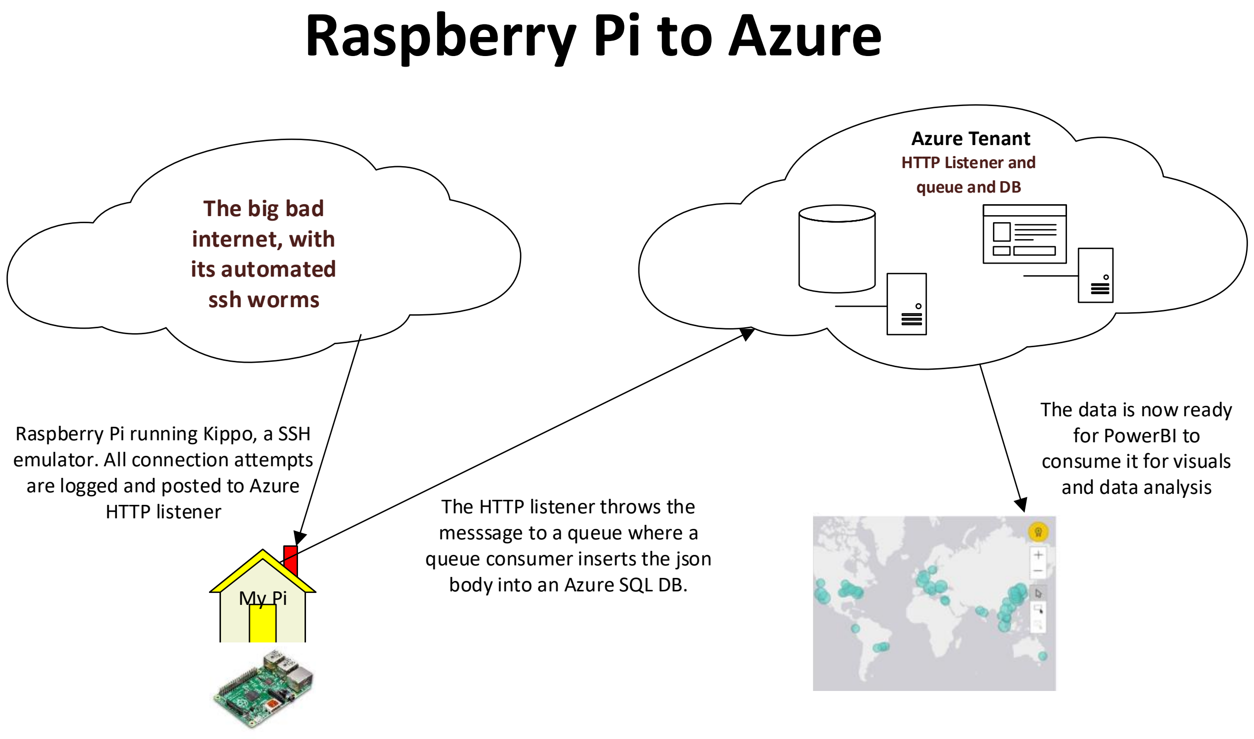
Raspberry Pi => Honeypot => Azure

Abstract:

Use a Raspberry Pi to host a honeypot ssh server (Kippo) and monitor the log file for new connection attempts. When a new attempt is detected, the PI posts a json body, consisting of the IP Address, latitude and longitude, to an Azure HTTP listener. The listener sends it to Azure queue. The queue consumer then sends it to an Azure DB. Power BI connects to the Azure DB to render some visuals.


Raspberry Pi to Azure
Azure Tenant
HTTP Listener and queue and DB
The big bad internet, with its automated ssh worms
Raspberry Pi running Kippo, a SSH emulator. All connection attempts are logged and posted to Azure HTTP listener
My Pi
The HTTP listener throws the messsage to a queue where a queue consumer inserts the json body into an Azure SQL DB.
The data is now ready for PowerBI to consume it for visuals and data analysis

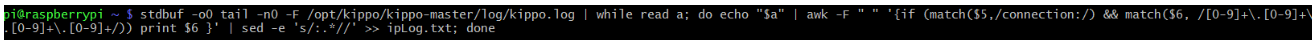
Overall Stats:

- Ran continuously for 2 days
- 472 connection attempts
- 70 distinct IP addresses
- China accounted for 86% of connection attempts

The Pi side of the equation:

