Platform as a Service using Raspberry Pi IOT

Meher Abhilash Komirishetty-1220259144
Sreekar Manchukonda-1224487368
Bharghav Rapolu-1219931960

PROBLEM STATEMENT:

The project's goal is to leverage Amazon Web Services to create a Lambda Function that can accept several concurrent user videos as input threads and outputs the anticipated user details back to the PI. The tasks entail creating the model's architecture, which includes using Raspberry PI to capture user input videos, AWS Lambda (Function as a Service) that extracts frames from the input video and runs image recognition logic on individual frames, AWS DynamoDB(Unstructured database) to store the user data in the form of records and the outputs result to Raspberry PI upon invocation, and AWS S3 to store the input files from PI that would be downloaded locally by AWS Lambda for processing. All the components used in the application are serverless components: S3, Lambda, and DynamoDB. The components are perceived as serverless to the user but are contained on servers internally. The usage of serverless functionalities minimizes overheads of managing server-oriented components when not being used, thus saving costs.

ARCHITECTURE:

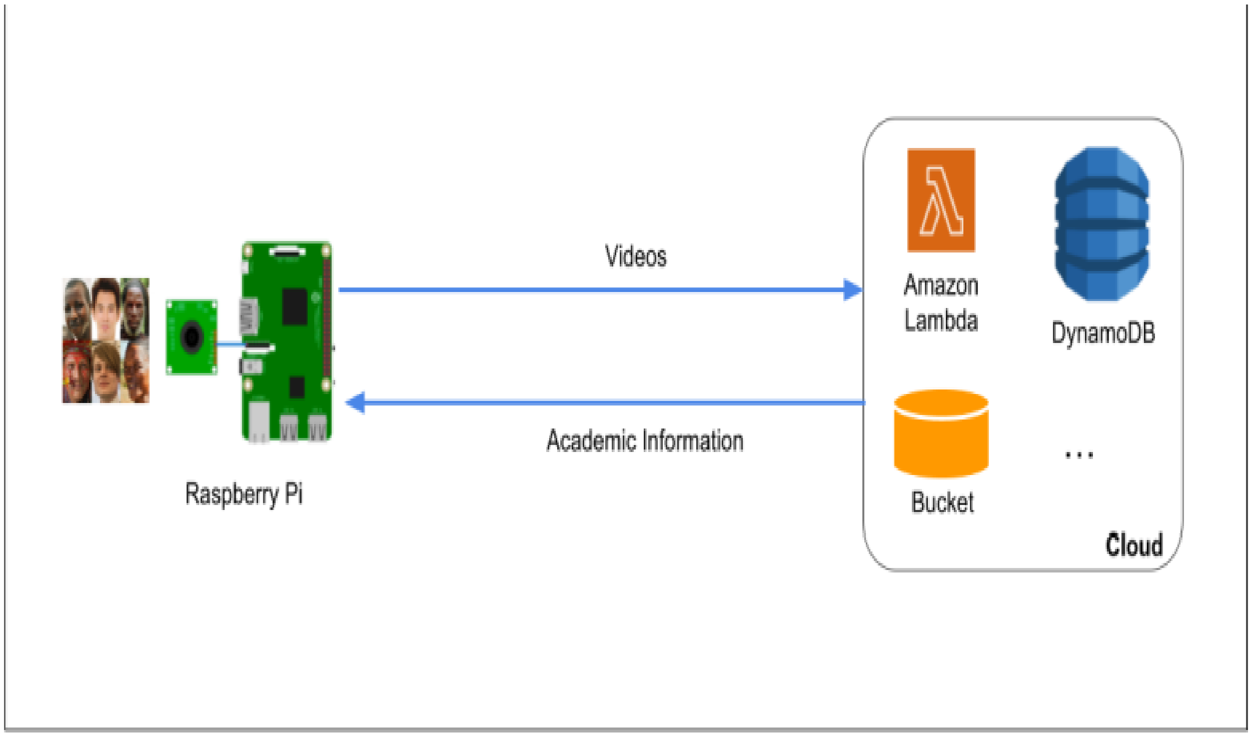


Figure 1- Architecture of the model

ARCHITECTURE DESCRIPTION:

