APAN5450 Group 10 Final Report

Business Case & Data

Problem statement & objective: Most digital translation tools and packages require a text input to function effectively. However, a significant amount of content, particularly in non-Latin scripts like Cyrillic, often exists only in handwritten forms. This content is inaccessible to standard digital translation tools, which limits the usability of these tools in certain contexts, such as historical document digitization, processing handwritten forms, or translating literary works in native scripts.

Leveraging AWS services, our objective is to develop a cloud-based image classification system specifically tailored for handwriting to text conversion. This tool aims to bridge the gap between handwritten content and digital translation services by providing a scalable and efficient means to convert Russian handwritten text into a digital format that can be input into translation software.

The system provides a crucial step in digital translation pipelines, particularly where the text to be translated is in handwritten format. By converting these texts into a machine-readable format, the tool opens up new possibilities for data accessibility, ensuring that users such as researchers, archivists, and businesses can include a broader range of materials in their digital transformation initiatives.

Data Procurement: We retrieved a dataset from Kaggle under the CC0: Public Domain license, which includes 73,830 segments of handwritten Russian texts. These segments, referred to as "crops," feature individual expressions with no more than 40 Cyrillic characters each. This substantial collection spans approximately 1.62 GB and is organized into training and testing sets, split 95% and 5% respectively. The significant size and variability of the dataset provide a solid foundation for training our model to accurately recognize and digitize handwritten Cyrillic text.

Group Member's Contribution

- Fei: Setup SageMaker notebook instance, train model, deploy model and endpoint. Setup API.
- Jianfei: Configure security features like VPC, Network ACLs, and Load Balancer for the model
- Madison: S3 data uploading and bucket configuration, Lambda function to invoke Sagemaker Endpoint

Cloud Architecture & Security

Our project used a set of AWS services to implement an optical character recognition system. The main components included:

- Amazon S3: Our project utilized S3 standard to store the training and test data for our classification model as well as the trained model weights. Our data was pulled from Kaggle and uploaded into the S3 bucket using the AWS S3 interface. We additionally had to compress the files before uploading, as the large training data set file was so large that it took an inordinate amount of time to upload. S3 was the logical choice for our project for a number of reasons, but primarily for its seamless integration with AWS Sagemaker. Sagemaker natively supports data stored in S3 buckets, and the integration between the services makes accessing and utilizing training data in Sagemaker. For our application, the scalability, reliability and durability of S3 will provide benefits for the theoretical future of the application as well; storing additional predictions

and training data for use in further model training from user inputs would be a simple process with the S3 and Sagemaker coordination.

- AWS SageMaker: In the project, AWS SageMaker was utilized to deploy a transformer model with a ResNet backbone, addressing the need for GPU acceleration and substantial memory by leveraging the g4dn.xlarge instance. The model training was streamlined by integrating a pre-trained model from Kaggle, given the extensive computational requirements. This allowed for efficient handling of epochs and model evaluation, achieving an accuracy of approximately 0.9 on test data. The deployment involved packaging the model artifacts, including weights and necessary scripts, into an Amazon S3 bucket followed by creating a model endpoint using SageMaker's deployment functions, providing a seamless workflow from model training to endpoint deployment. AWS SageMaker was chosen over other AWS services like Lambda, EC2, Elastic Beanstalk, ECS/EKS, and AWS Batch due to its specialized features tailored for machine learning pipelines. SageMaker simplifies the lifecycle of machine learning projects from building to monitoring models. It provides an integrated and secure environment that facilitates seamless deployment and future scalability without the need of manual setup for configuration and scaling in EC2. In the scope of this project, SageMaker is particularly effective for our goal without the need of managing infrastructure.
- Lambda: In our project, we created an AWS Lambda function to invoke our Amazon SageMaker endpoint. The purpose of the Lambda function is to handle HTTP requests for processing images using our sagemaker endpoint, and to return a prediction from the model. The Lambda function is organized around a central handler function, lambda_handler(event, context). Upon receiving an HTTP request, the Lambda function initiates a check to ascertain if the incoming data is base64 encoded. If the data is not base64 encoded, it responds indicating that the image wasn't uploaded. Conversely, if the data is indeed base64 encoded, the function proceeds to decode the image and trigger the SageMaker endpoint utilizing the boto3 SageMaker runtime client. During this invocation, it specifies the content type as 'image/png' and designates JSON as the expected response format.After receiving the response from the endpoint, the Lambda function processes it by reading and decoding the JSON body. It then extracts the prediction result, if available, and formats it as a JSON string. Lastly, the Lambda function returns a JSON response with a status code of 200, containing the prediction result in the body. From a comparative perspective, we could have used an EC2 instance to host our application, however this approach would have required a larger software stack to run our web application which would have incurred higher costs and involved several idle resources; Lambda’s pay-as-you-go and serverless model was the better choice to reduce unneeded software bloat, management and cost.
- Amazon API Gateway: A HTTP API endpoint was created to handle users’ API requests. It invokes the Lambda function to allow users to interact with the sagemaker endpoint. Choosing Amazon API Gateway for handling API interactions was appropriate due to its seamless integration with other AWS services such as AWS Lambda and SageMaker, and its ability to scale up effortlessly. Compared to other potential solutions, API Gateway offers a more direct and optimized route for deploying and managing APIs, especially in cloud-based environments. Its capability to scale automatically and handle bursts of traffic makes it ideal for applications requiring high availability and performance.

Design Principles:

- Reliability & availability: Our cloud architecture uses managed services like AWS S3, sagemaker and Lambda, offering high reliability and availability due to robust infrastructure and fault tolerance.
- Performance & scalability: By using Lambda and SageMaker, our design has high scalability. Amazon API gateway and Lambda automatically scales with the number of requests and S3 can also handle increased volumes of data easily.
- Cost optimization: Our system leverages AWS services in a cost-optimized manner, utilizing scalable storage, managed machine learning environments, serverless compute, and efficient API management to minimize operational expenses while maximizing performance and scalability. Zipping the training data, the test data and the model weights before uploading in S3 minimizes data transfer costs and optimizes upload times. Leveraging a pre-trained model for our classification system was another choice which led to cost and compute savings. The implementation of the Lambda function optimizes compute, and thus cost, for our specific application, as Lambda is serverless architecture and procures costs on a pay-as-you-go model; we have no idle resource costs and only incur cost when handling HTTP requests. Further, the choice of API Gateway, which auto-scales to match workloads ensures that we are not paying for over-provisioned resources.
- Security: We firstly created the fundamental VPC with 2 public subnets for later use, setting up HTTP:80 and SSH:20 as rules to mitigate network traffic. Then we apply ALB in the Load Balancer for attacks mitigation and prevention. We also have set up stateless network ACLs for enhanced security.

Security Features:

- VPC: We created public and private virtual public clouds for different users. In our case specifically we applied the public one as we are creating a user-facing interface.It can significantly prevent unauthorized accesses and creates an segregated environment. We may also control who has access to the resources deployed within the VPC managed by route tables and network ACLs. In addition, we configured a security group as a firewall and 2 subnet groups each for public and private VPCs, ensuring the customizable setting.
- Load Balancer: We applied Application Load Balancer to ensure availability and fault tolerance of our VPCs. It can also mitigate DDos attacks by distributing traffic spikes over a broader footprint. On the other hand, load balancers can handle SSL/TLS termination, thereby offloading the cryptographic responsibilities from the application servers. This not only optimizes resource usage but also centralizes certificate management for enhanced security.
- Network Access Control Lists(ACLs): Unlike security groups, network ACLs are stateless, meaning they do not track the state of network connections. Each packet is processed individually, allowing for fine-grained control over network traffic. When used in conjunction with security groups, network ACLs provide an additional layer of security. This helps to enforce a robust security posture where both inbound and outbound traffic can be controlled explicitly.

- IAM: Due to permission limitations, we are unable to access Identity and Access Management. In the future, we would intend to manage access to AWS resources and features regularly using IAM. Its roles allow entities (users, applications, or services) to assume temporary credentials that grant them access to AWS resources. This means that permissions can be more securely managed without having to distribute long-term credentials.

Future extensions and security considerations:
For enhancing availability, we could use multiple availability zones for our API services. For higher availability and fault tolerance, we could deploy our lambda function on multiple regions as well.
For future model updating, we could consider taking advantage of user input. We could expand our service by saving user uploaded data in S3 to enable them view historical recognition and provide feedback. We could also set up an automated model retrain pipeline in sagemaker utilizing more services like CloudWatch Schedule Trigger. As the new data in future extensions is sensitive, security techniques such as encryption of data in transit (Lambda, API gateway) and at rest (S3) using AWS key management service could be implemented.
With higher demand for API requests, we should enable detailed logging and monitoring of API usage in CloudWatch and CloudTrail to detect anomalies and respond to suspicious events in time.

