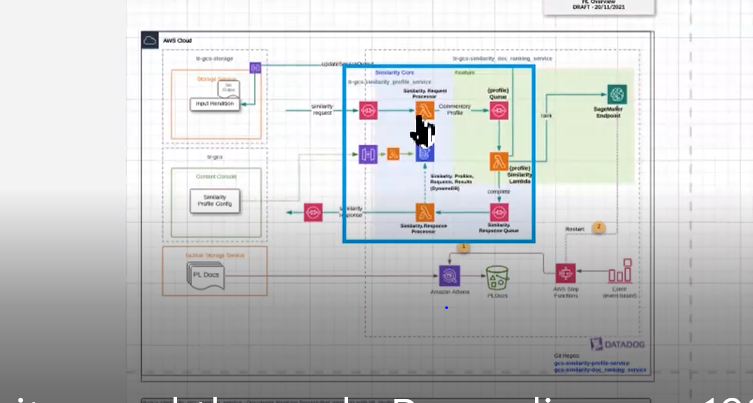
**Similarity Workflow:**

1. Acquisition workflow will update the massage into similarity request processor queue once acquisition workflow get completed.
2. Whenever massage is updated in request queue, Similarity request processor lambda function will get trigger and it will create request massage in dynamo DB.
3. We will get activity payload data from workflow service that will have profile details and it will process to Profile details to profile queue.
4. Sagemaker service will invoke based on the Profile Queue input that we have shared from core service.



There are two services that we are implementing in similarity,

1. **Compare HTML** - Not using Sagemaker Endpoint
2. **Document Ranking** - Using Sagemaker Endpoint

**Compare HTML:**

1. As per the discussion, we will be getting two renditionguids from acquisition workflow, one is Parent and Current renditionguids.



1. We are passing this two IDs to storage endpoint to get the HTML files. Once we get, compare those two HTML files, and find out added and removed.
2. We are preparing service output using added and removed links that we identified from the html files and storing into storage area.



1. Then sending Success or Failure response to similarity response queue.
2. Similarity response processor lambda will take the response from response queue and update status in request table.
3. Finally, it will send service response to similarity workflow queue.

**Document Ranking:**

1. We will get input document from the profile queue with document guid and details.
2. Using AWS glue jobs, we are processing PL Docs to sagemaker endpoint.
3. We are passing input document to sagemaker endpoint and it will give us the similar docs with rank.
4. Once we get similar docs based on that we are creating service output and storing into storage area.
5. Then sending service response to similarity response queue.
6. Similarity response processor will take the response from response queue and update in request table.
7. Finally, it will send service response to similarity workflow queue.

05-05-2022 Session Q & Ans:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. What is request queue will do.

Ans: Sim request queue will initiate when acquisition workflow is completed.

1. What data will store in dynamo DB.

Ans: Similarity request will get create in dynamo DB.

1. How many queues we will be having?

Ans: Two individual queues will be there for each service.

1. How we will get activity payload?

Ans: we will get it from workflow process, that is called from core side

1. What is going to store DB table once completed profile service?

Ans: Success or Failure status will get store in request table

1. How many tables we have in Dynamo DB?

Ans: Two tables, profile, request

1. What is diff between request and profile table?

Ans: Request table will store request queue details and profile table will store Profile details.

1. How many dead letter queues we have?

Ans: Each queue we have one dead letter queue. If request is failed it will go to the queue dead letter queue and this will trigger the error lander lambda that function will trigger the master dead letter queue.

1. Dose Compare HTML required Sagemaker service?

Ans: No

1. How request processor lambda will put msg into profile queue?

Ans: Using the queue name it will process the msg to profile queue

1. What sim response queue will have?

Ans: It will store profile status weather it is success or failure.

6-05-2022 Session Q and Ans:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Where we are putting service response either workflow queue or core queue?

Ans: We will be putting response in workflow queue

1. Do we need to prepare input to the sagemaker endpoint?

Ans: Yes, we have to prepare document input

1. What is that Glue jobs and PL Docs?

Ans: Running glue jobs to process data from PL Docs to sagemaker

1. What we are doing in the core service?

Ans: Once acquisition workflow completed, they will put msg into core similarity request queue.

1. How we are configuring lambda with queue?

Ans: In the request queue configuration we are enabling Lambda triggers

17.How we are running CloudFormation templates?

Ans: Using code pipeline we are running CloudFormation templates

18.What is SAM build?

Ans: The SAM build command processes your AWS SAM template file, application code, and any applicable language-specific files and dependencies.

19.Deployement purpose what we are using?

Ans: Git Hub we are using, As and when code is merged to main branch code pipeline will get trigger.

20.what is CloudFormation template?

