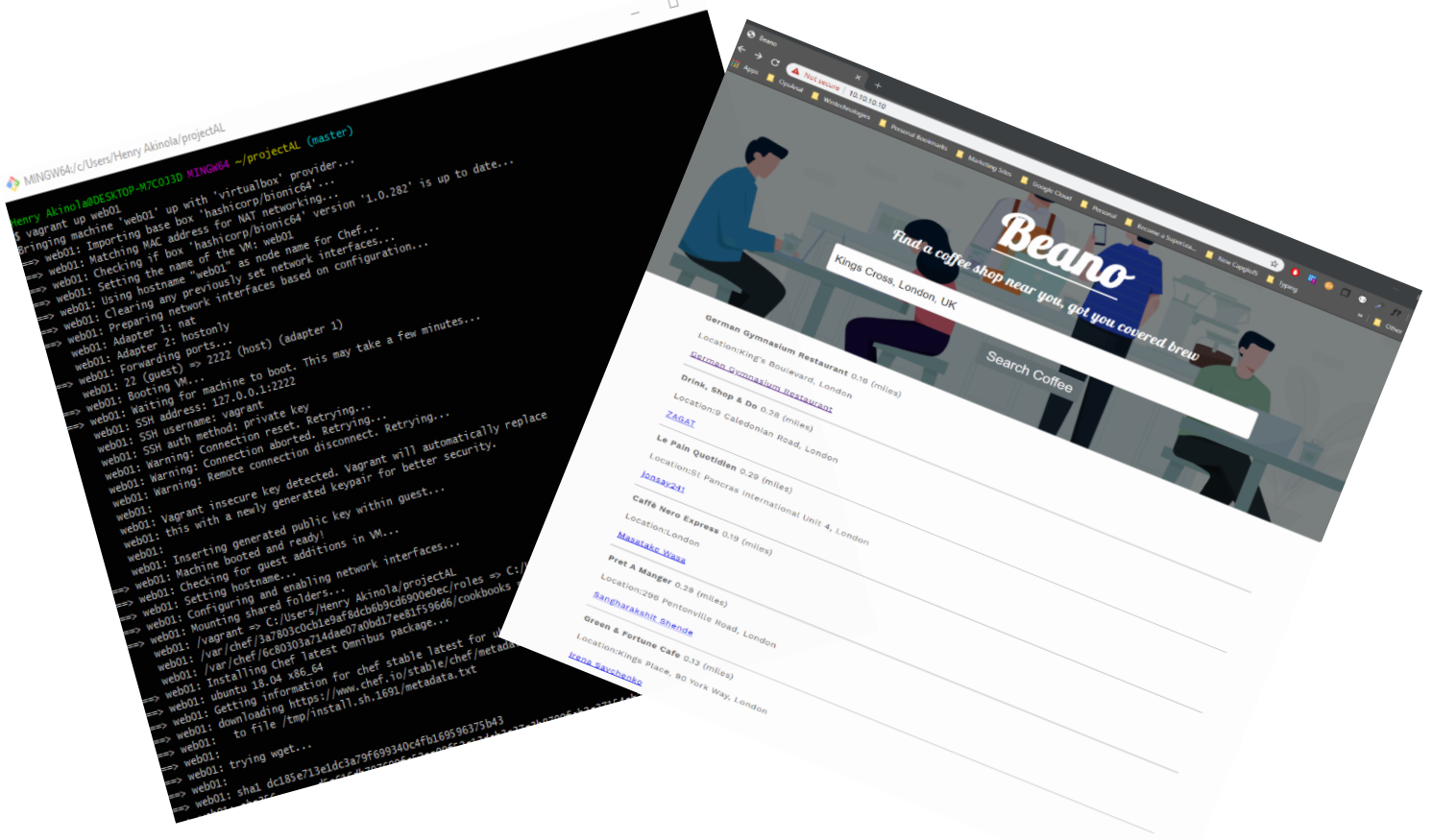
Written By
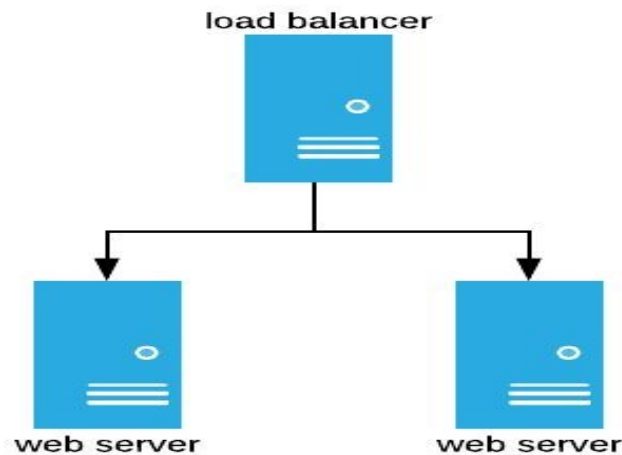
Henry Akinola

Summary

This document contains the steps taken to complete this project. It will walk you through the pre-requisite installation and setup, automated deployment process, infrastructure and descriptions of the tools used.

Project requirements:

2x Web servers – Hosting a web app (**Beano – find a cafe near you using google places API**)

*The webservers will host an app that will dynamically retrieve cafés within a 500m radius of the location entered in the search input (**Google Places API**).*

1x Load Balancer – Will act as proxy to evenly distribute traffic to the webservers

Vagrant – Will provide the ability to build and provision the servers with associated files using a simple *vagrant up* command (More information to follow)

Chef – Will be used as the infrastructure configuration manager (cookbooks *no pun*, will be distributed to the servers based on roles applied to the servers)

Test Script – shell script to test the deployed web apps

Pre-requisites

PLEASE NOTE: In order to achieve a consistent outcome, it is advised to use the same version stated in this document. The steps shown is on a windows device, however providing the same tools and versions installed, should produce the same result.

Enable virtualisation on PC (Windows Only)

This is required to allow the ability of being able to run virtual machines on your machine. The guide below provides good clear instructions on how to enable this feature

https://mashtips.com/enable-virtualization-windows-10/

VirtualBox – Installation steps

In use version 6.1

1. Select > https://www.virtualbox.org/wiki/Downloads

> Downloads

Download Vagrant

Below are the available downloads for the latest version of Vagrant (2.2.7). Please download the proper package for your operating system and architecture.

You can find the SHA256 checksums for Vagrant 2.2.7 online and you can verify the checksum's signature file , which has been signed using HashiCorp's GPG key. You can also download older versions of Vagrant from the releases service.

Check out the v2.2.7 CHANGELOG for information on the latest release.

Debian
32-bit | 64-bit

Windows
32-bit | 64-bit

Centos
32-bit | 64-bit

Linux
64-bit

macOS
64-bit

Select the download that is compatible with your operating system

2. Once downloaded navigate to your downloads folder and run the installation

3. Select **Next** all the way through, if prompted to install network interfaces select **Yes**

4. On The last screen select **Finish** to complete

Vagrant – Installation steps

Vagrant is a tool for building and managing virtual machine environments in a single workflow. With an easy-to-use workflow and focus on automation, Vagrant lowers development environment setup time, increases production parity, and makes the "works on my machine" excuse a relic of the past.