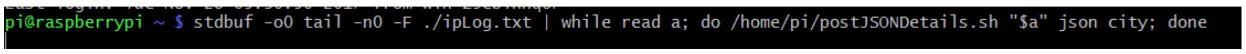
A BASH shell scripting guru I am not. The setup I am using requires two running “tail” processes. I am positive it could be consolidated into one, but I have not yet been able to do so. The first tail process monitors the kippo log for “new connection” attempts and then strips the IP address out of the message. It then writes it to a text file where another “tail” process listens for a new entry, runs another bash script to get the latitude and longitude of that IP address and then finally uses cURL to post the json body to the Azure HTTP listener.

```
stdbuf -o0 tail -n0 -F /opt/kippo/kippo-master/log/kippo.log | while read a; do echo
"$a" | awk -F " " '{if (match($5,/connection:/) && match($6, /[0-9]+\.[0-9]+\.[0-
9]+\.[0-9]+/)) print $6 }' | sed -e 's/:.*//' >> ipLog.txt; done
```



```
pi@raspberrypi ~ $ stdbuf -o0 tail -n0 -F /opt/kippo/kippo-master/log/kippo.log | while read a; do echo "$a" | awk -F " " '{if (match($5,/connection:/) && match($6, /[0-9]+\.[0-9]+\
.[0-9]+\.[0-9]+/)) print $6 }' | sed -e 's/:.*//' >> ipLog.txt; done
```

```
stdbuf -o0 tail -n0 -F ./ipLog.txt | while read a; do /home/pi/postJSONDetails.sh "$a"
json city; done
```



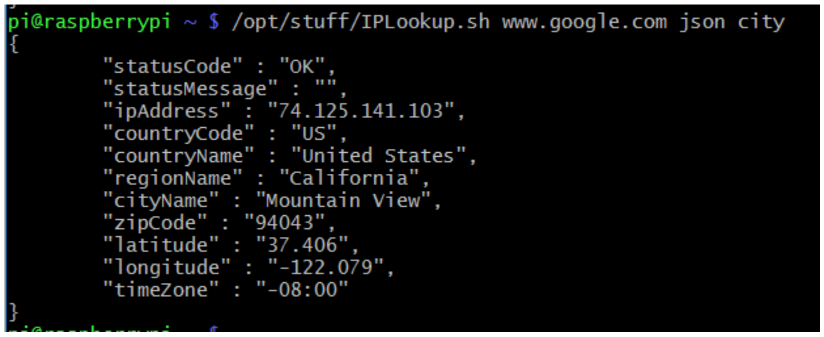
```
pi@raspberrypi ~ $ stdbuf -o0 tail -n0 -F ./ipLog.txt | while read a; do /home/pi/postJSONDetails.sh "$a" json city; done
```

postJSONDetails.sh:

```
#!/bin/bash

jsonBody="$(/opt/stuff/IPLookup.sh $1 json city)"
curl -H "Accept: application/json" -H "Content-Type: application/json; charset=UTF-8"
-X POST -d "'${jsonBody}'" "https://prod-
18.centralus.logic.azure.com:443/workflows.......<subscription key>
```

*IPLookup.sh comes from a free service called IPInfoDB that takes an ip address as input and produces json as output…..complete with latitude and longitude:

