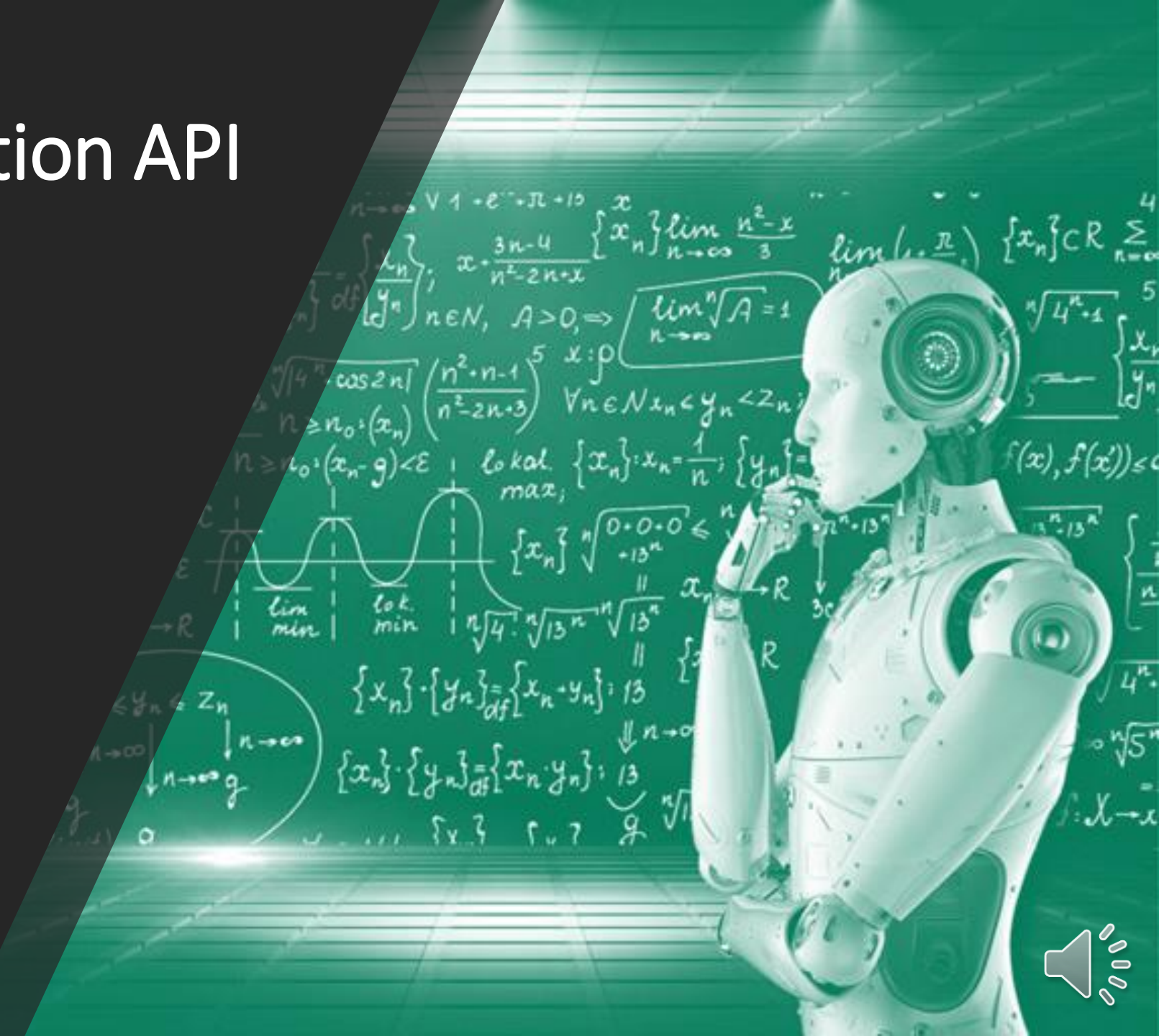
DGA Domain Detection API



Group 3 Achievers:

- Aftab Alam
- Srijani Basak
- Srinivasa Murri
- Harpreet Nanrhay
- Vineeta Singla

Safe Harbor

- Project is done as part of a class assignments
- Results may vary in real world
- Budget /Cost analysis would vary depending on type of AWS license or service used.
- Code in GitHub is not production ready.
- Other unknown risks.

