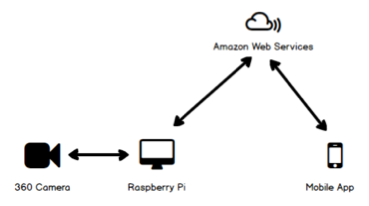
**CAMERA 360 DEGREE**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Description** |
| 1.0 | 11 Dec 2017 | Ngo Thuc Dat | Step by step of process |
|  |  |  |  |

**Introduction**



**360 Degree Camera**

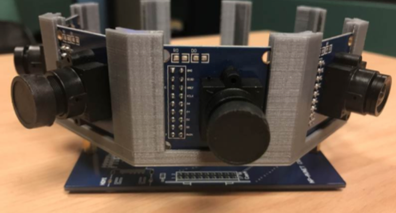
Take video streams from camera; setup a web server locally with the video stream embedded in an internal website. The camera is currently set with IP  192.168.1.73.

* Stream URL :

The website URL is 192.168.1.71:8080/jpegstream.html.

* Snapshot URL:

Take snap shot URL is http://192.168.1.22:8080/?action=snapshot





**Raspberry Pi**

* **Set up**

Raspberry Pi 3 connect to Camera module through Ethernet port.

Raspberry Pi 3 need static IP add to access ‘stream’ and ‘snapshot’ and follow : 192.168.1.x ( x > #22)

* **Task**

Raspberry runs Python scripts to accomplish following tasks repeatedly.

* Access camera website and extract 4 images.
  + Naming of the images are suggested as following. For micro-zed unit “001”(id = 1) with four cameras “1”, “2”, “3”, “4”, the image from the 4 cameras are “cam01\_1.jpg”, “cam02\_1.jpg”, “cam03\_1.jpg” and “cam04\_1.jpg”. We may consider add a same timestamp in front of all 4 images, e.g. “20171023\_080000\_a.jpg”, “20171023\_080000\_b.jpg”.
* Sending 4 image to S3\_AWS to do “Stitching” and “Motion Analysis”. For each image, it calls web service “s3\_signed\_url/upload” to get a signed URL for uploading.
  + The Key used for the signed URL will be in format of “001/20171023\_080000\_a.jpg”, this is so that images of same micro-zed will be in same folder.
* It calls a webservice “sqs\_send\_queue” to send a message to SQS.
  + The SQS queue name is “iot-centre-projects”. This queue is shared by all projects in IoT Centre.
  + Each message contains attributes: “project”, “action”, “bucket”, “input”, “output”.
  + “project” is used to identify which project the message belongs to.
  + “action” is used to identify the action are required: “stitching”, “motion-analysis”.
  + “input” are the required information for the action.
  + “output” are the required output from the action.

**Local computer Emulator**

* **Fetching Image Emulator:**

Local already fetching a lot of Image for 4 continuous camera and store to local memory. (Try to setup Micro-Zed snapshot picture with Objects far away on that pictures. If Objects too near the camera, algorithms of Stitching may be throw Errors in real time executing).

* **Copy Image:**

A main program now instead of fetching real time Image from camera, It will get Images are stored in local by copying. It’s Emulating fetching images. So, when hardware set up finish, we change to fetching directly from Micro-zed Unit by “Comment Copying Command: shutil.copyfile()” and Uncomment function “fetching(MIN\_INDEX, MAX\_INDEX)” in “main.py” file.(MIN\_INDEX = MAXINDEX = ID of Micro-Zed for now)

(MAX\_INDEX and MIN\_INDEX may be confusing because it create can fetching a lot of Image one time for testing - So for real time working MIN\_INDEX = MAX\_INDEX=ID will fetching 4 images for Micro-zed with one time fetching).

* **Upload Image:**

After fetching process finished, 4 new images will be store on Cam01, Cam02, Cam03 and Cam04. That folder stays constant on “Data” folder. They will be uploaded to S3 of AWS by invoking: s3\_signed\_upload.

It will return information of images location on S3.