Cost Analysis

Note:
- As the notebook instance is needed only for retraining the model. Suppose we update our model on updated training data every 3 days, resulting running 10 days * 24 hours.
- Endpoint running 24/7

AWS

Services	Price	Monthly Costs	1-year-costs Monthly	3-year costs Monthly
S3	$0.023 per GB 10 GB storage Inbound data transfer: $0.005 plus outbound data transfer $0.0004 to SageMaker	$0.34 USD	$0.34 USD	$0.34 USD
SageMaker	US East (N.Virginia) ml.g4dn.xlarge $0.736/hour for both notebook instance and real-time inference	Notebook: $176.64 Endpoint: $529.92 Total: $706.56	Saving plans rate for 1-year: $0.5334 Total: $512.064	Saving plans rate for 3-year: $0.3822 Total: $366.912

Lambda	US East (N.Virginia) x86 $0.2 per million requests for first Billion GB-seconds $0.0000000021 per 128 MB allocated	128MB for memory allocated, 512 MB for ephemeral storage, monthly total: $3.75 USD	Suppose our first year usage would not exceed 1 billion $3.75 USD	Suppose our 3-year usage would not exceed 1 billion: $3.75 USD
API Gateway	HTTP API: Price for First 300 million requests is $1/million $0.9/million for 300+ million Increment data size is 512KB	Estimated usage: 1 million / month Total: $1	$1	$1
Total		$711.65	$517.70	$372

Google cloud platform:

Services	Price	Monthly Costs	1-year-costs Monthly	3-year costs Monthly
Cloud Storage	$ 0.023/GB	$0.43	$0.43	$0.43
AI Platform (Compute + GPU)	n1-standard-4 with a Tesla T4 GPU Approximately $0.123 per hour for training and $0.4025 per hour for the Tesla T4 GPU. Prediction n1-standard east4: $0.2461/hour T4 GPU: $0.4025/hour	$(0.123 + 0.4025)/hour×24 hours/day × 10 days =126.12USD Endpoint: $(0.2461+0.4025)*24*30 = 466.992 Total: $593.112	10% discount: $593.112 * 0.9 = $ 533.801	25% discount: $593.112 * 0.75 = $444.83
Cloud Function	2nd generation, us-east4, 128 MB memory allocated with average data transfer per request: 0.09 MB, 5000 ms per request.	Serverless: $27.14	$27.14	$27.14

API Gateway Http API	$0 for the first 2M usage $3 per million for 2M to 1B usage Data Transfer: $0.105-$0.190/GB depending on destination	$3 + $0.1 Total: $3.1	After the first 300 million calls, the price would be $0.9, in our case it stays the same. $3.1	After the first 300 million calls, the price would be $0.9, in our case it stays the same. $3.1
Total		$623.78	$564.471	$475.5

Google cloud platform is cheaper for usage without commitment whereas AWS is cheaper for 1-year savings and 3-year savings.

Further extension:

- Suppose our data size grows from the initial 10 GB to 60 GB due to growing data volume by user uploading images.
- Suppose our project expands our services from cyrillic OCR to more languages, deploying multiple SageMaker endpoints to accommodate different usage scenarios.
- Our usage increases to 3 million per month.
- Model retraining and fine-tuning takes 12 hours per job and 5 jobs per model.

	AWS		GCP	
Services	Description	Monthly cost	Description	Monthly cost
S3	60 GB data storage 3 million put requests due to user upload Get requests will be mainly from SageMaker model retrain, within 100 requests.	60 GB x 0.023 USD = 1.38 USD Total cost = 1.38 USD 3,000,000 PUT requests x 0.000005 USD per request = 15.00 USD 15 GET requests in a month x 0.0000004 USD per request = 0.00 USD 1.38 USD + 15.00 USD = 16.38 USD S3 Standard cost : $16.38	Location type: Multi-region Location: United States (us) Storage class: Standard Storage Source region: North America Destination region: North America	Data Transfer within Google Cloud 10 GB $0.00 2 Million "Class A" operations: $20.00 0 ”Class B" operations $0.00 Default replication$1.12 Total amount of storage 60 GB $1.45

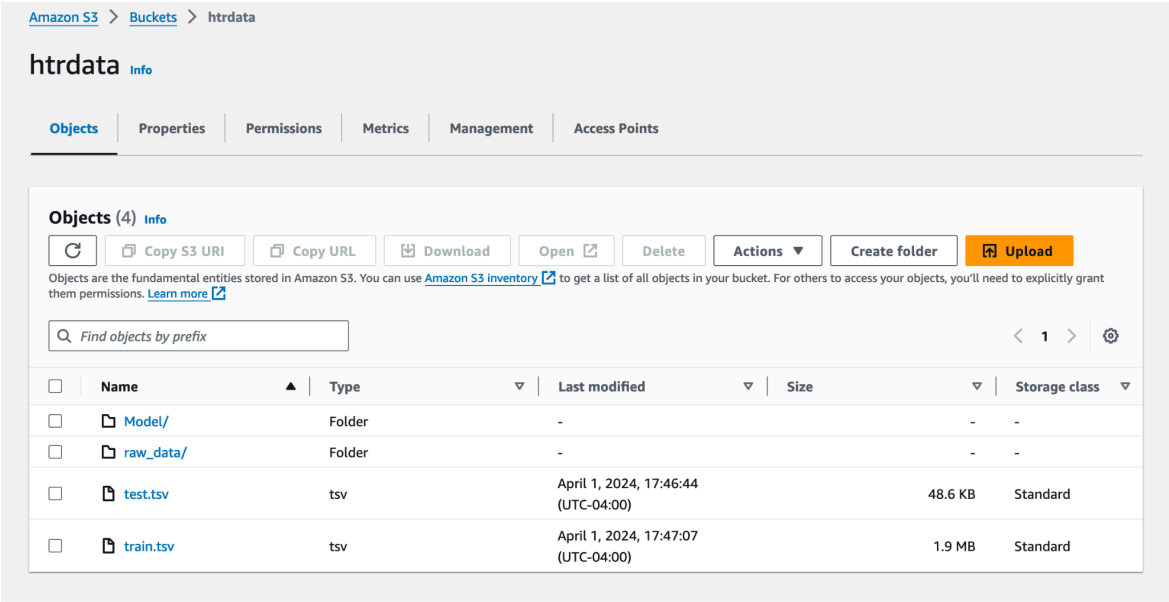
SageMaker	US East (N.Virginia) Training jobs: 5 jobs * 12 hour on 2 instances for each model 3 endpoints with instance type: ml.g4dn.2xlarge	180.00 hours per month x 0.94 USD per hour instance cost = 169.20 USD (monthly On-Demand cost) 2,160.00 hours per month x 0.94 USD per hour instance cost = 2,030.40 USD (monthly On-Demand cost) Total: $2,234.72	N1-standard-8 instance type US East (South Carolina) Training jobs: 5 jobs * 12 hour on 2 instances for each model with 1 T4 GPU on each instance Prediction running 24/7	Total using GCP calculator: $298.87 + $1,814.40 Total: $2,113.27
Lambda / Cloud Functions	With free-tier; 256 MB memory allocation, 3 million requests	3,350,000 GB-s x 0.0000166667 USD = 55.83 USD Total tier cost = 55.8334 USD 3,000,000 requests - 1000000 free tier requests = 2,000,000 monthly billable requests Total: $56.47	Tier 2: $0.000000648 / 100ms	Memory: $153.66 3 million Requests per month: $0.40 Total: $154.06
API Gateway	3 million usage per month	$1/million * 3 = $3	$3/million calls for 2M to 1B usage Data Transfer: $0.105-$0.190/GB depending on destination	$3* 3 + $0.2 = $9.2
Total		$2310.57		$2277.98

The monthly costs in AWS and GCP for the extended version is roughly the same where GCP is slightly cheaper.