High level Requirements

- Design a RESTful API that predicts if a FQDN or URL is a DGA or not.
- Sub-second response time
- API must have ability to throttle usage and protect itself from unauthorized used.
- API must adhere to industry standard high scalability and availability (99.99%)
- Provide Billing API to monitor usages and cost for each customer
- Provide Auditing Capability at each request/response level
- Prediction model must achieve high accuracy(85+%)

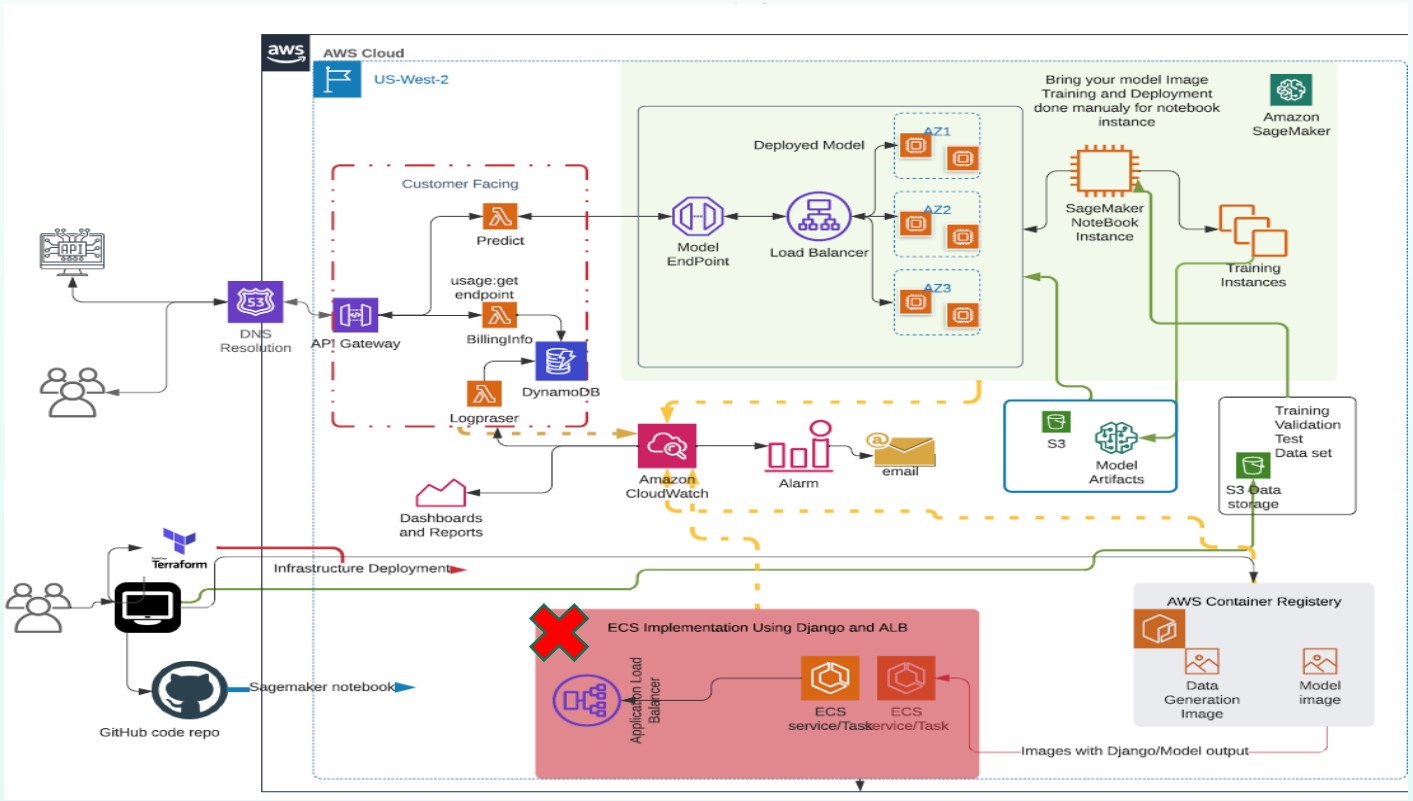
System Architecture

Design Considerations:

In order to make highly scalable and available API with low cost and effort, we have selected AWS as cloud platform and used AWS serverless and managed services as much as possible.