Ans: AWS infrastructure will get create based on CloudFormation templates. when we deploy the code into code pipeline, it will get create the AWS infrastructure as per the CloudFormation templates.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

AWS: SAM: Serverless application model: [What is the AWS Serverless Application Model (AWS SAM)? - AWS Serverless Application Model (amazon.com)](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html)

1. The AWS Serverless Application Model (AWS SAM) is an open-source framework that you can use to build [serverless applications](https://aws.amazon.com/serverless/) on AWS.
2. A **serverless application** is a combination of Lambda functions, event sources, and other resources that work together to perform tasks.
3. Note that a serverless application is more than just a Lambda function—it can include additional resources such as APIs, databases, and event source mappings.
4. **AWS SAM template specification**. We can use this specification to define your serverless application. It provides you with a simple and clean syntax to describe the functions, APIs, permissions, configurations, and events that make up a serverless application.
5. **AWS SAM command line interface (AWS SAM CLI)**. We can use this tool to build serverless applications that are defined by AWS SAM templates. The CLI provides commands that enable you to verify that AWS SAM template files are written according to the specification, invoke Lambda functions locally, step-through debug Lambda functions, package and deploy serverless applications to the AWS Cloud.

**Benefits of using AWS SAM**

1. **Single-deployment configuration**. AWS SAM makes it easy to organize related components and resources and operate on a single stack. You can use AWS SAM to share configuration (such as memory and timeouts) between resources, and deploy all related resources together as a single, versioned entity.
2. **Extension of AWS CloudFormation**. Because AWS SAM is an extension of AWS CloudFormation, you get the reliable deployment capabilities of AWS CloudFormation. You can define resources by using AWS CloudFormation in your AWS SAM template. Also, you can use the full suite of resources, intrinsic functions, and other template features that are available in AWS CloudFormation.

1. **Built-in best practices**. You can use AWS SAM to define and deploy your infrastructure as config. This makes it possible for you to use and enforce best practices such as code reviews. Also, with a few lines of configuration, you can enable safe deployments through CodeDeploy, and can enable tracing by using AWS X-Ray.

1. **Local debugging and testing**. The AWS SAM CLI lets you locally build, test, and debug serverless applications that are defined by AWS SAM templates. The CLI provides a Lambda-like execution environment locally.
2. **Deep integration with development tools**. You can use AWS SAM with a suite of AWS tools for building serverless applications. You can discover new applications in the [AWS Serverless Application Repository](https://docs.aws.amazon.com/serverlessrepo/latest/devguide/). For authoring, testing, and debugging AWS SAM–based serverless applications, you can use the [AWS Cloud9 IDE](https://docs.aws.amazon.com/cloud9/latest/user-guide/). To build a deployment pipeline for your serverless applications, you can use [CodeBuild](https://docs.aws.amazon.com/codebuild/latest/userguide/), [CodeDeploy](https://docs.aws.amazon.com/codedeploy/latest/userguide/), and [CodePipeline](https://docs.aws.amazon.com/codepipeline/latest/userguide/).

DLQ: Dead Letter Queue: [Amazon SQS dead-letter queues - Amazon Simple Queue Service](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-dead-letter-queues.html)

1. Amazon SQS supports *dead-letter queues* (DLQ), which other queues (*source queues*) can target for messages that can't be processed (consumed) successfully.
2. Dead-letter queues are useful for debugging your application or messaging system because they let you isolate unconsumed messages to determine why their processing doesn't succeed.
3. A dead-letter queue is a queue that one or more source queues can use for messages that are not consumed successfully.
4. Amazon SQS does not create the dead-letter queue automatically. You must first create the queue before using it as a dead-letter queue.
5. The dead-letter queue of a FIFO queue must also be a FIFO queue. Similarly, the dead-letter queue of a standard queue must also be a standard queue.

### Standard queues:

[Standard queues](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/standard-queues.html) keep processing messages until the expiration of the retention period. This continuous processing of messages minimizes the chances of having your queue blocked by messages that can't be processed. Continuous message processing also provides faster recovery for your queue.

**FIFO queues:**

[FIFO queues](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/FIFO-queues.html) provide exactly-once processing by consuming messages in sequence from a message group. Thus, although the consumer can continue to retrieve ordered messages from another message group, the first message group remains unavailable until the message blocking the queue is processed successfully.

1. How we are handling errors in our similarity?

Ans: DLQ has been created to handle the exceptions while sending the request massages.

1. How we are handling errors in response queue?

Ans: Response DLQ are created to handle exceptions.

1. Does request queue will configure DQL automatically?

Ans: No, it won’t get configure automatically, we have to configure while creating the Queue

1. Does DLQ will configure automatically?

Ans: No, we must enable DLQ option while creating

1. What is flow error handling?

Ans: whenever request msg is failed to profile request queue it will send to profile request dead letter queue. when DLQ is updated error handler lambda will get trigger and send failure msg to master DLQ

1. How Error handler lambda will get trigger?

Ans: Whenever failed msg updated in DLQ, the error handler lambda will get trigger and send the msg to master dead letter queue.

1. How DLQ will know to trigger the error handler lambda?

Ans: we must configure error handler lambda in DLQ

1. Once SQS templates are created to update the changes in AWS what you need to do?

Ans: We must merge the changes to master branch and run code pipeline.

1. What is SAM?

Ans: SAM is a serverless application model, is an open-source framework that you can use to build [serverless applications](https://aws.amazon.com/serverless/) on AWS.