Implementation Quality & Demo

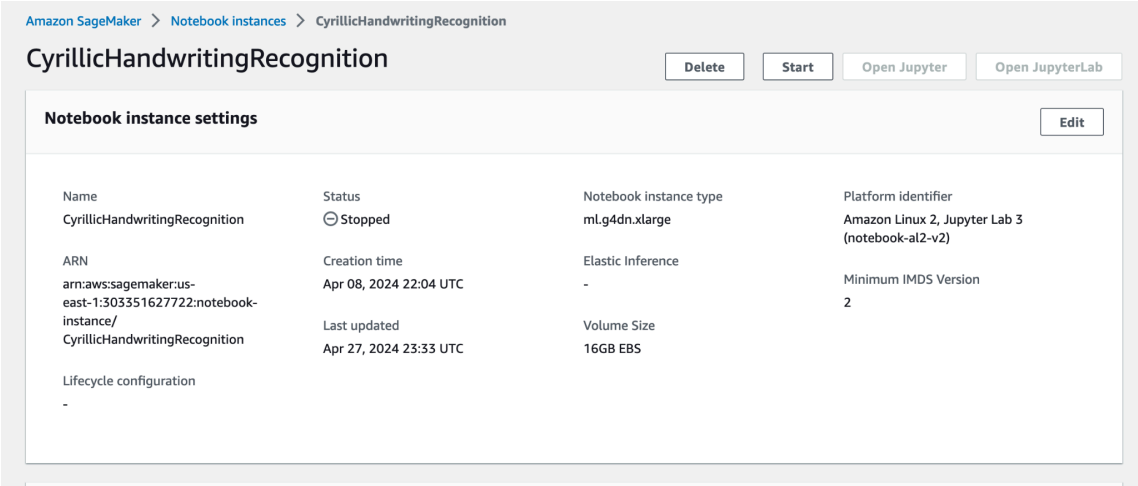
S3

S3 bucket structure:



SageMaker

Sagemaker notebook for training, testing, and model deployment



Notebook instance general structure:

- Training.ipynb

Data is retrieved from S3 bucket through boto3 library and unzipped to the train and test folders.

In [1]:
```
import sagemaker

sagemaker_session = sagemaker.Session()
role = sagemaker.get_execution_role()
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
```

In [2]:
```
import boto3, os,
import zipfile

bucket = 'htrdata'
train_file = 'raw_data/train.zip'
test_file = 'raw_data/test.zip'
local_train = '/home/ec2-user/SageMaker/train/'
local_test = '/home/ec2-user/SageMaker/test/'
checkpoint = '/home/ec2-user/SageMaker/checkpoint/'

s3 = boto3.client('s3')
if not os.path.exists(local_train):
    os.makedirs(local_train)
if not os.path.exists(local_test):
    os.makedirs(local_test)
if not os.path.exists(checkpoint):
    os.makedirs(checkpoint)

#download unzip train & test images
file_train = 'train.zip'
file_test ='test.zip'
s3.download_file(bucket, f'{train_file}', f'{local_train}{file_train}')
s3.download_file(bucket, f'{test_file}', f'{local_test}{file_test}')
with zipfile.ZipFile('/home/ec2-user/SageMaker/test/test.zip', 'r') as zip_ref:
        zip_ref.extractall('/home/ec2-user/SageMaker/test/')
os.remove('/home/ec2-user/SageMaker/test/test.zip')
with zipfile.ZipFile('/home/ec2-user/SageMaker/train/train.zip', 'r') as zip_ref:
        zip_ref.extractall('/home/ec2-user/SageMaker/train/')
os.remove('/home/ec2-user/SageMaker/train/train.zip')

#download labels
local_label = '/home/ec2-user/SageMaker/label/'
os.makedirs(local_label)
s3.download_file(bucket, 'train.tsv', '/home/ec2-user/SageMaker/label/train.tsv')
s3.download_file(bucket, 'test.tsv', '/home/ec2-user/SageMaker/label/test.tsv')
```

The raw image is processed through resizing and normalization:

The training process is as follows:

```
In [21]: model = TransformerModel('resnet50', len(hp.cyrillic), hidden=hp.hidden, enc_layers=hp.enc_layers, dec_layers=hp.dec
                           nhead=hp.nhead, dropout=hp.dropout).to(device)
         optimizer = optim.SGD(model.parameters(), lr=hp.lr)
         criterion = nn.CrossEntropyLoss(ignore_index=char2idx['PAD'])
         scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min')

         if WANDB_LOG:
             wandb.init(project="OCR-transformer", config={
                 "learning_rate":hp.lr,
                 "dropout": hp.dropout,
                 "batch_size": hp.batch_size,
                 "architecture": "RESNET50 + TRANSFORMER",
                 "dataset": "64x512 train 29k",
                 "classes": "92",
             })
             config = wandb.config


         train_all(model, optimizer, criterion, scheduler,train_loader,val_loader, epoch_limit=80)
```

```
/home/ec2-user/anaconda3/envs/pytorch_p310/lib/python3.10/site-packages/torch/nn/functional.py:5076: UserWarning: S
upport for mismatched key_padding_mask and attn_mask is deprecated. Use same type for both instead.
  warnings.warn(

[ 500 / 4066 ]
[ 1000 / 4066 ]
[ 1500 / 4066 ]
[ 2000 / 4066 ]
[ 2500 / 4066 ]
[ 3000 / 4066 ]
[ 3500 / 4066 ]
[ 4000 / 4066 ]
train loss : 2.491482118228006

-----------valid------------

100%|██████████| 7228/7228 [01:19<00:00, 90.44it/s]

validation loss : 2.2939682286552254
```

We were not able to train all epochs, so we retrieved fully trained model weights from Kaggle. And we evaluate the model accuracy with the final 0.899 accuracy on test data in terms of character. The word accuracy (for every character) is 0.538.

```
In [49]: word_accur, char_accur = test(model,PATH_TEST_DIR,PATH_TEST_LABELS,char2idx,idx2char,case=False,punct=False)
         print(word_accur, char_accur)
```

```
('test1317.png', {'predicted_label': 'Более', 'p_values': 1})
('test440.png', {'predicted_label': 'определии', 'p_values': 1})
('test8.png', {'predicted_label': 'воведем', 'p_values': 1})
('test1305.png', {'predicted_label': 'Лебеденно Н.', 'p_values': 1})
('test802.png', {'predicted_label': 'Полыпко', 'p_values': 1})
('test699.png', {'predicted_label': 'выписывала', 'p_values': 1})
('test495.png', {'predicted_label': 'правый', 'p_values': 1})
('test212.png', {'predicted_label': 'заявление', 'p_values': 1})
('test1477.png', {'predicted_label': 'Стинендию получаю', 'p_values': 1})
('test186.png', {'predicted_label': 'предпологим.', 'p_values': 1})
('test426.png', {'predicted_label': 'продажа', 'p_values': 1})
('test362.png', {'predicted_label': 'шисст', 'p_values': 1})
('test1563.png', {'predicted_label': '15.09.03', 'p_values': 1})
('test935.png', {'predicted_label': 'пастерту', 'p_values': 1})
('test866.png', {'predicted_label': 'их успех', 'p_values': 1})
('testt1211.png', {'predicted_label': 'Байона', 'p_values': 1})
('test1238.png', {'predicted_label': 'с детьми', 'p_values': 1})
0.898772664120926 0.5382124352331606
```

```
libpng warning: cHRM: invalid chromaticities
```

- model & model.tar.gz
  This directory stores the model artifacts, structured as:
  |– – model.pt
  |– – code
  | – – inference.py
  | – – requirements.txt

- inference.py

  In this script, we defined functions required for PyTorch model deployment in SageMaker, including 4 main functions. Code snippet in the appendix section 1- a.

