RMIT AIDA
Proof of Concept

System design & architecture

6th Sep 2018

Contents

Introduction

Objective:

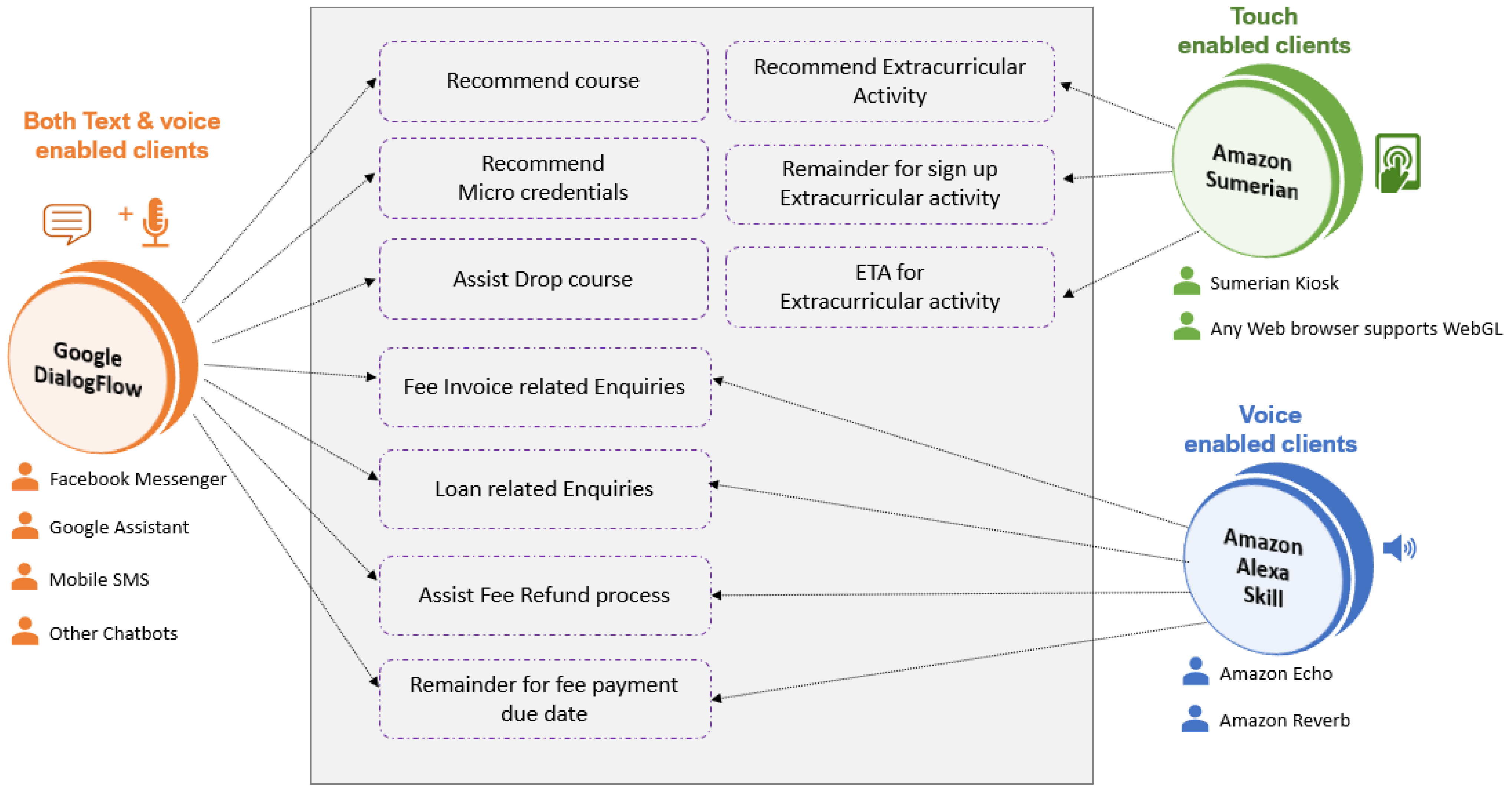
Deliver a Proof of Concept (POC) project towards implementing a tangible prototype of an Artificial Intelligence Digital Assistant (AIDA) (RMIT virtual assistant) and outlining the AIDA roadmap for the next 2 years, which takes input from the RMIT60 project and AIDA discovery work performed by RMIT over the previous year.

Background:

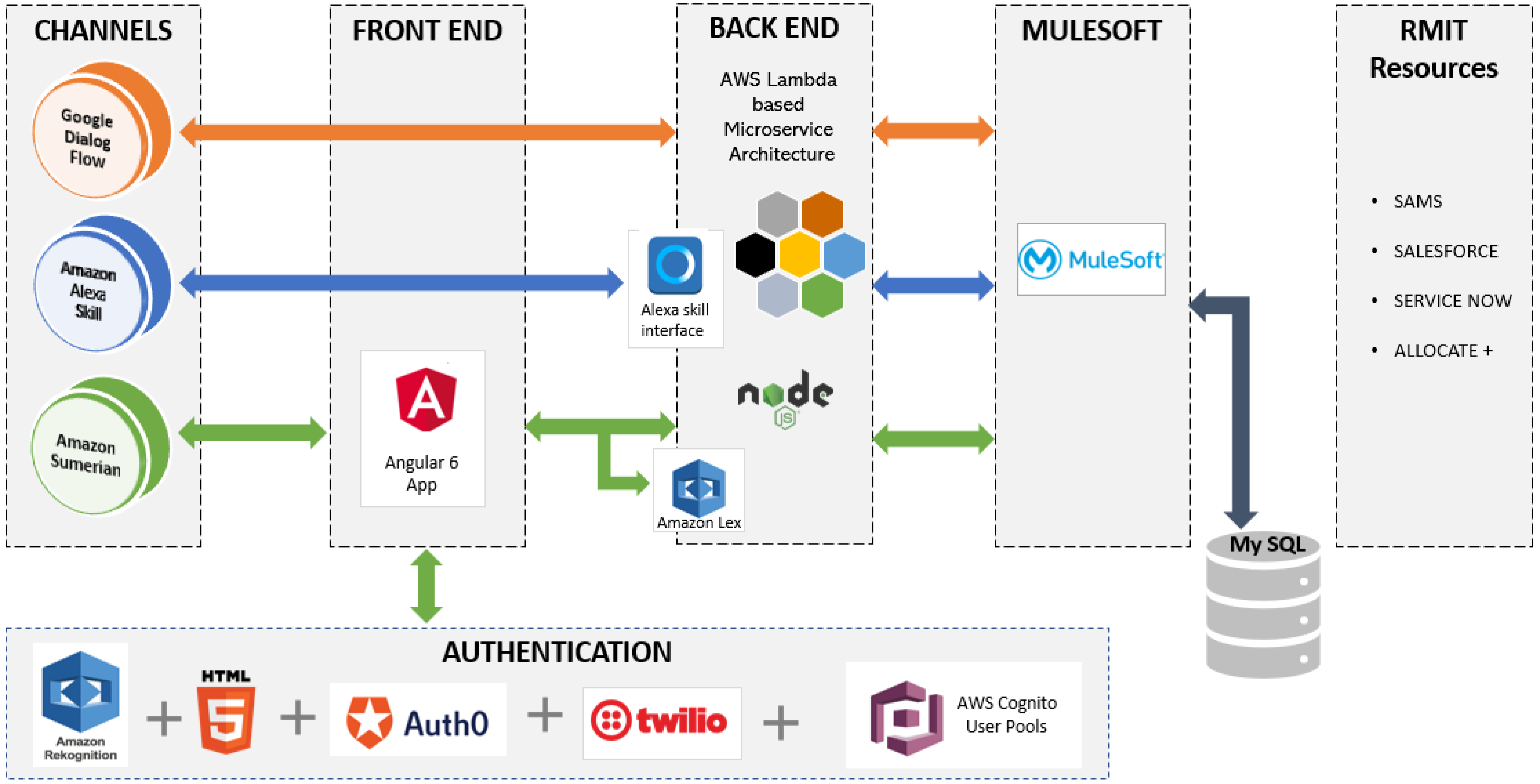
AIDA will be instrumental in driving student engagement and RMIT's belonging strategy, and it is intended to leverage Artificial Intelligence and cloud technologies by Amazon Web Services - AWS, alongside conversational design capability.