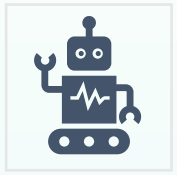
Technologies Used:

- AWS cloud
  - API Gateway
  - Sage maker
  - Lambda
  - S3
  - Docker
  - AWS ECS
  - CloudWatch
  - DynamoDB
- Terraform for IAAC
- Python programming language



Approach

System Design



POCs

System Integration

Testing

Exploring prior Arts

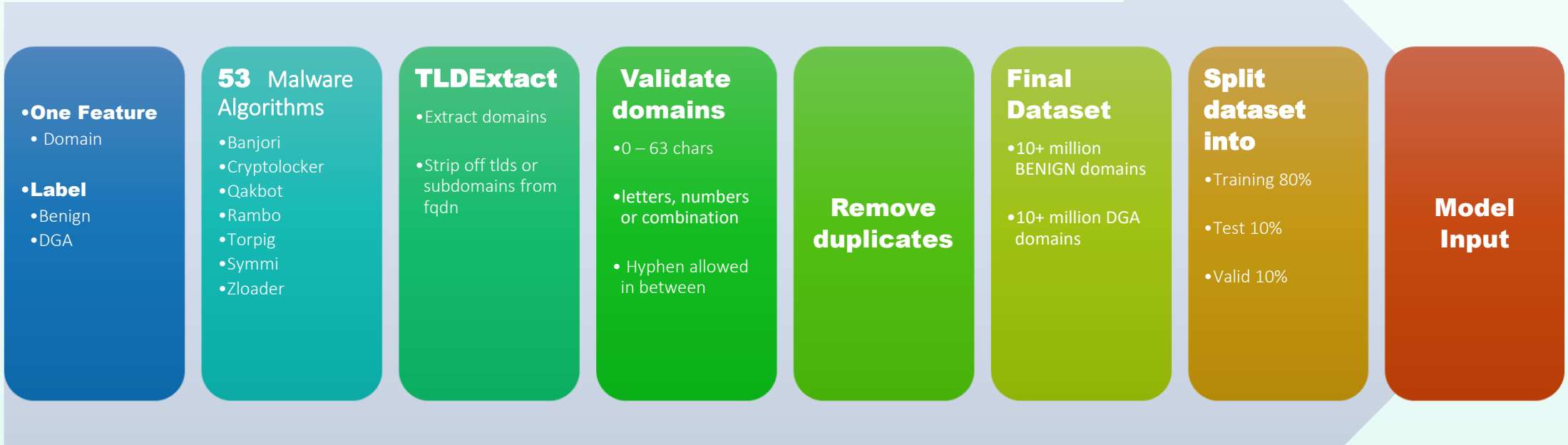
Data Creation

Feature Engineering

Model selection

Model Tuning at Scale

Training Dataset



Data Sources

DGA Families

bamital	dnschanger	matsnu	padcrypt	monerodo wnloader
banjori	dyre	murofet	pitou	qsnatch
bobax	enviserve	mydoom	pizd	ramdo
ccleaner	fobber	necurs	proslikefan	ramnit
chinad	gameoverzeus	newgoz	pushdo	ranbyus
corebot	gozi	nymaim	pykspa	reconyc
cryptolocker	kraken	nymaim2	qadars	rovnix
dircrypt	locky	tinba	qakbot	shiotob
sisron	symmi	tofsee	torpig	simda
suppobox	tempedreve	vidro	zloader	vawtrak

Benign

1.Alexa Top 1M
2.Cisco Umbrella
3.Majestic Million
4.Domcop top 10 mn
5.Google Search API

Feature Engineering

AS TEXT MUST BE PROCESSED NUMERICALLY IN ORDER TO BE FED INTO THE MODEL, WE DID SOME REQUIRED FEATURE ENGINEERING ACTIVITIES:

CLEANSED THE DATA , LIKE REMOVAL OF PUNCTUATIONS, REMOVAL OF SPACES, TAGS

GENERATED A DICTIONARY OF VALID CHARACTERS EXTRACTED FROM THE DATA

REPRESENT STRINGS AS NUMERICAL ARRAYS, CONVERTED CHARACTERS TO INTEGERS AND PADDED TO CREATE A STANDARD NUMBER OF FEATURES

CONVERTED THE DEPENDENT CLASS VARIABLE TO NUMBERS 0, 1 FOR BENIGN AND DGA RESPECTIVELY

WE CREATED A TEST-TRAIN SPLIT WITH 80:20 RULE.

Model Selection

We have good amount of data and so flexibility to do cross validation.

