Abstract

This document provides more details on Meraki Portal Counter demo project implemented for retail store case study. It describes project implementation, setup and execution, as well as assumptions and logical flow of tools used.

Overview

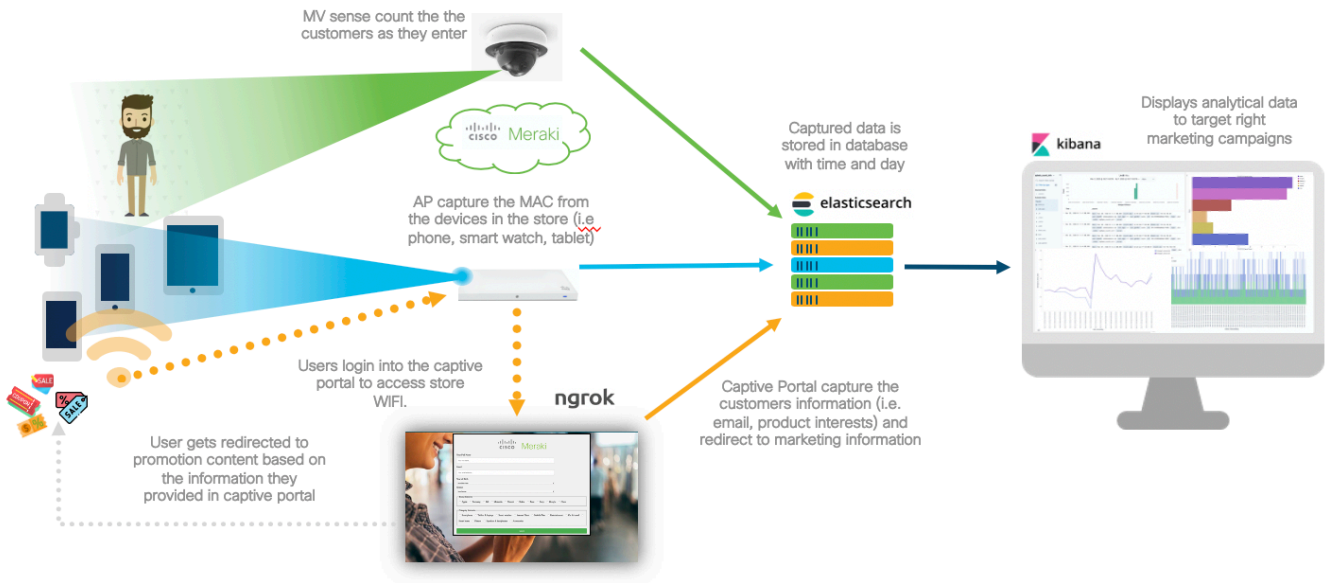
The purpose of this project is to provide a solution for the retailer so they can get people traffic information easily. By using the analytical results, retailers have insights of the market trends and customer interests, which contribute in a targeted marketing campaign and successful business strategy.

This project is not a fully working solution. This is more of a proof of concept to illustrates the capabilities of Meraki products and how they help achieve desired outcome of the case study.

This project uses the following:

- Meraki cameras to detect and count people entering and leaving the store
- Meraki access points to detect and count devices in the store
- Meraki Captive Portal to capture the customers' information, and redirect them to products of interest.
- Kibana to visualize the customer traffic and device information.
- Backend code written in python.

The following diagram illustrated the project.



Tools Used

- Python3
- Flask
- Elasticsearch
- Kibana
- Ngrok
- Meraki:
    - SDK
    - MV Sense
    - Splash Page
    - Location Analysis
    - Camera
    - Access Point
    - Switch

File Structure

- **camera_count_v2.py**
    - Used for capturing how many people enter and exit the store as seen by Meraki camera.
- **location_scanning_v2.py**
    - Used to detect nearby devices that use WiFi.
- **meraki_captive_portal.py**
    - Used to capture data from splash page and to host UI for splash page.
- **elastic_handler.py**
    - Contains functions used by other scripts to query and push data to Elasticsearch.
- **config.py**
    - Contains all sensitive information that should not be shared publicly. Things like Meraki API keys, secret key, Ngrok URLs, names for Elasticsearch indices, etc.
    - You would need to acquire most of this information from your Meraki dashboard or from your Ngrok process.
    - This file is part of **git ignore** so any changes you make to it will not be pushed to main repo. We recommend you do not remove it from ignore file.
- **elastic_onetime.py**
    - Contains code to create all indices required for this project. You would need to run this only once : the first time when you are setting up Elasticsearch.
- **ngrok.yml**
    - Config file for ngrok. You should copy this to your ngrok/config folder (usually placed in *$HOME/.ngrok/*) so you can run multiple tunnels at once.
- **requirements.txt**

    - Contains all python moudles used by other scripts. To install these modules run **pip3 install -r requirements.txt**. You would need to run this only once (i.e. when you are setting up project initially).
- **static** and **templates** directories
    - Contain all front end files for Splash Page. Files like html, css, javascript, images, etc...

Ngrok

In order for Meraki to access server you need to have a public domain.

For this we used ngrok, which is secure way to link your localhost to publicly accessible domain.

To install ngrok follow their getting started guide.

In GitHub you will find file called **ngrok.yml**. Copy that file to ngrok conifig folder once you install it.

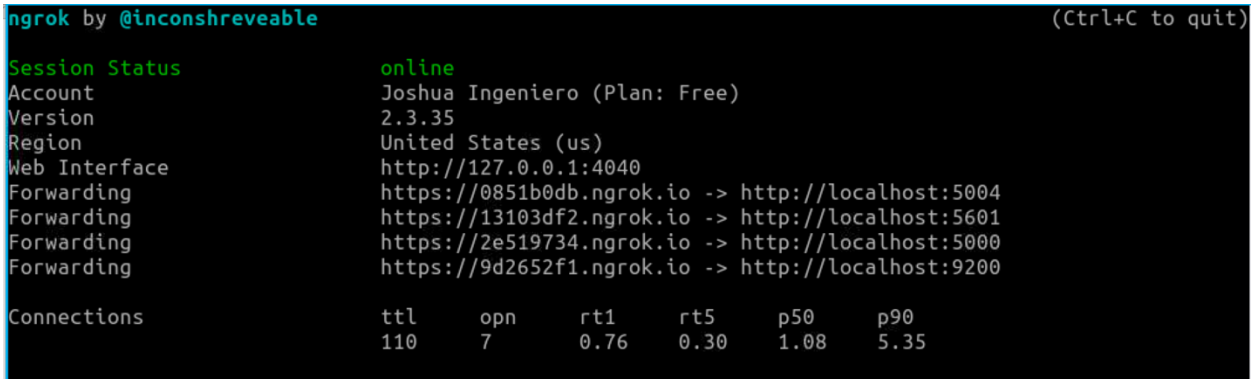
Config file is already set up to generate 4 different public domains each one pointing to, localhost port 5000, 5004, 5601 and 9200 respectively, where:

- 5004 – splash page server
- 5000 – scanning server
- 5601 – Kibana
- 9200 – Elasticsearch

Note: Every time you run ngrok it will provide you with different URL (i.e. domain) hence you will need to change these in **Meraki dashboard** and in **config.py** file in the code. More on this later.