The AIDA POC is building an AIDA prototype by adopting a number of sample use cases to help inform and validate identified key risk areas and the viability of AIDA, alongside providing input into the future stages and product roadmap for production releases in the future.

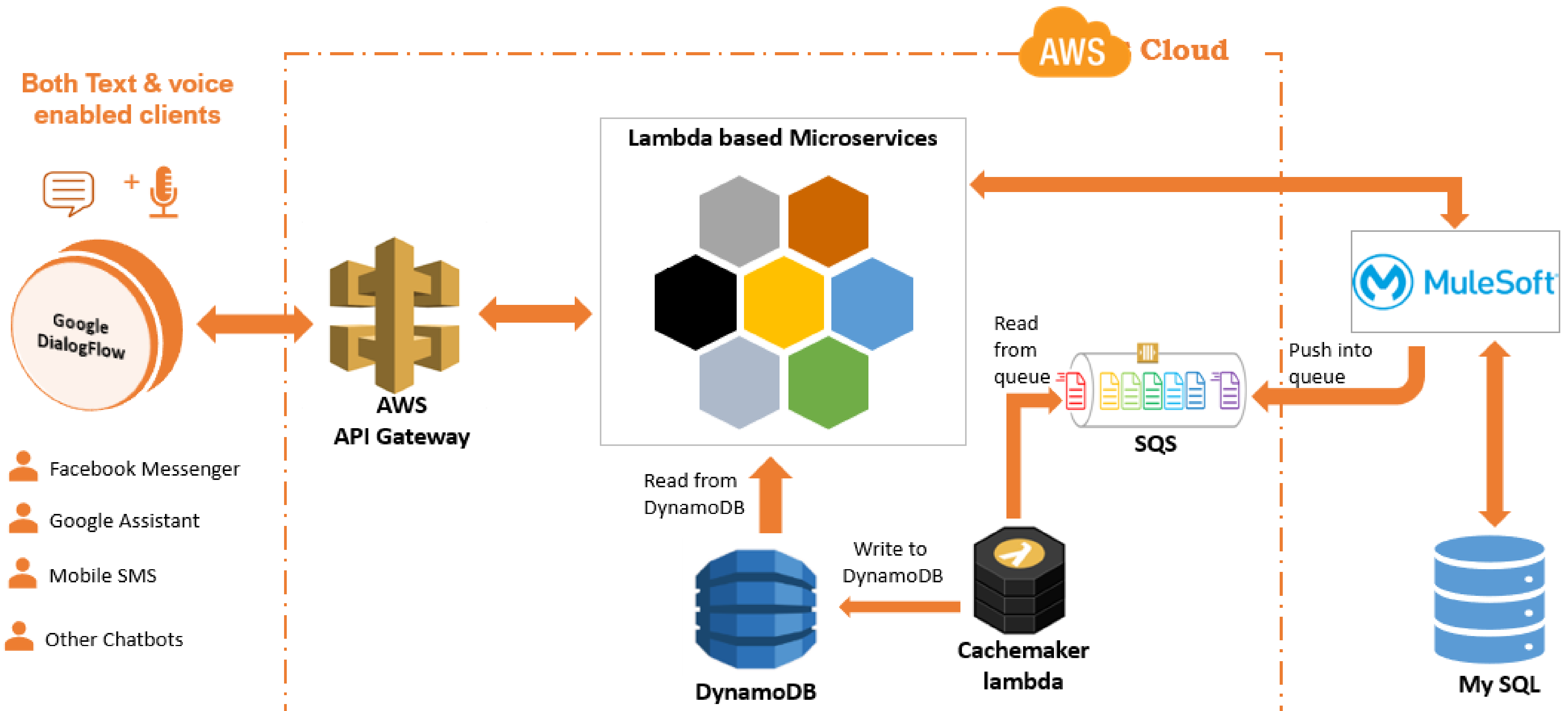
Use cases diagram


Both Text & voice enabled clients
Google DialogFlow
Facebook Messenger
Google Assistant
Mobile SMS
Other Chatbots
Recommend course
Recommend Micro credentials
Assist Drop course
Fee Invoice related Enquiries
Loan related Enquiries
Assist Fee Refund process
Remainder for fee payment due date
Recommend Extracurricular Activity
Remainder for sign up Extracurricular activity
ETA for Extracurricular activity
Touch enabled clients
Amazon Sumerian
Sumerian Kiosk
Any Web browser supports WebGL
Voice enabled clients
Amazon Alexa Skill
Amazon Echo
Amazon Reverb