- With prior knowledge we shortlisted some applicable algorithms.
- Tried multiple algorithms like SVC, Decision Tree , Logistic Regression etc. and shortlisted XGBoost and LSTM based on Accuracy and other performance metrics.
- Considered the other key factors like Performance, Accuracy , Cost, operational ease , Training complexity
- Finally , We traded XGBoost over LSTM

Model Tuning

- Increase training dataset size(1mn to 21 mn)
- Different xgbooster – gbtree, gblinear, dart
- Different Number of rounds from 1000-3000
- Binary logistics vs SquareError
- Several round of hyper tuning jobs with below Parameters
- Added ratios of unique number of char/length of domain in feature
- TensorFlow Sequential model with LSTM
- Trained on Sage maker using script mode

```
# Define exploration boundaries (default suggested values from Amazon SageMaker Documentation)
hyperparameter_ranges = {
    'alpha': ContinuousParameter(0, 1000, scaling_type="Auto"),
    'colsample_bylevel': ContinuousParameter(0.1, 1,scaling_type="Logarithmic"),
    'colsample_bytree': ContinuousParameter(0.5, 1, scaling_type='Logarithmic'),
    'eta': ContinuousParameter(0.1, 0.5, scaling_type='Logarithmic'),
    'gamma':ContinuousParameter(0, 5, scaling_type='Auto'),
    'lambda': ContinuousParameter(0,100,scaling_type='Auto'),
    'max_delta_step': IntegerParameter(0,10,scaling_type='Auto'),
    'max_depth': IntegerParameter(0,10,scaling_type='Auto'),
    'min_child_weight': ContinuousParameter(0,10,scaling_type='Auto'),
    'num_round': IntegerParameter(1000,3000,scaling_type='Auto'),
    'subsample': ContinuousParameter(0.5,1,scaling_type='Logarithmic')}

objective_metric_name = 'validation:accuracy'

tuner_log = HyperparameterTuner(
    xgb,
    objective_metric_name,
    hyperparameter_ranges,
    max_jobs=20,
    max_parallel_jobs=1,
    strategy='Bayesian'
)
```

```
def get_model(learning_rate,max_features=MAX_FEATURES,maxlen=MAXLEN):
    model = Sequential()
    model.add(Embedding(max_features, 128, input_length=maxlen))
    model.add(LSTM(128))
    model.add(Dropout(0.5))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='rmsprop')
    return model
```

Accuracy after tuning

Results:
After Running
- 50 Tuning jobs in sage maker
- ~200+ tuning hours on single ml.c5.4xlarge

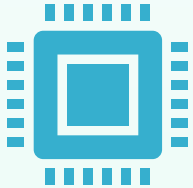
Accuracy :
75% to 92% on test data set
78% to 90% on unknown 500 data set

Technologies (Pro's & Con's & Options Explored)

Model Tuning

Local

SageMaker



Model Hosting

API Gateway, lambda, SageMaker

API Gateway, ALB, ECS with Django/Docker

Transaction Logging

Cloudwatch

DynamoDB

What can be done better

Training Data

- Obtaining Benign data set from trusted source
- Proper representation/Weightage of DGA family

Model Selection

- Try Different Model and find a right balance between accuracy, cost and effort e.g. LSTM

Features

- Try bigram vector instead of char vector?

Hosting Service

- Check if Azure/GCP machine learning plateform has better model selections

Security

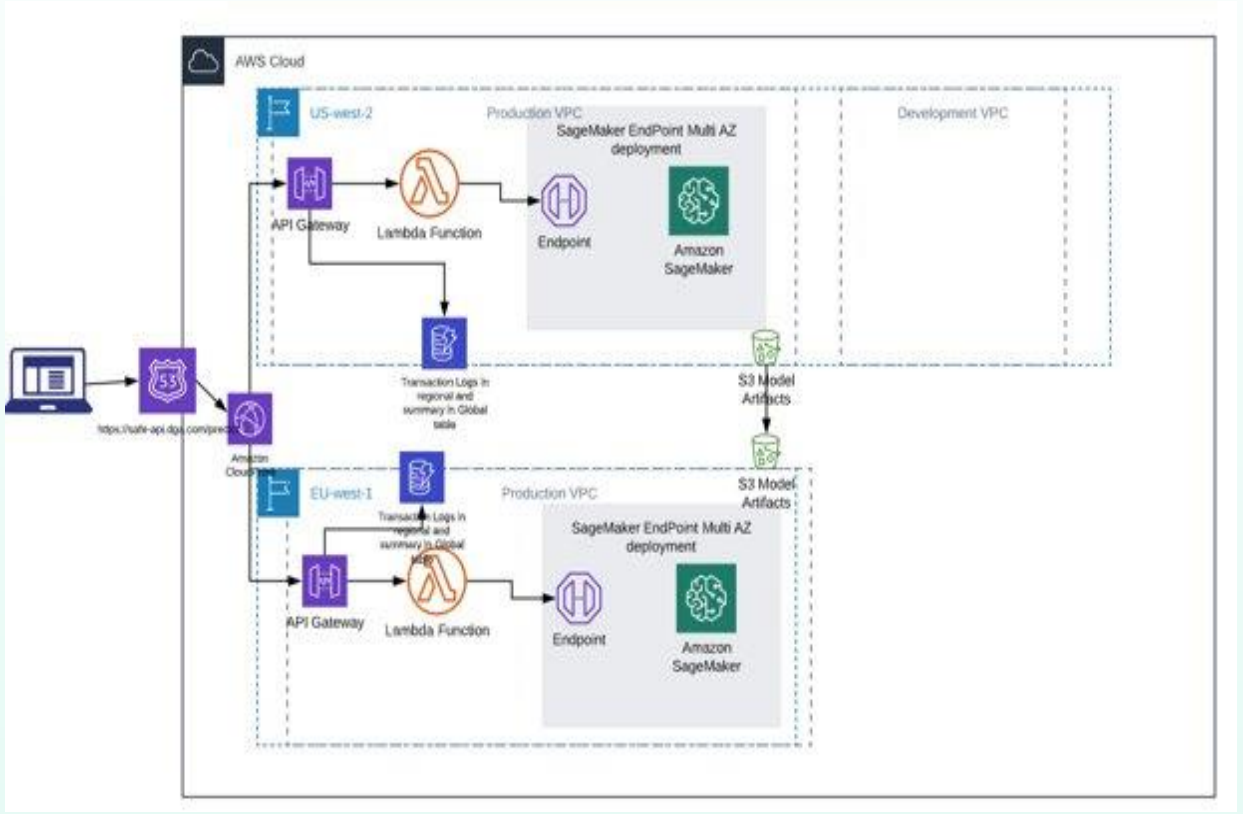
- Authorization and Authentication
- Short live token/API Keys