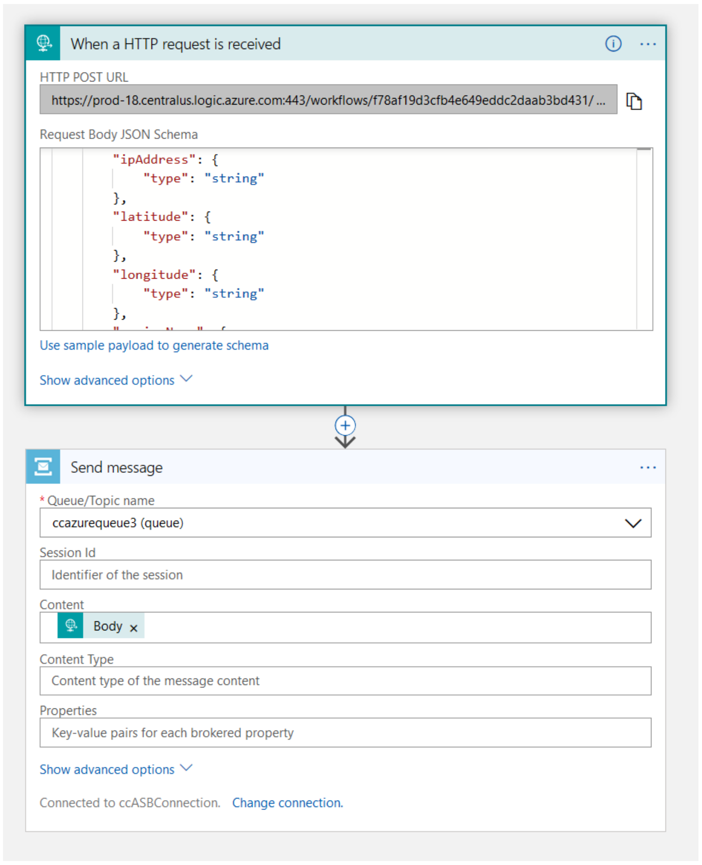


```
pi@raspberrypi ~ $ /opt/stuff/IPLookup.sh www.google.com json city
{
        "statusCode" : "OK",
        "statusMessage" : "",
        "ipAddress" : "74.125.141.103",
        "countryCode" : "US",
        "countryName" : "United States",
        "regionName" : "California",
        "cityName" : "Mountain View",
        "zipCode" : "94043",
        "latitude" : "37.406",
        "longitude" : "-122.079",
        "timeZone" : "-08:00"
}
```

The Azure side of the equation:

In my Azure account, I have an HTTP listener expecting a post with the json body show above. The received message is then thrown into an Azure service bus queue.

`Azure http listener:`



When a HTTP request is received

HTTP POST URL

https://prod-18.centralus.logic.azure.com:443/workflows/f78af19d3cfb4e649eddc2daab3bd431/ ...

Request Body JSON Schema

```
        "ipAddress": {
            "type": "string"
        },
        "latitude": {
            "type": "string"
        },
        "longitude": {
            "type": "string"
        },
```

Use sample payload to generate schema

Show advanced options

Send message

* Queue/Topic name: ccazurequeue3 (queue)

Session Id: Identifier of the session

Content: Body

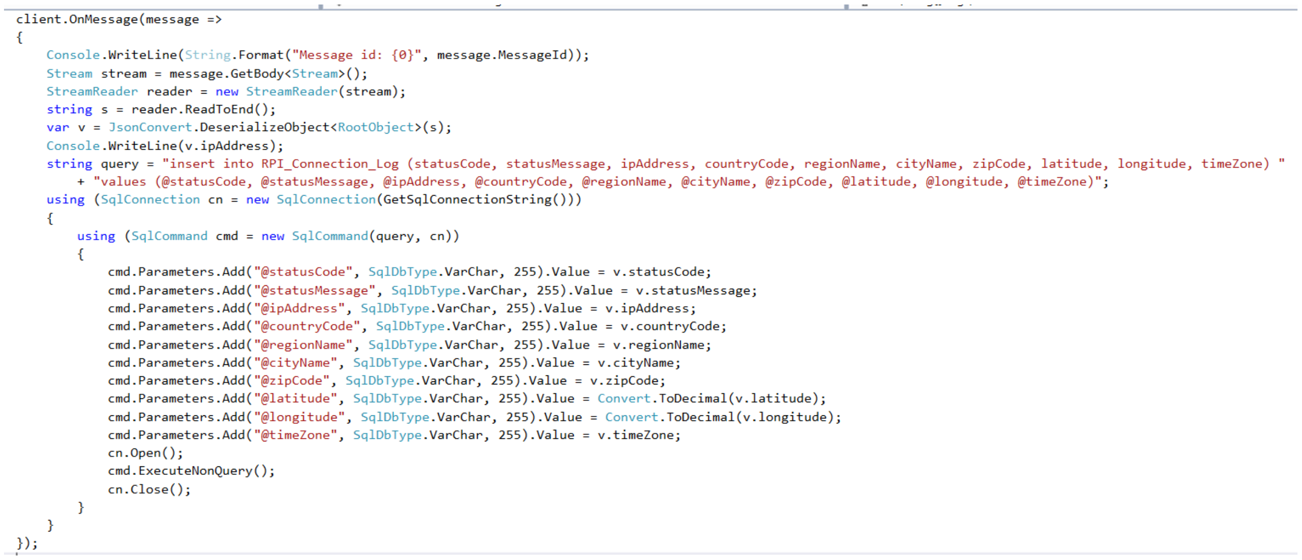
Content Type: Content type of the message content