Note: You will also need to register for ngrok if you wish to run 4 tunnels concurrently. It's free but they just want you to register.

To run ngrok (assuming you have placed ngrok.yml in appropriate place) simply open the terminal and run `ngrok start -all`. You should see something like this:



Elasticsearch

For database we decided to use Elasticsearch. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents In this case elastic is hosted locally and can be accessed using APIs. You can read up more about Elastic here.

Data is *indexed (*i.e. placed) in Elasticsearch. An Elasticsearch index is a collection of documents that are related to each other. Elasticsearch stores data as JSON documents. Each document correlates a set of keys (names of fields or properties) with their corresponding values (strings, numbers, Booleans, dates, arrays of values, geolocations, or other types of data).

Once data is indexed in Elasticsearch, users can run complex queries against their data and use aggregations to retrieve complex summaries of their data. It is provides fast and easy way to store and retrieve information.

However, one big reason for choosing Elasticsearch was because of Kibana plug-in. Using Kibana, users can create powerful visualizations of their data, share dashboards, and manage the Elastic Stack.

Setup

1. Installing Elasticsearch and Kibana
    a. For Elasticsearch refer to this page.
    b. For Kibana refer to this page.

2. Startup Elastic and Kibana
    a. In a terminal navigate to where you exported your Elastic
    b. From there simply run: `./bin/elasticsearch`
    c. In a separate terminal navigate to where you extracted kibana.
    d. Run: `./bin.kibana`