To summarise why will be using vagrant for this project, is to mainly speedup the deployment of the virtual machines not having to manually download an iso mount the drive and repeat this for each server.

With the use of vagrant, you can simply point your Vagrantfile to a hosted iso in vagrants repo and easily spin up a VM with a single command **vagrant up** <optional machine name>. More details to come as we continue to build this project.

Manages Virtual machines,
- Easily create/destroy VMs
- Start/Stop and restart VMs
- Access to VMs
- Networking VM settings

In use version 2.2.7

```
Henry Akinola@DESKTOP-M7COJ3D MINGW64 ~/projectAL (master)
$ vagrant --version
Vagrant 2.2.7
```

1. Download Vagrant installer from https://www.vagrantup.com/downloads.html selecting your operating system.
2. Run Vagrant installer, Select **Next** all the way through accepting licensing agreements
3. Once installed if prompted to restart select **Yes**

Vagrant plugins – ominubus, berkshelf (optional no community cookbooks used)

Git - Installation steps (Command line Tool)

Git is an opens source version control system used by developers. Version control systems keep these revisions straight, storing the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute.
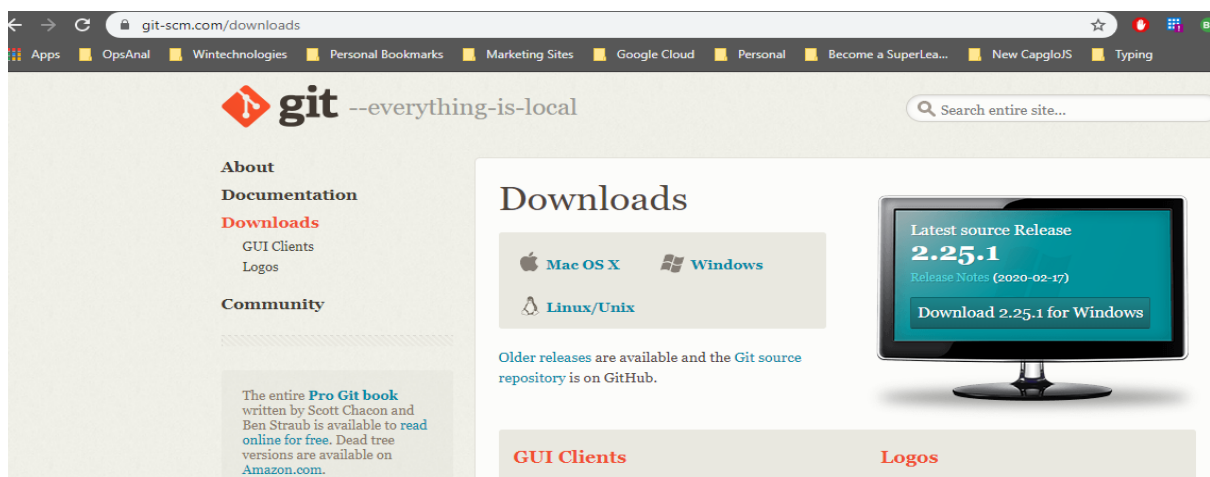
Download Git installed from https://git-scm.com/downloads
1. Once downloaded, the install should be found in your downloads folder
   c:/users/<yourusername>/Downloads
2. **Run** The download by double clicking
3. Select **Next** all the way through (**I Suggest to keep all the shortcut options checked**)

Chef-solo and not Chef Server?

https://blog.differentpla.net/blog/2014/11/13/which-chef/

Being that the scope of the project is to build a small infrastructure consisting of 3 servers, I thought it would be best to use chef solo instead of creating a chef server as it will require a lot more configuration.

Chef-solo is a much lighter version of chef, If I were to use Chef-server, this will require a workstation that has chef-development kit installed on it. Chef clients deployed to each server along with validator certs. Using Chef-solo chef and its required config is installed independently on the machine to run locally. With the use of vagrant, I am able to specify the location of the cookbooks and assign each server chef roles that will be used to deploy the correct configurations on the servers.

Manual Install is not required as this will be done at deployment phase.

Git Repository – Clone Project

Now that we have the tools that will be used out of the way. Next up is to clone the repository that contains all the vagrant file and associated project files.

You can either use your command prompt or simply use the terminal installed with the previous Git Bash install. **Git Bash Terminal used in steps shown**

To confirm the pre-reqs has been installed

command: **vagrant --version**
Check vagrant is installed and version

command: **vboxmanage --version**
Check Virtualbox version
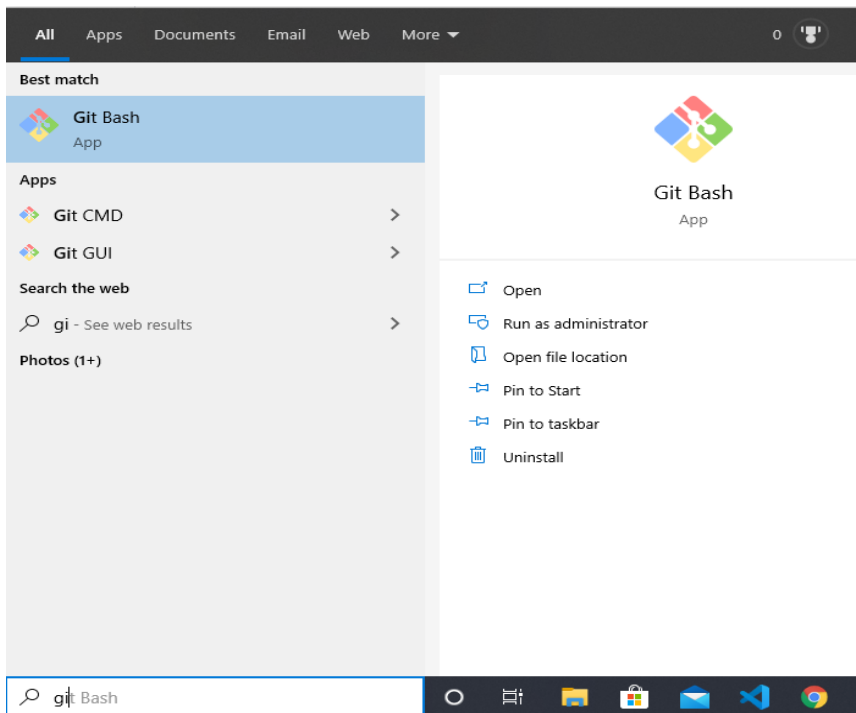
command (if not using git bash): **git --version**
Confirms git is installed

Providing no error found running the above commands, install the required plugins for our vagrant build:
command: **vagrant plugin install vagrant-omnibus**
command: **vagrant plugin install vagrant-vbguest**

1. Select **Start button** > type (**Git bash**)

2. Select **Git bash** you should be presented with a terminal screen

3. Make sure you are in a directory you wish to store the project files in. type **pwd** and press enter. This will show you the current directory you are in (Git bash usually opens in youre the root of your home directory)

4. *Optional step: if you prefer to load files in your Documents folder simply type cd ~/Documents*

