香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

Cloud-based Image Classification System with AWS Services

COMP4651
Big Data Systems and Cloud Computing

Date
30-11-2024

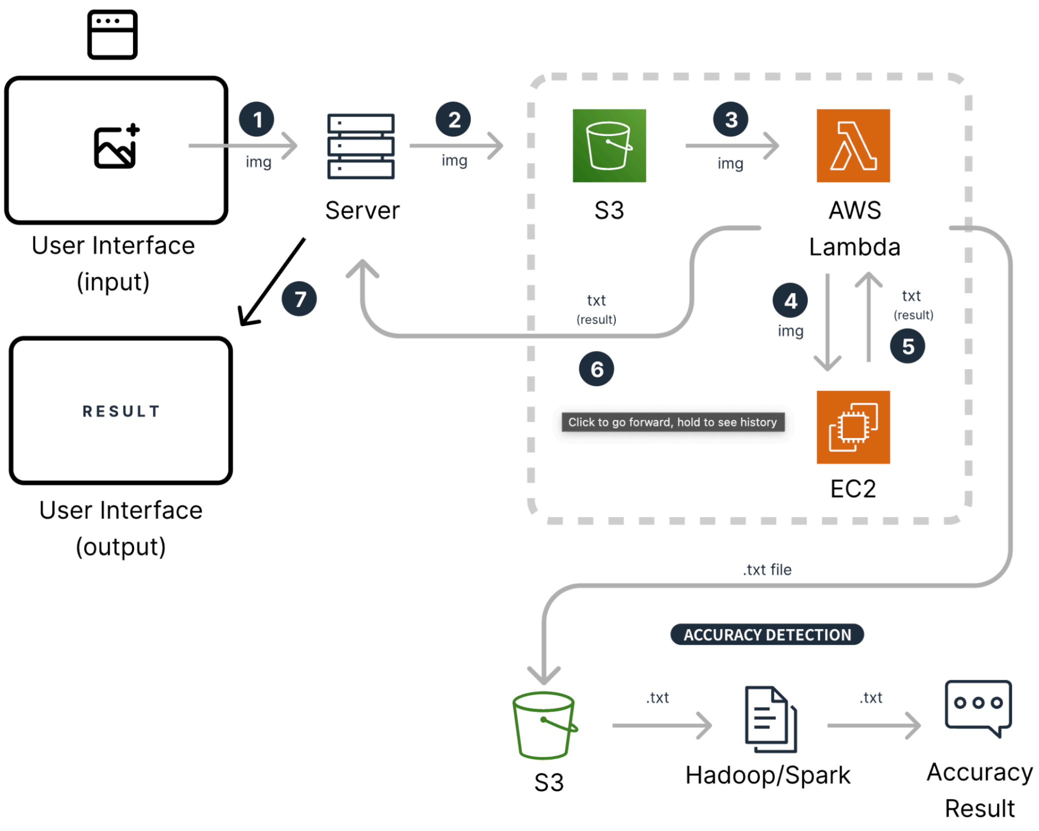
Authors
Group 9

Gonzalo Carretero	21158252	gch@connect.ust.hk
Siyoon Park	20739700	sparkay@connect.ust.hk
Yeung Kong Sunny Lam	20857162	ykslam@connect.ust.hk
Gunwoo Park	20635825	gparkab@connect.ust.hk

I. Introduction

In this project, we designed and implemented an image classification system using AWS Services. Our system processes images uploaded by users and returns a classification result. The accuracy of the classification model is also evaluated. By combining **serverless functionality**, **data analysis during model training**, and **distributed processing tasks**, we aim to build a model with high accuracy and operation for the analysis of model accuracy in actual usage to public users.

II. Design



Part 1: Classification

1. User selects an image from their computer and uploads it to the web app.
2. The backend of the web app receives the image and sends it to an S3 bucket.
3. A trigger event is triggered in the S3 bucket, allowing an AWS Lambda function to collect the image from the bucket.
4. The Lambda function sends the image to our classification model which is waiting in an EC2 instance.
5. The model computes the result and sends it back to our Lambda function.
6. The Lambda function sends the results to the web app backend to display it in the UI.

Part 2: Accuracy Evaluation

1. We store in a S3 bucket the predicted and actual labels of the image classes.
2. A Hadoop job running in AWS EMR processes the data from the S3 bucket to compute accuracy metrics.
3. The results are inserted back into the S3 bucket for storage.

III. Implementation

Task 1: Create an intuitive user interface and a backend server to receive input and display results. Deploy both frontend and backend in an EC2 instance.