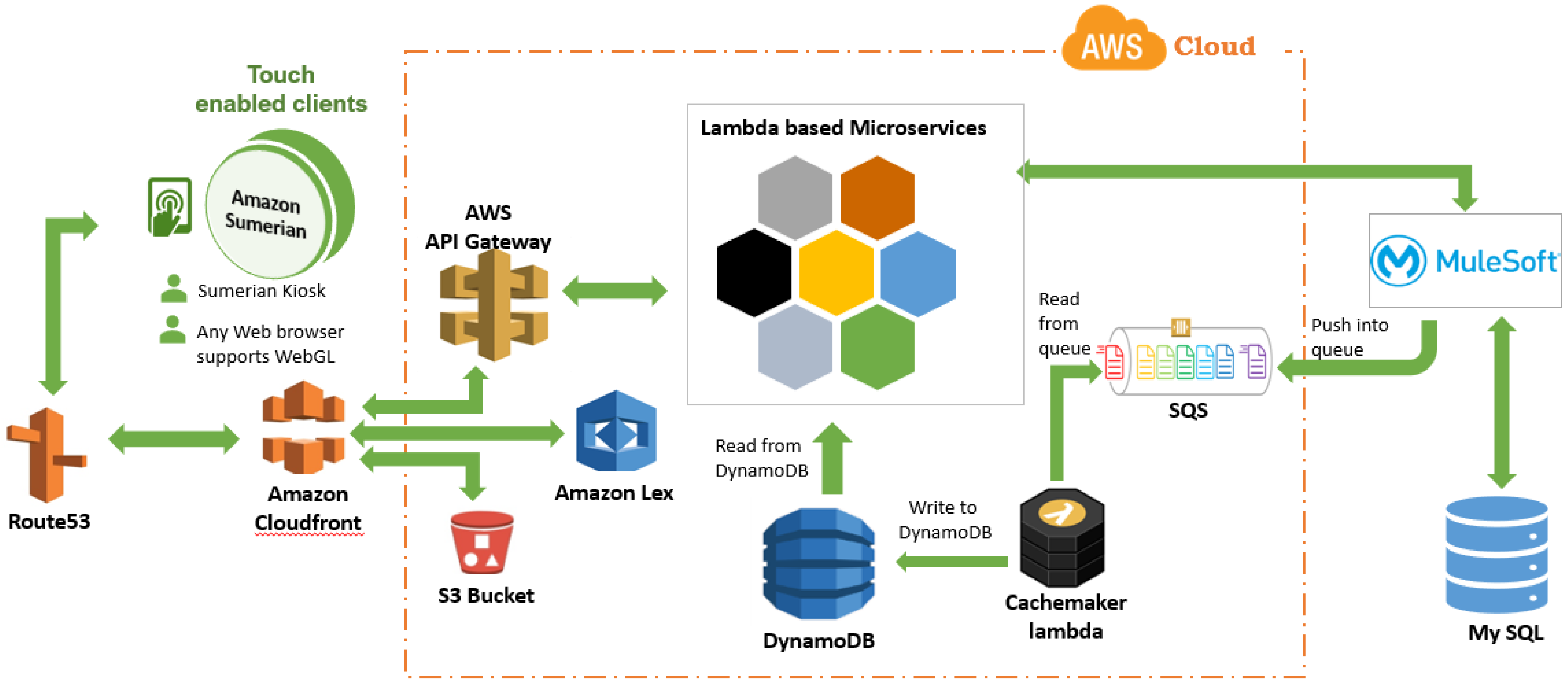
High level system view



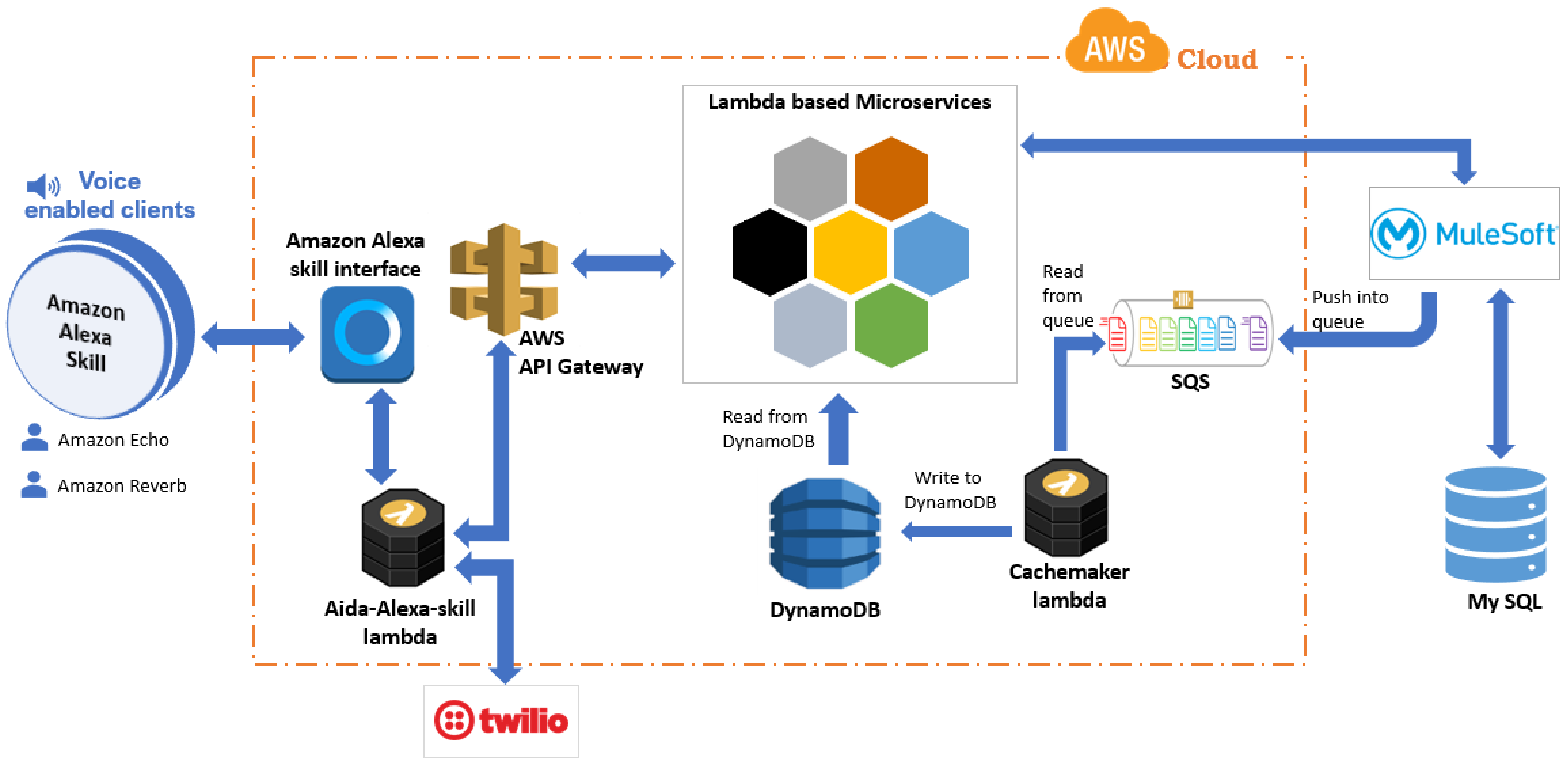
Google Dialogflow – overall view



Amazon Sumerian – overall view

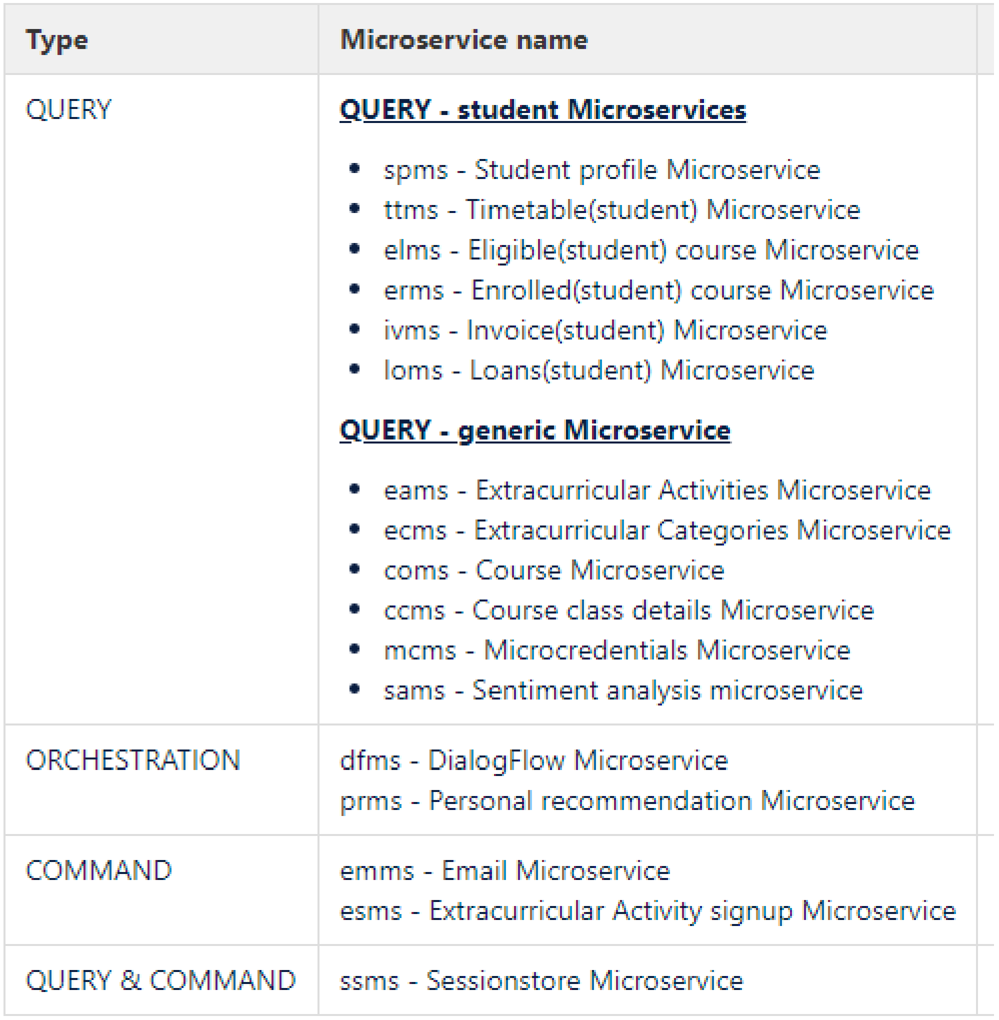


Amazon Alexa skill – overall view