Properties: Key-value pairs for each brokered property

Show advanced options

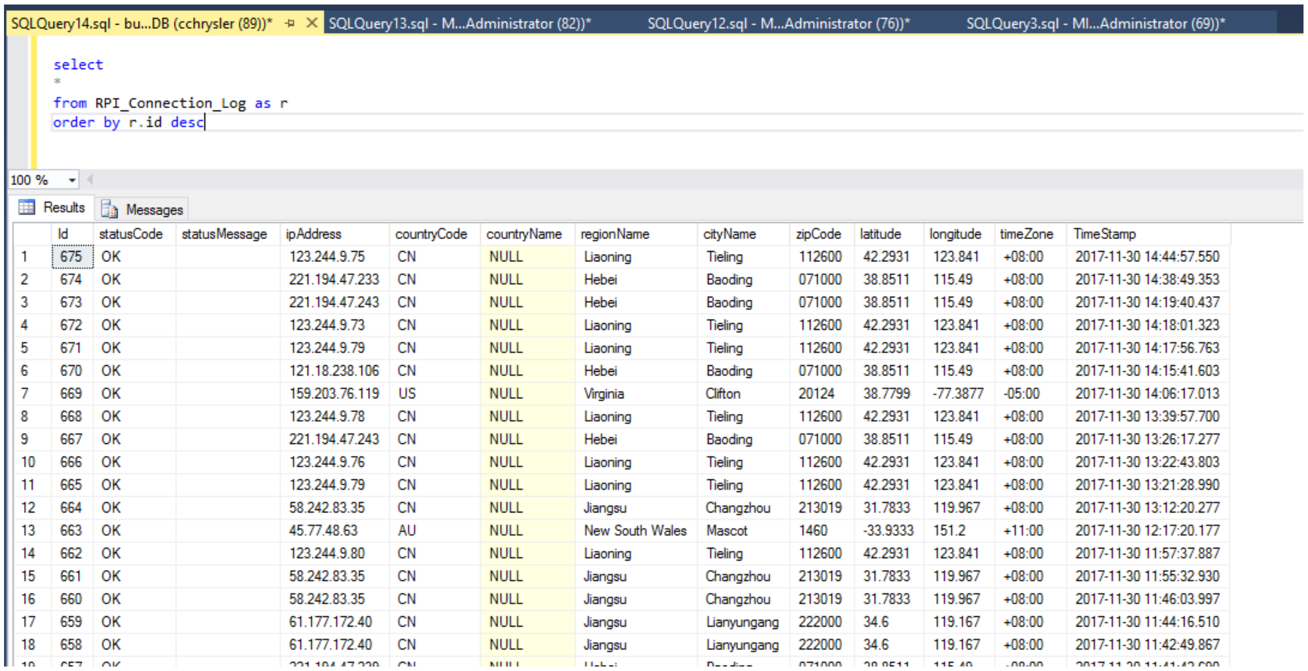
Connected to ccASBConnection. Change connection.

The queue consumer deserializes the message and inserts into an Azure SQL DB.



```
client.OnMessage(message =>
{
    Console.WriteLine(String.Format("Message id: {0}", message.MessageId));
    Stream stream = message.GetBody<Stream>();
    StreamReader reader = new StreamReader(stream);
    string s = reader.ReadToEnd();
    var v = JsonConvert.DeserializeObject<RootObject>(s);
    Console.WriteLine(v.ipAddress);
    string query = "insert into RPI_Connection_Log (statusCode, statusMessage, ipAddress, countryCode, regionName, cityName, zipCode, latitude, longitude, timeZone) "
        + "values (@statusCode, @statusMessage, @ipAddress, @countryCode, @regionName, @cityName, @zipCode, @latitude, @longitude, @timeZone)";
    using (SqlConnection cn = new SqlConnection(GetSqlConnectionString()))
    {
        using (SqlCommand cmd = new SqlCommand(query, cn))
        {
            cmd.Parameters.Add("@statusCode", SqlDbType.VarChar, 255).Value = v.statusCode;
            cmd.Parameters.Add("@statusMessage", SqlDbType.VarChar, 255).Value = v.statusMessage;
            cmd.Parameters.Add("@ipAddress", SqlDbType.VarChar, 255).Value = v.ipAddress;
            cmd.Parameters.Add("@countryCode", SqlDbType.VarChar, 255).Value = v.countryCode;
            cmd.Parameters.Add("@regionName", SqlDbType.VarChar, 255).Value = v.regionName;
            cmd.Parameters.Add("@cityName", SqlDbType.VarChar, 255).Value = v.cityName;
            cmd.Parameters.Add("@zipCode", SqlDbType.VarChar, 255).Value = v.zipCode;
            cmd.Parameters.Add("@latitude", SqlDbType.VarChar, 255).Value = Convert.ToDecimal(v.latitude);
            cmd.Parameters.Add("@longitude", SqlDbType.VarChar, 255).Value = Convert.ToDecimal(v.longitude);
            cmd.Parameters.Add("@timeZone", SqlDbType.VarChar, 255).Value = v.timeZone;
            cn.Open();
            cmd.ExecuteNonQuery();
            cn.Close();
        }
    }
});
```