Analytics and Reporting

- AWS EMR, Kinesis would better for large volume and Analytics

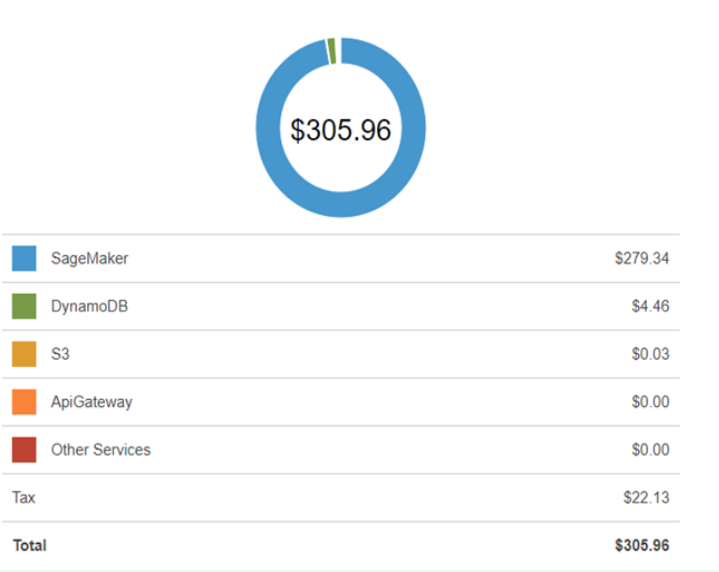
Scaling the System Horizontally across regions

- Depending of Custom from different regions we may decide to use more than two region or multiple cloud provides
- Could explore Kinesis /S3 Athena or other Analytics solution for transaction analytics
- AWS monitoring services like x-ray or Data Dog
- CloudFront distribution to increase performance and reduce repeated calls to source



Budget/Cost

Operation Cost Per month			$$$	
Expected number of customer		1000		
Number of calls Per months		1000000000		
Number of Lambda Calls		2000000000	816.6	Including requests and duration cost
CloudWatch Logs		100GB	267.5	.535 per GB for ingestion, scan and metrics
S3 storage		50 GB	10	To keep cost minimum, we will do data transfer within region
ECR		50 GB	Free	Mostly free
DynamoDB		500	1250	Mostly write operations, storage cost, reads will be minimum for billing enquiry
	2 region 3 AZ each	Instance Type		
Sagemaker Endpoint instance	9	ml.m5.large(.134 per hour)	868.32	Add autoscaling policy based on number of invocation and CPU Usage
Training Instance Cost	3	ml.c5.4xlarge	428.4	50 hours per month. Could use spot instance for training to lower cost
Total Operation cost per months		USD	**3640.82**	



Ethical Use of digital data

- Opensource Data is used
- Code sample in github are referred
- Privacy norms are maintained as no personal identification data is used
- Prior Arts and research reports mentioned appropriately wherever applicable

Weaponization risks?