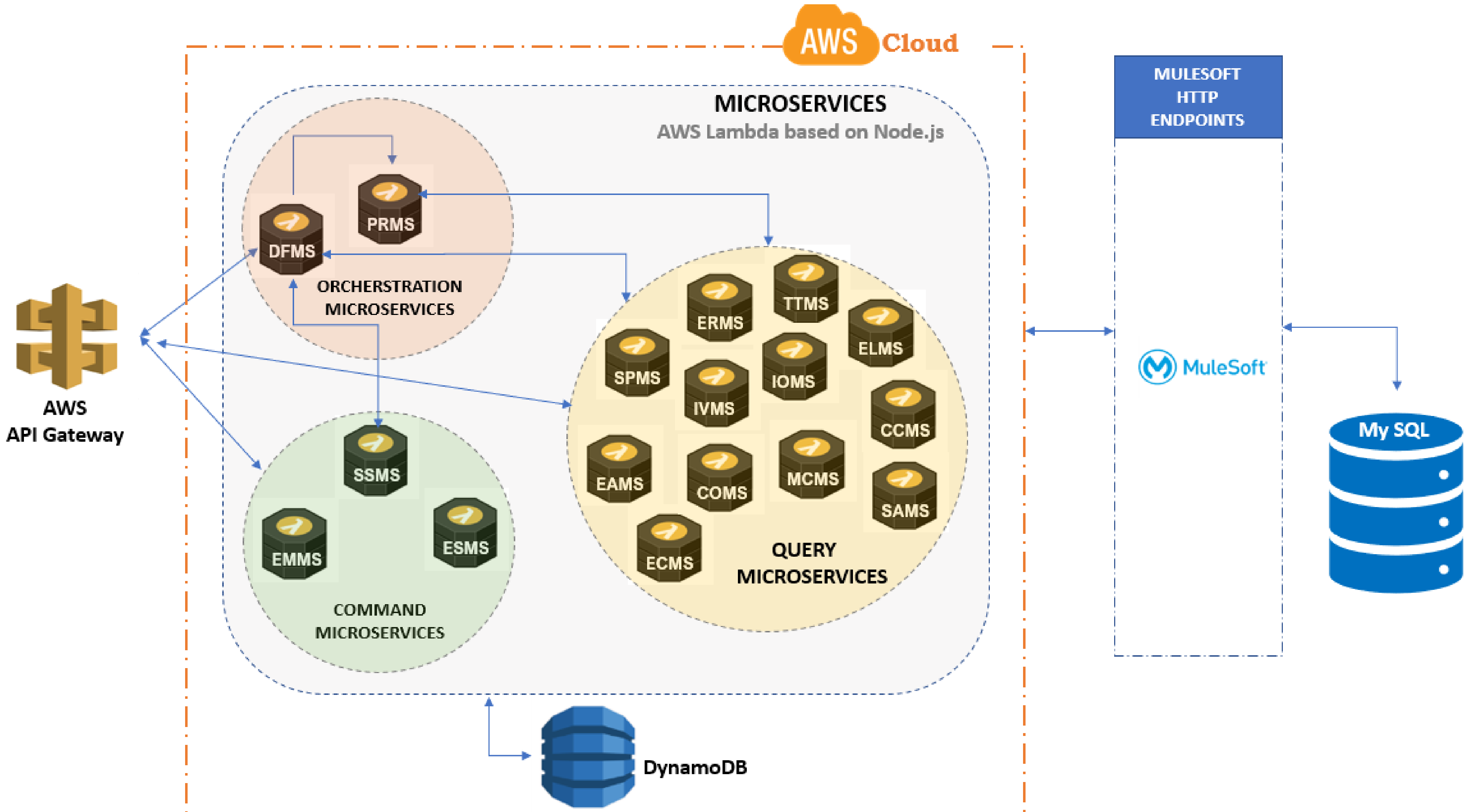
Microservices

Following domain related microservices were identified. All microservices are developed in node.js and deployed in AWS Lambda proxy and connected to AWS API gateway.