- Packaging&SampleEndpointUsage.ipynb

The endpoint deployment process is as follows:

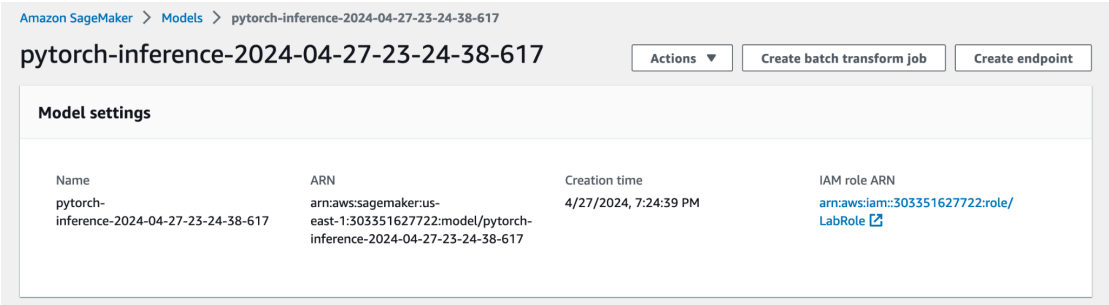
```
In [51]: !tar -czvf model.tar.gz model

         model/
         model/code/
         model/code/inference.py
         model/code/requirements.txt
         model/code/.ipynb_checkpoints/
         model/.ipynb_checkpoints/
         model/model.pt
```

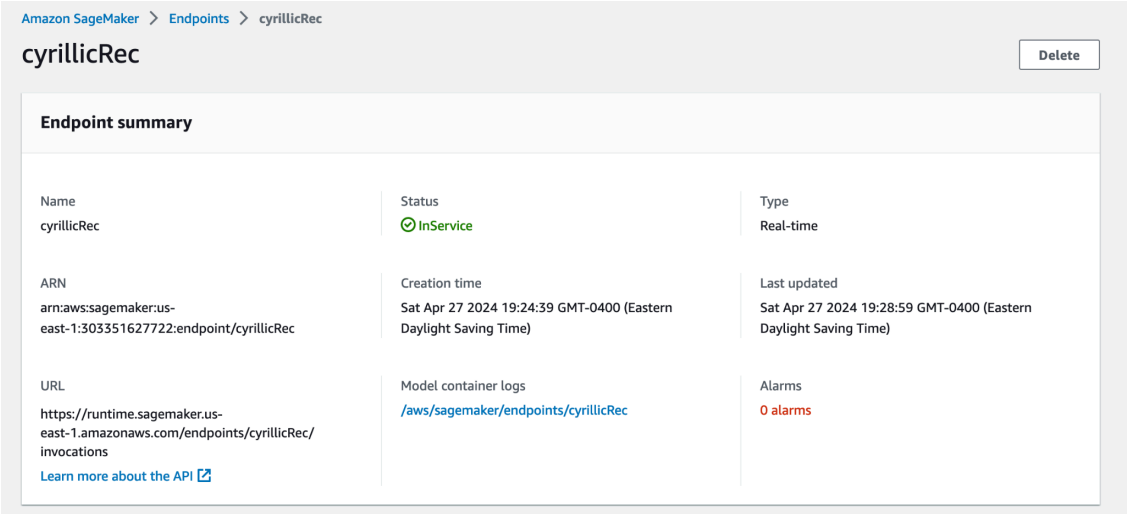
```
In [47]: import boto3

         # Create an S3 client
         s3 = boto3.client('s3')

         # Specify the file and bucket name
         filename = 'model.tar.gz'
         bucket_name = 'htrdata'

         # Upload the file
         s3.upload_file(filename, bucket_name, 'Model/model.tar.gz')
```

```
In [5]: #model deployment stage
        import sagemaker
        from sagemaker.pytorch import PyTorchModel

        sagemaker_session = sagemaker.Session()

        #IAM role
        role = 'arn:aws:iam::303351627722:role/LabRole'
        model_data = 's3://htrdata/Model/model.tar.gz'
        pytorch_model = PyTorchModel(model_data=model_data,
                                     role=role,
                                     entry_point='inference.py',
                                     framework_version='2.1.0',  #PyTorch version
                                     py_version='py310',
                                     source_dir='model/code')  #directory for inference.py
```

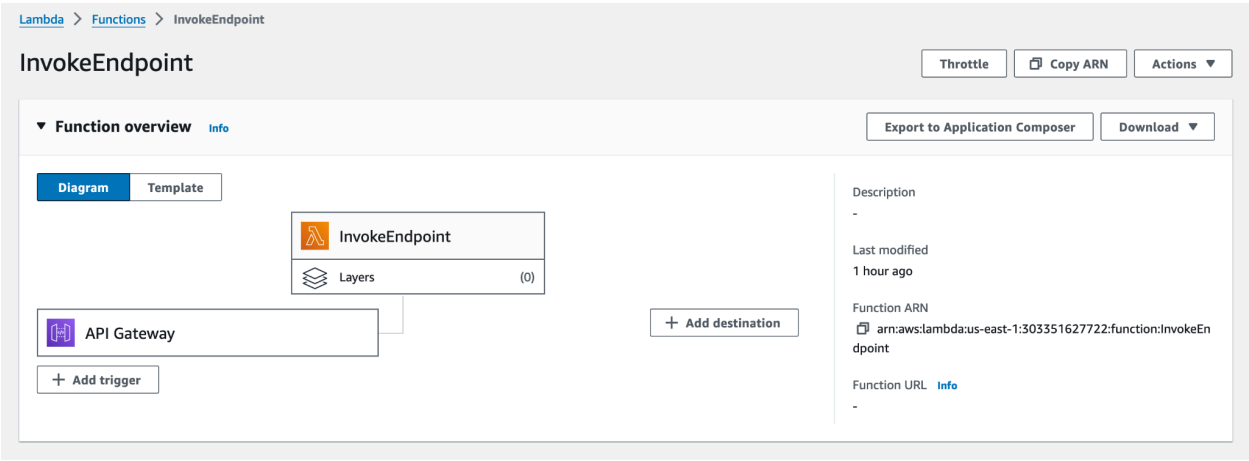
```
In [6]: predictor = pytorch_model.deploy(endpoint_name = "cyrillicRec",
                                         initial_instance_count=1,
                                         instance_type='ml.g4dn.xlarge')

        --------!
```

The model instance and endpoint:




Amazon SageMaker > Endpoints > cyrillicRec
cyrillicRec
Delete
Endpoint summary
Name
cyrillicRec
Status
InService
Type
Real-time
ARN
arn:aws:sagemaker:us-east-1:303351627722:endpoint/cyrillicRec
Creation time
Sat Apr 27 2024 19:24:39 GMT-0400 (Eastern Daylight Saving Time)
Last updated
Sat Apr 27 2024 19:28:59 GMT-0400 (Eastern Daylight Saving Time)
URL
https://runtime.sagemaker.us-east-1.amazonaws.com/endpoints/cyrillicRec/invocations
Learn more about the API
Model container logs
/aws/sagemaker/endpoints/cyrillicRec
Alarms
0 alarms

Lambda


Lambda > Functions > InvokeEndpoint
InvokeEndpoint
Throttle
Copy ARN
Actions
Function overview
Info
Export to Application Composer
Download
Diagram
Template
InvokeEndpoint
Layers
(0)
API Gateway
Add trigger
Add destination
Description
-
Last modified
1 hour ago
Function ARN
arn:aws:lambda:us-east-1:303351627722:function:InvokeEndpoint
Function URL
Info
-

lambda_function × Environment Var ×

```python
import boto3
import json, sys
import base64

def lambda_handler(event, context):
    #check whether image is empty
    decode = event['isBase64Encoded']
    if not decode:
        return {
            'statusCode': 200,
            'headers': {
                'Content-Type': 'application/json'
            },
            'body': json.dumps("image not uploaded :<")
        }
    img = base64.b64decode(event['body'])

    print("retrieving endpoint")
    runtime = boto3.client('sagemaker-runtime')
    endpoint_name = 'cyrillicRec'
    response = runtime.invoke_endpoint(
        EndpointName=endpoint_name,
        ContentType='image/png',
        Body=img,
        Accept="application/json"
    )

    response_body = response['Body'].read().decode('utf-8')
    result = json.loads(response_body)
    prediction = result['result']
    json_output = json.dumps(prediction, ensure_ascii=False)
    json_output = json_output + '\n'
    # return prediction result
    return {
        'statusCode': 200,
        'headers': {
            'Content-Type': 'application/json'
        },
        'body': json_output.encode('utf8')
    }
```