Multiple user requests [videos] are provided to the AWS Lambda through the Raspberry Pi. The requests are successively provided in the form of threads and stored in the S3 Input Bucket. Upon the video's arrival in the S3 Bucket, an S3 PUT trigger is activated that invokes the lambda function and sends the JSON event to the handler function of the AWS Lambda. The AWS Lambda downloads and extracts frames from the video, thus running image processing logic on the extracted frames and outputting the result locally. The outputted result name is queried in the AWS DynamoDB where the corresponding details for the result name are fetched and stored in s3 result bucket which is pulled by the Raspberry Pi. The entire system flow executes by creating a new thread every 0.5 seconds and the corresponding result for the thread is received back, thus calculating the latency of processing.

AWS SERVICES USED:

- *AWS LAMBDA*
  The AWS Lambda Service is a lightweight computing service that provides Function as a Service functionality that gets executed upon a trigger event and processes the outputs based on the logic included in the function module.

- *AWS DYNAMODB*
  AWS DynamoDB is a Serverless NoSQL database that holds the data records in the form of key-value pair based on the partition key and sort key.

- *AWS S3*
  AWS Simple Storage Service stores the data in the form of bucket objects. The S3 Input bucket holds the user input videos and the S3 Result bucket holds the predicted result of each video.

SERVERLESS:

Without having to manage servers, AWS provides technology for running code, maintaining data, and integrating apps. To boost agility and save costs, serverless solutions provide automatic scaling, built-in high availability, and a pay-per-use invoicing mechanism. These technologies also take care of infrastructure management responsibilities like capacity provisioning and patching, allowing you to concentrate on building code that benefits the users.

The Serverless Computing Functionality of AWS is advantageous in the following ways:

- Save costs by paying for used compute time instead of provisioning infrastructure upfront for peak capacity.
- Optimize code execution time and performance with the right function memory size.
- Respond to high demand in double-digit milliseconds with Provisioned Concurrency

MODULES:

DOCKER CONTAINER IMAGE

Aws_properties.py
Contains the dynamic variables that are accessed through the python modules. Storage of the AWS User Credential, URLs of AWS Resources, and the common parameters is performed.

Lambda_handler.py
Contains the logic for downloading the video from S3 that initiated the trigger, extracting the video to frames locally, processing the extracted frames into the image recognition module, and pushing the result as an argument to get the details module.

get_details.py
Contains the logic for extracting the user details in a required format from DynamoDB and stores the result in the S3 Result Bucket.

eval_face_recognition.py
Contains the face recognition module that takes the user image from the required path as input and outputs the result back to the module from which the script is invoked.

requirements.txt
This text file contains all the required packages and API to be installed to run the application.

models
Contains the Image processing logic.

checkpoint
Contains the user data used by the models directory.

EDGE

pi_detector.py
Contains logic to record user videos by invoking fresh threads every 0.5 seconds and invokes the uploader logic to upload the input to the S3 input bucket.

uploader.py
Contains the logic to upload the corresponding video given by a thread and fetch the result from dynamo DB thereby displaying it on the console.