3. Test Elastic and Kibana are up and running
    a. By default elastic runs on localhost port **9200**
    b. Kibana runs on localhost port **5601**
    c. In your browser browse to localhost:9200 you should see something like this:

```
{
  "name" : "UMIHAJLO-M-426L",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "gpDOaM4kTnGb7fH4RVYaqg",
  "version" : {
    "number" : "7.6.1",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "aa751e09be0a5072e8570670309b1f12348f023b",
    "build_date" : "2020-02-29T00:15:25.529771Z",
    "build_snapshot" : false,
    "lucene_version" : "8.4.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```


This indicates that Elasticsearch is up and running.

d. In a separate tab browse to localhost:5601 and you should see kibana interface. More on this later.
e. If you require further understanding on Elasticsearch and Kibana we would recommend looking into Getting Started Guide on their website.

4. Better yet, test if you can access them using **ngrok** URLs configured previously. Simply copy the **ngrok** URL and paste it in your browser of choice. It should take you to the respective interfaces.

5. **Note**: If you are setting up this project for the very first time. You would need to run **elastic_onetime.py** function to setup all indexes used in this project. Note you only need to run this code once assuming you do not reinstall Elasticsearch.

6. Install elasticsearch package for Python3:
   In a terminal, run: `pip3 install elasticsearch`

Overview

Indexes in elastic are as follows:

- **splash_count**
  - Used to store information regarding splash page.
- **camera_count_v2**
  - Used to store information regarding Meraki MV sense.
- **scanning_count**
  - Used to store information regarding Meraki location scanning.

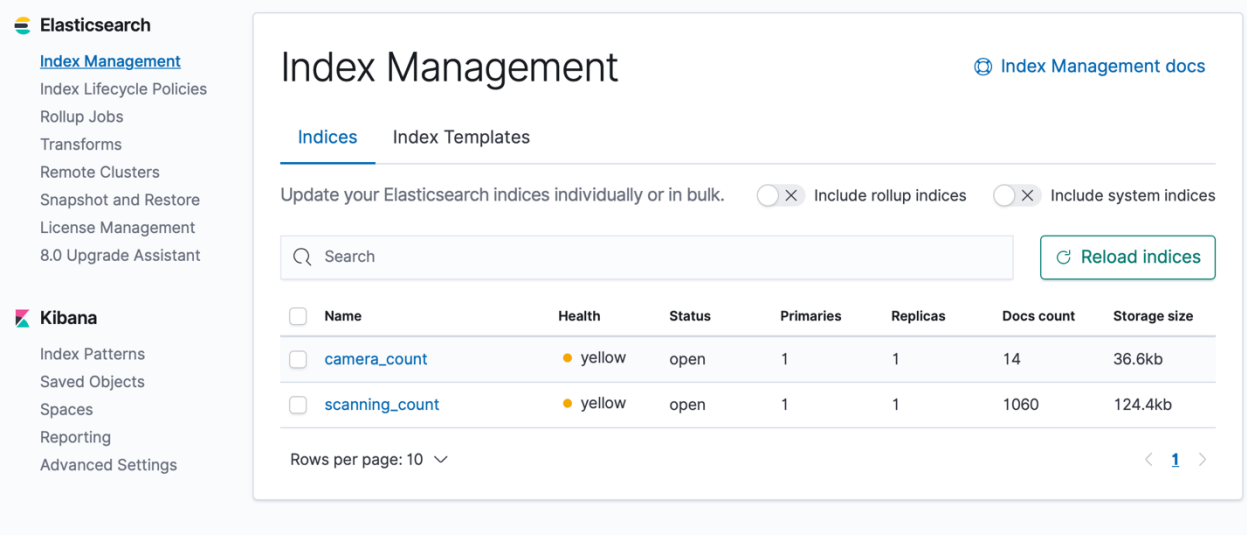
Note: Python code actually uses ngrok URL when trying to write to Elasticsearch. This information is stored in **config.py** file in a variable called ES_HOST_URL. If your ngrok is re-run you need to replace this with new URL. Otherwise you can change this value to **localhost:9200**, in which case you can only writer to elastic if you are running it on your local machine.

Kibana

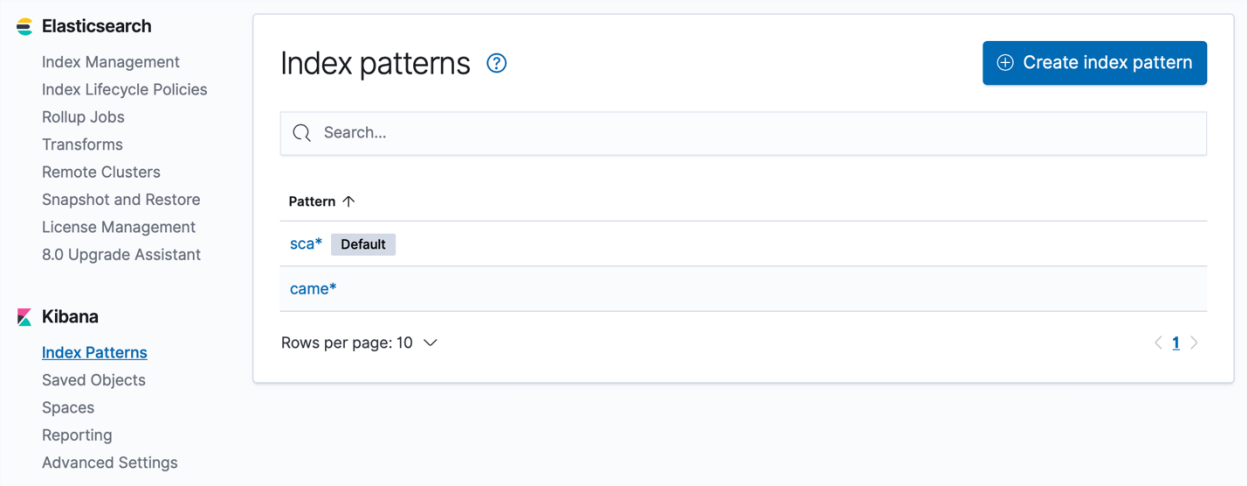
To view the data stored in Elasticsearch database, we use Kibana. It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster. You can create bar, line, and scatter plots, or pie charts and maps on top of large volumes of data. You can read more about Kibana here.

Setup

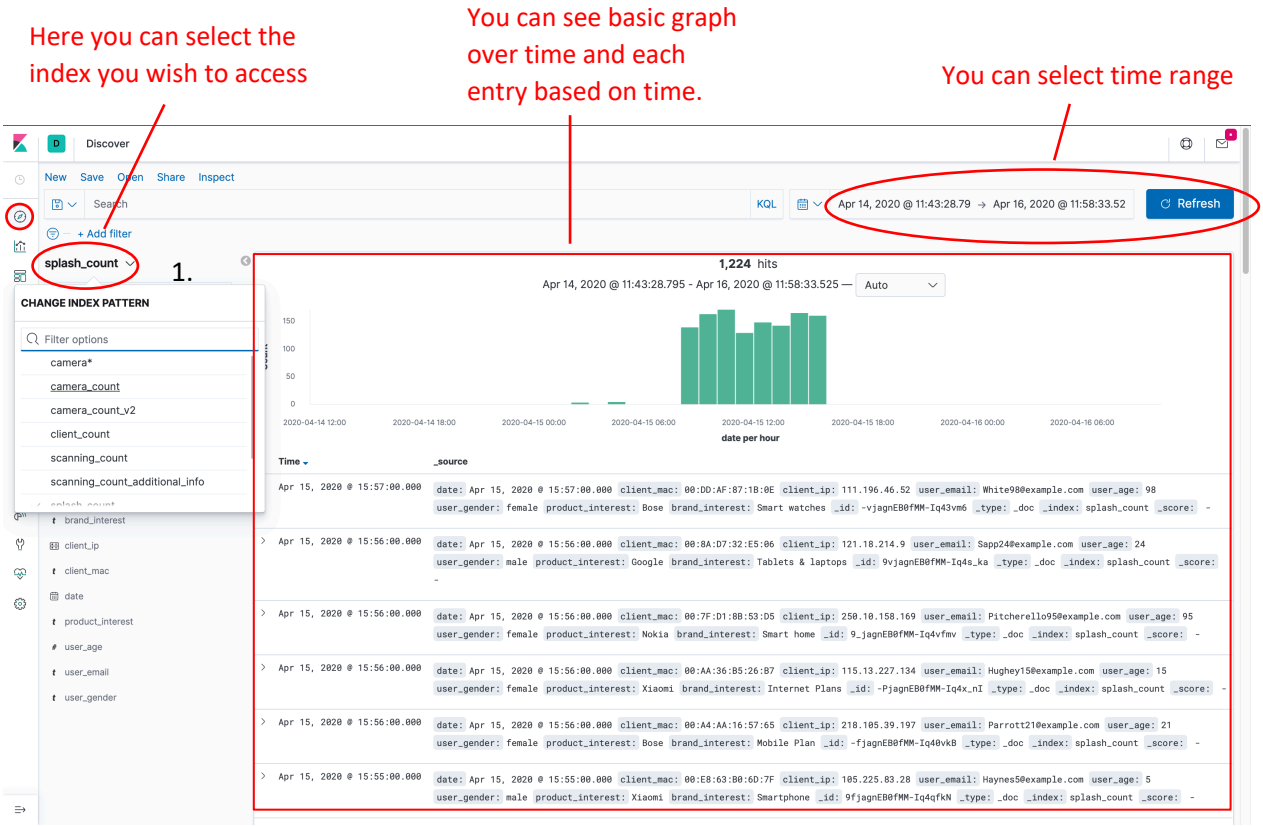