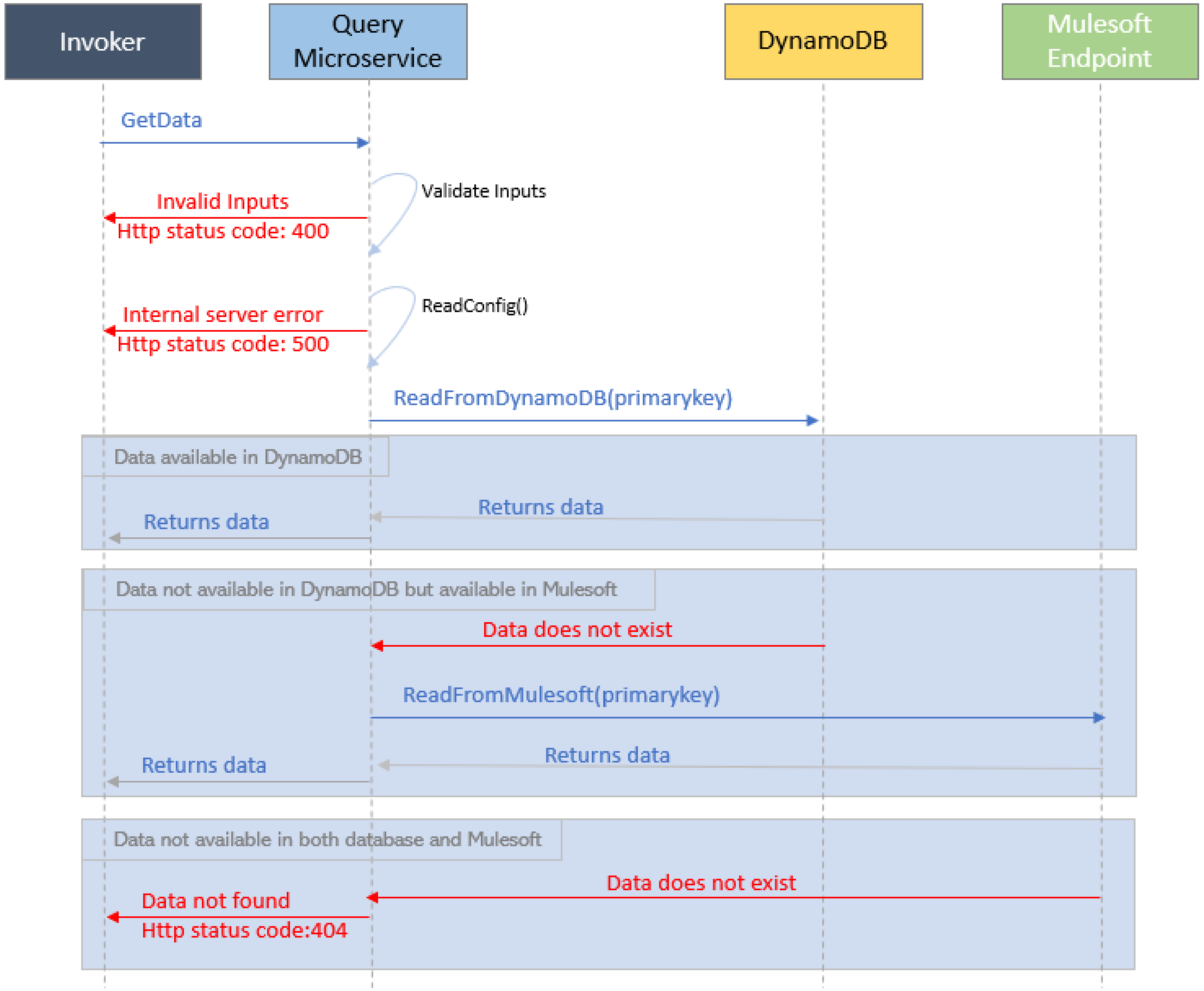


Type	Microservice name
QUERY	**QUERY - student Microservices** • spms - Student profile Microservice • ttms - Timetable(student) Microservice • elms - Eligible(student) course Microservice • erms - Enrolled(student) course Microservice • ivms - Invoice(student) Microservice • loms - Loans(student) Microservice **QUERY - generic Microservice** • eams - Extracurricular Activities Microservice • ecms - Extracurricular Categories Microservice • coms - Course Microservice • ccms - Course class details Microservice • mcms - Microcredentials Microservice • sams - Sentiment analysis microservice
ORCHESTRATION	dfms - DialogFlow Microservice prms - Personal recommendation Microservice
COMMAND	emms - Email Microservice esms - Extracurricular Activity signup Microservice
QUERY & COMMAND	ssms - Sessionstore Microservice

Microservices design

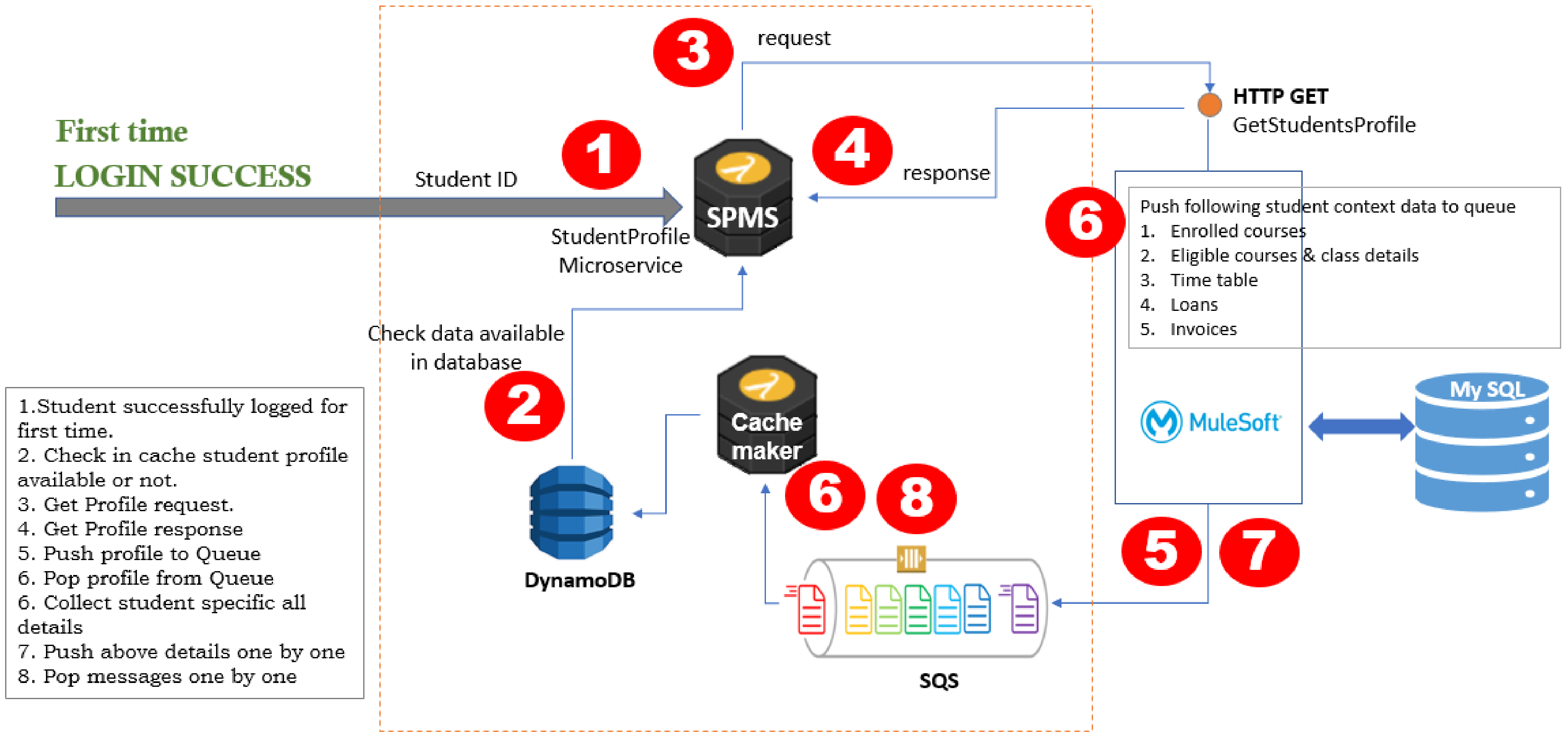


Query Microservices – sequence flow diagram



Caching design

Following diagram depicts caching design which follows early loading technique. All cache data is stored in DynamoDB database and will be populated asynchronously as soon as user logged in for the first time. Cache invalidation is not considered in POC.