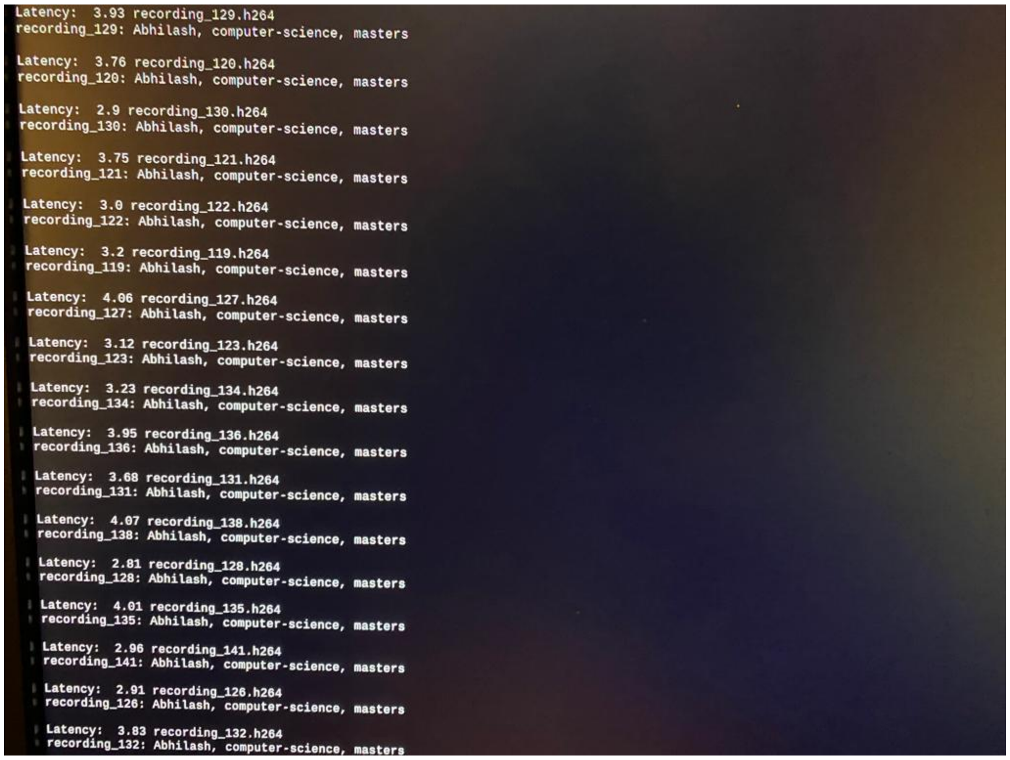
CONTROL FLOW:

- *EDGE PI*:
    - ➢ Concurrent Input videos are retrieved from the user every 0.5 seconds
    - ➢ Received videos are pushed into the S3 Input Bucket
    - ➢ Results are retrieved from S3 Result Bucket.

- *DOCKER CONTAINER IMAGE ON AWS LAMBDA:*
    - ➢ Invoked on S3 PUT Trigger and Fetches Data from the S3 Input Bucket
    - ➢ Input Images are fetched and the Predicted result is stored in a local file.
    - ➢ The output data is retrieved from dynamo DB using the result obtained earlier
    - ➢ The resulting file is stored in the S3 Result Bucket.

TESTING:

The Testing phase is performed by fetching input videos from the user through PI, processing it using AWS Lambda Functions, and checking the latency of obtaining the result back to the Edge by obeying the architecture. Several factors include multithreading video upload, Persistence of data in the S3 Buckets, functioning of face recognition, storage of data in DynamoDB, retrieval of accurate information for the respective user and the Predicted Results are checked, thus measuring the accuracy of the model.



```
Latency:  3.93 recording_129.h264
recording_129: Abhilash, computer-science, masters

Latency:  3.76 recording_120.h264
recording_120: Abhilash, computer-science, masters

Latency:  2.9 recording_130.h264
recording_130: Abhilash, computer-science, masters

Latency:  3.75 recording_121.h264
recording_121: Abhilash, computer-science, masters

Latency:  3.0 recording_122.h264
recording_122: Abhilash, computer-science, masters

Latency:  3.2 recording_119.h264
recording_119: Abhilash, computer-science, masters

Latency:  4.06 recording_127.h264
recording_127: Abhilash, computer-science, masters

Latency:  3.12 recording_123.h264
recording_123: Abhilash, computer-science, masters

Latency:  3.23 recording_134.h264
recording_134: Abhilash, computer-science, masters

Latency:  3.95 recording_136.h264
recording_136: Abhilash, computer-science, masters

Latency:  3.68 recording_131.h264
recording_131: Abhilash, computer-science, masters

Latency:  4.07 recording_138.h264
recording_138: Abhilash, computer-science, masters

Latency:  2.81 recording_128.h264
recording_128: Abhilash, computer-science, masters

Latency:  4.01 recording_135.h264
recording_135: Abhilash, computer-science, masters

Latency:  2.96 recording_141.h264
recording_141: Abhilash, computer-science, masters

Latency:  2.91 recording_126.h264
recording_126: Abhilash, computer-science, masters

Latency:  3.83 recording_132.h264
recording_132: Abhilash, computer-science, masters
```