* **Call webservice:**

After uploading image, we got information of images. We call WebService <https://jm307gwsej.execute-api.ap-southeast-1.amazonaws.com/api/sqs> and pass json data to send to SQS

* **How to run**

From now just call python complier to “main.py” file on “DatNT\_Raspi/Main” Folder. All process above will run.

**TESTING PROCESS STITCHING AND MOTION-ANALYSIS LOCAL**

* **Required software:** Computer need to install Python 2.7(not > 3) and OPENCV 3.0 to run.
* **Running**
* **Stitching:**
  + Stitching process will stitch four image one time executing. To run:
    - “cd DatNT\_Raspi\StitchingMaker”

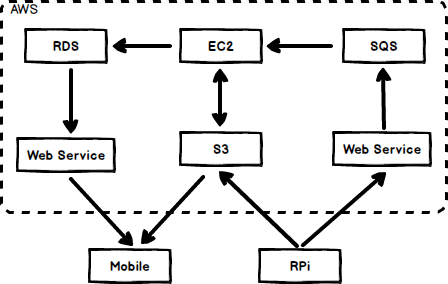
“python stitching.py”

* + (After finished, we got 4 \* (MaxIndex – Mindex) image(s). (Maxindex and MinIndex are specified in code)
* **Motion-analysis:**
  + - **On the Stitched Image:**
  + Need to run stitching finished.
  + The process will get stitched images and do algorithms analysis on them.
  + Run command: “python crowd\_new.py” on “DatNT\_Raspi\Demo\_Crowd” folder.
    - **On four Images:**
  + The process will get images from folder.
  + Run command: “python MID\_analysis.py” on “DatNT\_Raspi\Demo\_Crowd\_4\_Image” folder.

**Amazon Web Services (AWS)**

AWS is setup for following purpose:

* S3 for storage of images.
* SQS for queuing of jobs. The SQS message defines the type of job, inputs and outputs.
* Lambda + API Gateway to implement web service for coordinating works, e.g. uploading & serving of images, creating of new jobs in SQS, serving of data from database
* EC2 for running python scripts for image analysis.
* RDS for storing of analysis data.



**Web Services**

* Lambda + API
  + **Request**

|  |  |
| --- | --- |
| **Method** | **URL** |
| **GET** | https://jm307gwsej.execute-api.ap-southeast-1.amazonaws.com/api/<microzed-unit> |

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Params** | **Value** | **Remark** |
| URL | <microzed-unit> | string | Example:  “/api/1” |

* **Response**

|  |  |
| --- | --- |
| **Status** | **Response** |
| 200 | {         "percent": 0.00,  } |
| 400 | {"message":"Missing Authentication Token"} |

* + **Request**

|  |  |
| --- | --- |
| **Method** | **URL** |
| **GET** | https://jm307gwsej.execute-api.ap-southeast-1.amazonaws.com/api/sqs |

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Params** | **Value** | **Remark** |
| URL | message | string | Example:  {  "project": "360 Degree Camera",  "camera\_id": "1",  "camera\_name": "001",  "action": ["stitch", "montion-analysis"],  "bucket": "iot-centre-projects",  "output": "panorama",  "key\_prefix": "360-degree-camera/",  "input" : {  "cam01\_key": "Cam01/1512966860620\_1.jpg", "cam02\_key": "Cam02/1512966860620\_1.jpg", "cam03\_key":"Cam03/1512966860620\_1.jpg",  "cam04\_key": "Cam04/1512966860620\_1.jpg"  }  } |
|  |  |  |  |

* **Response**

|  |  |
| --- | --- |
| **Status** | **Response** |
| 200 | {         "status": done,  } |
| 400 | {"message":"Missing Authentication Token"} |

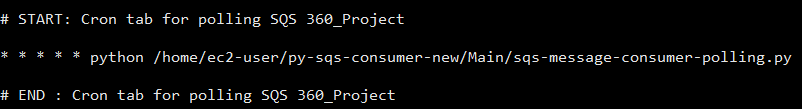
* **S3 signed url**
  + For RPi (local computer):
  + Grab four images are fetched from camera, local uploading to S3 by using “s3\_signed\_upload”. This will do 4 times for 4 images.
    - Call uploading webservice(lambda+api), request include key of file will locate on S3
    - Response of request success will return a “Signed URL”
    - Upload image to signed URL
  + After uploading success, call SQS webservice that will send information of 4 images and bring actions of user(for now is stitching and motion-analysis). A SQS message will create and stand on QUEUE of SQS after this task done.