5. Type: **git clone https://github.com/henry89/projectAL.git** and press enter. Desired outcome below

```
Henry Akinola@DESKTOP-M7C0J3D MINGW64 ~/Documents
$ git clone https://github.com/henry89/projectAL.git
Cloning into 'projectAL'...
remote: Enumerating objects: 267, done.
remote: Counting objects: 100% (267/267), done.
remote: Compressing objects: 100% (134/134), done.
remote: Total 267 (delta 71), reused 259 (delta 63), pack-reused 0
Receiving objects: 100% (267/267), 1.12 MiB | 2.50 MiB/s, done.
Resolving deltas: 100% (71/71), done.

Henry Akinola@DESKTOP-M7C0J3D MINGW64 ~/Documents
$
```

6. Navigate to the cloned folder. Type: **cd projectAL**

```
Henry Akinola@DESKTOP-M7C0J3D MINGW64 ~/Documents
$ cd projectAL/

Henry Akinola@DESKTOP-M7C0J3D MINGW64 ~/Documents/projectAL (master)
$
```

7. Providing that the pre-requisites install and setup has been completed, you should now be able to simply type: **vagrant up** in the terminal window

```
==> web01: +#https://nginx.org/en/docs/varindex.html
==> web01:
==> web01: * service[nginx] action enable
==> web01:   (up to date)
==> web01:    * service[nginx] action restart
==> web01: [2020-02-26T09:52:47+00:00] INFO: service[nginx] restarted
==> web01:
==> web01:    - restart service service[nginx]
==> web01: [2020-02-26T09:52:47+00:00] INFO: Chef Infra Client Run complete in 7
3.685747706 seconds
==> web01: Running handlers:
==> web01: [2020-02-26T09:52:47+00:00] INFO: Running report handlers
==> web01: Running handlers complete
==> web01: [2020-02-26T09:52:47+00:00] INFO: Report handlers complete
==> web01: Chef Infra Client finished, 17/20 resources updated in 01 minutes 15
seconds
==> web02: Importing base box 'hashicorp/bionic64'...
==> web02: Matching MAC address for NAT networking...
==> web02: Checking if box 'hashicorp/bionic64' version '1.0.282' is up to date.
..
```

The install should take around 10 minutes to deploy. Once completed you should be met with the following screen

```
==> loadbalancer: [2020-02-26T10:02:55+00:00] INFO: Chef Infra Client Run comple
te in 84.072170394 seconds
==> loadbalancer: Running handlers:
==> loadbalancer: [2020-02-26T10:02:55+00:00] INFO: Running report handlers
==> loadbalancer: Running handlers complete
==> loadbalancer: [2020-02-26T10:02:55+00:00] INFO: Report handlers complete
==> loadbalancer: Chef Infra Client finished, 7/10 resources updated in 01 minut
es 25 seconds

Henry Akinola@DESKTOP-M7C0J3D MINGW64 ~/Documents/projectAL (master)
$
```
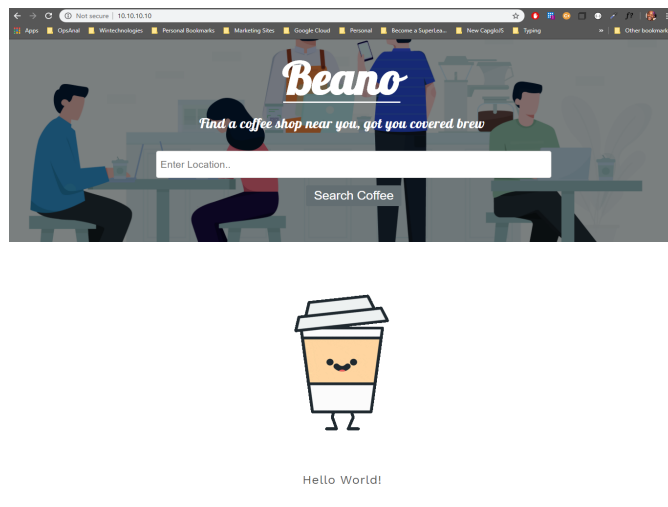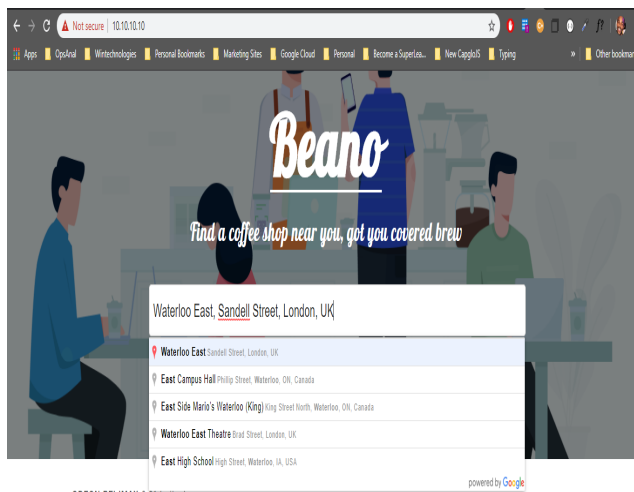
Testing the web App (Web browser)

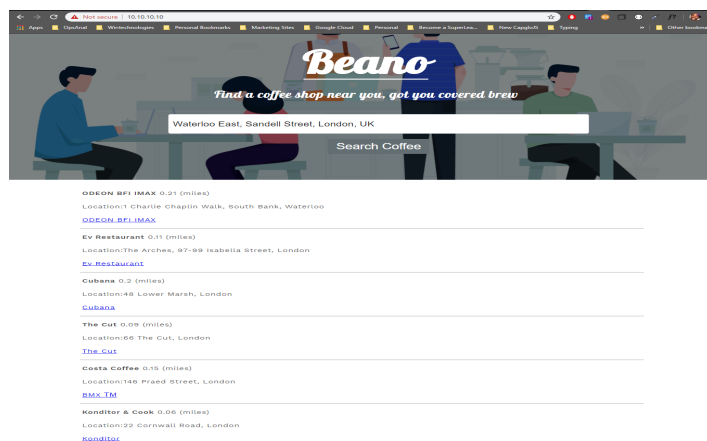- In your web browser navigate to https://10.10.10.10

You should be presented with the following webpage

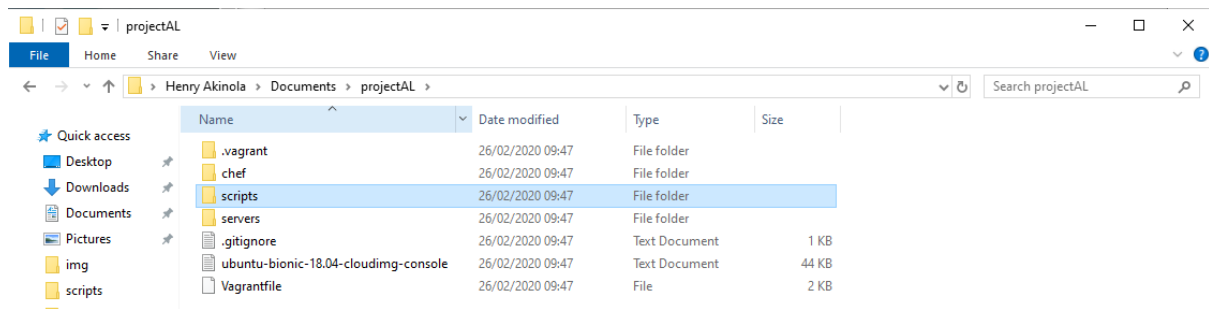- You can now type in and search a location within the input