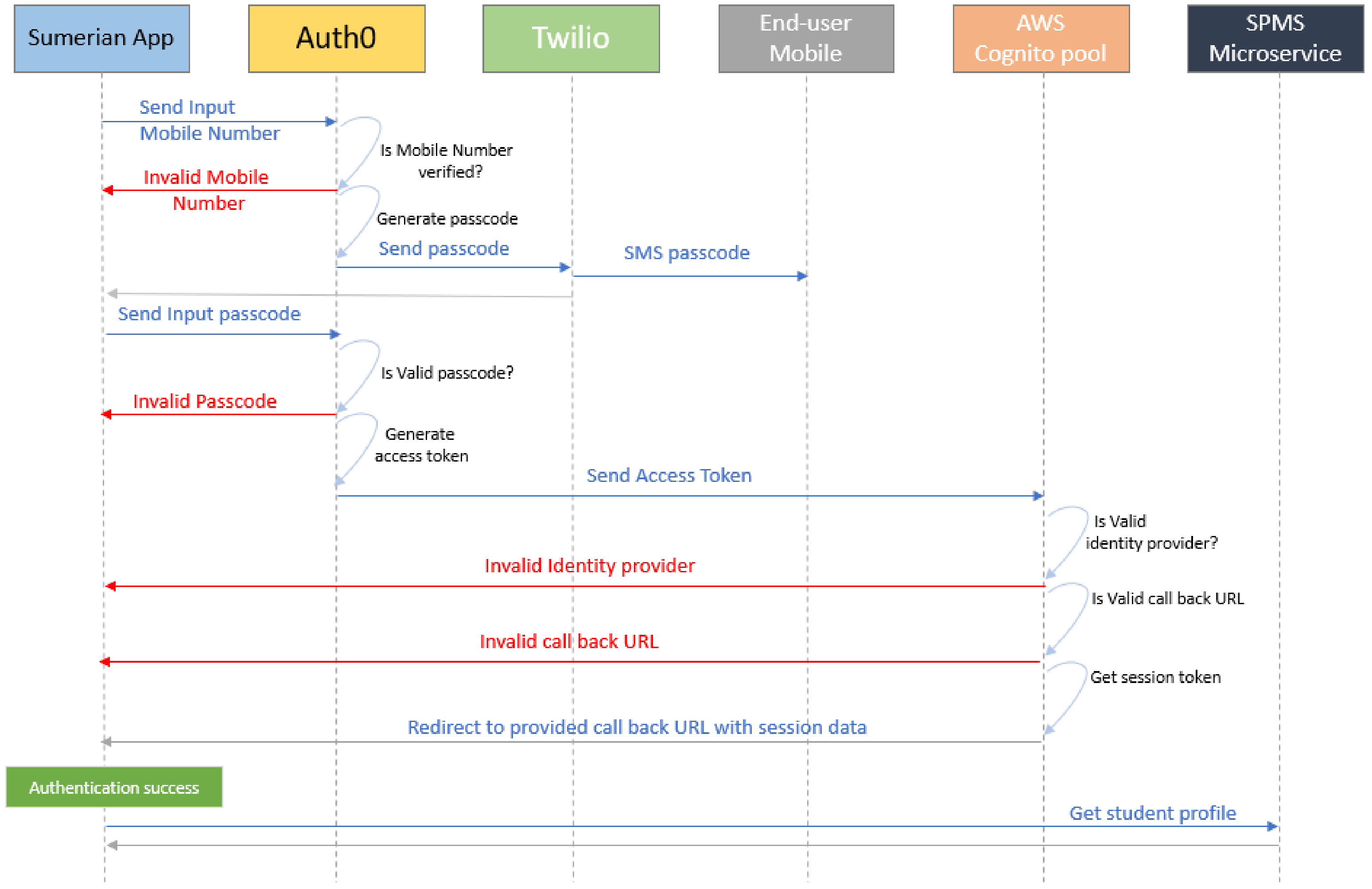


Push following student context data to queue
1. Enrolled courses
2. Eligible courses & class details
3. Time table
4. Loans
5. Invoices

1.Student successfully logged for first time.
2. Check in cache student profile available or not.
3. Get Profile request.
4. Get Profile response
5. Push profile to Queue
6. Pop profile from Queue
6. Collect student specific all details
7. Push above details one by one
8. Pop messages one by one