**Simple Queue Service**

* Is invoke by lambda function
* Automatic created a message that contain Json data of images
* Waiting for EC2 polling messages to get data content to process and delete after finish.

**EC2**

* Crontab is set up for polling every minute on SQS to check new “message” coming.



* Which each new message coming:
  + Get key S3 images from content of message JSON
  + Download all four images and store to local
  + Get key actions
  + Delete message
* Get key actions:
* Stitching actions:
  + Do Stitching images “from StitchingMaker.stitching import stitching” function. Stitching 4 images is downloaded one time only.
  + Send image stitched(panorama) back to S3 using “s3\_signed\_upload”
* Motion analysis:
  + Old version (in folder ‘py-sqs-consumer’):
    - Check if existed previous panorama image else break
    - Do motion analysis algorithm of new stitched image and older stitched image (previous panorama image)
    - Store value of motions analysis return when finished and images path in S3 to RDS database AWS service.
    - Store back S3:
      * 2 copied after analysis finished

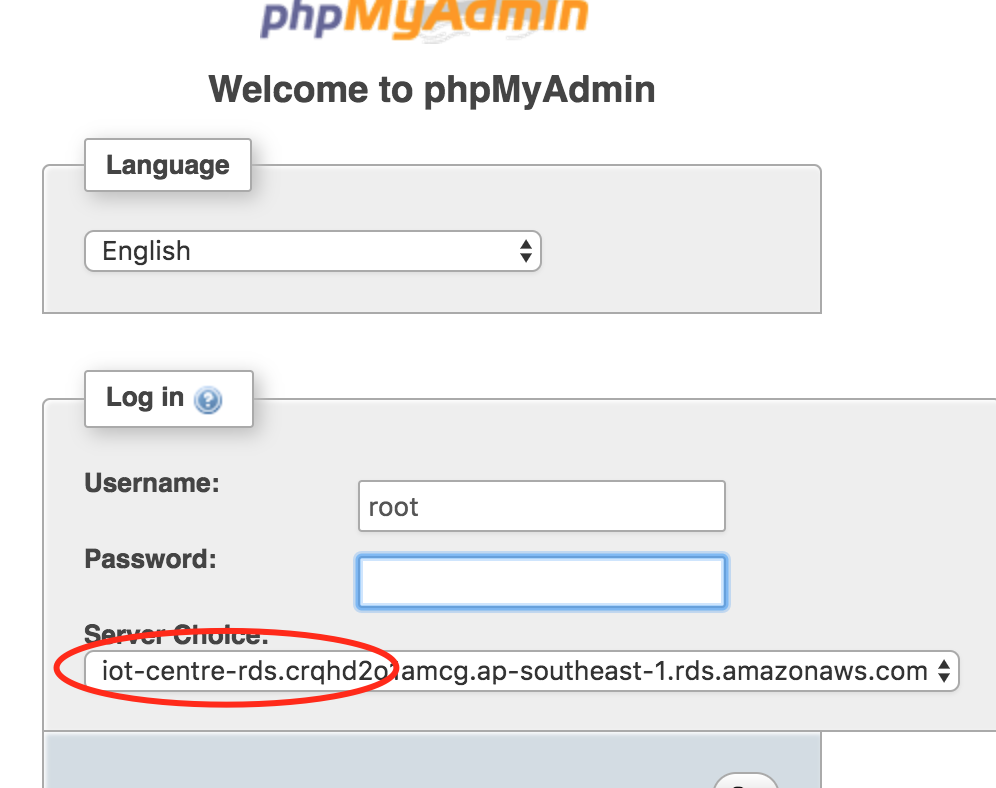
Key\_prefix/Timestamp\_Unit.jpg (1293478274\_001.jpg)