- Upon selecting your location, you will be able to see a list off cafes with in a 500m radius of the specified address. (Asynchronous request – no page refresh)
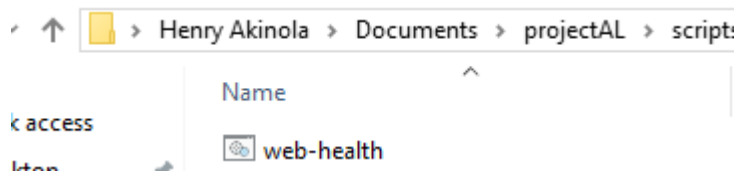
Testing the Web App (Shell Script)

Within the files/folders cloned from the git repository, there is a directory called scripts in the root folder

1. **Open** the scripts directory
2. **Double click** on the **web-health** shell script

The following screen should show with the output results of the test

Providing that requests where successful you should see a success message of status ok on all checks

web02/01 192.168.10.11 Status - OK

Configurations – Technical Overview

Directory list & Server Inventory (Tree)

Below is the server list configuration and the directory listings. I decided to go with putting the basic server configs into a yaml file instead of hardcoding it in the vagrant file, this will make it easier to make adjustments to the server set ups if/when required. I limited it to just sever names and IPs. this can be broken out even further supplying information such as the CPU, memory, OS and much more.
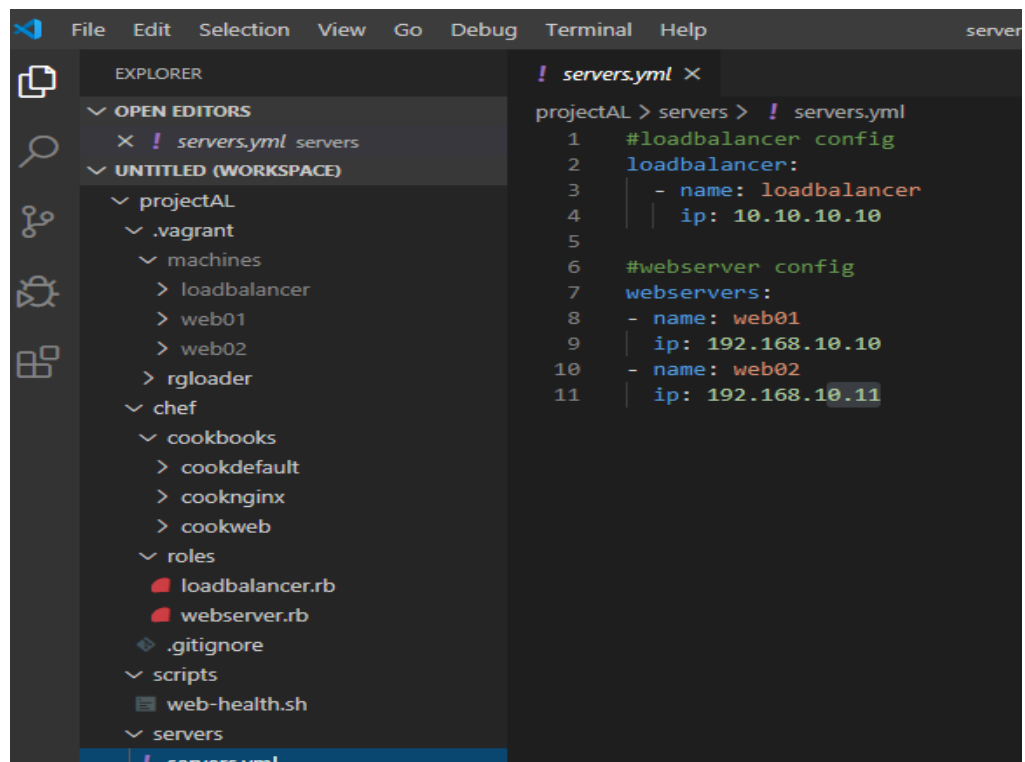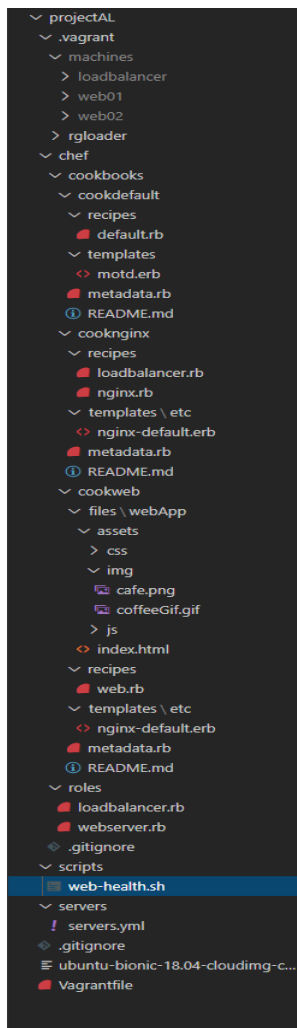
Vagrant setup

The following script executes the following:

- Includes the **YAML** module to allow for the import of the **server.yaml** file that will be used later on in the script.

- Vbguest has been set to auto update
  Experienced issues when attempting to install packages, installing the updated resolved the issue.

- The script first loops through the webservers listed in the **servers.yml** and begins to set up
  - The box **hashicorps/ubutu64** box is then pulled from vagrant cloud
  - Hostname of the server is set using the name entry found in the **servers.yml**
  - The hypervisor is the triggered to start creating the virtual machine with the specified name.

```ruby
### Web servers ###
servers["webservers"].each do |host|
  config.vm.box = "hashicorp/bionic64"
  config.vm.define host['name'] do |define|
    define.vm.hostname = host['name']
    define.vm.provider "virtualbox" do |vb|
      vb.name = host['name']
```

  - SSH is enabled and the private IP address is using the IPs specified in **servers.yml**

The chef provisioning block will take affect:
  - Taking care of licence agreement prompt (No user interaction required)
  - Assigning the provisioning path this is automatically set (Optional - details on why it has been set in Troubleshooting chapter)
  - Chef cookbooks path set (root of all cookbook directories) this will be assigned to a path within the virtual machine at build along with the roles

```ruby
define.vm.network "private_network", ip: host['ip']
define.ssh.forward_agent = true
define.vm.provision "chef_solo" do |chef|
  chef.arguments = "--chef-license accept"
  chef.provisioning_path = "/var/chef"
  chef.cookbooks_path = ["chef/cookbooks"]
  chef.roles_path = "chef/roles"
  chef.add_role ('webserver')
```

Chef Provisioning and cookbooks

Roles

Roles where used to avoid the need to have to specify the run list within the vagrant file for each server, run list meaning what recipes/configurations that should run against the server. Chef will automatically look in the roles directory to find the roles and run the recipes defined within its role file on servers tagged with that particular role.

