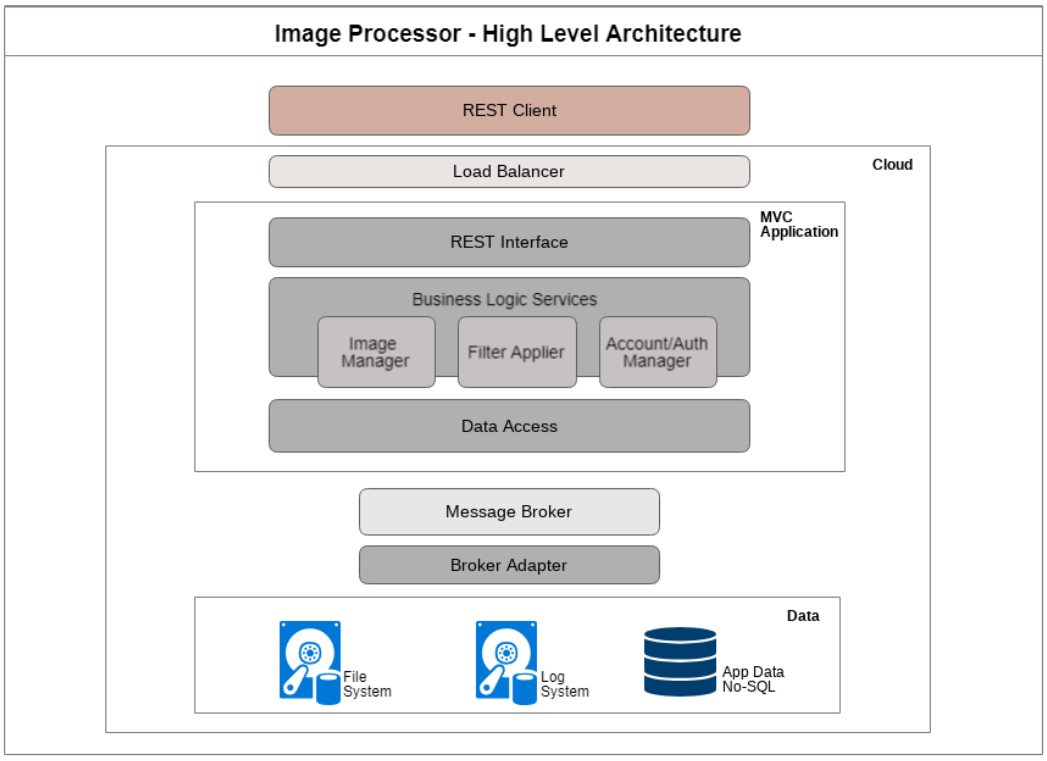
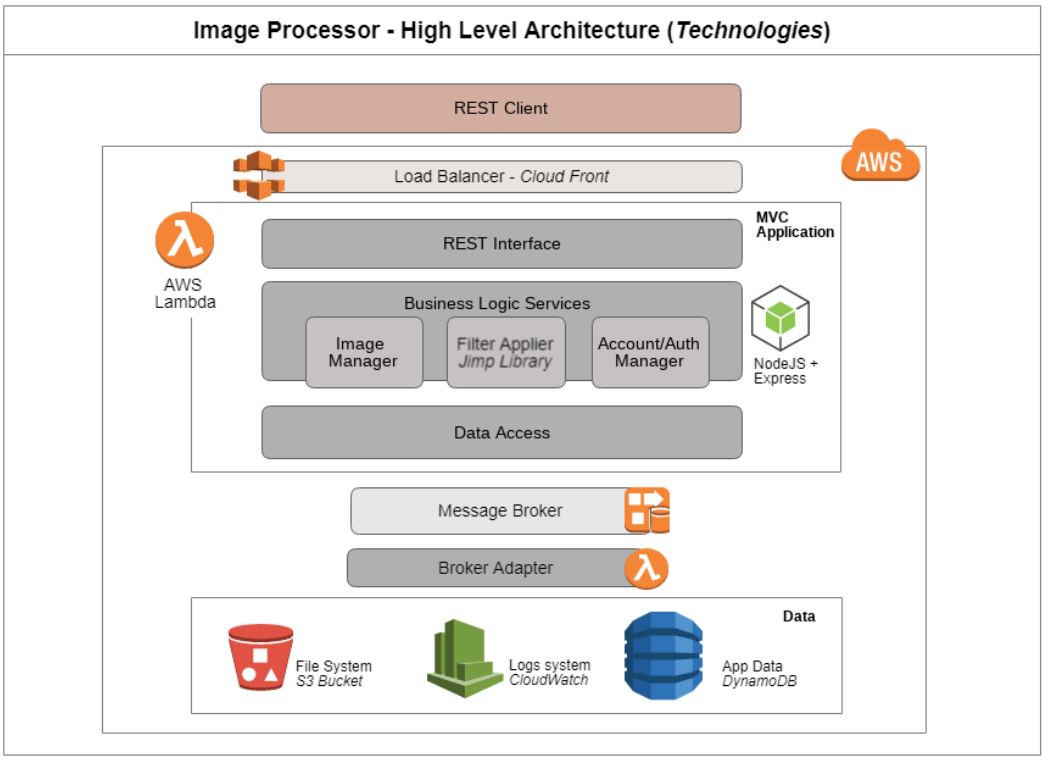
Individual Project – Image Processor

