**Title:** AWS Digital Marketing Email Subscription Platform

**Author:** Shobharani Polasa

**Abstract**

This thesis presents the design and implementation of a robust AWS-based digital marketing platform tailored for managing email subscriptions. Leveraging various AWS services, this platform offers a scalable, secure, and cost-effective solution that automates the capture, storage, and processing of user subscriptions through digital marketing campaigns. This project not only demonstrates the application of advanced cloud technologies in digital marketing but also showcases the integration of these services to optimize marketing strategies.

**Table of Contents**

1. Introduction
2. Literature Review
3. Methodology
4. System Architecture and Design
5. Implementation
6. Testing and Validation
7. Results and Performance Analysis
8. Discussion
9. Conclusion
10. References
11. Appendices

**1. Introduction**

The project addresses the need for sophisticated digital marketing tools that integrate seamlessly with cloud technology to enhance user engagement and optimize resource management. The AWS Digital Marketing Email Subscription Platform is designed to facilitate efficient email marketing campaigns through automated subscription management.

**2. Literature Review**

Research in digital marketing and cloud computing forms the theoretical basis for this project. Prior studies have emphasized the impact of cloud technologies in scaling and securing digital marketing solutions. This review covers various approaches to integrating AWS services in different sectors, particularly focusing on S3 for static hosting, Lambda for serverless computing, and DynamoDB for NoSQL data storage.

**3. Methodology**

An agile project management approach guided the development, with iterative sprints used to plan, execute, and review incremental deliverables. AWS CloudFormation was used to script the infrastructure setup, ensuring replicable and consistent deployments across environments.

**4. System Architecture and Design**

**AWS Services Utilized:**

* **Amazon S3:** Hosts static website files (index.html, style.css, main.js) ensuring fast content delivery.
* **AWS Lambda:** Executes backend logic in response to API requests, interacting with DynamoDB to store subscriber data.
* **Amazon API Gateway:** Manages secured HTTP endpoints for form submissions, acting as a conduit between the frontend and Lambda functions.
* **Amazon DynamoDB:** Stores subscription data with high availability and performance.
* **Amazon SES:** Automates the sending of emails to subscribers.

**Architectural Diagrams:** Detailed visual representations are provided to illustrate the flow of data through these services.

**5. Implementation**

**Step 1: Planning Your Project**

* **Objective**: Build a landing page for a digital marketing campaign that captures lead information.
* **Key AWS Services**:
  + Amazon S3 for hosting static website content (HTML, CSS, JS).
  + AWS Lambda for running backend code without provisioning servers.
  + Amazon DynamoDB for storing lead information.
  + Amazon SES for sending email notifications.

**Step 2: Setting Up Your AWS Account**

* Sign up or log into your AWS account.
* Ensure you have the necessary permissions to create and manage resources in S3, DynamoDB, Lambda, and SES.

**Step 3: Creating and Configuring the S3 Bucket**

1. **Create a New Bucket** in S3 for your website files.
   * Disable public access blocking settings.
   * Enable static website hosting.
2. **Upload Your Website Files** (HTML, CSS, JavaScript).
3. **Update Bucket Policy** to allow public read access to the website files.

**Step 4: Setting Up DynamoDB**

1. **Create a New Table** in DynamoDB.
   * Table name: Leads
   * Partition key: email (String)

**Step 5: Setting Up AWS Lambda**

1. **Create a New Lambda Function**.
   * Runtime: Node.js
   * Trigger: API Gateway
   * Permissions: Create or use an existing role that has permissions to access DynamoDB and SES.
2. **Implement Function Logic** to insert data into DynamoDB and send email notification via SES.
3. **Deploy the Lambda Function** and note the API Gateway endpoint URL.

**Step 6: Integrating Lambda with Your Landing Page**

* Modify your form submission code to POST data to your Lambda function's API Gateway endpoint.

**Setting Up API Gateway**

**Create a New REST API:**

1. **Navigate to API Gateway**: In the AWS Management Console, find and select the API Gateway service.
2. **Create API**: Choose "REST API" and click "Build". Select "New API" and provide a name, e.g., EmailSubscriptionAPI.
3. **Create Resource**: With your API selected, click on "Actions" and choose "Create Resource". Name it accordingly, e.g., subscription.
4. **Create Method**: Select the newly created resource, click on "Actions", and choose "Create Method". Select "POST" and click the checkmark.
5. **Configure Method**: Select "Lambda Function" for the integration type, and enter the name of your Lambda function. Save and grant permissions when prompted.

**Deploy API:**

1. **Create Deployment**: Click on "Actions" and select "Deploy API". Create a new stage, e.g., prod, and deploy your API.
2. **Note the Invocation URL**: After deployment, you'll be given an invocation URL. This is what you'll use as the endpoint for your subscription form's action attribute.