If Dga identification API is provided as service, Hacker could use it to check which dga algorithms can go undetected and then target business that are using this service to protect themself.

To mitigate this risk we need to consider following

- Training dataset should be updated frequently with new DGA algorithms.
- Work with security expert and white hackers
- Do business with reputed organization(prevent hacker getting access to api)
- Strong Defense mechanism to detect abnormal use of API and block access


Thank You

References

- Prior Arts:
  - Research papers
    - http://faculty.washington.edu/mdecock/papers/cchoudhary2018a.pdf
  - Model examples
    - https://towardsdatascience.com/xgboost-in-amazon-sagemaker-28e5e354dbcd
    - https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/xgboost_abalone/xgboost_abalone.ipynb
    - https://aws.amazon.com/blogs/machine-learning/call-an-amazon-sagemaker-model-endpoint-using-amazon-api-gateway-and-aws-lambda/
  - Benign Data set:
    - http://s3.amazonaws.com/alexa-static/top-1m.csv.zip
    - http://s3-us-west-1.amazonaws.com/umbrella-static/top-1m.csv.zip
    - http://downloads.majestic.com/majestic_million.csv
    - https://www.domcop.com/files/top/top10milliondomains.csv.zip)
  - DGA data set:
    - https://github.com/andrewaeva/DGA/blob/master/dga_wordlists/main.py
    - https://github.com/baderj/domain_generation_algorithms
- Project Code base:
  - https://github.com/aftabalam01/machinelearningpipeline

Appendix

Extract domain TLD extract

In [129]:
```
%%time

def extract_domain_subdomain(record):
    domain = record.domainName
    ret=''
    try:
        ext = tldextract.extract(domain)
        ret = ext.domain
    except :
        print(record)
    return ret
def get_y(row):
    if row.label.lower()=='bad':
        return 1
    elif row.label.lower()=='good':
        return 0
    else :
        return 1
```

```
Wall time: 0 ns
```

Create new columns for domain and dga binary

In [ ]:
```
%%time
df_domains.loc[:,'domain_subdomain'] = df_domains.apply(lambda row : extract_domain_subdomain(row), axis=1 )
```

In [130]:
```
%%time
## 1 for dga and 0 for benign
df_domains.loc[:,'Y'] = df_domains.apply(lambda row : get_y(row), axis=1 )
```

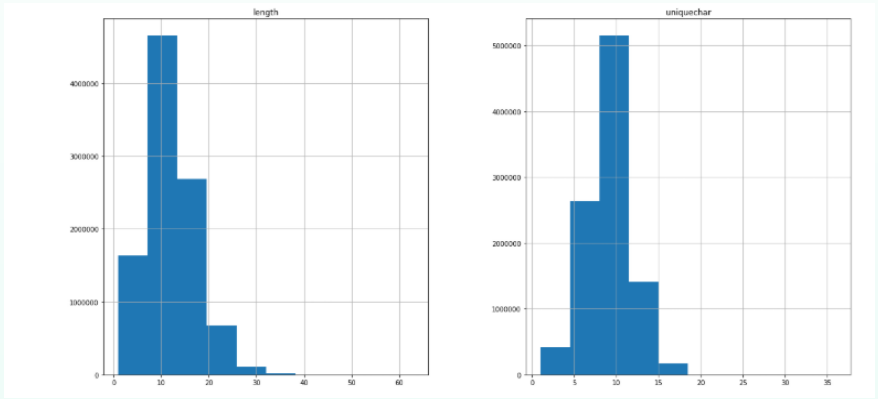
Create char vector from domain

In [139]:
```
VALID_CHARS = 'abcdefghijklmnopqrstuvwxyz0123456789-_.'
LOOKUP_TABLE = None
def pad(l, content, width):
        l.extend([content] * (width - len(l)))
        return l

def check_validchar(domain):
    for c in domain.lower():
        if c not in VALID_CHARS:
            return False
    return True



def features_extract(domain):

    global VALID_CHARS
    global LOOKUP_TABLE
    if not LOOKUP_TABLE:
        LOOKUP_TABLE = dict()
        idx = 1
        for c in VALID_CHARS:
            LOOKUP_TABLE[c] = int(idx)
            idx += int(1)
    #ds = tldextract.extract(fqdn)
    #domain = ds.domain
    #ratio = len(set(domain))/len(domain)

    rvalue = list()
    if len(domain)<=63:
        for c in domain.lower():
            try:
                rvalue.append(LOOKUP_TABLE[c])
            except:
                print(f"Char error out in {domain}: {c}")
    else:
        #print(domain)
        pass

    rvalue=pad(rvalue,0,63)
    return rvalue
```