API Gateway

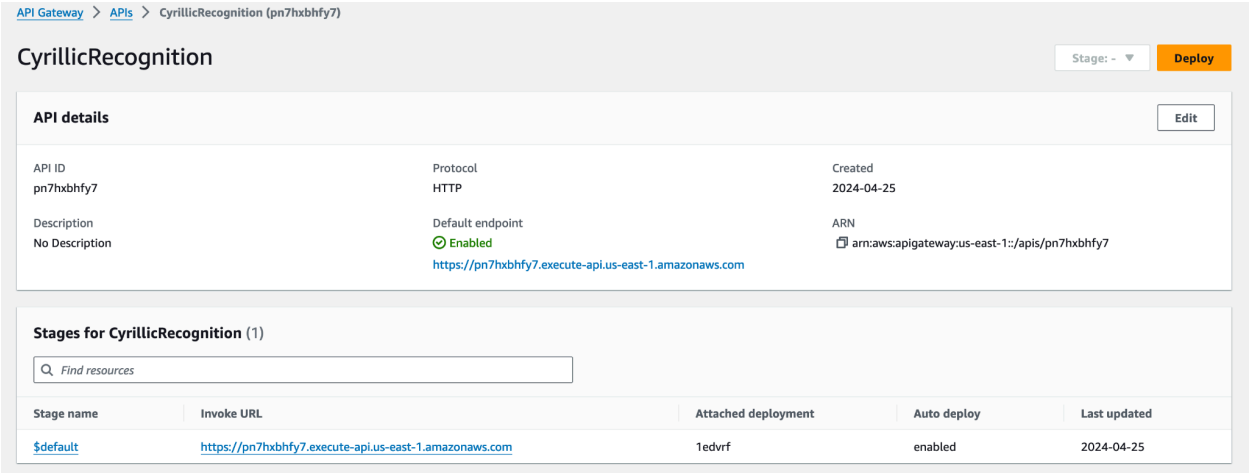
Triggers (1) Info

Fix errors | Edit | Delete | **Add trigger**

Find triggers

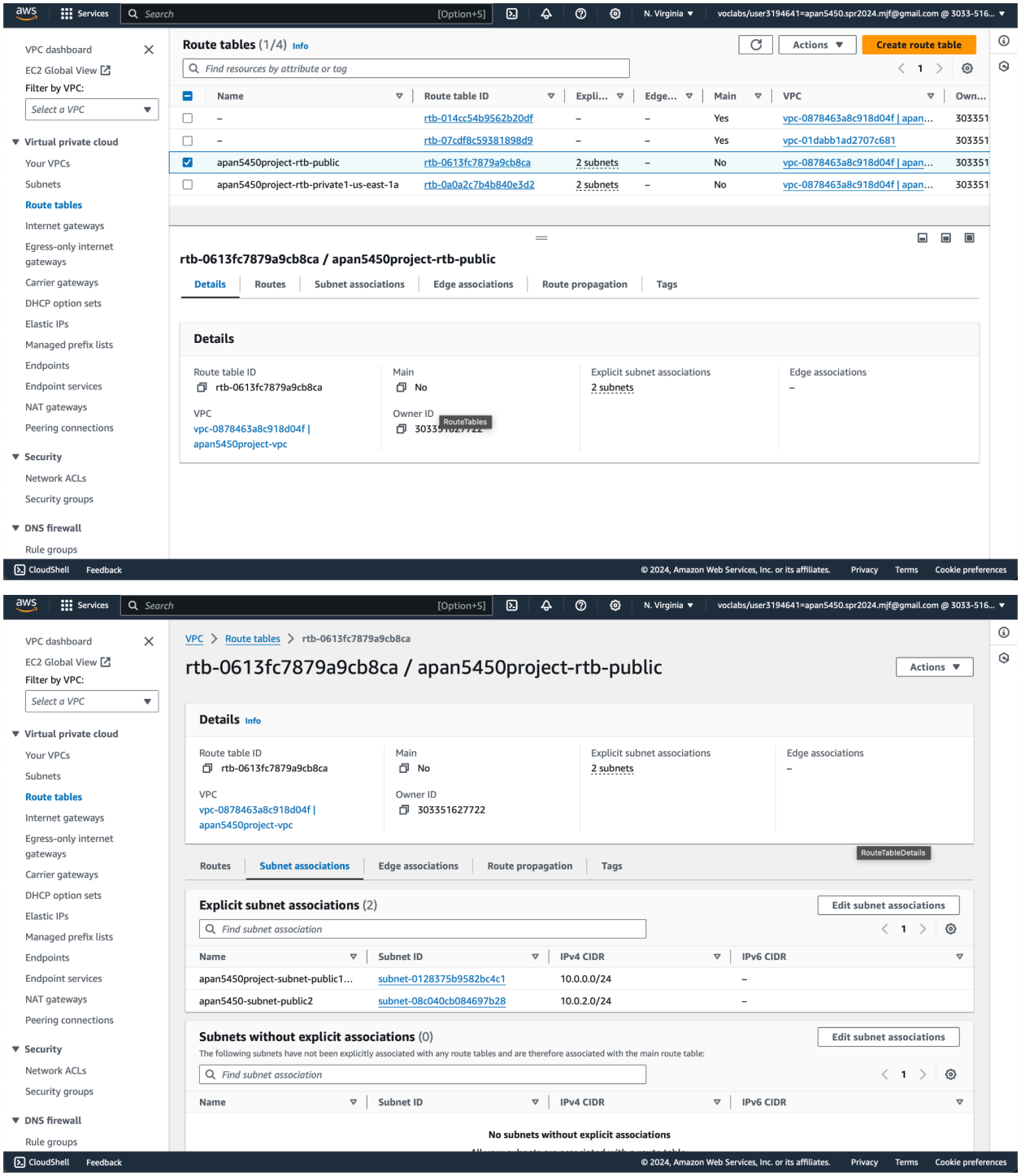
1

Trigger
API Gateway: CyrillicRecognition arn:aws:execute-api:us-east-1:303351627722:pn7hxbhfy7/*/*/getPrediction API endpoint: https://pn7hxbhfy7.execute-api.us-east-1.amazonaws.com/getPrediction ▼ Details API type: **HTTP** Authorization: **NONE** CORS: **No** Detailed metrics enabled: **No** isComplexStatement: **No** Method: **ANY** Resource path: **/getPrediction** Service principal: **apigateway.amazonaws.com** Stage: **$default** Statement ID: **0c3f5f27-ac22-5cf3-8724-9e411b83817d**