INDIVIDUAL CONTRIBUTIONS

1. Meher Abhilash Komirishetty(1220259144)

Design

- I came up with the final design of the project after conceptualizing several architectures and concepts.
- My role was to implement the flow of the design from the architecture given and implement the design in the form of code.
- I also developed the core task of sending multithreaded videos from edge to the S3 Bucket.
- I was also involved in the lambda function implementation which is the core part of the problem statement, extracting frames from uploaded video, processing the uploaded video to the image recognition algorithm, and sending the result to the result pi bucket
- I also solved critical issues in the architecture design such as handling multiple threads and retrieving necessary results
- I am also responsible for evaluating and improving the performance of the entire model and making code changes to reduce the complexity.

Implementation

I completed the following tasks:

❖ Retrieval of Concurrent videos from edge
❖ Received videos are pushed into the pi input bucket
❖ Implementation of the lambda function to extract frames and process the videos
❖ Creation of the docker container image
❖ Multithreading logic to upload input videos into S3 Input Bucket
❖ Result Retrieval into Raspberry PI from the pi result bucket.

Testing

After implementing the above tasks in python, my job was to conduct unit, integration and end-to-end testing of our Image Recognition Service.

○ *Unit Testing:* Results of the image processing model are checked individually and accuracy is computed

○ *Integration Testing:* Frames are extracted after video is uploaded to pi input bucket and the result is obtained through image processing module

○ *End to End Testing:* We all participated in end-to-end testing of the model through recording concurrent videos and calculating latency of retrieval

2. Sreekar Manchukonda (1224487368)

Design:

- I was involved in understanding the architecture of the problem statement and structuring the implementation modules accordingly. We worked together as a group to evaluate the entire design of the application.
- I was primarily involved in building the lambda function logic and the flow of control behind running image processing logic to the extracted frames and retrieving necessary results from the DynamoDB
- I am also responsible for evaluating the performance of the model and improving it wherever necessary reducing the time and space complexity.
- I managed the security aspects of the AWS resources by configuring the security groups and IAM policies for them.
- Building Docker container locally and uploading the container to AWS ECR and generating a lambda function through the docker container

Implementation:

My tasks include:

- ❖ Writing logic to push input videos to the pi input bucket and fetch results from the result bucket
- ❖ Implement the Image Processing module over the input image and push the result to the user
- ❖ Creating and configuring lambda function configurations and deploying docker container over the lambda module
- ❖ Deployment of the source code into a docker container

Testing

○ Unit Testing: I created each and every code module and tested them individually by keeping different parameters and scenarios in mind.

○ Integration Testing: I tested the flow of the application after setting up all the code modules on the lambda function

○ End-to-End Testing: The entire working of the model has been tested where We tried different parameters and uploaded different numbers of images to verify the correctness of our code.

3. Bharghav Rapolu (1219931960)

Design

● The design of the system was implemented in collaboration with my coworkers. In addition, I worked on training and testing the image processing module

●I am responsible for creating the necessary AWS resources and managing them

●I am also involved in creation of docker containers and managing it

●I took responsibility of installing and configuring the edge thereby ensuring proper functioning of the Pi

Implementation.

● Implemented the raspberry pi and docker container.
  - ❖ Installing python and required apis
  - ❖ Managing network aspects and configurations whenever necessary
  - ❖ Training the image recognition module and validating it
  - ❖ Delete the unwanted modules from aws resources whenever required through automation

Testing

●On my local PC as well as on lambda, I thoroughly tested the working of the logic.Unit testing, integration testing, and end-to-end testing were the three phases in which I tested.

Unit testing: I tested each module separately (on a local server) to confirm that the functionality was correct.

Integration Testing: I uploaded the logic to lambda function and tested the triggering of lambda function and processing of the result

End-to-End Testing: We tested the end-to-end flow of the project and validated the accuracy and the obtained latency