Design Document

Fortinet IP Prefix Lookup Service

Written By	Rahul Prajapati
Version	1.0.0

Table of Contents

Introduction

This project is about lookup IP predix in optimal way.

Requirements

```
# IP Prefix Lookup Optimal Solution

Assume there's a script that collects a list of IP subnets owned by all the public service providers. These IP subnets with their prefixes are provided in `prefixes.json` file. Each subnet has a Cloud Service Provider and some tag(s) attached to them. Each IP Subnet will consist of multiple IP addresses, based on the prefix. The biggest subnet we can have is /8 (16,777,216 IPs) and smallest subnet is /32 (1 IP).

As a user, we are going to provide either a single IP address or list of IP addresses via REST API. These IP(s) might or might not belong to the above IP subnets. Also these (IPs) could belong to multiple subnets. The expected output is name of the Cloud Service Provider and related tag(s), if matched for user input.

Example: The provided data contains “184.51.33.0/24” with “Akamai” as provider and “Cloud”, “CDN/WAF” as tags. So if user provides “184.51.33.230” as an input, the API should return something like `{“result”: [{“subnet”: “184.51.33.0/24”, “provider”: “Akamai”, “tags”: [“Cloud”, “CDN/WAF”]}]}` as output. As mentioned above, there could be multiple matching subnets.

Your task is to find a solution that's:
* The most efficient way to store this data.
* The fast way to figure out if a user provided IP address belongs to a certain Cloud Provider Prefix or not?
* The ideal time for looking this up should be less than ~300ms for a batch of 10 IP addreses.
* Searching a single IP address should be less than ~50ms. (excluding N/W trip time)
* A single IP could belong to multiple subnets too.
* You can prepare/model the data as per your preference, if needed.
* Feel free to use any database for the task. Using a database is not mandatory.
```

```
Build RESTful endpoint(s) to search in Python (Flask RESTful/FastAPI preferred):
* Single IP address
* Multiple IP addresses (batch)

Follow the RESTful design standards and coding standards. The code should be up to
prod standards and follow best practices.

## References

You can read more about IP Prefixes and Subnets here:
* https://www.cloudflare.com/learning/network-layer/what-is-a-subnet/
* https://medium.com/netdevops/what-are-network-prefixes-e1923a1d6a3e
* https://avinetworks.com/glossary/subnet-mask/
* https://www.calculator.net/ip-subnet-calculator.html
```

Assumptions

- Source to find the data already exists.
- Source file will be in json format.
- Authentication/Authorisation is not required.
- Frontend part is not covered in the system. User can interact with Restful services.
- Requested data store is not required (Optional).

Failure Mode

- If source file is not available, then service will raise exception.
- Invalid input data such is IP formatting is not valid then it will raise exception.
- In batch IP processing, any IP is found to be invalid or any unknown exception arise then also service will continue for other IP in list. In the response it will send empty result and message with code for partial failure.

Acceptance Criteria

- Service should process the single IP and return service and all matching tags associated with requested IP.
- For IP batch, service should continue if any ip search is broken.
- Service should be Idempotence; In every time service should give the result in ordered way response.
- In case any exception arise duing IP finding, service should retry and break once max_retries is over.
- Service should be fast and efficient to find the IP details.
- Service should take less than ~300ms for a batch of 10 IP addreses.
- Searching a single IP address should be less than ~50ms. (excluding N/W trip time)

Solution overview

This service is to find IP subnet details in single and batch ip.
For both of the case we will process request in different way.

- First we process the sources data in order to faster searching.
- As this is prefix search, so using Tries to store the source for better performance in searching operation.
- On every requires we are not processing the source data (i.e convert to tries), this also takes times, hence using caching.
- Once we process the source data, we will save that data into cache, We are using redis for caching and Write through caching strategies,
- Storing data in binary format by pickling the object in caching in order as it is space efficient and better way to store the object.

Service is built with 5 different modules;

- Data Validation
- Data Processor
- Search Engine
- IP processor
- Batch IP processor

Data Validators:
This takes care on input data, weather the data is correct, request type.
IP formate is correct and other request param validation.

Data Processor:
This is about convert the source data to trie, This can be used as s cron, which will fetch thed ata from sources and create a json file.
Once data is present, then convert that to Tries, and store the trie object to cache in redis in binary format.

Search Engine:
This is responsible for searching a IP in preprocessed trie object.
If it doesnt find the object, it will try to process the raw json file and then find the ip from that object and again set the key in cache for future request.

IP Processor
This take IP from request and make call to search engine, accept single IP process it and send response to called. This is like a bridge between search engine and controller.

Batch Processor
We have two type of batching here.
Single worker batch processing:
All IP id processed by same worker, This will call IP processor sequential in loop, collect the result and send the response back with all IP< If any IP is failed then it will set empty result for that with error message.