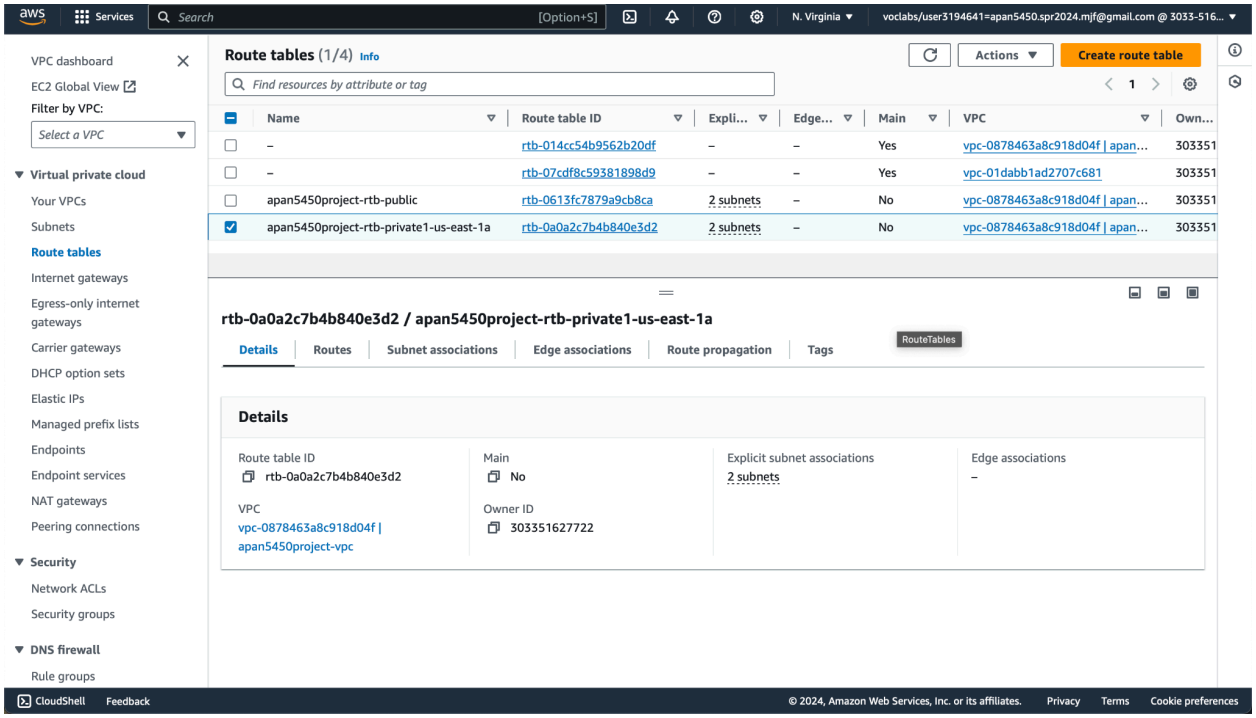


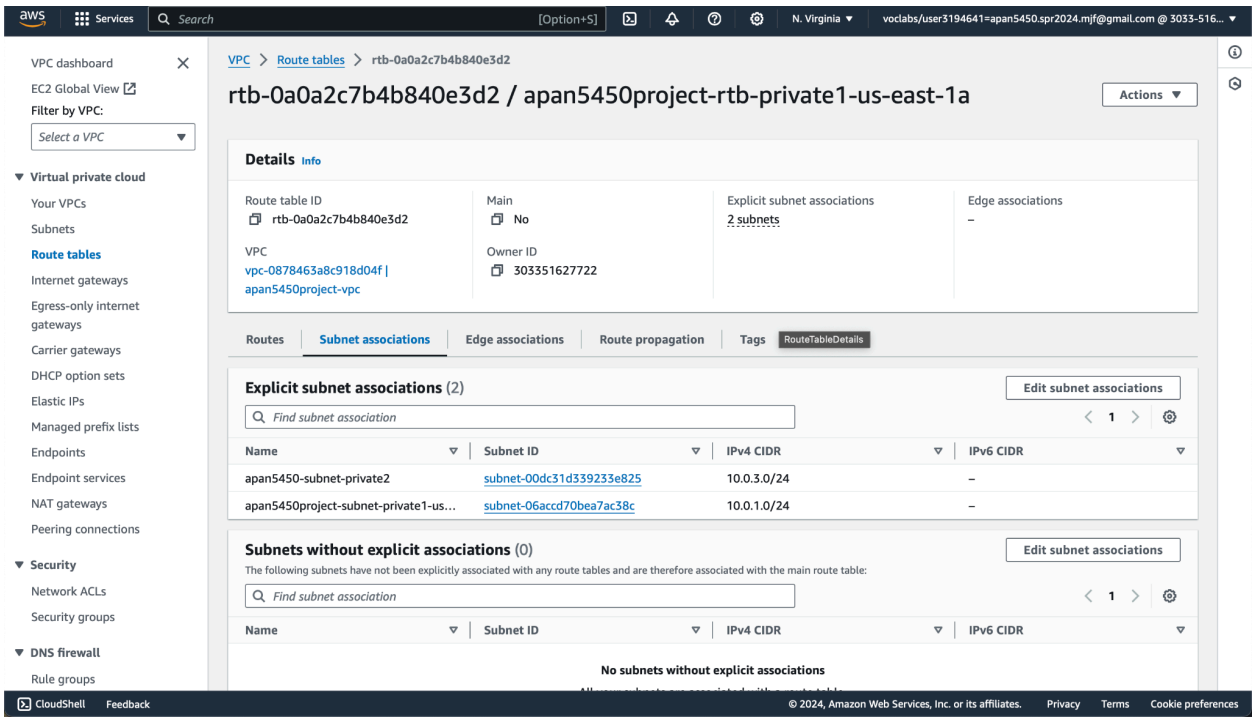
Security Configuration

VPC Route Tables(Public)

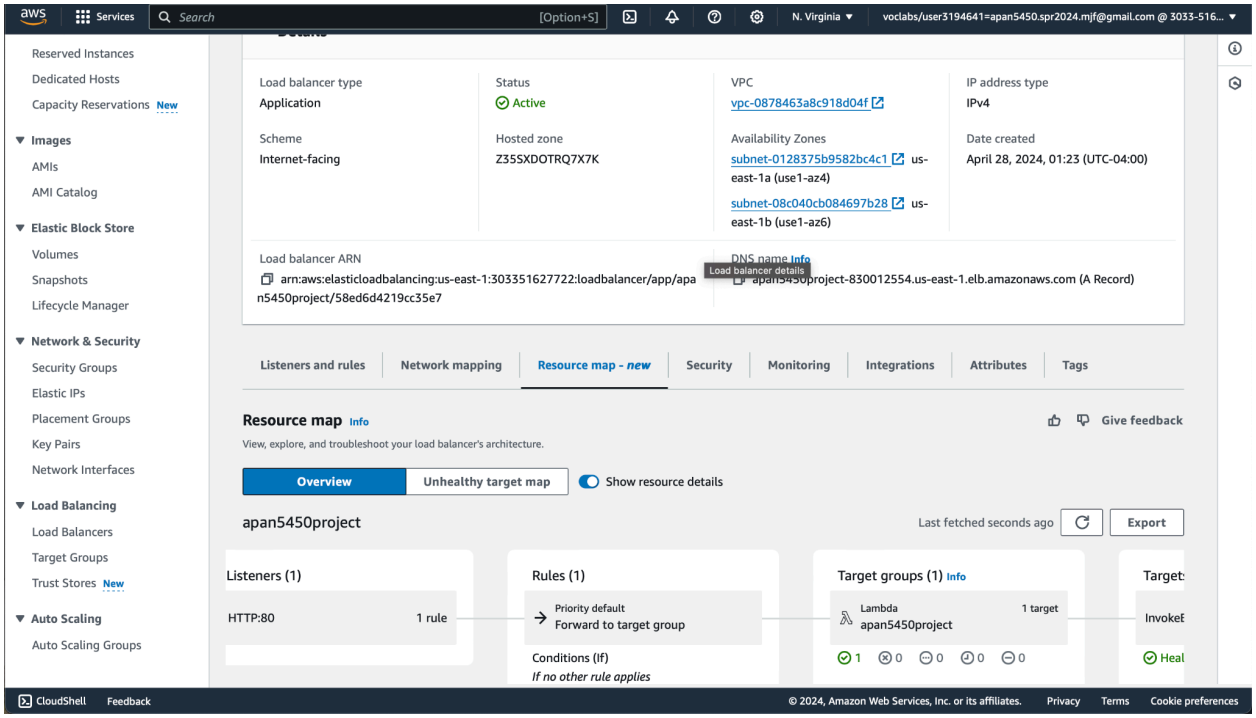


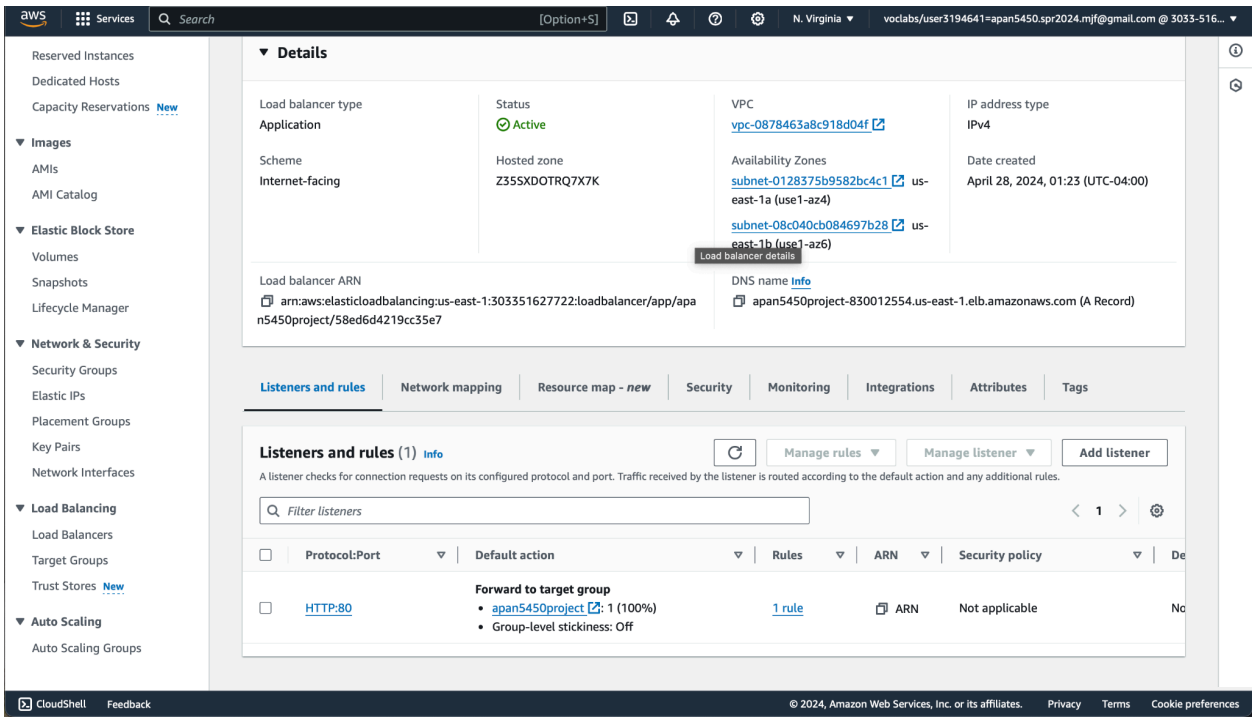
VPC Route Tables(Private)