Azure DB:

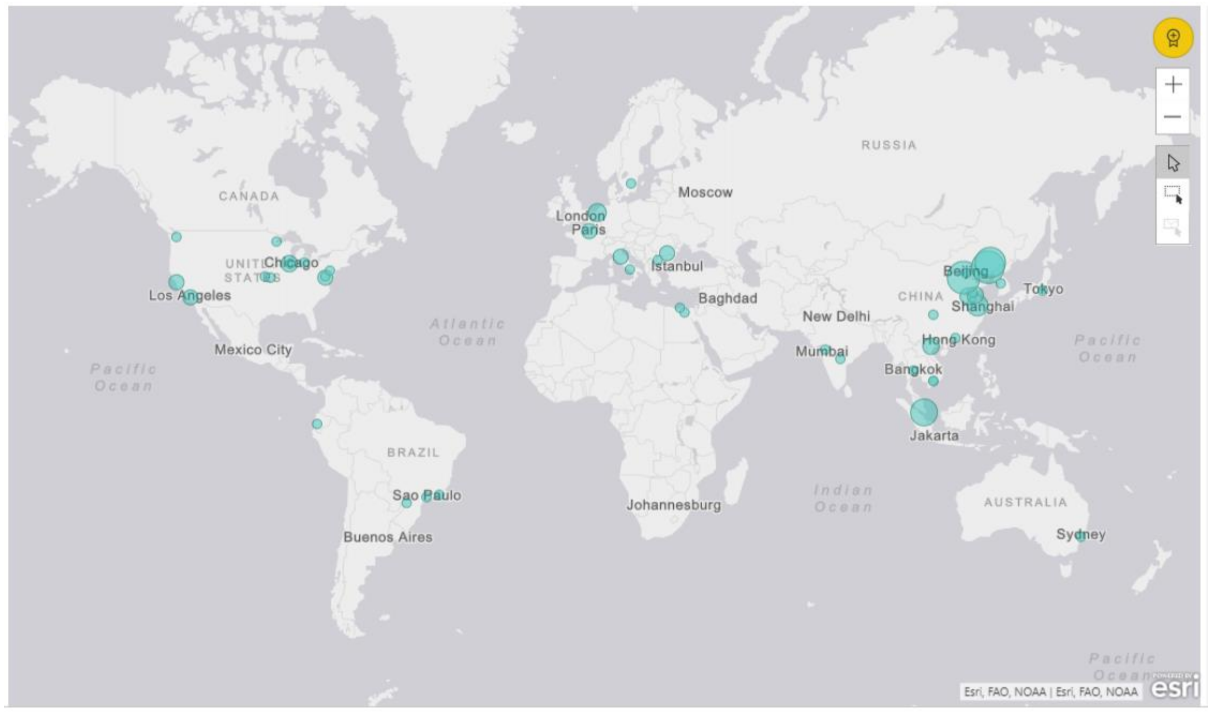

SQLQuery14.sql - bu...DB (cchrysler (89))*
SQLQuery13.sql - M...Administrator (82))*
SQLQuery12.sql - M...Administrator (76))*
SQLQuery3.sql - MI...Administrator (69))*
100 %
Results
Messages

```
select
*
from RPI_Connection_Log as r
order by r.id desc
```