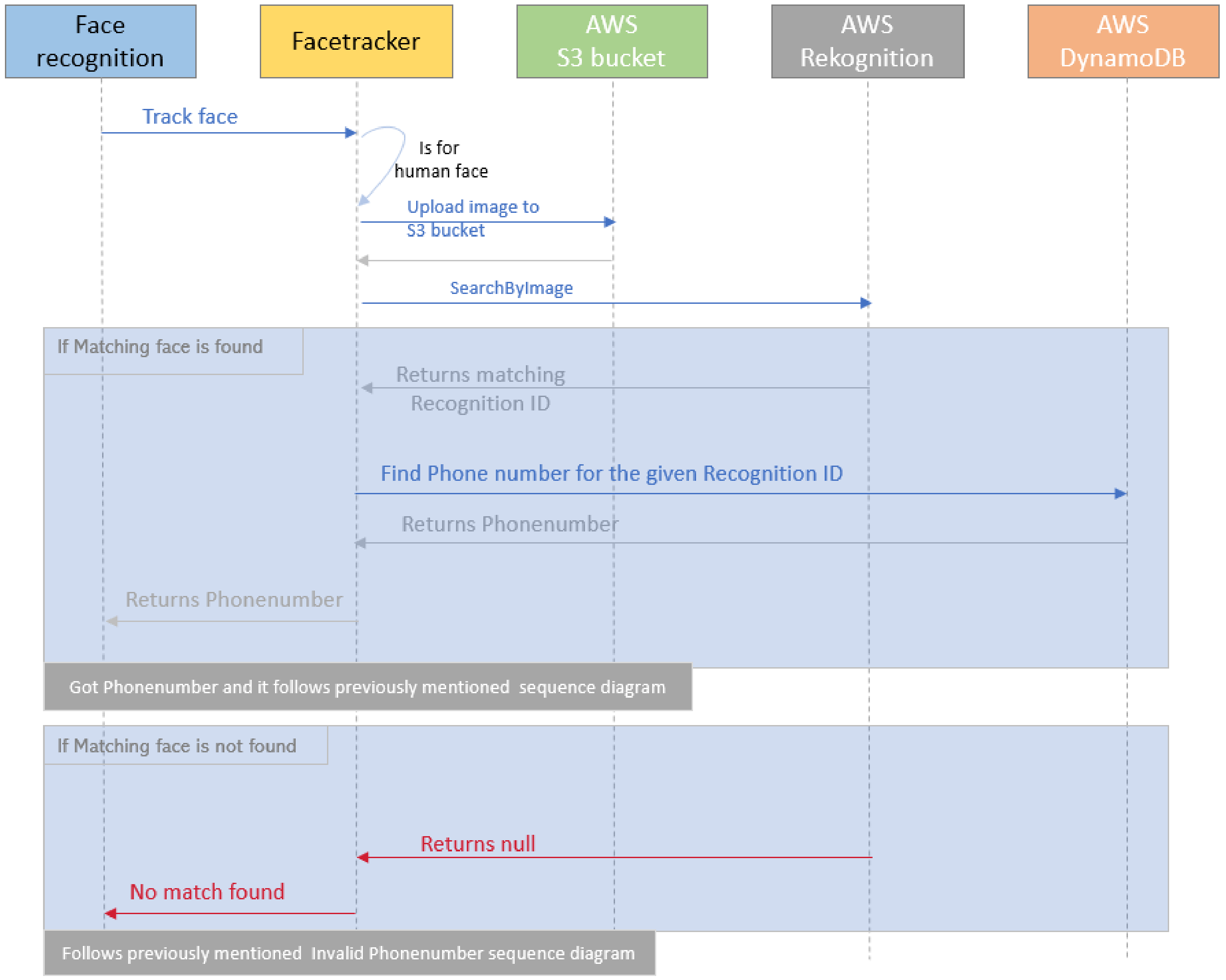
Authentication with PhoneNo– sequence diagram

without facial recognition



Authentication with face – sequence diagram

Below sequence diagram should be seen in conjunction with previous diagram for completeness.

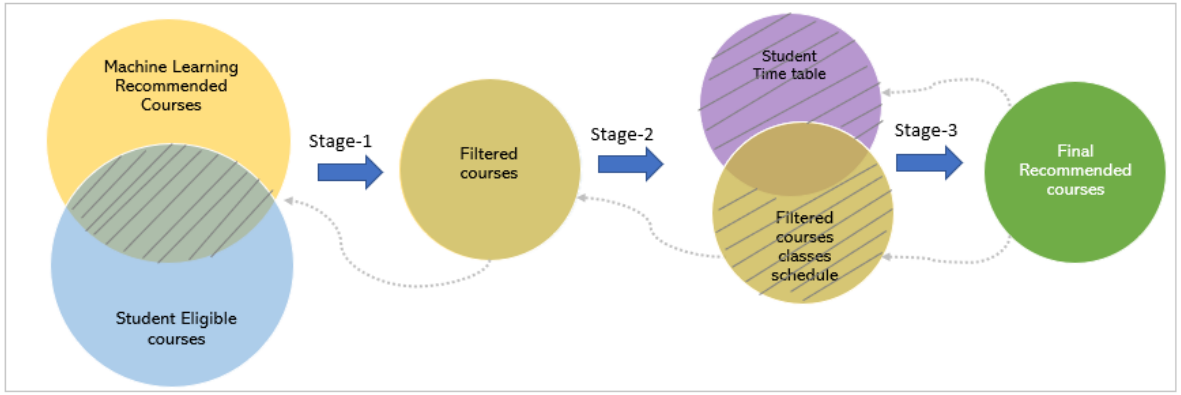

Face recognition
Facetracker
AWS S3 bucket
AWS Rekognition
AWS DynamoDB
Track face
Is for human face
Upload image to S3 bucket
SearchByImage
If Matching face is found
Returns matching Recognition ID
Find Phone number for the given Recognition ID
Returns Phonenumber
Returns Phonenumber
Got Phonenumber and it follows previously mentioned sequence diagram
If Matching face is not found
Returns null
No match found
Follows previously mentioned Invalid Phonenumber sequence diagram

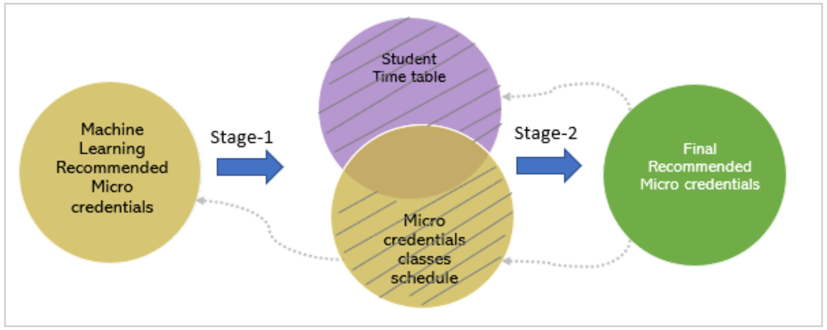
Recommendations pipeline

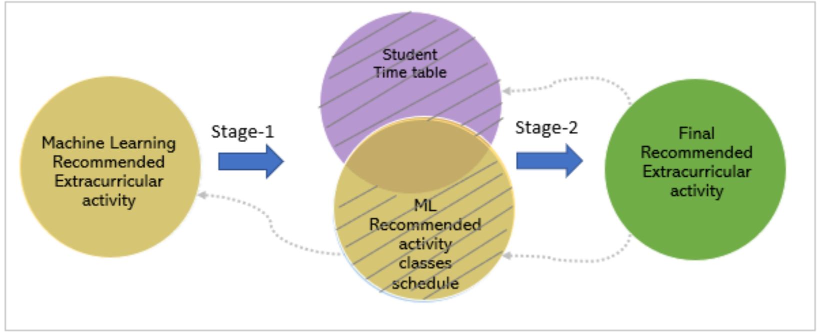
Currently AIDA uses 3 personalized recommendations namely

1. General course recommendation
2. Micro credential recommendation
3. Extracurricular activity recommendation

These recommendations are generated by Machine learning module and then gone through multiple stage of filtering before it goes to front end. Refer following diagram.


Machine Learning Recommended Courses
Student Eligible courses
Stage-1
Filtered courses
Stage-2
Student Time table
Filtered courses classes schedule
Stage-3
Final Recommended courses


Machine Learning Recommended Micro credentials
Stage-1
Student Time table
Micro credentials classes schedule
Stage-2
Final Recommended Micro credentials


Machine Learning Recommended Extracurricular activity
Stage-1
Student Time table
ML Recommended activity classes schedule
Stage-2
Final Recommended Extracurricular activity

Nudging

Currently AIDA got following 2 nudging

- **Fee Nudge**
  The fee nudge producer service searches through the invoice table upon being triggered and finds any invoices that are due in 14 days from the current date. It gathers theses invoices along with the corresponding student information (name and phone number) and sends it to the DFMS for it to send a text message to the student to warn them of an upcoming fee.