1. In your browser, navigate to: http://localhost:5601. It will open up Kibana page. Alternatively, access it using the ngrok URL.
2. Got to the setting by clicking the icon: ⚙ to add new indexes into the portal:
    a. You can view all indexes from elasticsearch database:



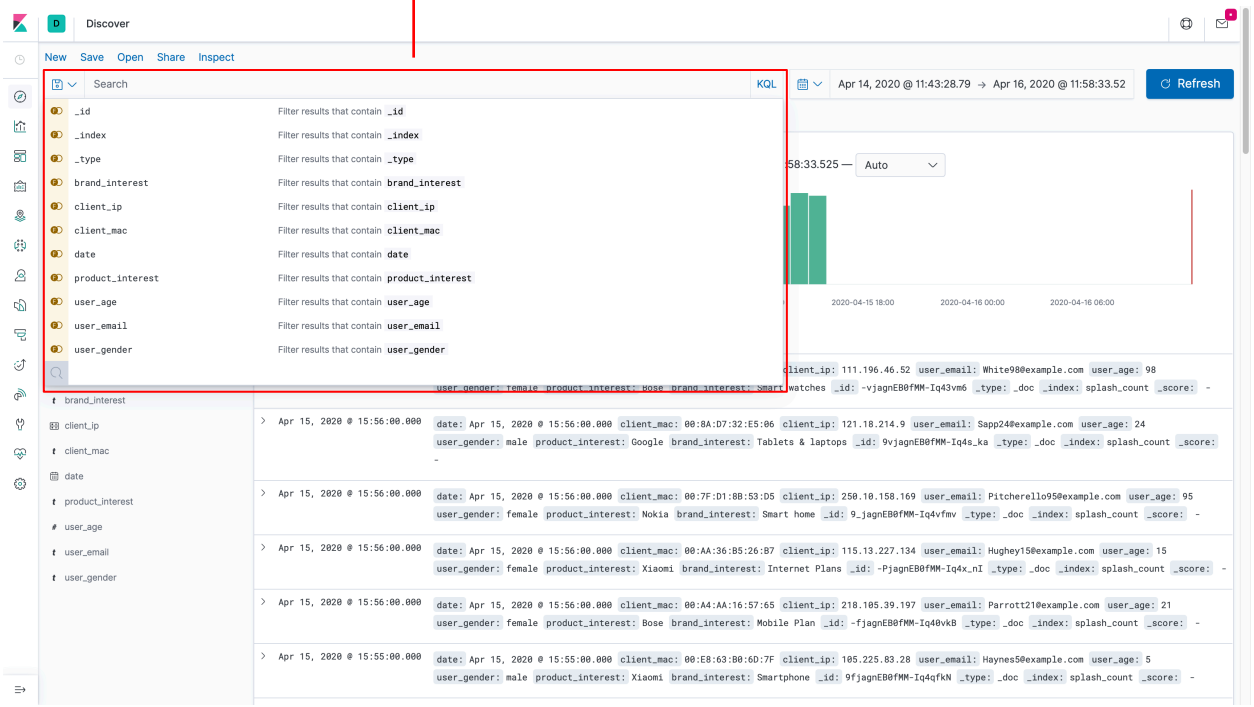
    b. To import the index into Kibana, click the **Index Patterns** under Kibana category, and click **Create Index Pattern** as the screenshot shows:



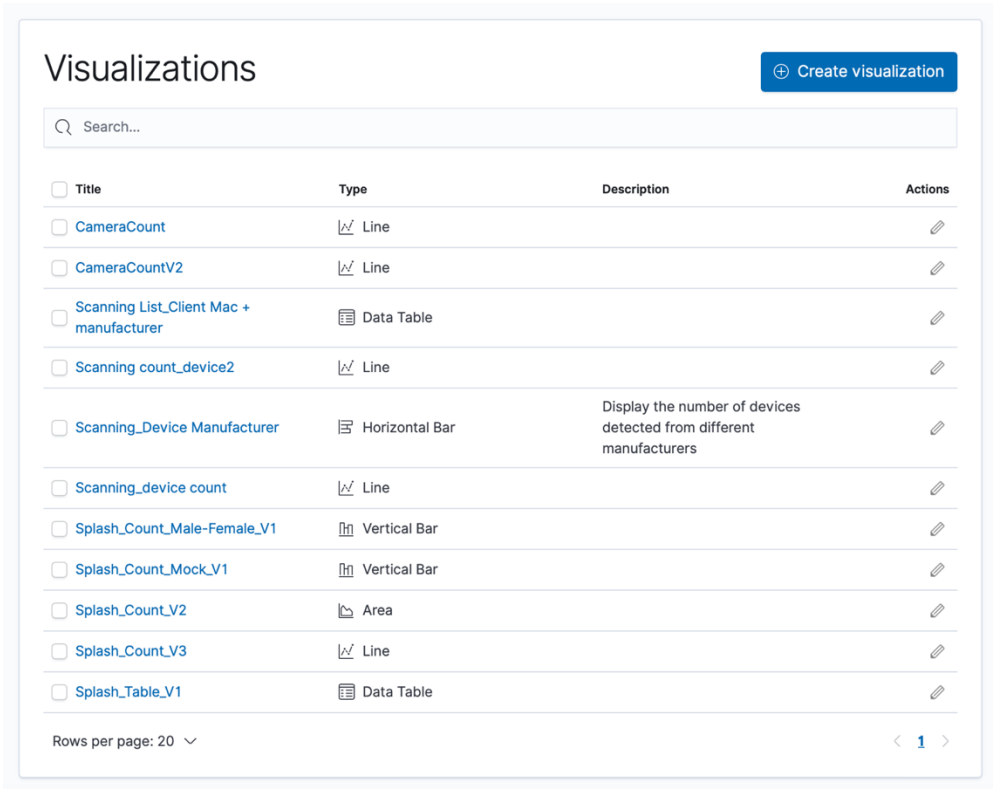
3. Click the icon:  to view all available data captured from location analysis, splash page and MV sense:



Search allows you to filter results based on index values. You can do search for each key.



4. Click the icon:  to generate and view data in graph:

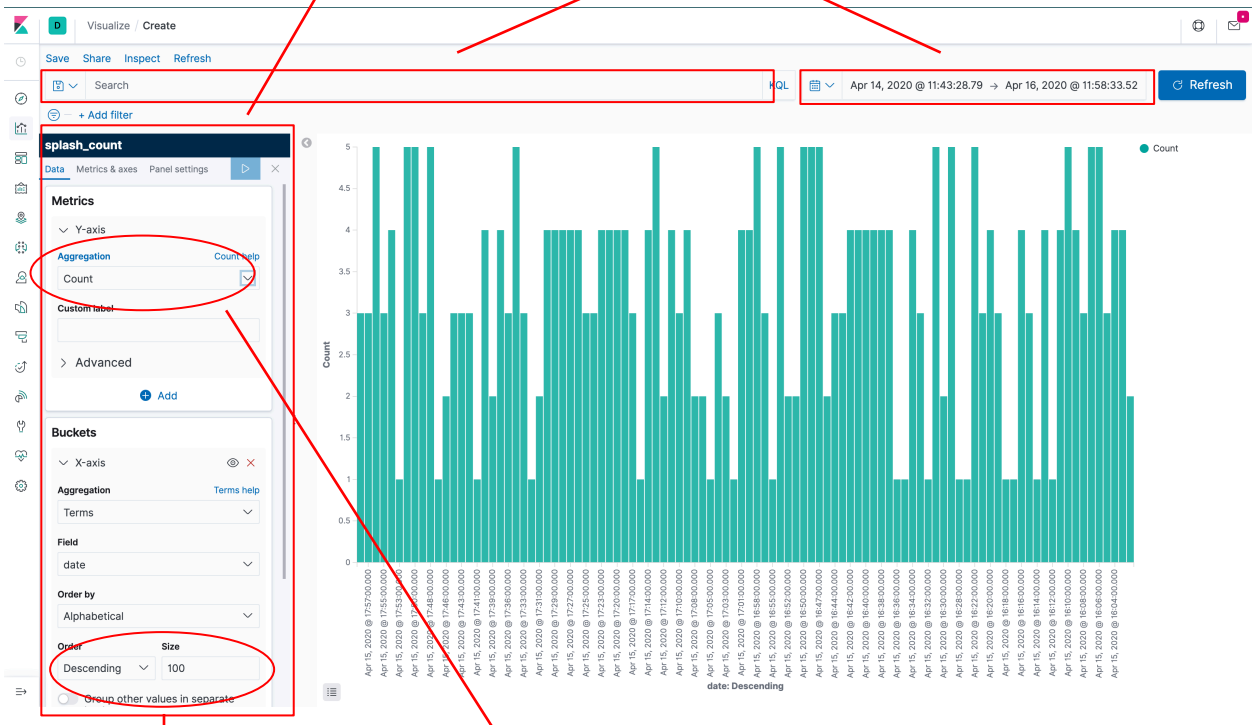


Click **Create Visualization** to generate new graph. You can view previously generated graph from the list.

For example, let's take splash_count. Once you create your visual you need to setup axis and time range.

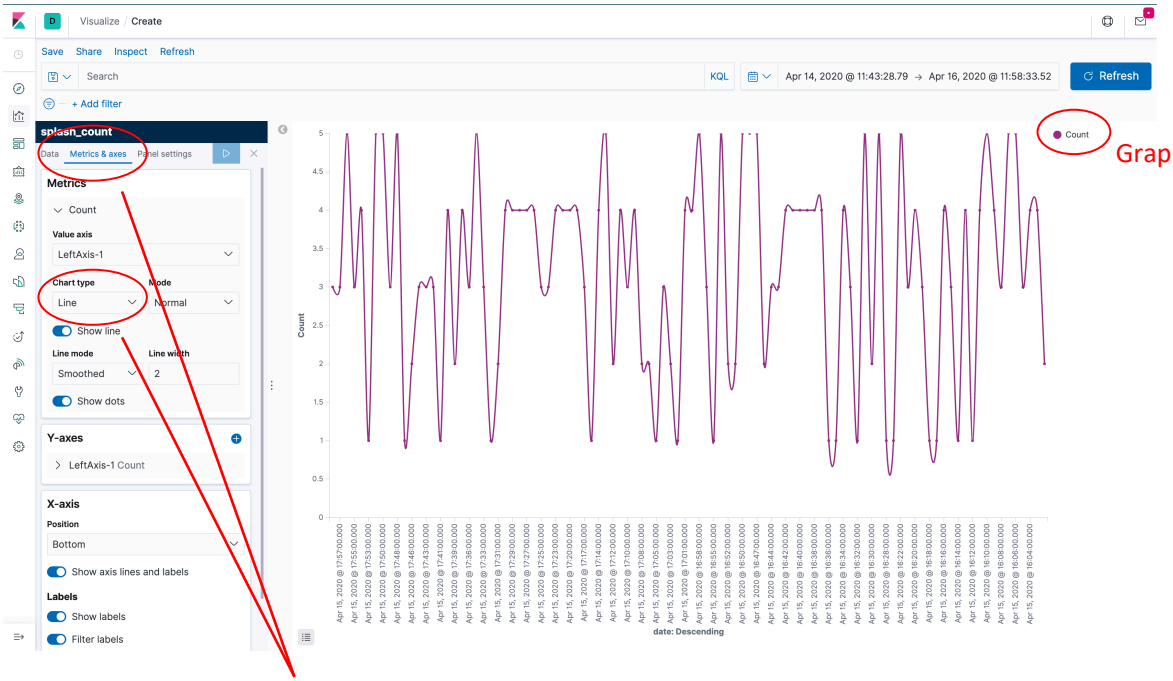
You need to set your x-axis and your y-axis. Generally, you would place date on x-axis and data on y-axis.

You can still filter data through search bar and change time range



You would need to specify how many values you want to display on x-axis

You can also display which value you want to display from the index.



Graph Color

You can change the type of the graph and other settings.

Location Analysis

Location scanning code is based of [example code provided on Meraki website].

More information regarding location scanning Can be found on [Meraki website].

The Meraki access points listen for WiFi clients that are searching for a network to join and log the events in a real-time manner. The client information is collected temporarily in the cloud, and the additional information can be collected too, such as location data of the clients and its manufacturer. With the in-built end-to-end system, Meraki can aggregate data from thousands of endpoints to collect, analyze and present both online and offline data effectively. This enables retails or enterprise have insights of trends and user engagement.

This location scanning application can receive the client information from the Meraki Cloud. It will extract, analyze, store and present the client data to your server. The information can be obtained from the data includes access point location, access point MAC address, client MAC address, client manufacturer, client location data and the average client count over a period.