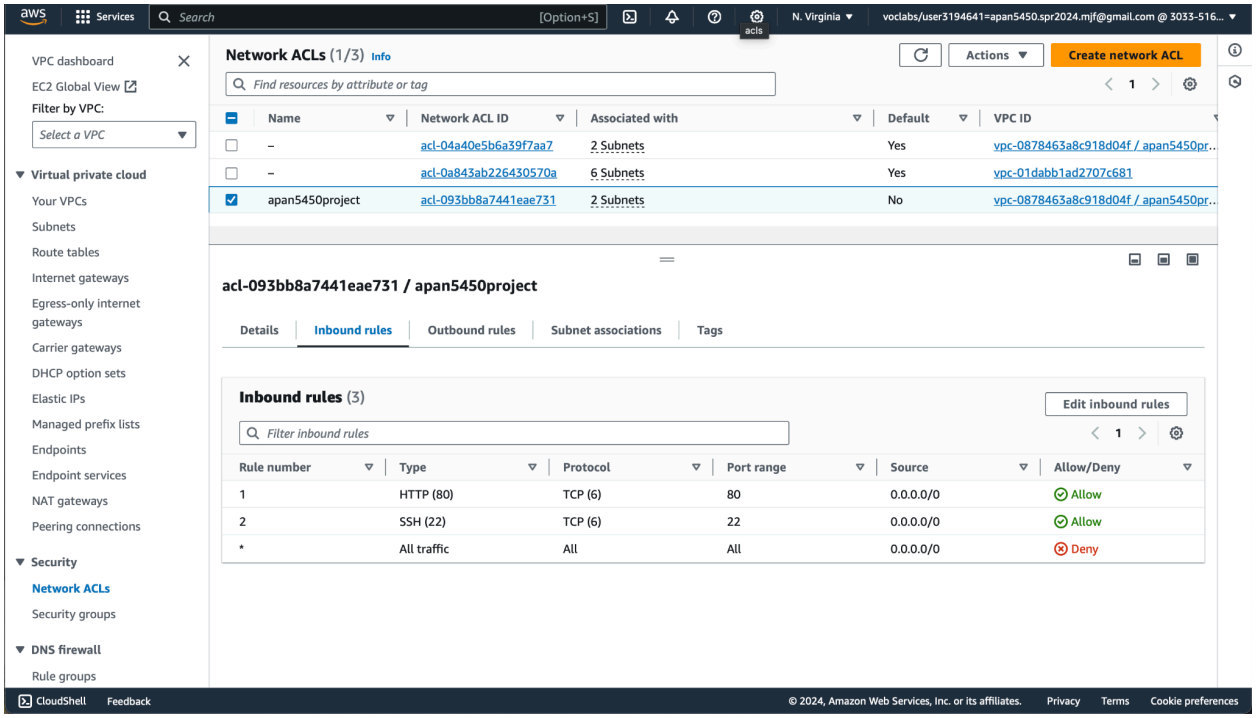
Load Balancer

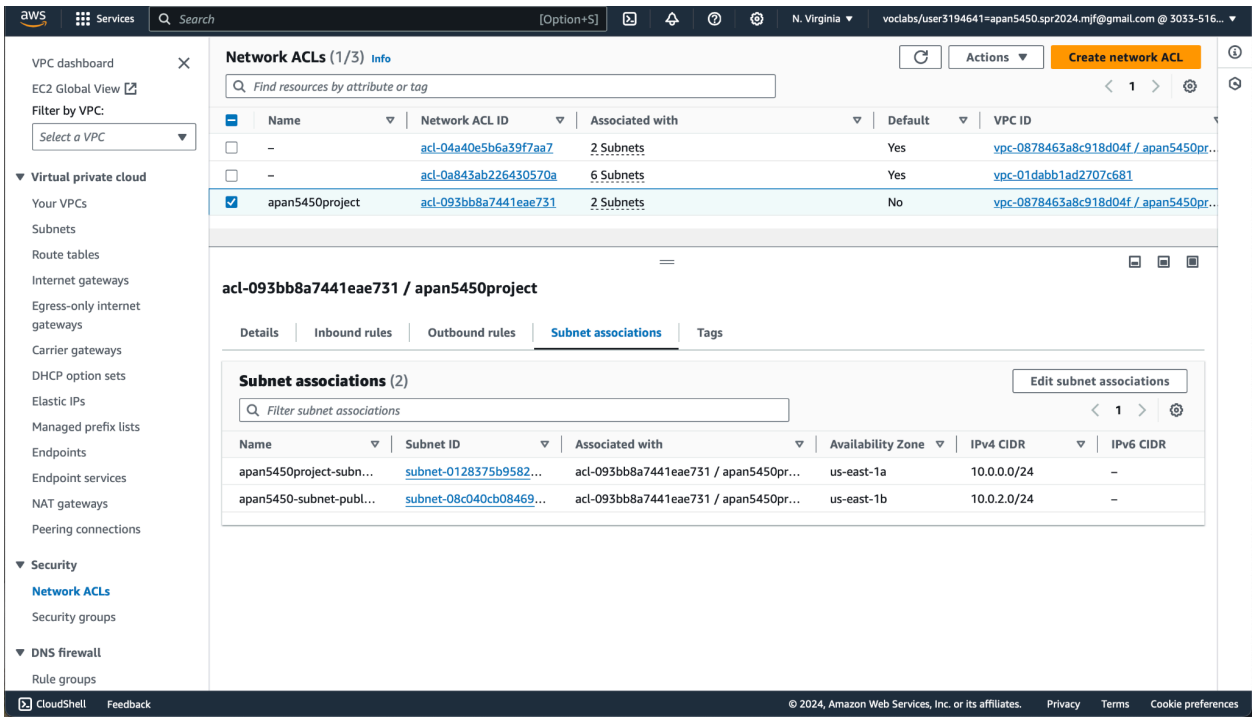




***the architecture is designed for the targeted group 'InvokeEndpoint' on Lambda function**

Network ACL





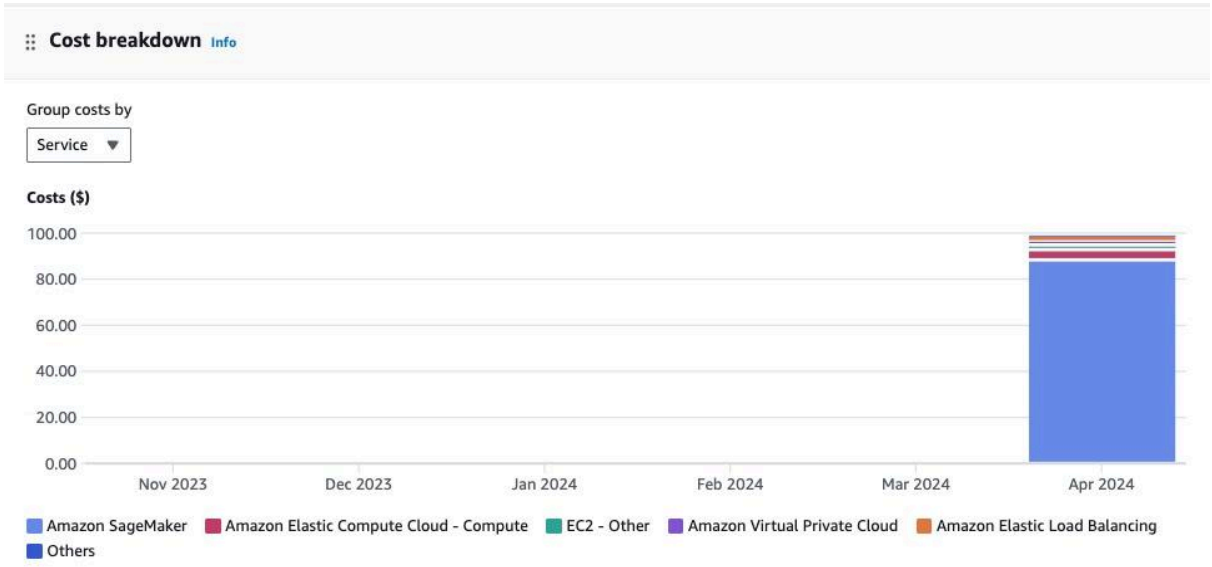
***the network acl controls traffic into and out of the public subnets with inbound rules same as VPC**