Multi worker batch processing.
In future is we receive high amount if batch size requests, so out service should be reliable and efficient enough for scaling, Hence we are using second type batch processing by multiple worker .
This are celery worker are will be running as standalone and decoupled from main application, this will run in isolated container, so we can scale it according to situation.
We send call the request to celery brokers and worker will process the data and and back to call.This way service can be scalable enough to handle high load.

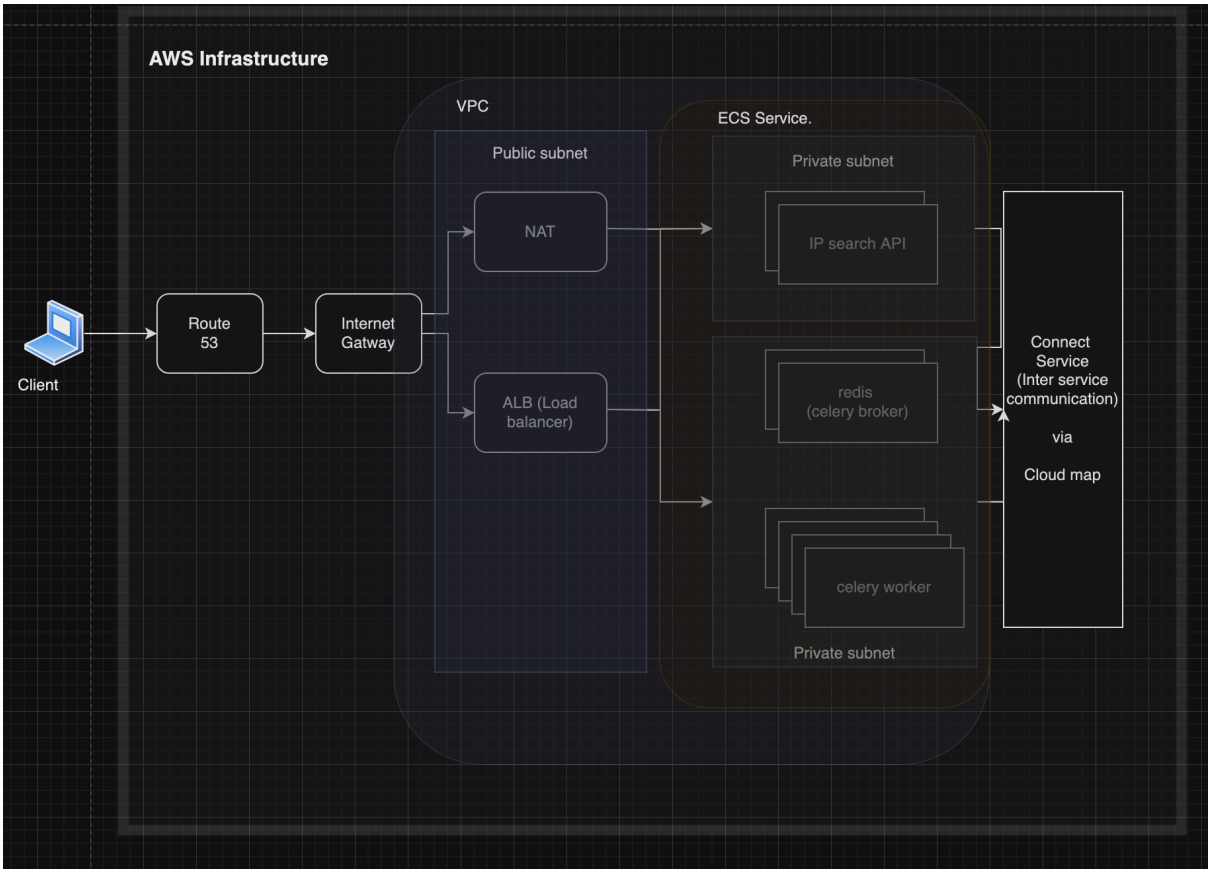
Tech stack

Flask rest: web framework to process the requests.
Uwsgi - To run flask application will use uwsgi as application server. It has a large request handling capacity and easy to scale.
Celery: Tasks queue for processing large number of data.
Redis: For as caching and celery broker, for broker we can use SQS as well in future
Docker: To deploy the service anywhere across different system.
AWS cloud services:

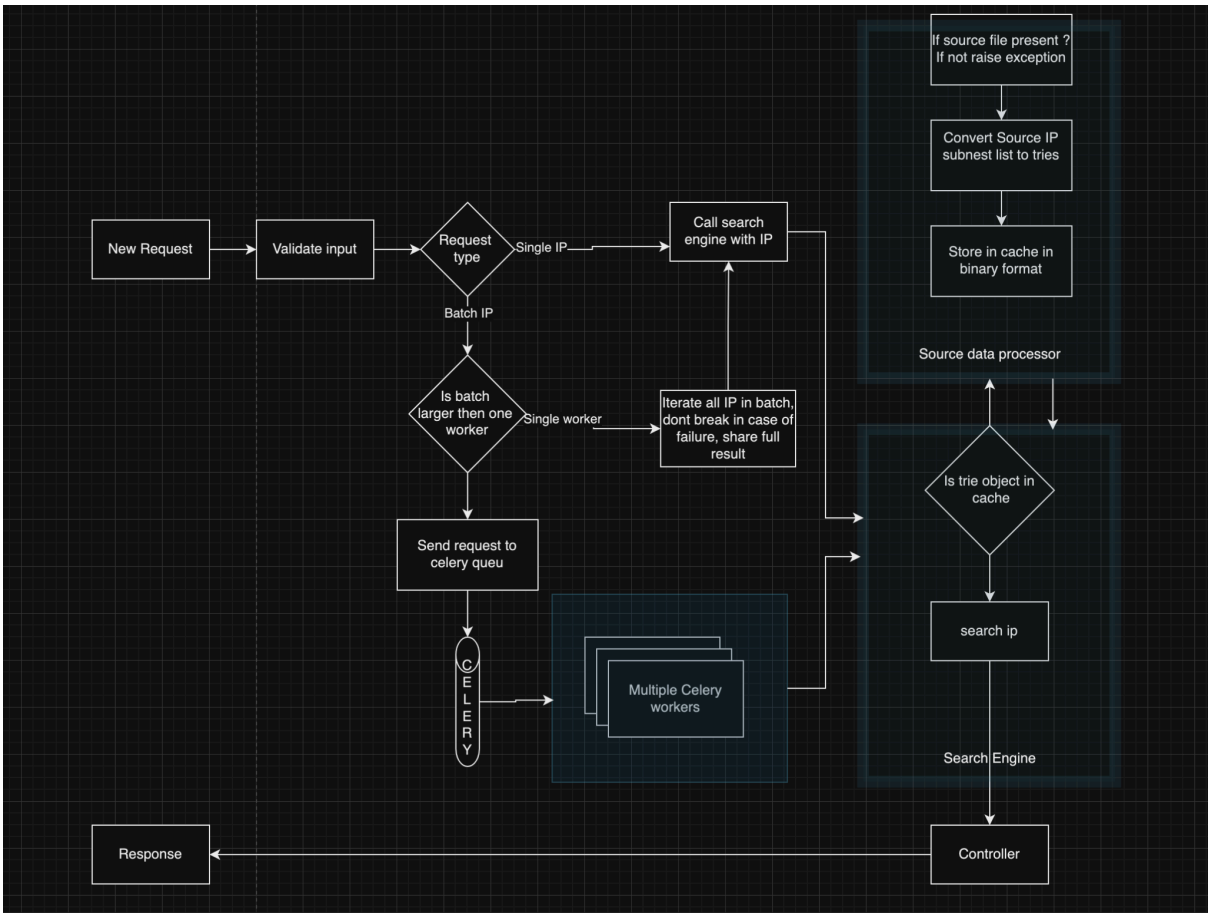
- ECS: As a main infra, application will be running in ECS cluster.
- This is containerised services and using 3 main services;
  Web container (As ECS service, autoscaling ,NAT and services connect, private subnet)
- Redis container (As ECS service,autoscaling, private subnet)
- Worker container (As ECS service,autoscaling,, private subnet
- ALB Load balancer- This will be calling web service, which accept incoming request and send to uwsgi app.
- And Internet gateway, WAF, Rout53, ECR

Git: Code management system, github action for CI-CD automation.
AWS Cloud-formation for infra as code,auto deployment on different env.

Infrastructure architecture



Application architecture



API - Restful services

Name: Search IP
Description: Search IP and return all possible subnet with services and tags.
URI: search-ip-prefix/v1/search-ip
Method: Post
Param: {“ip”: Union[list, str], “request_type”: “single/batch[str]”
Example:

1) {"ip": ["18.154.10.90","18.154.10.91"],"request_type": "batch"}
2) {"ip": "18.154.10.91","request_type": "single"}

Output Example:


```
"data": {
       "18.154.10.39": [
           {
               "18.154.0.0/15": {
                   "service": "AWS",
                   "subnet": "18.154.0.0/15",
                   "tags": [
                       "Cloud",
                       "CDN"
                   ]
               }
           },
           {
               "18.154.0.0/15": {
                   "service": "Akamai",
                   "subnet": "18.154.0.0/15",
                   "tags": [
                       "Cloud"
                   ]
               }
           }
       ],
```

TODO:
Due to time constraints many feature is missed includes-
Decouple single and batch ip search API endpoints.
Implement Celery workers
Implements Loggers
Format response from for batch IP.
CI Pipeline setup
Cloudformation setup.