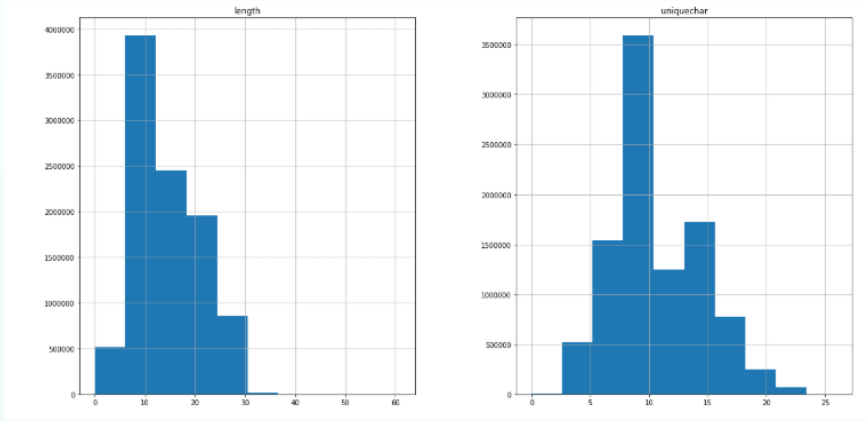
In [75]:
```
%%time
df_temp = df_domains.head(10)
x = [features_extract(D) for D in df_temp.domain]
```