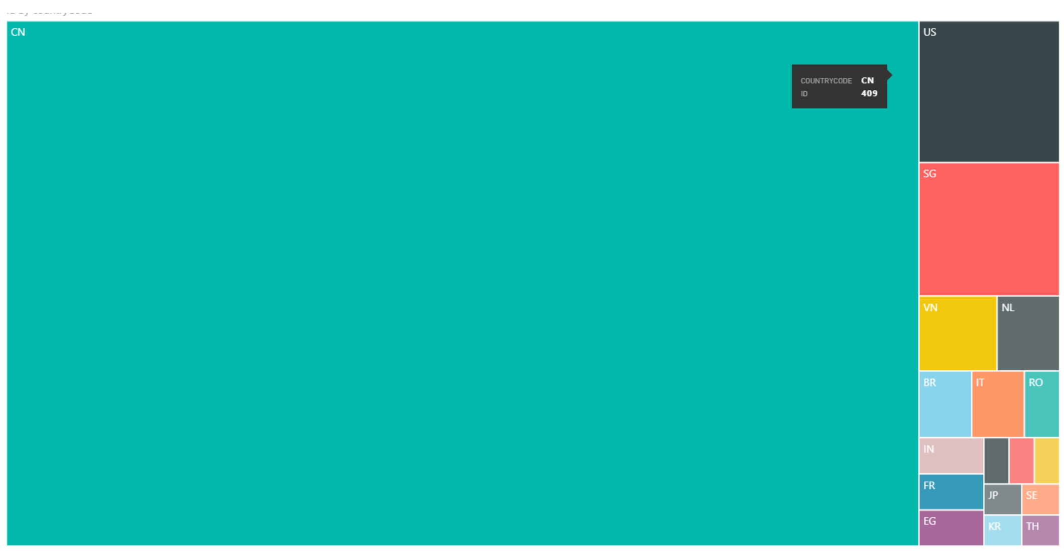
	Id	statusCode	statusMessage	ipAddress	countryCode	countryName	regionName	cityName	zipCode	latitude	longitude	timeZone	TimeStamp
1	675	OK		123.244.9.75	CN	NULL	Liaoning	Tieling	112600	42.2931	123.841	+08:00	2017-11-30 14:44:57.550
2	674	OK		221.194.47.233	CN	NULL	Hebei	Baoding	071000	38.8511	115.49	+08:00	2017-11-30 14:38:49.353
3	673	OK		221.194.47.243	CN	NULL	Hebei	Baoding	071000	38.8511	115.49	+08:00	2017-11-30 14:19:40.437
4	672	OK		123.244.9.73	CN	NULL	Liaoning	Tieling	112600	42.2931	123.841	+08:00	2017-11-30 14:18:01.323
5	671	OK		123.244.9.79	CN	NULL	Liaoning	Tieling	112600	42.2931	123.841	+08:00	2017-11-30 14:17:56.763
6	670	OK		121.18.238.106	CN	NULL	Hebei	Baoding	071000	38.8511	115.49	+08:00	2017-11-30 14:15:41.603
7	669	OK		159.203.76.119	US	NULL	Virginia	Clifton	20124	38.7799	-77.3877	-05:00	2017-11-30 14:06:17.013
8	668	OK		123.244.9.78	CN	NULL	Liaoning	Tieling	112600	42.2931	123.841	+08:00	2017-11-30 13:39:57.700
9	667	OK		221.194.47.243	CN	NULL	Hebei	Baoding	071000	38.8511	115.49	+08:00	2017-11-30 13:26:17.277
10	666	OK		123.244.9.76	CN	NULL	Liaoning	Tieling	112600	42.2931	123.841	+08:00	2017-11-30 13:22:43.803
11	665	OK		123.244.9.79	CN	NULL	Liaoning	Tieling	112600	42.2931	123.841	+08:00	2017-11-30 13:21:28.990
12	664	OK		58.242.83.35	CN	NULL	Jiangsu	Changzhou	213019	31.7833	119.967	+08:00	2017-11-30 13:12:20.277
13	663	OK		45.77.48.63	AU	NULL	New South Wales	Mascot	1460	-33.9333	151.2	+11:00	2017-11-30 12:17:20.177
14	662	OK		123.244.9.80	CN	NULL	Liaoning	Tieling	112600	42.2931	123.841	+08:00	2017-11-30 11:57:37.887
15	661	OK		58.242.83.35	CN	NULL	Jiangsu	Changzhou	213019	31.7833	119.967	+08:00	2017-11-30 11:55:32.930
16	660	OK		58.242.83.35	CN	NULL	Jiangsu	Changzhou	213019	31.7833	119.967	+08:00	2017-11-30 11:46:03.997
17	659	OK		61.177.172.40	CN	NULL	Jiangsu	Lianyungang	222000	34.6	119.167	+08:00	2017-11-30 11:44:16.510
18	658	OK		61.177.172.40	CN	NULL	Jiangsu	Lianyungang	222000	34.6	119.167	+08:00	2017-11-30 11:42:49.867