Roles Directory

```
∨ roles
   🔴 loadbalancer.rb
   🔴 webserver.rb
```
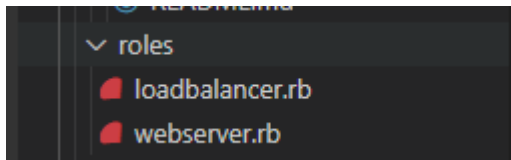
Vagrant file – applying roles

```
53          chef.cookbooks_path = ["chef/cookbooks"]
54          chef.roles_path = "chef/roles"
55          chef.add_role  ('loadbalancer')
56        end
```

Load balancer role – displays run list

```
projectAL > chef > roles > 🔴 loadbalancer.rb
1    name "webserver"
2    description "installs all recipes required to for the loadbalancer"
3    run_list 'recipe[cookdefault::default]','recipe[cooknginx::nginx]','recipe
     [cooknginx::loadbalancer]'
4    |
```

Webserver role – displays run list

```
projectAL > chef > roles > 🔴 loadbalancer.rb
1    name "webserver"
2    description "installs all recipes required to for the loadbalancer"
3    run_list 'recipe[cookdefault::default]','recipe[cooknginx::nginx]','recipe
     [cooknginx::loadbalancer]'
4    |
```
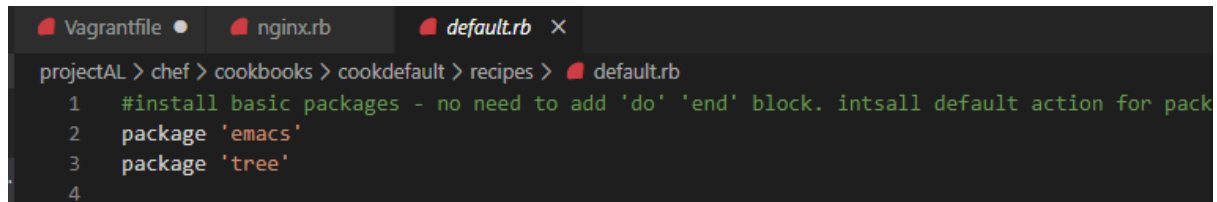
Default recipe

This was set up initially to test cookbook settings

- Installs emacs and tree

```
Vagrantfile ●     nginx.rb        default.rb  ×

projectAL > chef > cookbooks > cookdefault > recipes >  default.rb
   1    #install basic packages - no need to add 'do' 'end' block. intsall default action for pack
   2    package 'emacs'
   3    package 'tree'
   4
```

Nginx recipe
- Install Nginx

- Comment out **octect stream** line – when navigating on the website, it causes the web page response to be returned as binary data ultimately leading to the browser downloading it as it does not know how to interpret it
-
- Updated the **/etc/nginx/sites-enabled** default file with a commented out line.
  This file is included in the main nginx config **/etc/nginx/nginx.conf.**

  I could have simply used the same function to replace the included sites default line in the nginx config, however I wanted to utilise the chef **file resource** functionality to show some of the other options available within **Chef**.

- The Last config is to enable and start **Nginx**

```
projectAL > chef > cookbooks > cooknginx > recipes >  nginx.rb
   1    #install nginx, enable and start service and auto start when server is on
   2    package 'nginx'
   3
   4    ruby_block 'comment line out' do
   5        block do
   6            f = File.open('/etc/nginx/nginx.conf').read
   7            f.each_line do |line|
   8                #comment out octet-stream
   9                if line.match('default_type application/octet-stream;')
  10                    print (" - Octet stream line found")
  11                    print (" - Updating default_type application/octet-stream; to #defaul
                        application/octet-stream; - ")
  12                    fe = Chef::Util::FileEdit.new('/etc/nginx/nginx.conf')
  13                    fe.search_file_replace_line('default_type application/octet-stream;',
                        "#default_type application/octet-stream;")
  14                    fe.write_file
  15                end
  16            end
  17        end
  18    end
  19
  20
  21    #remove the contents in sites enabled
  22    file '/etc/nginx/sites-enabled/default' do
  23        content '#will not be used - please refer to the file inside /etc/nginx/conf.d
            directory \n'
  24        action :create
  25        owner 'root'
  26        group 'root'
  27    end
  28
  29
  30    service 'nginx' do
  31        supports :status => true, :restart => true, :reload => true
  32        action [ :enable, :start ]
  33    end
  34
  35
  36    #resource to replace line
  37    #https://stackoverflow.com/questions/14848110/how-can-i-change-a-file-with-chef
```

Loadbalancer recipe

- Creates a file in **etc/nginx/conf.d** directory using the template file provided in the templates directory
- Restarting the Nginx service to make sure changes take affect

```
projectAL > chef > cookbooks > cooknginx > recipes >  loadbalancer.rb
 1    #this will have the loadbalancer configuration
 2
 3
 4    #creating the load-balancer configuration file
 5    template '/etc/nginx/conf.d/load-balancer.conf' do
 6        source 'etc/nginx-default.erb'
 7        action :create
 8    end
 9
10    service 'nginx' do
11        action [ :enable, :restart ]
12    end
13
```

Template file – nginx-default.erb
- Listening on port 80 distributing requests to the specified servers

```
projectAL > chef > cookbooks > cooknginx > templates > etc > <> nginx-default.erb
 1
 2    upstream app {
 3        server 192.168.10.10:3000;
 4        server 192.168.10.11:3000;
 5    }
 6
 7        # This server accepts all traffic to port 80 and passes it to the upstream.
 8        # Notice that the upstream name and the proxy_pass need to match.
 9
10        server {
11            listen 80;
12
13            location / {
14                proxy_pass http://app;
15            }
16        }
17
```

Web recipe

- Very similar to the load balancer recipe, it contains a chef remote_directory resource enabling the feature of copying directories to a specified folder. In this case I used it to copy the web app to the /var/www directory within the server.

  Setting the permission on that folder to read/write/execute for root and only read for users.

projectAL > chef > cookbooks > cookweb > recipes > 🔴 web.rb

```
 1    remote_directory "/var/www" do
 2        source 'webApp'
 3        owner 'root'
 4        group 'root'
 5        mode '0755'
 6        action :create
 7    end
 8
 9    template '/etc/nginx/conf.d/web.conf' do
10        source 'etc/nginx-default.erb'
11        action :create
12    end
13
14    service 'nginx' do
15        action [ :enable, :restart ]
16    end
17
```

Template used for the Nginx config
- Listening on port 80

projectAL > chef > cookbooks > cookweb > templates > etc > <> nginx-default.erb

```
 1    server {
 2
 3        listen 3000;
 4        server_name $host;
 5
 6        root /var/www/;
 7
 8        location /health {
 9            return 200 '\n $hostname $server_addr Status - OK \n';
10        }
11    }
12
13    #removed ruby interpolation and used nginx default vars
14    #https://nginx.org/en/docs/varindex.html
15
```

Nginx configuration

/etc/nginx/nginx.conf

```
        # gzip_vary on;
        # gzip_proxied any;
        # gzip_comp_level 6;
        # gzip_buffers 16 8k;
        # gzip_http_version 1.1;
        # gzip_types text/plain text/css application/json applicati
text/xml application/xml application/xml+rss text/javascript;

        ##
        # Virtual Host Configs
        ##

        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;
```

By default nginx will run both the configuration in **"/etc/nginx/sites-enabled/default"** and **"/etc/nginx/conf.d/".**

As mentioned in the cookbooks chapter of this document, the nginx config files is created and pushed to the **"/etc/nginx/conf.d/"** directory.