Natalia Manriquez - CPSC 5200 - March 10, 2019

High Level Overview





Architecture Diagram Details

Two diagrams are presented to depict how the system looks in a high level and with the choice of technologies.

General Details

- The API highest level architecture style is 3-tiers which includes the application, middleware, and data tiers.
- Model-View-Controller (MVC) is the architecture style that will be used to implement the application. Even though there is no a conventional “view” the response in JSON format is considered as the view.
- For accessing to a shared memory, a message broker will be used. Also, a piece called adaptor is needed to communicate with the data persistence. So that, multiple servers can communicate with multiple instances of database through the broker using a queue.
- A log system is used to keep track of all the actions on the system. In terms of security it will allow to identify suspicious users, files, or behaviors.
- The images will be stored on files in disk by the application

Technology Details

- The cloud technology to use will be AWS. This decision will drive most of the technology’s choices of the system.
- The load balancer functionalities will be supported using AWS CloudFront.
- There are two AWS lambda instances, one to implement the MVC application and another to implement the broker adapter.
- The No-SQL database to use will be DynamoDB, supported by AWS.
- NodeJS will be the programming language to implement the application and broker-adapter. It will be used with express framework and dynamoose library.
- The logging system will be incorporated using Amazon CloudWatch.
- To store the images the application will access an S3 bucket.

Discussions

Components and Connectors

Components	Description
REST Client	This is the consumer of the API, or the client system which works based on contract design
Load Balancer	Component that allows different requests to be assigned to the corresponding host. Needed, because several machines will run the App.
REST Interface	Is the “view” of the MVC application which will only vend the API responses.
Business Logic Services	The representation of business services functionality. It serves the role of controller for MVC.

Filter Applier	The service that apply image transformations in order. It will restrict the file format accepted.
User Account/Auth Manager	Allows a user to create an account and login to be authorized to use the system. The authentication will be supported with sessions and SHA-3 encryption.
Data Access	It is the object that interacts with the databases. It corresponds to the Model of MVC application.
Message Broker	Maintains the consistency of data between several servers and database instances.
Broker - Adapter	Connects the message broker technology to be consistent with the database language.
Log system	Stores the actions made by users, including data time, IP address. Supported by CloudWatch.

Connectors	**Description**
Express routes	Express serves as a connector to serve the interactions of the REST based API. It allows to exchange parameters, queries and responses between the server and the client.
AWS Lambda connections	Features of AWS lambda serve as connector, for example • One of them with the logging system, where lambda needs to establish a connection between the application and CloudWatch. • Another connection is generated between the application and the S3 bucket to access the images files.