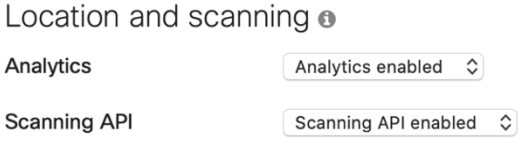
Requirements

- Ensure that **ngrok** is up and running pointing to correct ports as per previous instructions.
- You also need code from GitHub.
- Install Flask
- Meraki AP and Meraki Dashboard access.

Setup

1. The **ngrok** instance should already be running as per previous instructions.
    a. If ngrok is not active
        i. Run: `ngrok start –all`
        ii. Or Run: `ngrok http 5000` (if you want to only test splash page)
    b. Remember your ngrok URL you will need it for the next step.

2. Logon to the Meraki Dashboard then navigate to **Network Wide > General**
    a. Turn on the API by selecting Location API enabled in the dropdown box.



b. Specify a post URL from your ngrok and the authentication secret (the secret is used by your HTTP server to validate that the JSON posts are coming from the Meraki cloud)

c. Specify which Location API version your HTTP server is prepared to receive and process. In this code we use version 2.0.

3. Specify the parameters in **config.py**:
    a. Enter the validator shown from the Meraki Dashboard
    b. Enter the secret you specified from the last step
    c. Save your changed to the python file.

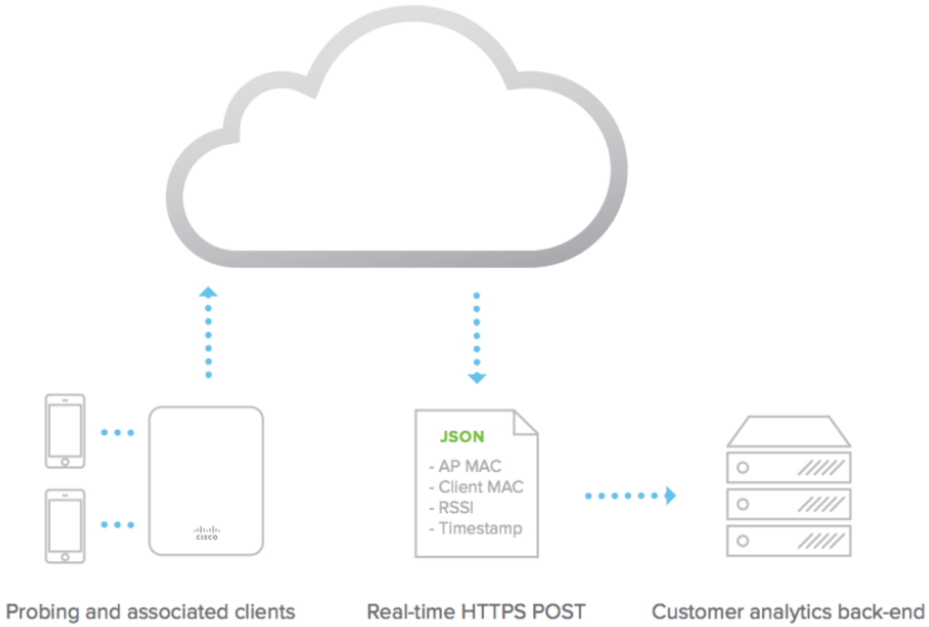
4. Configure and host your HTTP server to receive JSON objects. Run python script
    a. In a terminal, install Flask: `pip3 install flask`
    b. Run: `python3 location_scaning_v2.py` in a terminal.
    c. This should start the server on localhost:5004

5. Back to the Meraki Dashboard, under **Network Wide > General**
    a. Click Validate button

b. Upon the first connection, the Meraki cloud will perform a single HTTP GET; the server must return the organization-specific validator string as a response, which will verify the organization's identity as the Cisco Meraki customer. The Meraki cloud will then begin performing JSON posts.

6. Provided information should be displayed in Kibana.

Workflow



The Scanning API delivers data in real-time from the Meraki cloud and can be used to detect WiFi (associated and non-associated) and Bluetooth Low Energy (BLE) devices in real-time. The elements are exported via an HTTP POST of JSON data to a specified destination server. The raw data is aggregated from all access points within a network on the Meraki cloud, and sent directly from the cloud to an organization's data warehouse or business intelligence center. The JSON posts occur frequently, typically batched every minute for each AP.

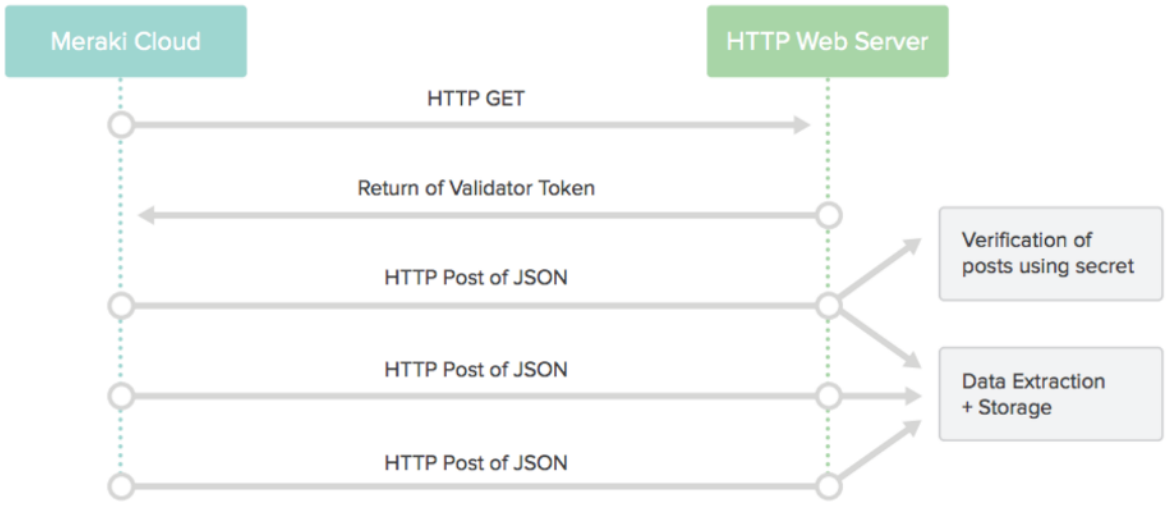