**API Gateway**

For API Gateway, you can check the resource and path details as follows:

1. **Navigate to the API Gateway Console**:
   * Open the AWS Management Console.
   * Select "Services" and go to "API Gateway" under the Networking & Content Delivery category.
2. **Select Your API**:
   * You'll see a list of your APIs. Click on the API name for which you want to check the resource and path details.
3. **Resources Tab**:
   * By default, you'll be taken to the "Resources" tab, where you can see a hierarchical list of your API's resources and methods.
   * Here, you can click on any resource to see its path relative to the API's base URL. Resources are typically represented as parts of the URL path (e.g., /users, /products).
4. **Method Details**:
   * Clicking on a method (GET, POST, etc.) attached to a resource will show you detailed information about the method, including its configuration and the Lambda function it's connected to (if any).
5. **Stage and Invoke URL**:
   * To find the complete path you can access an API at, navigate to the "Stages" section on the left sidebar.
   * Select a stage (e.g., prod, dev) to view the stage's details, including the Invoke URL at the top. You can append your resource paths to this base URL to form the complete URL used for invoking the API.

**Checking Path Parameters and Query Strings**

* When configuring a method in API Gateway, you can specify path parameters and query strings under the "Method Request" section. This can be useful for configuring how your API handles different parts of a URL path or query string parameters for more dynamic behavior.

**Assign an IAM Role to the Lambda Function**

1. **Open the AWS IAM Console**: Go to the IAM dashboard within the AWS Management Console.
2. **Create a New Role**:
   * Click on "Roles" on the sidebar, then "Create role".
   * Choose AWS service as the type of trusted entity and select "Lambda" as the service that will use this role.
   * Click "Next: Permissions".
3. **Attach Policies**:
   * Search for and attach the AmazonDynamoDBFullAccess and AmazonSESFullAccess policies. This grants your Lambda function permissions to interact with DynamoDB and SES.
   * Click "Next: Tags" (optional) and "Next: Review".
4. **Review and Create Role**:
   * Name your role, e.g., LambdaDynamoDBSESRole, and provide a description.
   * Click "Create role".
5. **Assign Role to Lambda Function**:
   * Go back to your Lambda function in the AWS Lambda Console.
   * Under the "Configuration" tab, find the "Execution role" section and edit it to select the role you just created.

**Set Up API Gateway**

1. **Create a New REST API**:
   * Navigate to the API Gateway console.
   * Choose "REST API" and click "Build".
   * Select "New API", give it a name, and then click "Create API".
2. **Create a New Resource**:
   * Select your API's root resource and click "Create Resource".
   * Name the resource, e.g., subscription, and click "Create Resource".
3. **Create a POST Method**:
   * Select the resource you just created.
   * Choose "Create Method" and select "POST" from the dropdown menu.
   * Configure the method to use Lambda Function integration, selecting the region where your Lambda function is located and entering the name of your Lambda function.
4. **Deploy the API**:
   * In the "Actions" dropdown, select "Deploy API".
   * Create a new Deployment stage, e.g., prod, and note the "Invoke URL" provided after deployment.

**Integrating the Subscription Form with Your Backend**

* **Update Your script.js**:
  + Replace 'YOUR\_API\_GATEWAY\_ENDPOINT' in the fetch call with the actual invocation URL of your deployed API Gateway endpoint.

**Configuring Amazon SES for Email Communications**

1. **Verify Email Address or Domain**:
   * In the SES console, verify the email address or domain you intend to use as the "From" address for sending confirmation emails.
2. **Update IAM Role if Necessary**:
   * Make sure the IAM role assigned to your Lambda function has AmazonSESFullAccess. This should already be covered if you followed the role creation steps above.

**Step 7: Setting Up Amazon SES**

1. **Verify Your Email Address** in SES to send and receive emails.
2. Ensure your Lambda function IAM role has permissions to send emails via SES.

**Step 8: Deployment and Testing**

* **Deploy your updated website** to S3.
* **Test the form** on your landing page to ensure leads are captured in DynamoDB and notification emails are sent.

**Step 9: Monitoring and Maintenance**

* Monitor AWS service usage via the AWS Management Console.
* Regularly check the performance and deliverability of your emails.

Setting up the IAM:

1. **IAM Console**: Navigate to the IAM console within the AWS Management Console.
2. **Find Your IAM User/Role**: Locate the IAM user or role you're using to access the S3 bucket.
3. **Permissions**: Check the permissions attached to your user or role. Ensure you have the s3:PutBucketPolicy permission. This permission is required to edit bucket policies.
4. **Update Policy if Necessary**: If you don't have the required permission, you may need to attach a policy granting you the s3:PutBucketPolicy permission. This might require assistance from an AWS account administrator if you don't have permissions to modify your own IAM roles or policies.

Source code files and their roles are extensively documented:

* **index.html:** Main landing page with subscription form.
* **style.css:** Contains CSS rules for the website's design.
* **main.js:** Manages form submissions, making AJAX calls to the API Gateway.
* **lambda.js:** Contains AWS Lambda function code that processes the incoming data and interacts with DynamoDB and SES.

Configuration details include IAM roles for secure AWS service interaction, CORS settings on API Gateway for cross-origin resource sharing, and environment variables in Lambda for managing sensitive data securely.