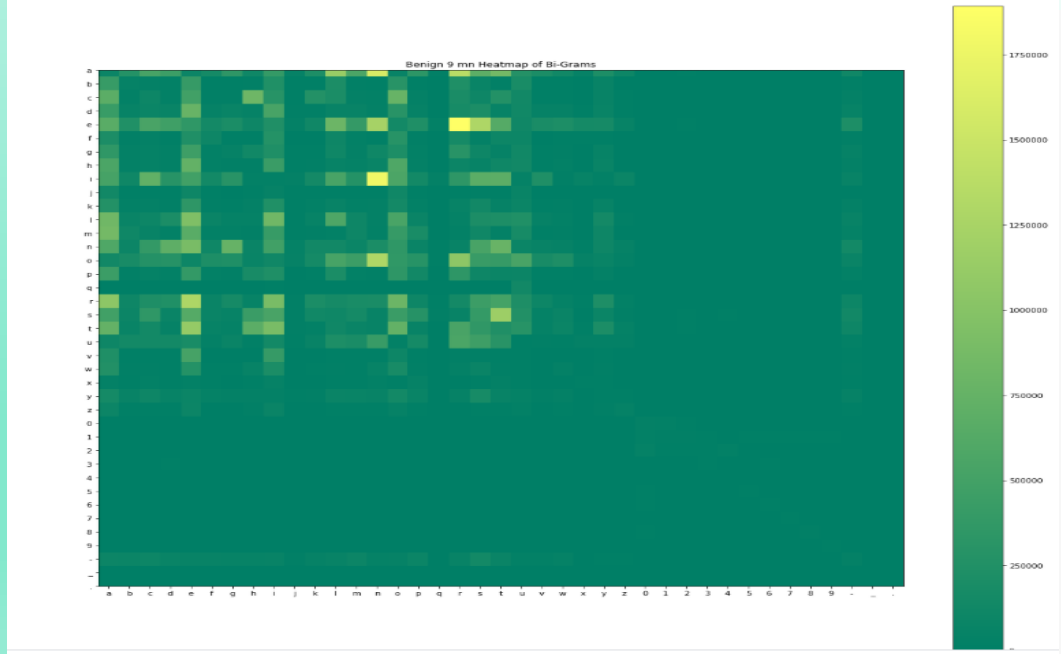
Appendix

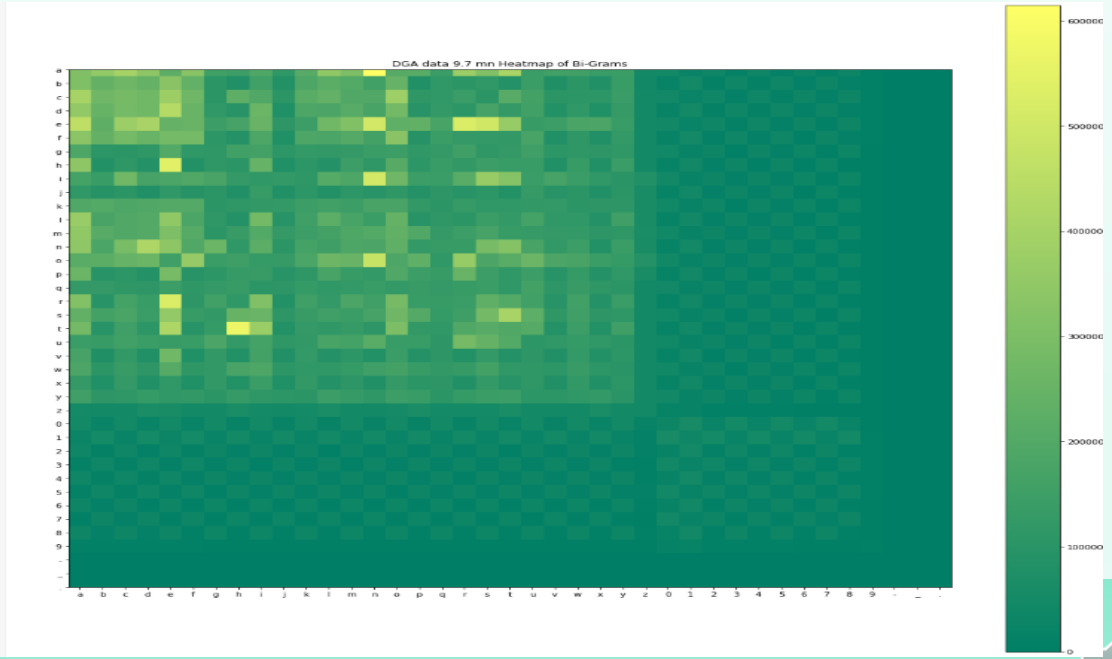
Begin Data



DGA data







Some DGA algorithms uses rare combination of char that are not seen in begin data but it is possible to generate domain using dga algorithms that gives similar char comination which will be hard to detect

Appendix

Train xgboost model and Deploy on sagemaker

```
%%time
# train model
from sagemaker.amazon.amazon_estimator import get_image_uri

container = get_image_uri(region, 'xgboost','1.0-1')

xgb = sagemaker.estimator.Estimator(container,
                                    role,
                                    train_instance_count=1,
                                    train_instance_type='ml.c5.4xlarge',
                                    output_path='s3://{}/{}/output'.format(bucket, prefix),
                                    sagemaker_session=sagemaker_session)

# fitting model with paramter from previously best tune model for this data set
# or we can start default and tune model later
xgb.set_hyperparameters(base_score=0.5,
                        booster='gbtree', #['gbtree', 'gblinear', 'dart']
                        colsample_bylevel=0.3328968814794882,
                        colsample_bynode=1,
                        colsample_bytree=0.7460086251908613,
                        gamma=4.36472704596215,
                        #reg_lambda=18.34813124562997,
                        alpha=458.20153739471834,
                        max_delta_step=8, max_depth=6,
                        min_child_weight=7.4485695445680005,
                        scale_pos_weight=1, subsample=.9, tree_method='auto',
                        eta=0.4008765966370876,
                        silent=1,
                        objective='reg:squarederror', #reg:squarederror
                        num_round=200
                       )

xgb.fit({'train': s3_input_train, 'validation': s3_input_validation})
## Deploy trained XGBoost model endpoint to perform predictions
xgb_predictor = xgb.deploy(initial_instance_count = 1, instance_type = 'ml.t2.medium')

# make sure to set content type to csv as we have data in csv format
xgb_predictor.content_type = 'text/csv'
xgb_predictor.serializer = csv_serializer
xgb_predictor.deserializer = None

## Function to chunk down test set into smaller increments

def predict(data, model, rows=500):
    split_array = np.array_split(data, int(len(data) / float(rows) + 1))
    predictions = ''
    for array in split_array:
        #print(array[0])
        predictions = ','.join([predictions, model.predict(array).decode('utf-8')])

    return np.fromstring(predictions[1:], sep=',')

%%time
## Generate predictions on the test set for the difference models
with open('domainsDataSet.test', 'r') as f:
    payload = f.read().strip()
labels = [int(line[0]) for line in payload.split('\n')]
test_data = [line[2:] for line in payload.split('\n')]
predictions = predict(test_data, xgb_predictor)
#xgb_predictor.predict(payload[2:]).decode('utf-8')
from sklearn.metrics import accuracy_score ,confusion_matrix
thresh = 0.5
y_pred = predictions
y_pred_binary = np.where(predictions > thresh, 1, 0)
accuracy_score(labels,y_pred_binary)
confusion_matrix(labels,y_pred_binary)

```