Using the physical placement of the access points from the Map & Floorplan on the Dashboard, the Meraki cloud estimates the location of the client. The geo-location coordinates (latitude, longitude) and X,Y location data accuracy can vary based on a number of factors and should be considered a best effort estimate. AP placement, environmental conditions, and client device orientation can influence X,Y estimation; experimentation can help improve the accuracy of results or determine a maximum acceptable uncertainty for data points.

Protocol flow

As figure below shows, An HTTP GET request will first be sent from Meraki Cloud to your server, expecting the server return a "validator" key that matches the Meraki network's validator.

Then the Meraki Cloud will send JSON message containing client information to the local server.

The JSON is checked with the secret, version and observation device type. After validation, the data is extracted and sent to the database.


Meraki Cloud
HTTP Web Server
HTTP GET
Return of Validator Token
HTTP Post of JSON
HTTP Post of JSON
HTTP Post of JSON
Verification of posts using secret
Data Extraction + Storage

Captive Portal – Splash Page

Splash page code is based of example code provided on Meraki website.

More information regarding Splash Portal Can be found on Meraki website.

Requirements

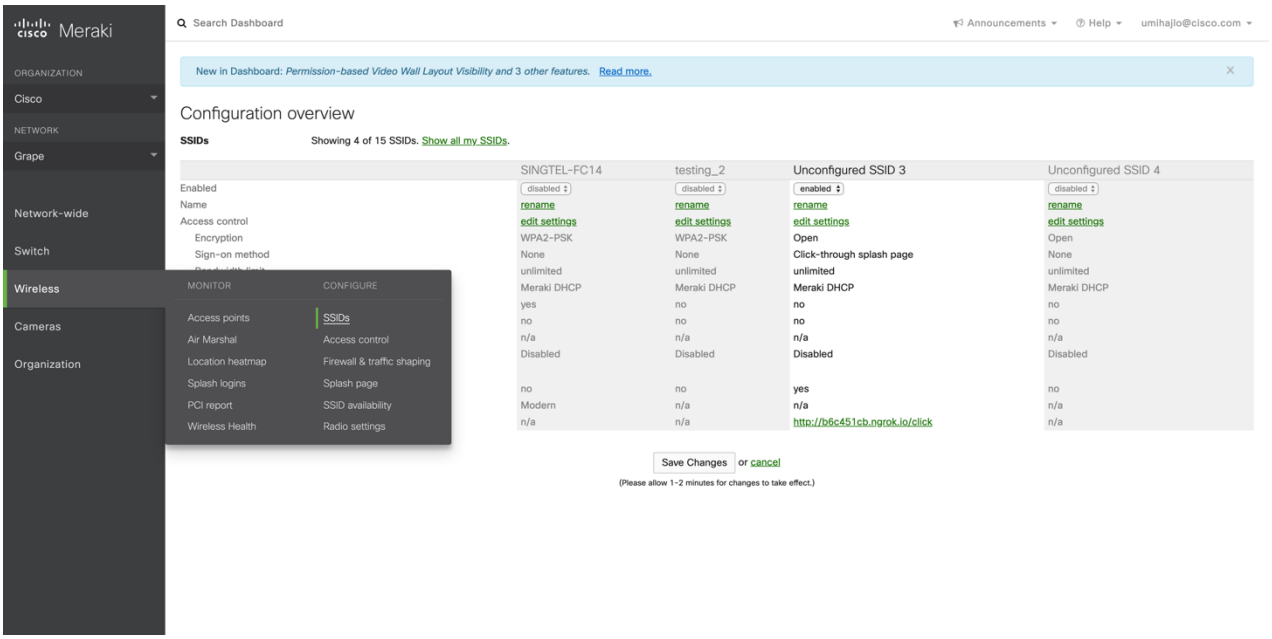
- Ensure that **ngrok** is up and running pointing to correct ports as per previous instructions.
- You also need code from GitHub.
- Install Flask
- Meraki AP and Meraki Dashboard access.

Setup

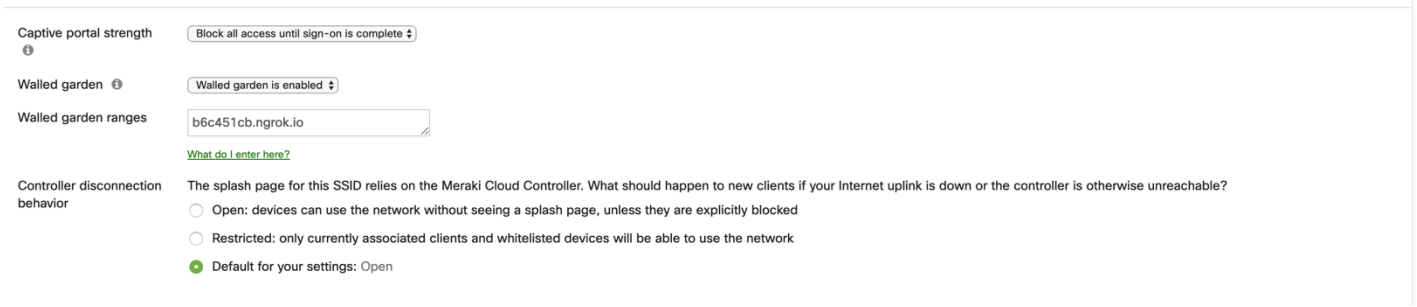
1. Run python script
    a. Run: `python3 Meraki_captive_portal.py` in a terminal.
    b. This should start the server on localhost:5000

2. The **ngrok** instance should already be running as per previous instructions.
    a. If ngrok is not active
        i. Run: `ngrok start –all`
        ii. Or Run: `ngrok http 5004` (if you want to only test splash page)
    b. Remember your ngrok url you gonna need it for the next step.

3. Ensure that at least one SSID is enabled on your AP.
    a. In Meraki Dashboard side panel naviaget to Wireless -> Configuration -> SSIDs
    b. Ensure that at least one SSID is enabled (For example, we used "*Unconfigured SSID 3".)*



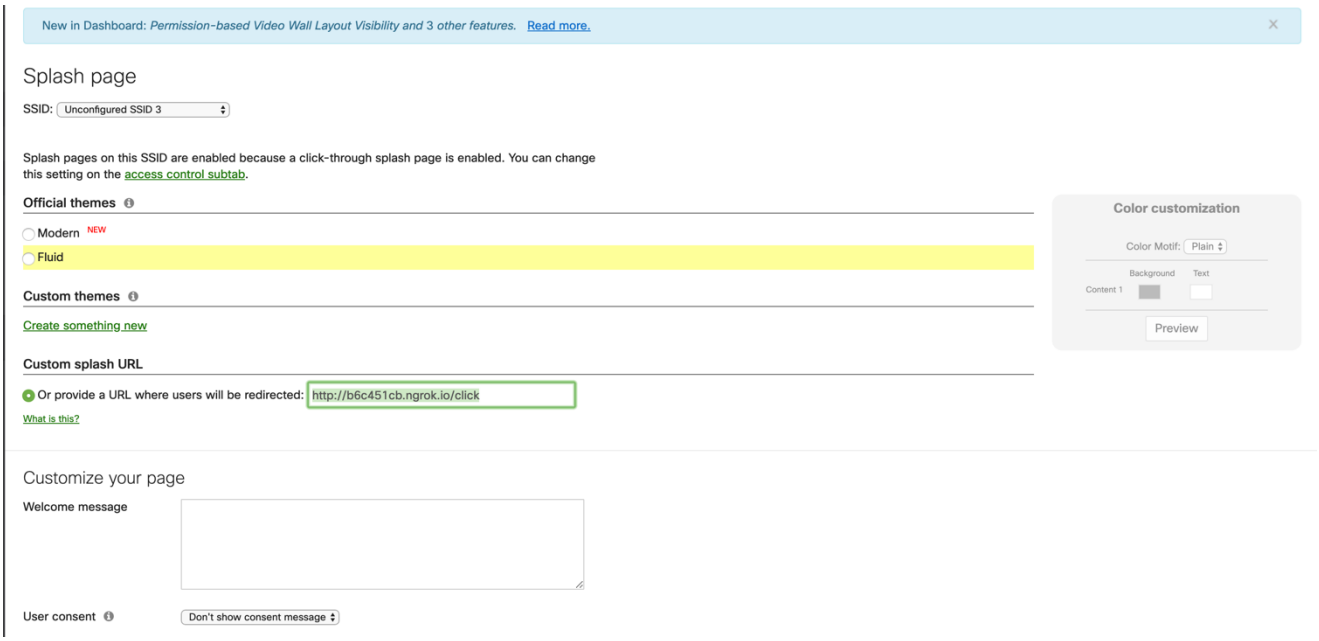
4. Setup Access Control for SSID
    a. Click on edit settings underneath SSID of your choice
    b. Access Control page should open up
    c. Under **Splash Page** section select **click-through** option.
    d. **Enable wall garden** and set **wall garden ranges** to ngrok url.