Web App (Frontend)

- The main code is located in the `App.tsx` file, which is written using TypeScript and the React framework. The UI execution is managed through Vite.
- The user can upload a file from their local device by clicking the 'Upload File' button. The web page will then display the selected image for confirmation. Upon clicking the 'Classify' button, a POST request is made to the backend to transmit the image for processing.
- The UI will receive the classification prediction from the backend and display the result on the screen.
- Important note: When running the UI with the *npm run dev* command, the IP addresses of the links suggested should be replaced with the public IP address of the corresponding EC2 instance where the code is running, in order to access the web app from a local device.

Web App (Backend)

- The code can be found inside `server.js`. We used Node.js with Express. It has three APIs:
  - app.post('/upload'): When the backend receives an image from the UI, it will send it to an S3 bucket, using aws-sdk.
  - app.post('/display-classification'): The backend will store the classification result it receives from AWS Lambda.
  - app.get('/get-prediction/:uploadId'): The UI will periodically check if the image being processed has already been classified. If so, the backend will send the results to the UI through this GET request.
- Important note: a .env file should be made to save the AWS access keys in order to be able to put files in the S3 bucket from the backend:

AWS_ACCESS_KEY_ID=...

AWS_SECRET_ACCESS_KEY=...

AWS_ACCESS_TOKEN=... // Only needed for temporal credentials, like the ones generated during lab sessions. Not needed if using a personal AWS account.

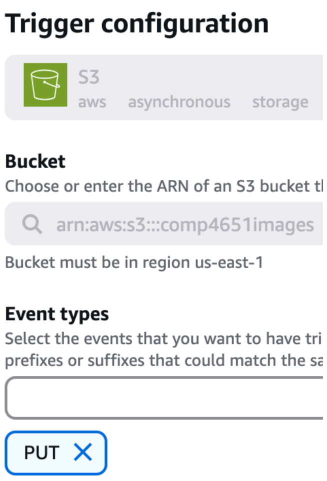
AWS_BUCKET_NAME=comp4651images // This is the name we gave to our S3 bucke

Task 2: Create an S3 trigger that invokes a Lambda function when an image is uploaded. Implement the Lambda function so that it retrieves the image from S3, sends it to the EC2 instance running the AI model and gets a classification result from it, and lastly sends the result to the EXC2 instance running the web app.

S3

We first created a new bucket in S3. It will receive an image from the web app backend. It stores it and a trigger we have configured will run our AWS Lambda function.

The trigger has been configured as shown in the image on the right (selecting our comp 4651 images bucket and PUT event types):



Trigger configuration

S3
aws asynchronous storage

Bucket
Choose or enter the ARN of an S3 bucket tl

arn:aws:s3:::comp4651images

Bucket must be in region us-east-1

Event types
Select the events that you want to have tri
prefixes or suffixes that could match the sa

PUT

AWS Lambda

When the trigger from S3 is executed, the AWS Lambda function will receive an event. We can extract the image from that event as follows:

```
bucket = event['Records'][0]['s3']['bucket']['name']
key = event['Records'][0]['s3']['object']['key']
response = s3Client.get_object(Bucket=bucket, Key=key)
image_data = response['Body'].read()
```

We will send the image through a POST request to the Pytorch model running on an EC2 instance. After, the function will receive the response from the server with the classification result and confidence. We will send this data back to the web application's backend with the POST API we mentioned earlier.

Task 3: Load the classification model on an EC2 instance to process the image for predictions

Model

The first simple model is the digit classification model. It is used for demonstration purposes. The second model is trained in a data analysis manner in a way that achieves high accuracy during training processing and aims to evaluate the accuracy of collected data stored in S3 in actual usage from public users. Please check the GitHub directory "./AImodel" for details.

EC2

For our application, we used two different EC2 t3.large instances.

- EC2 for web app: It runs the web app backend and front end
- EC2 for the AI model: It runs the CNN model that classifies digits.

One crucial aspect is that the communication in and out of EC2 instances using APIs requires to specify the particular public IP address of the EC2 instances when needed, as well as give enough security group permission to the instances to receive HTTP requests into the appropriate ports.

Additionally, for our project we gave 30 GB of disk space to each instance to allow installing all the necessary requirements to run the code (e.g. torch, numpy, etc.). It is vital to install all the libraries that our codes import before running the programs.