The load balancer is listening on port 80 and distributing the traffic to the two web servers on port 3000.

```
vagrant@loadbalancer:~$ cat /etc/nginx/conf.d/load-balancer.conf

upstream app {
   server 192.168.10.10:3000;
   server 192.168.10.11:3000;
}

   # This server accepts all traffic to port 80 and passes it to the upstream.
   # Notice that the upstream name and the proxy_pass need to match.

   server {
      listen 80;

      location / {
         proxy_pass http://app;
      }
   }
}
vagrant@loadbalancer:~$
```

The webservers are set to only accept web traffic on port 3000

```
vagrant@web01:~$ cat /etc/nginx/conf.d/web.conf
server {

    listen 3000;
    server_name $host;

    root /var/www/;

    location /health {
        return 200 '\n $hostname $server_addr Status - OK \n';
    }
}

#removed ruby interpolation and used nginx default vars
#https://nginx.org/en/docs/varindex.html
vagrant@web01:~$
```

Improvements

Currently there is no way of blocking users attempting to directly access the webserver providing they know the IP address on port 3000.

If I was to improve this setting, I will update the firewall settings to only accept request from the load balancer on that port.

Test Script

There is a shell script in scripts/web-health.sh
The script is fairly straightforward, it simply sends a curl request multiple times to both web servers and to the load balancer to **/health** route. Returning the status.

```bash
projectAL > scripts > ≡ web-health.sh
  1    #!/bin/bash
  2    echo '
  3    -------------------------------------------------
  4    |    |    |    |         Test directly to web01
  5    -------------------------------------------------'
  6    curl 192.168.10.10:3000/health --connect-timeout 20
  7
  8    echo '
  9    -------------------------------------------------
 10    |    |    |    |         Test directly to web02
 11    -------------------------------------------------'
 12    curl 192.168.10.11:3000/health --connect-timeout 20
 13
 14    for i in {1..2}
 15    do
 16    echo '
 17    -------------------------------------------------
 18    |    |    |    |         Test on load-balancer
 19    -------------------------------------------------'
 20    curl 10.10.10.10/health --connect-timeout 20
 21    done
 22
 23    read
```

Troubleshooting steps

- Cookbooks not being found

When initially deploying the cookbooks, I was specifying the full path of each cookbooks in the vagrant file which when loaded caused the virtual machine to mount them into their own individual directory.

When attempting to run the cookbooks, I kept getting met with the cookbook not found.

First attempt
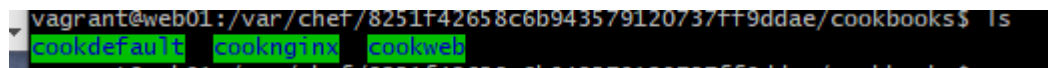Was to explicitly set the chef provisioning directory.
Doing this did not fix it, as it was doing the same thing within the newly set provisioning directory – **Daft move**

Second attempt
As the box was already built but not properly configured, I was still able to SSH on to it. I then attempted to run the chef cookbook manually.

I navigated to the chef cookbooks directory as seen in the image and executed the following command to run the cookbook recipe locally- **chef-client –z cooknginx/nginx.erb**
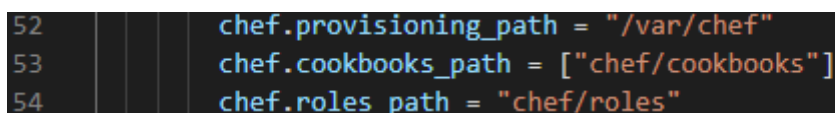
That worked!
 However, the main purposes of the project is to keep the user interaction at build and configuration to a minimum.

Resolution
Simple fix in the end was to not put the full path of the cookbooks and filename but to just provide the path of where all the cookbooks reside.
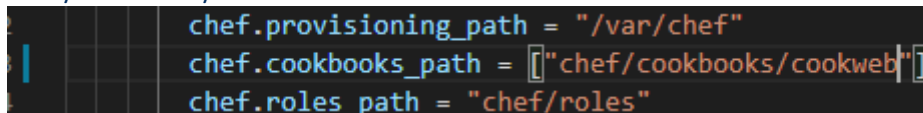
Correct
"chef/cookbooks"

Incorrect – cookbook name not required
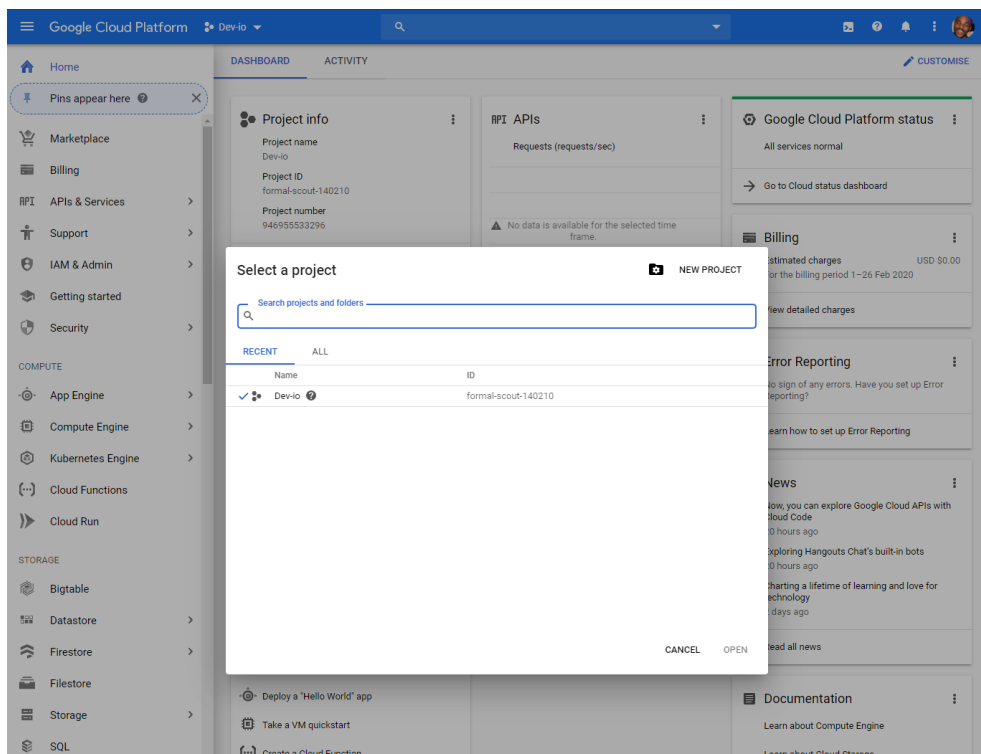"chef/cookbooks/cookweb"