The Power BI side of the equation:

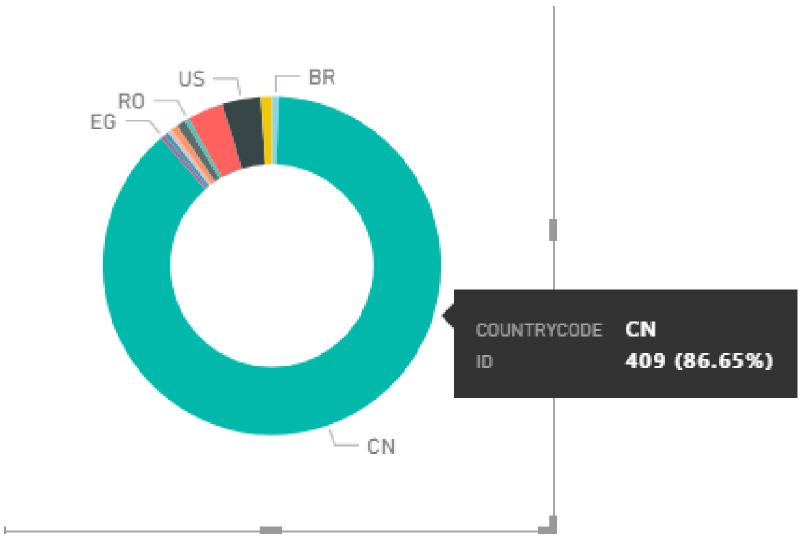
`Traffic by latitude and longitude`



`Traffic broken down by country code`

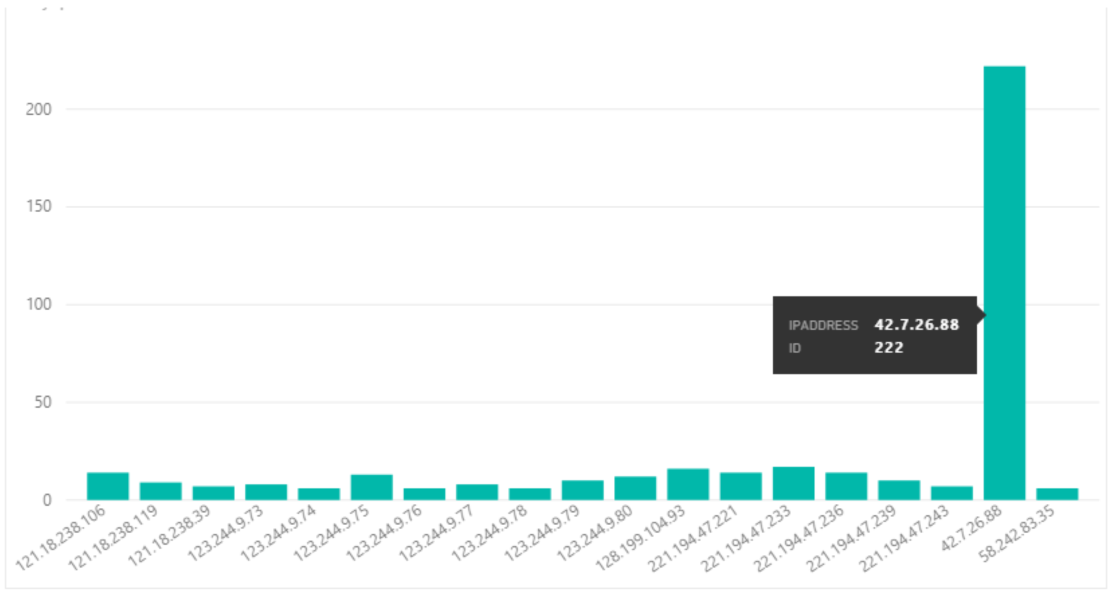


Percentage of traffic by country code



One IP (42.7.26.88, China) was more persistent than the others. I had to alter the firewall to not allow more than one connection attempt per minute.

Connection Attempts per IP address


IPADDRESS 42.7.26.88
ID 222