Communication Protocols

- This application is thought to be implemented on a web browser, so it will use HTTP and TCP for the exchange of data in the network between client and server. Also, different servers will connect each other through the network using TCP/IP.
- For security reasons it will be necessary to support HTTPS protocol to encrypt the data. That's because it will use sensible information such as passwords and personal image files. Kinesis message broker also uses HTTPS protocol.
- It is important to mention that the exchange of data file images will be supported by base64 strings, so in case that this API is not used directly by a web browser it is expected that images be encoded using network byte order.

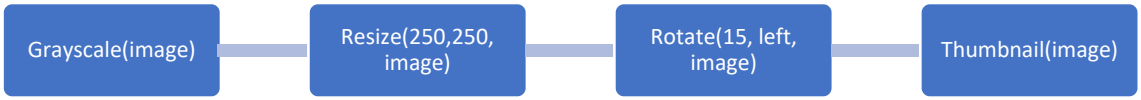
Justify the Architecture

- Architectural style 3-tiers: Is used to give flexibility and legibility to the development process. Also, separating these concerns will be useful to apply different cloud technologies at different layers. In addition, the scalability of the system will be improved by modifying the system by layers.
- Architectural Style MVC: The architectural style MVC was chosen because of the interaction between these components to separate the concerns and clarify the functionalities. Also it will limit the attributions from the client to the model. Even though there is no a real view with HTML documents, I will consider the json retrievals as the "view".
- The idea of using the cloud service will allow to deal well with scalability. I have chosen AWS because of the manageability, and the complete suit of applications that serves for different purposes such as load balancing and logging. Also, there are an intuitive interaction between all these components. However, if this application scale too much I should be aware that the price of storage in AWS could be higher that other technologies like Azure [1] and be prepared to support that cost.
- Include database: The database will store the images metadata, and user's data. I decided to store them to access the data easily and relate them to the users on the application. I also decided to use the database to enhance the service in the future. Thus, a user could keep the images to not upload them all the time or review all the processed images that he/she has. I decided to use a No-SQL database due to the faster retrieval, to improve the real-time effect. In addition, I included the database in the cloud system for security reasons, otherwise the data would be only on one server acting as a single point of failure.
- For security reasons the system will require login and should work with sessions, so that the users can be individualized as well as their submissions. The system is designed to only logged users can do actions on the API.
- NodeJs: Even though it is not an architecture component itself, I've mentioned to notice how will be managed the events from the user, in the application. It will be done with the NodeJs event loop that serves the requests by automatically queueing them.

Service Implementation Details

This service is thought to be implemented in different stages. In this first stage the API only offers user account register and image processing. Future implementations will focus on more features for the user images access, return all images of a user, delete them. Also, some features for reporting could be supported along with an administrator user features.

During the creation of the system, some patterns could be identified and though to be used. One of the them is the façade design pattern. The idea is to work with the cloud technology without compromising all the software pieces to it and abstract the direct interactions as much as possible. So, in case of a change of decision and migrate from AWS to AZURE localize the pieces to alter. Another design pattern to use is singleton for the database to open only one connection for the entire application. In terms of architecture the data broker architectural style was applied to the distributed system. Also, I could recognize pipe and filters. It can be applied for the image transformations. That's because after each service will process the image that another service processed before. For example, applying grayscale, rotation 15, thumbnail will apply changes one after the other to the same image.



Assumed Constraints

- Apply the same processing with the same parameters to an image doesn't have an effect after the second time.
- Received image is in base64 including the metadata with the format image.
- Resize receives the new width and height as parameters.
- Thumbnail applies scales to fit 250x250.
- Only the following image formats are supported: jpeg, png, bmp, tiff, gif.
- Only a logged user can use the system.
- Update a user (PUT) will replace only the values provided in the body request.

References

[1] Cloudhealthtech, 2019. A Look At Azure vs AWS Pricing In 2018-2019. Retrieved from https://www.cloudhealthtech.com/blog/azure-vs-aws-pricing