This will then mount all the cookbooks in that directory into one directory on the server
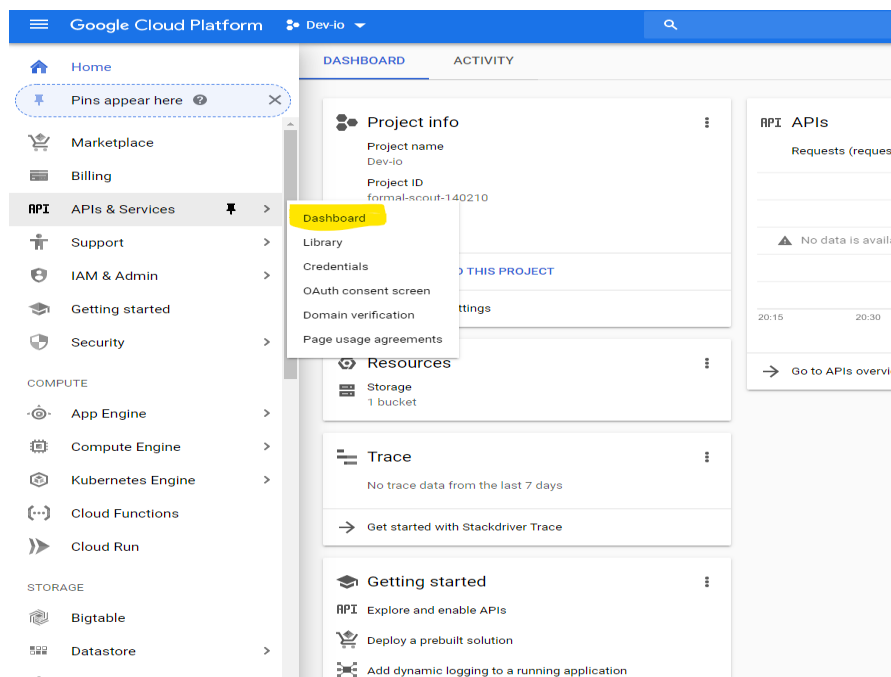
Google API set up for places information
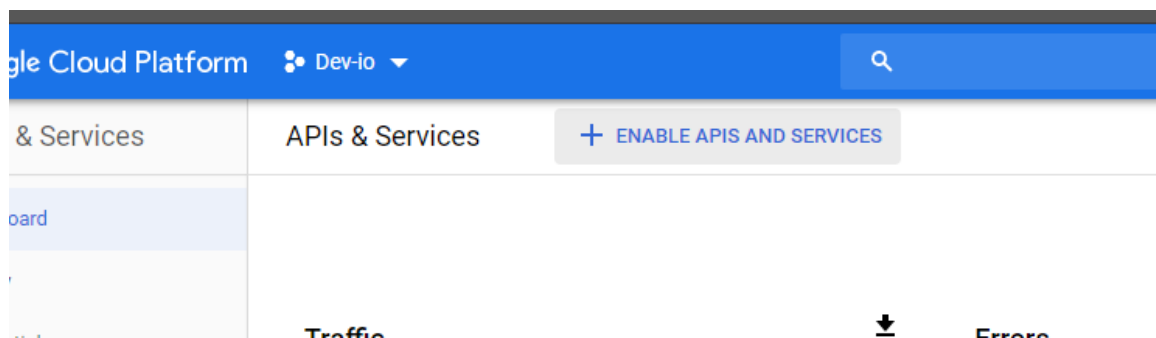
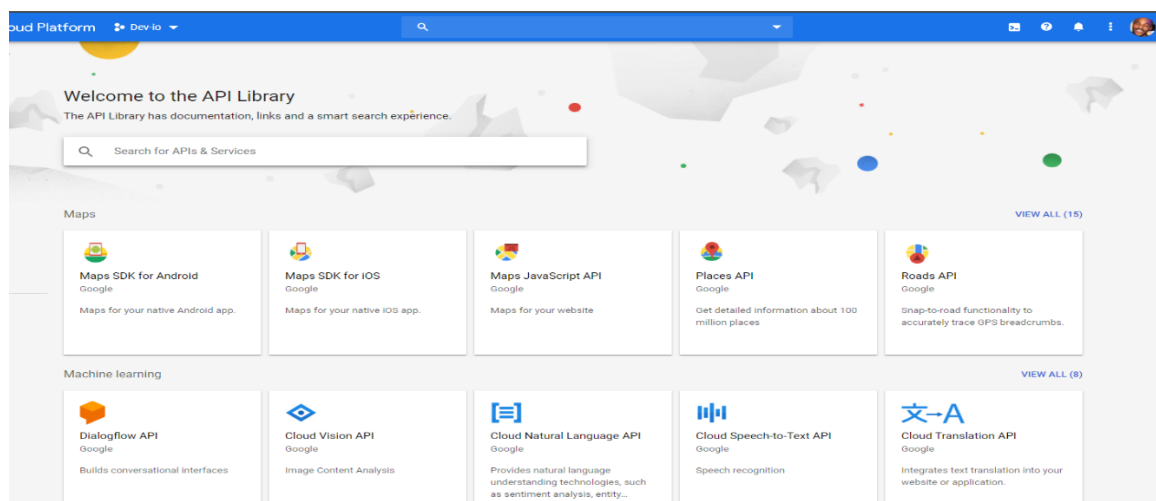1. Sign up to google cloud - https://cloud.google.com/
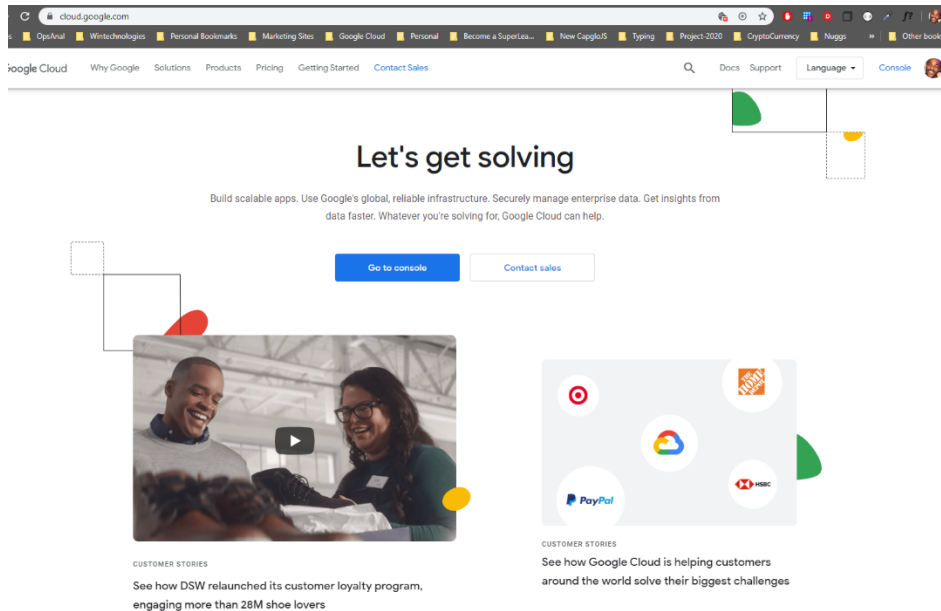
2. Create a project

3. Select API libraries

4. **Select** ENABLE APIS AND SERVICES

5. **Search** Places (you will be prompted to add authentication mechanisms – required for you to make requests to Google API)

6. Billing will need to be enabled to use the place API
https://developers.google.com/places/web-service/usage-and-billing

API key generation

PLEASE NOTE: I have currently created a API Key for the purpose of this project once used I will be removing the key.