- **Extracurricular activity nudge**
  The extracurricular nudge producer service searches through the extracurricular table for an activity that starts within an hour from the current time or an inputted time. It gathers the student data for the currently registered students (name and phone number) along with the activity data and sends it to the DFMS for it to send a text message to the student to notify them of an upcoming extracurricular activity

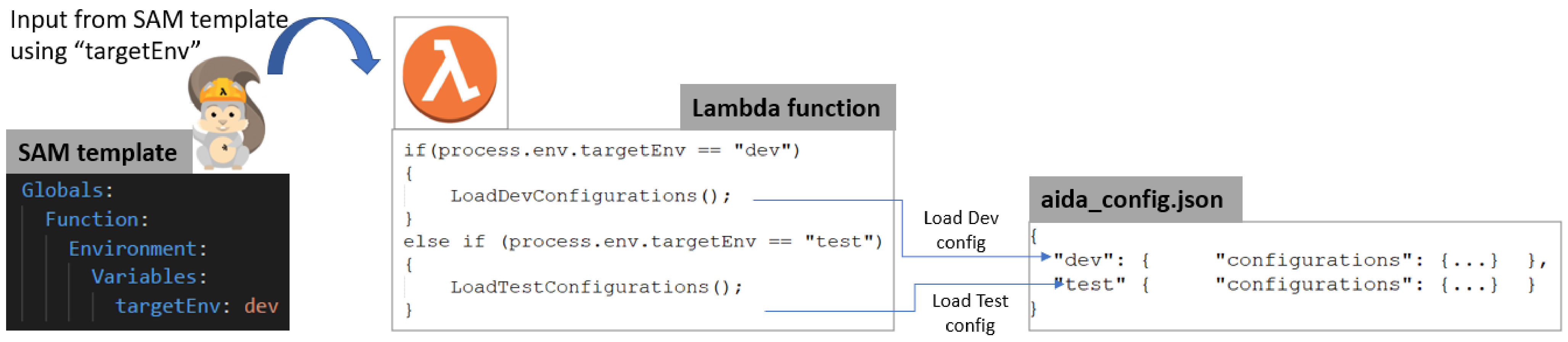
Deployment

Currently AIDA-POC system deployed in 2 environments namely

- Development – master branch of source code
- Test – demo branch of source code

Here goes design tactics to load variables depends upon the target environment at run time.

- Define global environment variable "targetEnv" SAM template.
- "targetEnv" is the variable for the configuration target environment. It contains either "dev" or "test".
- Be default all global SAM template variables are available to all lambda functions defined in that SAM template.
- So "targetEnv" is read in lambda functions by process.env.targetEnv.
- All variables relevant to development and test environment is configured in separate JSON file called aida_config.json
- Based on "targetEnv", corresponding environment variables are loaded at the run time.

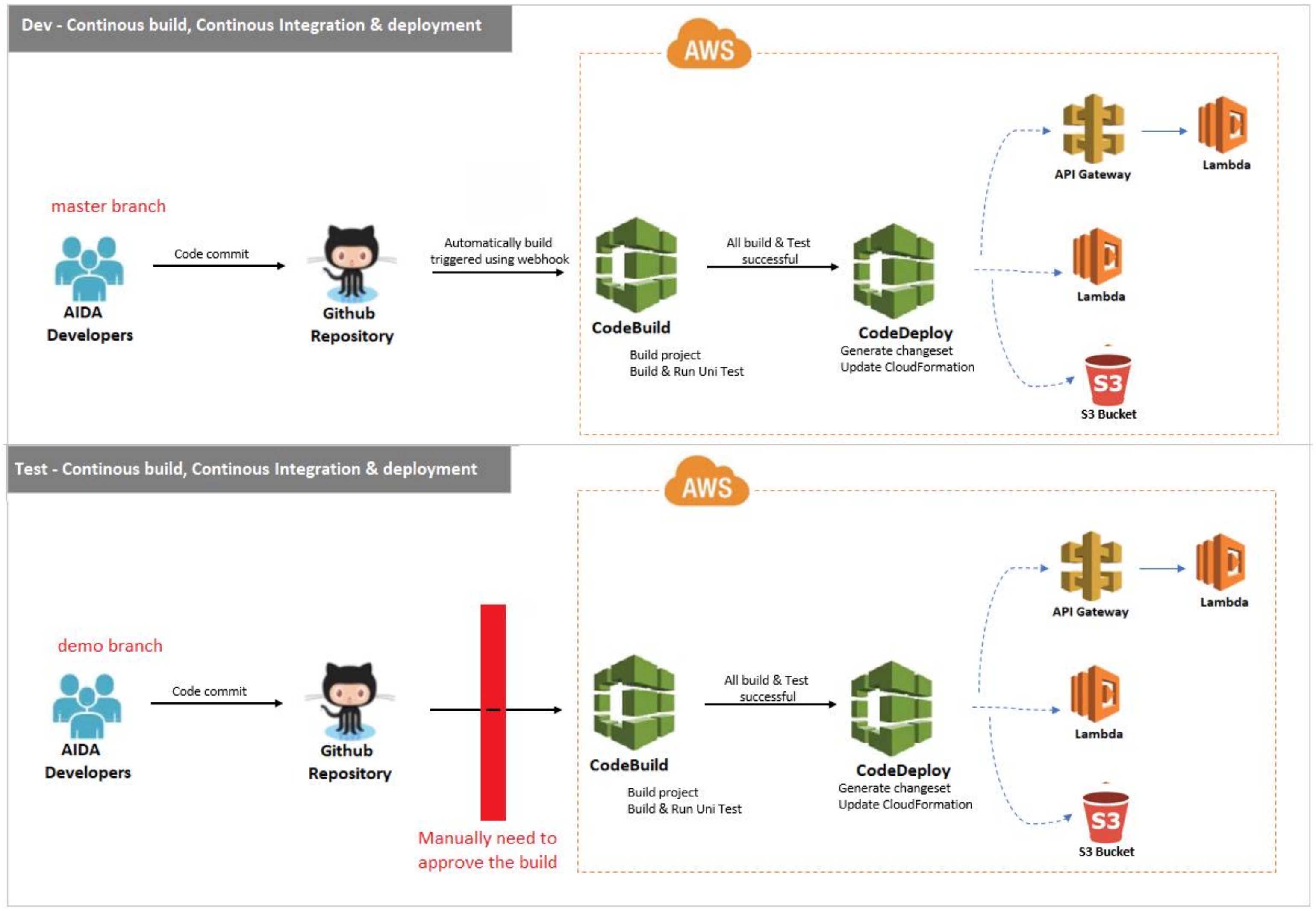


```
Globals:
  Function:
    Environment:
      Variables:
        targetEnv: dev
```

```
if(process.env.targetEnv == "dev")
{
    LoadDevConfigurations();
}
else if (process.env.targetEnv == "test")
{
    LoadTestConfigurations();
}
```

```
{
 "dev": {      "configurations": {...}  },
 "test" {      "configurations": {...}  }
}
```

Deployment process



END OF SLIDE