**6. Testing and Validation**

Comprehensive testing strategies encompassed unit tests, integration tests, and UI tests to ensure the robustness of the platform. Automated scripts using AWS CodeBuild and CodePipeline were implemented to execute these tests upon code changes.

**Considerations**

* **Be Cautious with Public Access**: Allowing public access to your S3 bucket can expose your data to anyone on the internet. Use public access only when necessary and understand the implications.
* **Security Best Practices**: Always adhere to AWS security best practices, especially when modifying permissions and access policies.

**Troubleshooting**

1. **Logs and Errors**:
   * Check CloudWatch Logs for any errors returned by your Lambda function or API Gateway. This can provide insights into issues like permission errors, code exceptions, or integration problems.
2. **Permissions**:
   * Common issues often relate to IAM permissions. Verify that the execution role associated with your Lambda function has the necessary permissions for DynamoDB and SES.
3. **Email Verification**:
   * Ensure that the email address used to send confirmation emails is verified in SES. Unverified emails will not be allowed to send emails.
4. **API Gateway Deployment**:
   * Remember to redeploy your API Gateway after making changes to its configuration or linked Lambda function to ensure the latest changes are active.

**Additional Considerations**

* **IAM Role for Lambda**: Ensure that the IAM role assigned to your Lambda function has the necessary permissions to access SES for sending emails and DynamoDB for inserting records. You may need to attach policies like AmazonSESFullAccess and AmazonDynamoDBFullAccess. It's best practice to scope these permissions down to the minimum required for your application.
* **Test Your Setup**: After setting up SES and DynamoDB, and writing your Lambda function, test the entire flow to ensure everything works as expected. Use the Lambda console to create test events, and check DynamoDB and your email inbox for the results.
* **Secure Your Application**: Consider using environment variables in Lambda to store sensitive information such as email addresses. Also, review your API Gateway settings to ensure that your endpoint is secure, considering aspects like rate limiting, CORS, and API keys if necessary.
* **Monitor and Log**: Utilize AWS CloudWatch for monitoring and logging your Lambda function executions and API calls. This will help in debugging and understanding the performance of your application.

**7. Results and Performance Analysis**

The deployment of the platform showed excellent scalability and reliability. Metrics such as latency, error rates, and throughput were monitored using Amazon CloudWatch, providing insights into the system's operational status and helping fine-tune its performance.

**8. Discussion**

This section delves into the scalability challenges encountered during peak loads and the strategies employed to mitigate them, such as adjusting the read/write capacity of DynamoDB and scaling policies for Lambda.

**Challenges**: The error message "An error occurred while subscribing. Please try again." suggests that there's an issue when the front end tries to communicate with the backend services, possibly the AWS Lambda function through API Gateway. Here are steps to diagnose and resolve the issue:

1. **Check the Browser Console for Errors:**
   * Open the web page in a browser.
   * Right-click anywhere on the page and select "Inspect" or "Inspect Element" to open the developer tools.
   * Go to the "Console" tab to see any JavaScript errors that may have occurred during the subscription request.
2. **API Gateway:**
   * Go to the API Gateway service in the AWS Management Console.
   * Click on your API and go to the "Logs/Tracing" tab to make sure that CloudWatch logging is enabled for your API stages.
   * After trying to subscribe again on your site, check the CloudWatch logs for detailed error messages.
3. **Check the Lambda Function:**
   * Go to the Lambda service in the AWS Management Console.
   * Find your subscription function and look at the monitoring and logs to identify any errors during execution.
   * You can also test the Lambda function directly in the AWS Console by creating a test event that mimics the API Gateway's JSON structure for invoking the Lambda function.
4. **CORS Configuration:**
   * If your API Gateway and front end are on different domains, ensure CORS is enabled on your API Gateway.
   * In API Gateway, select your API resource and method, then click on "Actions" and "Enable CORS". Enter the necessary headers and methods, then deploy your API.
5. **Network Issues:**
   * Check for any network issues that might be preventing communication between AWS Amplify and API Gateway.
   * Confirm that there are no VPC configurations or security groups that might be blocking access.
6. **IAM Role and Execution Policy:**
   * Ensure the Lambda function's execution role has the necessary permissions for accessing the resources it needs, such as DynamoDB and SES.
   * Go to IAM and review the policies attached to the role to ensure they are correctly configured.
7. **Deployment and Configuration:**
   * Make sure that your API Gateway changes are deployed to the correct stage.
   * Verify that the environment variables (if any) and resource configurations in Lambda are correct.
8. **Amazon SES Email Verification:**
   * Make sure the email addresses used for sending emails via SES are verified.
   * Go to the SES dashboard in the AWS Management Console to verify or add new email addresses.

**9. Conclusion:** The project successfully demonstrates how AWS services can be orchestrated to build a comprehensive digital marketing solution. Future enhancements could include integrating machine learning to predict user behavior and further personalizing email content.

**10. References:** AWS Documentation

* Journals on Digital Marketing and Cloud Computing
* Recent academic papers on marketing automation