7. Navigate back to the Dashboard settings **(Step 3)**
8. **Select** Credentials (Left pane)
9. On the credentials screen select +CREATE CREDENTAILS (Top of screen)
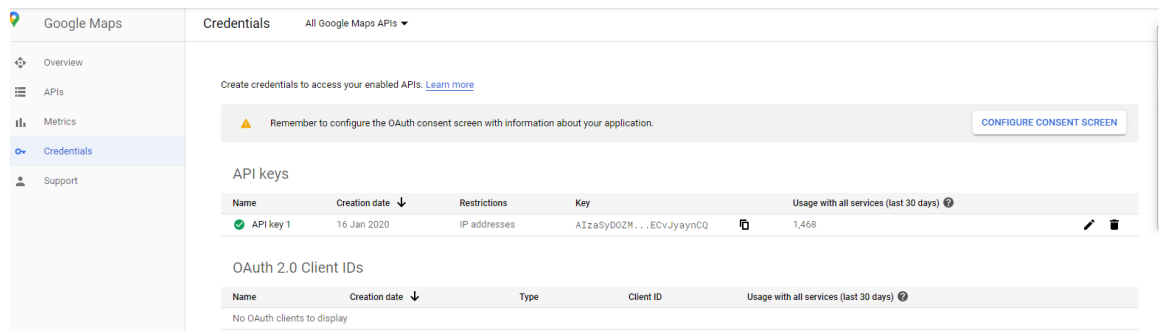
10. You will be presented with a popup with options to copy and restrict key
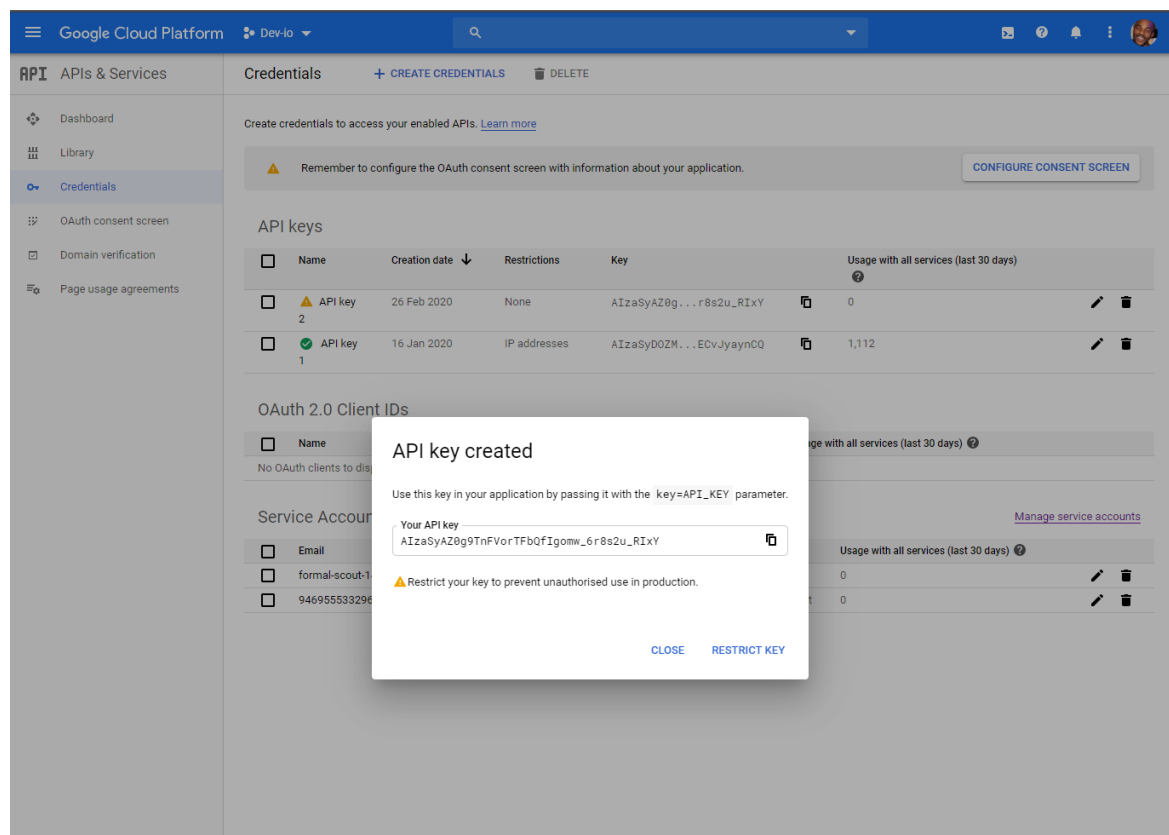
https://developers.google.com/maps/documentation/javascript/places#place_search_requests

At the bottom of the index.html file I added the places api link below above the custom script that will be handling the search form on the page.

```
48    </div>
49    <script type="text/javascript" src="https://maps.googleapis.com/maps/api/js?
      key=AIzaSyAZ0g9TnFVorTFbQfIgomw_6r8s2u_RIxY&libraries=places"></script>
50    <script src="./assets/js/custom.js"></script>
```

Order of operation this will ensure the google components is available within the custom script.

Without boring you with the details of the custom script as it is out of the scope of the project. Upon entering a location in the search field using the longitude and latitude data to retrieve cafés within a 500m radius from Google Places API. Then transformed to output the results in a list format on to the page.

Please see Testing chapter for app demonstration

Vagrant commands

Vagrant init - will create a vagrant file

Vagrant up *<optional machine name>* - will spin up specified machine or any configured machine specified in the vagrant file

Vagrant halt *<optional machine name>* - will shut down all or specified machine

Vagrant destroy *<optional machine name>* - Will shut down and delete any specified virtual machines

More on: https://www.vagrantup.com/docs/cli/

Time Taken

Research/Study - 5/6 hrs
Vagrant setup – 1 hrs
Chef Configurations/cookbook creation – 3 hrs
Documentation – 3/4 hrs
Testing/Troubleshooting – 2/3 hrs
Creating the web app – 2 hrs
Creating shell test script - .5 hrs

Resources/Learning

Vagrant Docs
https://www.vagrantup.com/docs/index.html
https://www.vagrantup.com/docs/provisioning/chef_solo.html

Vagrant/Chef Video Tutorials
https://www.udemy.com/course/vagrant-up/
https://www.udemy.com/course/chef-fundamentals-a-recipe-for-automating-infrastructure/

Chef-Solo
https://docs.chef.io/chef_solo.html

Nginx load balancing
https://www.digitalocean.com/community/tutorials/how-to-set-up-nginx-load-balancing

Google docs
https://developers.google.com/maps/documentation/javascript/places-autocomplete
https://developers.google.com/places/web-service/web-services-best-practices

Haversine Algorithm
https://cloud.google.com/blog/products/maps-platform/how-calculate-distances-map-maps-javascript-api