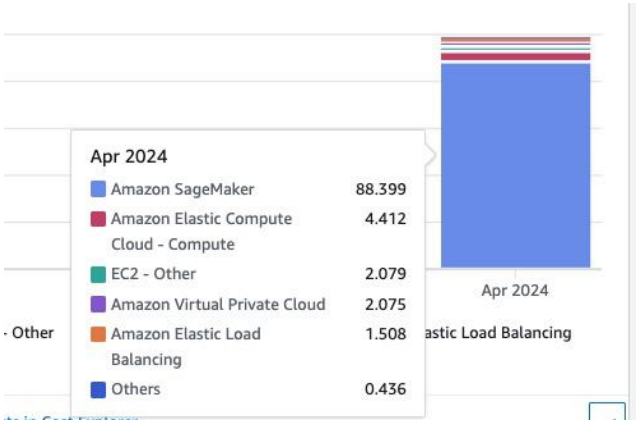
IAM Roles and Policies

While I was trying to create IAM roles for API Gateway to ensure secure access to AWS resources, the user(me) does not have necessary permissions to 'IAM:GETRoles'

“User: arn:aws:sts::303351627722:assumed-role/voclabs/user3194641=apan5450.spr2024.mjf@gmail.com
Service: iam
Action: GetRole
On resource(s): role
Context: voclabs with an explicit deny in an identity-based policy”

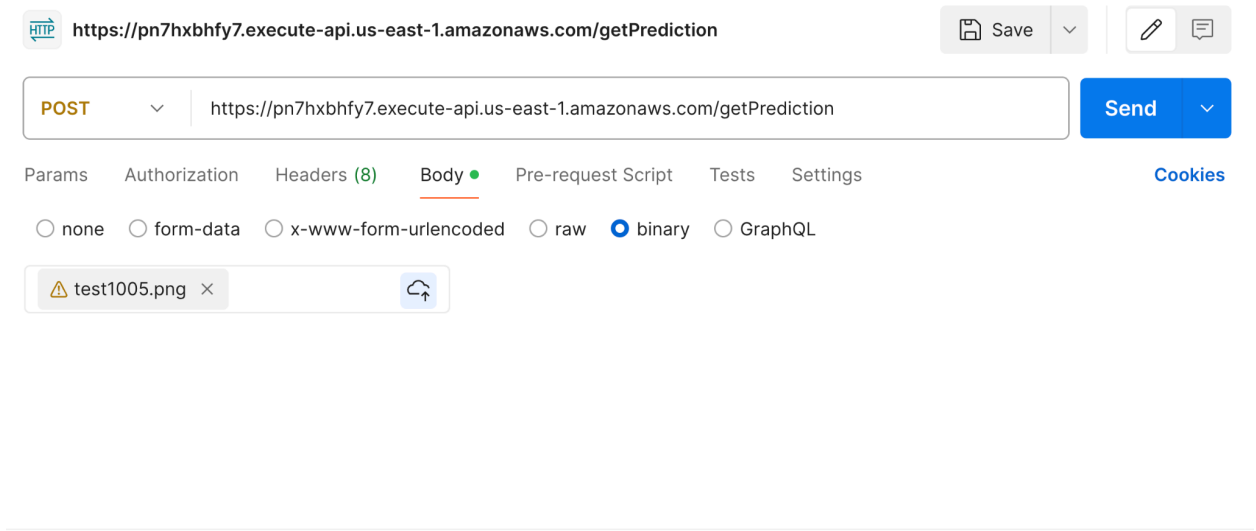
Cost Breakdown

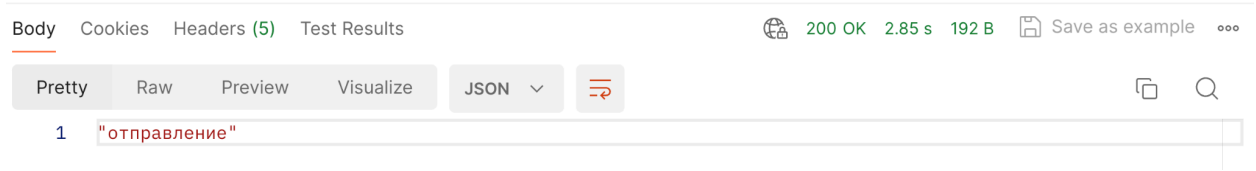




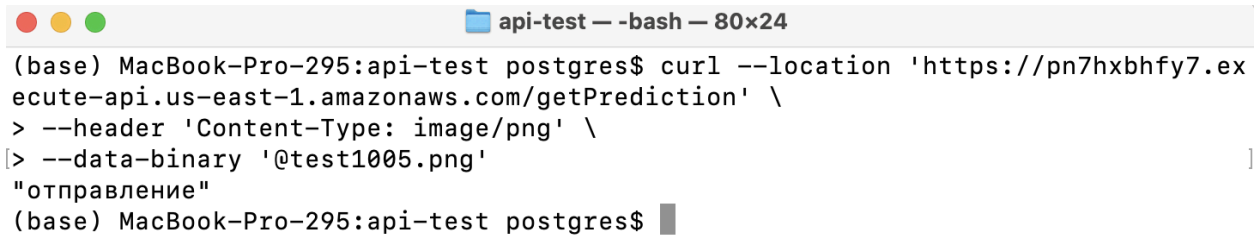
Demo

Sample Usage in Postman



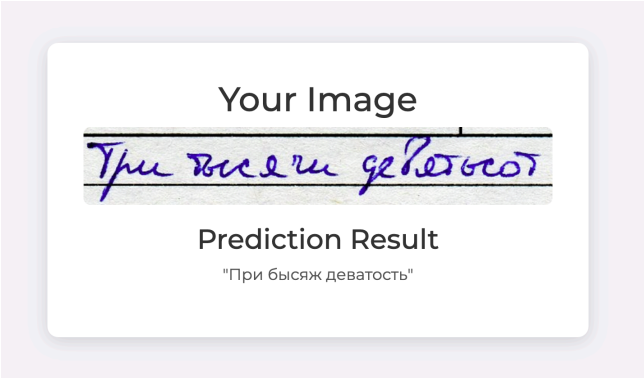


Sample usage in shell command



Sample usage of API in Python with local flask interface:


Handwritten Cyrillic Recognition
Select image to upload:
选择文件
test1073.png
UPLOAD IMAGE


Your Image
Prediction Result
"При бысяж деватость"

Appendix

1-a

```python
def model_fn(model_dir):
    """
    Load the PyTorch model from the `model_dir` directory.
    """
    print("Loading model.")
    model = TransformerModel('resnet50', len(hp.cyrillic), hidden=hp.hidden, enc_layers=hp.enc_layers, dec_layers=hp.dec_layers,
                         nhead=hp.nhead, dropout=hp.dropout, pretrained=False)
    model_path = os.path.join(model_dir, "model/model.pt")
    state = torch.load(model_path)['model']
    model.load_state_dict(state, strict=False)
    print("model loaded")
    model.eval()
    return model

def input_fn(request_body, request_content_type):
    """
    prepare input.
    """
    if request_content_type == 'image/png':
        image = np.frombuffer(request_body, dtype=np.uint8)
        image = cv2.imdecode(image, cv2.IMREAD_COLOR)  # Convert bytes to an image
        image = process_image(image).astype('uint8')
        image = image / image.max()
        image = np.transpose(image, (2, 0, 1))
        tensor = torch.FloatTensor(image).unsqueeze(0)
    else:
        raise ValueError(f"wrong content type: {request_content_type}")

    return tensor


def predict_fn(input_data, model):
    """
    make a prediction using the model and input data.
    """

    with torch.no_grad():
        hp = Hparams()
        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        char2idx = {char: idx for idx, char in enumerate(hp.cyrillic)}
        idx2char = {idx: char for idx, char in enumerate(hp.cyrillic)}
        out_indexes = [char2idx['SOS'], ]
        for i in range(100):
            trg_tensor = torch.LongTensor(out_indexes).unsqueeze(1)
            output = model(input_data,trg_tensor)
            out_token = output.argmax(2)[-1].item()
            out_indexes.append(out_token)
            if out_token == char2idx['EOS']:
                break
        output = labels_to_text(out_indexes[1:], idx2char)

    return output



def output_fn(prediction_output, accept):
    """
    Serialize the string prediction output to the specified format.
    """
    if accept == 'application/json':
        output = json.dumps({'result': prediction_output}, ensure_ascii=False)
        return output
    else:
        raise ValueError(f"wrong 'accept' type: {accept}")
```