Key\_prefix/Pano/Unit.jpg(001.jpg - constant)

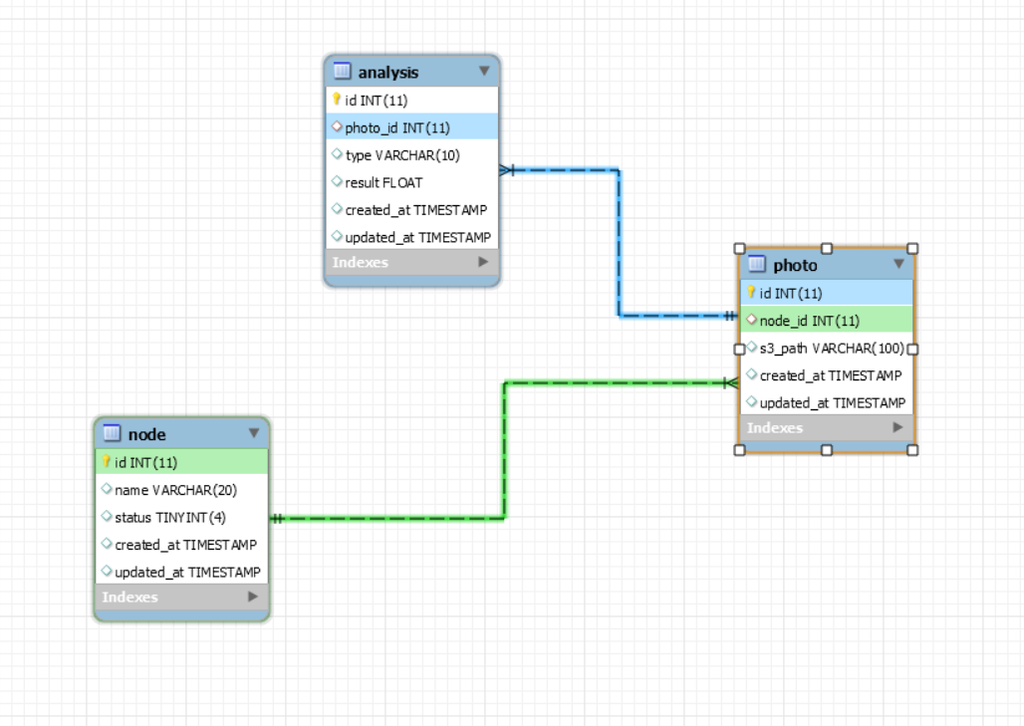
* + - With analysis stitched image, store with name “previous\_pano.jpg” in local.
  + New version: If want to analysis individual image, store with name each image “previous\_cam0x\_x.jpg” (x stands for “id” of image)
    - Check if existed previous 4 images else break
    - Do motion analysis algorithm of 4 new image and 4 older images (one by one correspond). And return 4 value of motion analysis.
    - Calculate total 4 values, divide 4 and store that result value follow with 4 images path, panorama path to RDS service.
    - Store 4 analysis images back S3 with name: key\_s3/cam0x/cam0x\_als\_{camera\_id}.jpg
    - With 4 analysis images, store with name “cam01/previous\_cam0x\_x.jpg”

**Database**

* Using RDS service AWS:
* Connect: http://13.228.113.29/phpmyadmin/



* Model:

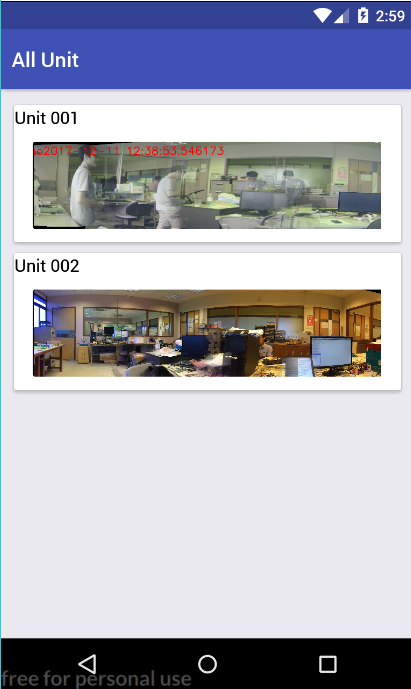
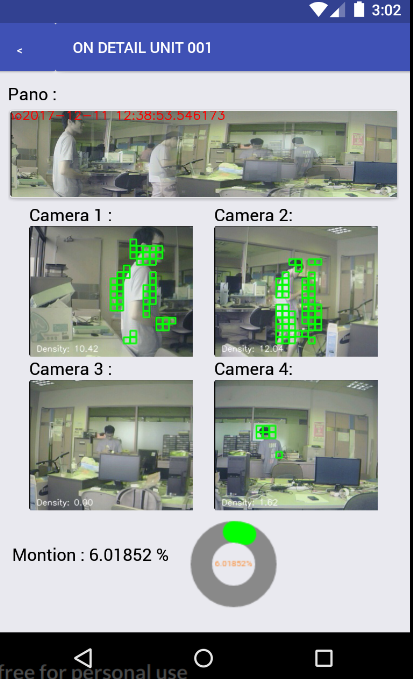
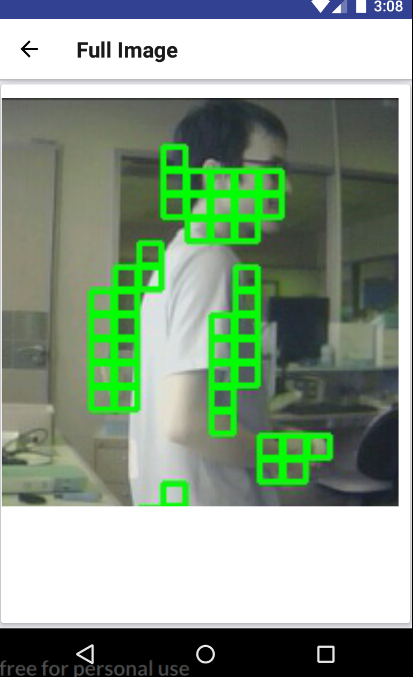


* Database content three model analysis, photo, node.
  + One node can have “n” photo.
  + One photo can have “n’ analysis.
* The job stores value is executed by python code on EC2 by calling service using peewee.
* The job gets newest value motion analysis is executed by python code on lambda function that invoke by Rest API

**Mobile App**

https://github.com/qinjie/360-Degree-Camera/tree/Datnt\_Mobile\_Dev/Cameras360Degree

This app is developed by React native language include Three Screen ( MainScreen, OnDetailScreen and OnViewerScreen)

* + All method get image is s3\_signed\_url
  + Main Screen
    - Get all panorama images is standing on S3, each panorama of each micro-zed unit is stand “key\_s3/Pano/pano\_<micro-zed name>.jpg”
    - 
  + On Detail Screen
    - Panorama image is passed from MainScreen
    - Get more 4 individual images using s3\_signed\_url
    - Data of motion analysis is got from Webservice Rest API with get method.
    - This screen will refresh every 10 seconds and check new data update on sever.
    - 
  + On Viewer Screen
    - Image data is passed from anywhere. This screen will take the data and show in View mode
    - 

Appendix:

EC2 Dev Code : <https://github.com/qinjie/360-Degree-Camera/tree/Dev_EC2_DATNT>

* 360-lambda folder: chalice dev serverless for lambda + rest API
* py-sqs-consumer: Server handle for stitching and analysis

Mobile react-native code : <https://github.com/qinjie/360-Degree-Camera/tree/Datnt_Mobile_Dev>

App uploaded to Play store at : https://play.google.com/store/apps/details?id=com.cameras360degree