    For example:



Notice that you exclude ***https*** part in front of ngrok url

    e. Everything else can be left by default.
    f. Don't forget to save changes.

5. Setup custom Splash Page
    a. In the side panel navigate to **Wireless -> Configure -> Splash Page**
    b. Select Custom splash URL
    c. Insert your full ngrok url together with */click.* For example:

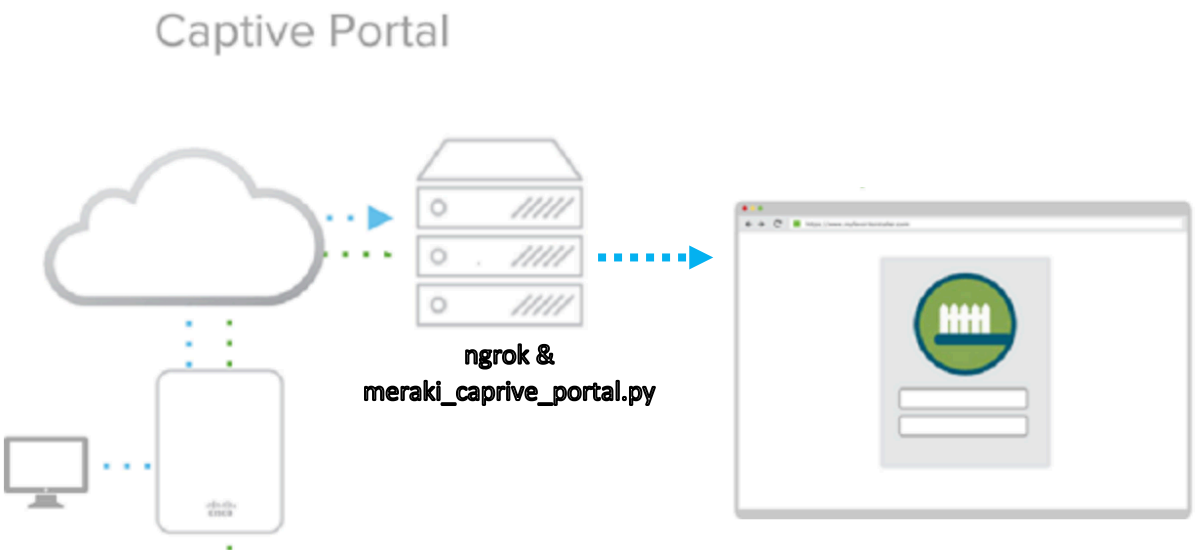


6. Don’t forget to save changes.
7. To test, connect to AP and you should be redirected to the following portal

8. Once you follow through the portal instructions you should be redirected to Verizon website showing content related to your selected interests.
9. Provided information should be displayed in Kibana.

Note: If you re-run ngrok you would need to change above described parameters with new ngrok URL.

Workflow



1. Once the user connects to Meraki access point, they get redirected to Splash Page
2. Meraki then makes http request to specified web server.
3. In this case web server is represented by python code (i.e. **Meraki_captive_portal.py)** which is hosted on ngrok.
4. Python code hosts splash page and also gets all information from http request from Meraki.

Meraki MV Sense

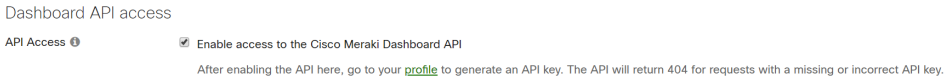
Requirements

- Ensure that **ngrok** is up and running pointing to correct ports as per previous instructions.
- You also need code from GitHub.
- Install Meraki-sdk
- Meraki AP and Meraki Dashboard access.

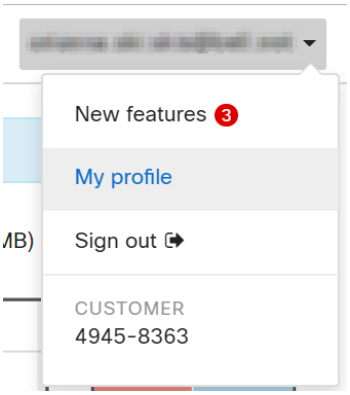
Setup

1. The MV Camera is framed and placed such that the left zone captures the people exiting the store and the right captures people entering the store.

2. First, logon to Meraki Dashboard and enable API access:
    a. Navigate to **Organization > Configure > Setting**s.
    b. Scroll down to Dashboard API access and click to enable access

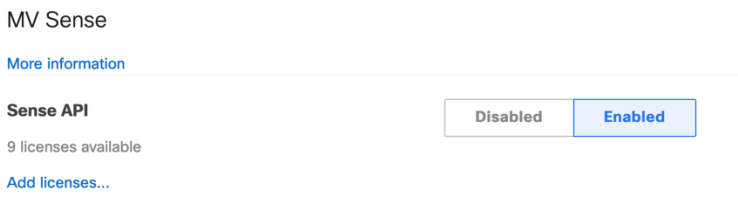


c. Go to your profile page to generate an API key. To access, click on your account email located in the top right of the Dashboard and select My Profile.



d. Scroll down to API Access and generate a new API key.

3. Now that you have Dashboard API access, it's time to enable MV Sense!
   a. Start by navigating to Cameras > Monitor > Cameras and selecting the camera you would like to enable MV Sense on.
   b. Once the camera is selected, go to Settings > Sense.
   c. Click Enabled.



   d. If this is your organization's first time using MV Sense, you will have 10 free perpetual licenses available to use. If you have exceeded this 10 free-license counts, you must activate more licenses by navigating to Organization > Configure >License info and claiming more licenses.