Task 4: Evaluate the accuracy model using MapReduce

Setting up and preparing AWS:

- Create S3 bucket: Create an S3 bucket to store predicted and actual value data.
- EMR Cluster Settings: Set up an EMR cluster to run Hadoop, which will run MapReduce operations.
- Set up EC2 instances: Create and set up EC2 instances for data processing and analysis.
- Upload Data: Upload the predicted and real value data to the S3 bucket

MapReduce job design:

- Implement Mapper Class: The Mapper class receives and processes input data and generates intermediate output data. The input data is divided into several small pieces and passed to each Mapper. The Mapper class reads each piece one by one, and converts the data according to the given logic. The converted data is output in the form of an intermediate key-value pair.
- Implement Reducer Class: The Reducer class receives the intermediate output data generated by the Mapper and generates the final result. The Reducer class receives all the intermediate data with the same key as one group. It collects the data from each group and processes it according to the desired logic, and outputs the final result in the form of a key-value pair.

- Implement Driver Class: The Driver class controls the flow of the entire task. First, create a configuration object, and initialize the Job object. Set the Mapper and Reducer classes through the Job object, specify the input and output paths. Submit the Job, and wait for the task to complete. This ensures that the MapReduce task runs on the Hadoop cluster.

Run the job in AWS EMR:

Runs the MapReduce operation on the EMR cluster, where the input path points to the data in the S3 bucket, and the output path sets the processed results to the S3 bucket path.

Analyzing Results:

- The output result data is downloaded from S3 and analyzed by the local system.
- Can write a simple script using Python to calculate the accuracy.

IV. Results

As a result of the implementation of all the components mentioned above, we have been able to create a fully functional cloud-based image classification system leveraging AWS services. The system successfully allows users to upload images, which are then processed by the AI model running on an EC2 instance. The classification results, including prediction labels and confidence levels, are displayed back to the users on the web app UI. Here is a demo video of the system working: https://drive.google.com/file/d/1mG-5ArxXxegJ4jyc-2LxEFV8596YTRCR/view?usp=sharing

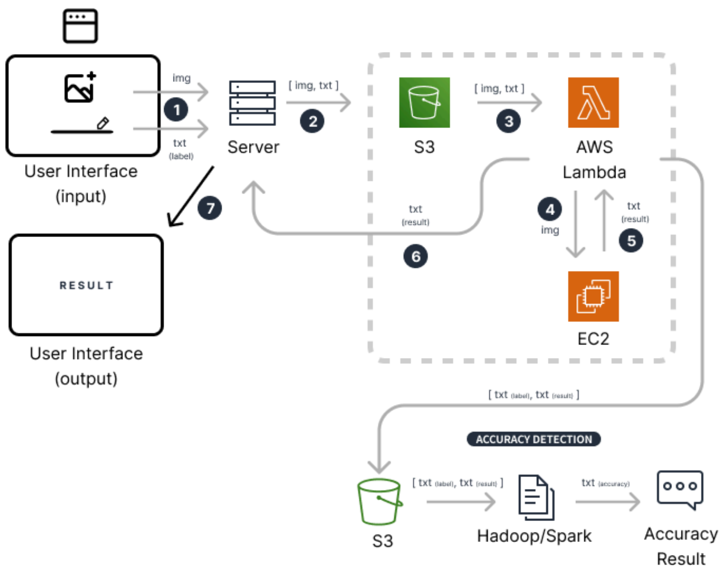
The system's architecture was designed to be scalable, ensuring that multiple users can upload images concurrently, with the backend processing handled efficiently using AWS Lambda and EC2 instances. Additionally, the modularity of our architecture allows us to easily change between different AI models to use for classification without affecting our web application, storage or communication components. We have implemented and integrated into the system both a digit image recognition CNN and a more robust sports image recognition model to test and demonstrate the aforementioned.

Lastly, we have leveraged map reduce with Hadoop in AWS EMR to distributedly and parallelly process a vast amount of image classification data stored in S3, allowing us to further extract insights from multiple AI model's performance.

V. Discussion

Future versions:

A possible ampliation of the project would be for the own users to give the true labels of the pictures they upload and the AWS Lambda will store it together with the image prediction in an S3 bucket for the map reduce to process. In the current version, the true label is already set. The new architecture would look as follows, where the user would also give the true label of the image:


User Interface
(input)
img
txt
(label)
1
Server
[img, txt]
2
S3
[img, txt]
3
AWS
Lambda
4
img
txt
(result)
5
EC2
txt
(result)
6
7
RESULT
User Interface
(output)
[txt (label), txt (result)]
ACCURACY DETECTION
S3
[txt (label), txt (result)]
Hadoop/Spark
txt (accuracy)
Accuracy
Result