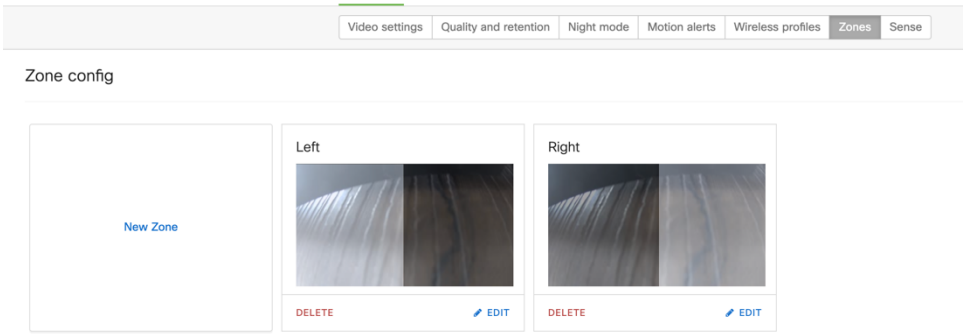
(Check here for latest information)

4. Install the Meraki SDK for Python3.
   a. In a terminal, install Meraki-sdk: `pip3 install Meraki-sdk==1.5.0`
   b. Alternatively, go to this site, and follow the instructions.

5. Fill up the **config.py** file as follows:
   a. CAMERA_INDEX_NAME : Name of the index you want to store the information in Elastic

b. MERAKI_API_KEY : The key you get from Step 1(d).
c. CAMERA_SERIAL : The serial number of the camera.
    i. Login to Meraki Dashboard, go to Cameras > Monitor > Cameras and select the camera.
    ii. Click on Network tab and you will find the Serial Number on the left.

6. Setting up split zones:
    a. Login to Meraki Dashboard and go to **Cameras > Monitor > Cameras** and select the camera.
    b. Go to **Settings > Zones**.
    c. Drag and drop to select the zone such that it captures the exit (Left) and call it Left or any other name. Do the same for entrance (right) and call it Right or any other name.



7. Alternatively, if you want, you can have 2 cameras – one facing the entrance and the other for exit, and there would be no need to create zones. The code would have to be slightly modified for this use-case.

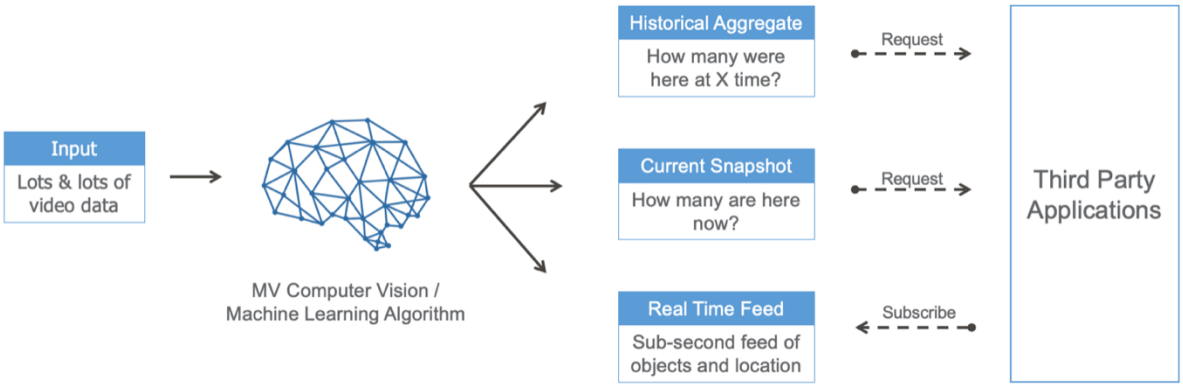
8. In a terminal, run: `python3 camera_count_v2.py`
The following explains how it works:
    a. The script checks every 5 minutes – how many people entered and exited the store in the last 5 minutes. It does this by detecting the number of people in the 'Left' and 'Right' Zones.
    b. It uses these numbers to add to the total counter and pushes the numbers to Elastic Database.

9. To view the results, go to Kibana and choose the appropriate index (camera_count_v2 in this case), the type of chart and create a visualization. Then choose the time period you would like to view results.

Workflow

MV Sense is designed to allow you to truly integrate the edge-computing capabilities of Meraki MV Smart Cameras into business solutions while keeping in-line with Meraki simplicity.

Through both REST and MQTT API endpoints, request or subscribe to historical, current, or real-time data generated in the camera to feed into your application and start using your camera for more than just security.



MV Sense has a number of endpoints which provide aggregate data on the following:

- People Detection
- Vehicle Detection (cars, bikes, trucks)
- Light Level Readings