WHITEPAPER

BudgetWise: Your Financial Advisor

The following document is a comprehensive whitepaper on the BudgetWise application. This document highlights the following:

A. Executive Summary

BudgetWise aims to help with personal finance management by offering a comprehensive cloud-based solution that simplifies budgeting and expense tracking for users. Traditional methods often involve cumbersome manual processes, leading to disorganization and uninformed financial decisions. By leveraging AWS services, BudgetWise provides an intuitive platform equipped with features such as expense management, budget tracking, visual analytics, and timely notifications. This whitepaper outlines the problem statement, our proposed solution, the technologies utilized, associated costs, recommendations, challenges, and the anticipated benefits of BudgetWise.

B. Problem Statement

Traditional methods of budgeting and expense tracking are time-consuming, error-prone, and lack real-time insights. Users face challenges in staying organized, making informed financial decisions, and planning for the future due to the limitations of manual processes and outdated tools. Furthermore, traditional budgeting tools often lack the functionality and flexibility required to meet the diverse needs and preferences of users.

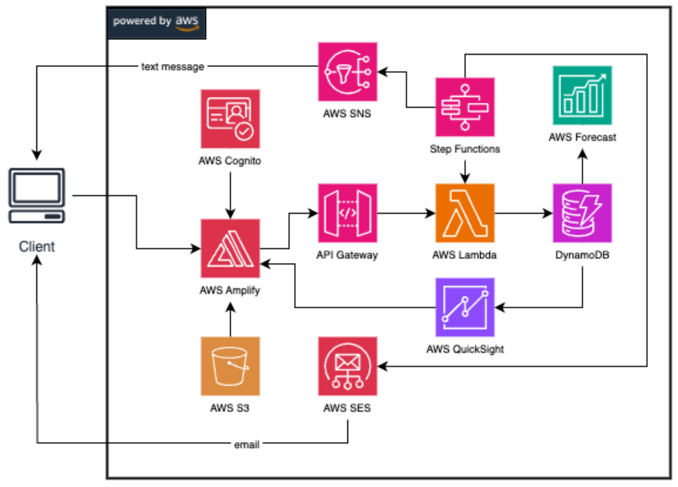
In light of these challenges, there is a pressing need for a modern and comprehensive solution that leverages the power of cloud technology to streamline personal finance management. Such a solution should offer users a user-friendly platform equipped with advanced features such as automated expense tracking, dynamic budgeting capabilities, visual analytics, and timely notifications.

C. Architected Solution

a. Proposed Solution

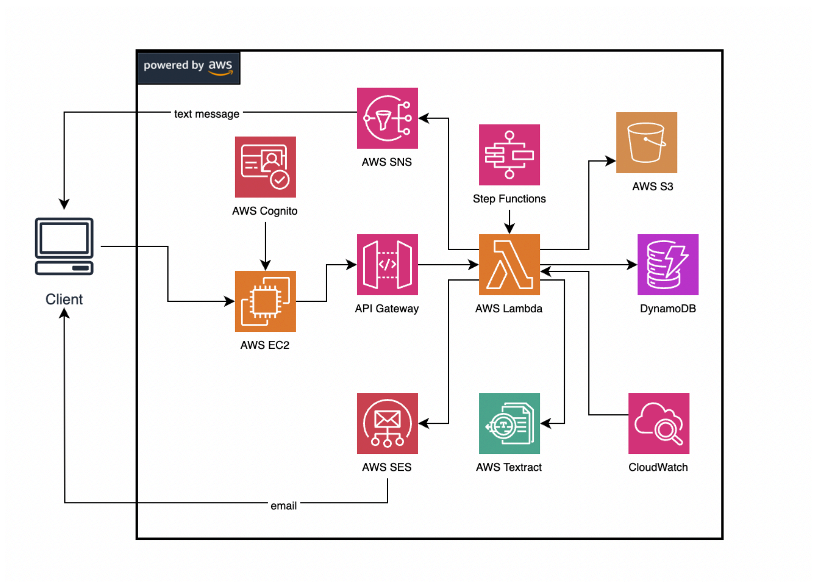
The proposed solution architecture for BudgetWise encompassed AWS services tailored to deliver a seamless and efficient personal finance management platform. Leveraging **AWS Cognito** for secure user authentication, **AWS Amplify** for hosting the frontend (React) of the application, and **AWS DynamoDB** as the database for storing application data, our aim was to provide users with a robust and scalable infrastructure. Additionally, **AWS Lambda** served as the serverless backend of the application, while **AWS StepFunction** facilitated error handling and orchestration of Lambda functions. The integration of **AWS API Gateway** ensured smooth communication between frontend and backend components, while **AWS QuickSight** enabled the creation of visualization analytics embedded within the application. Furthermore, **AWS Forecast** was planned to forecast spending based on past user data, while **AWS SNS** would alert users via SMS on budget exceeds, and **AWS SES** would provide users with monthly spending reports via email.

While my proposed solution architecture remained subject to potential adjustments, the overarching goal of empowering users to manage their finances effectively remained unchanged.



b. Current Solution

After conducting spikes and tutorials within the team, I refined the solution architecture, occasionally substituting certain services to better align with the overall architecture. As a result, the current solution architecture includes **AWS EC2** with SG, which hosts the user interface (React) of the application instead of AWS Amplify. Additionally, **AWS Textract** with S3 enables OCR functionality for scanning and storing receipts, enhancing expense tracking capabilities. CloudWatch has been integrated to implement rules for cron-jobs, facilitating daily total calculation for forecasting purposes. During automation, I also had to include and work with **AWS IAM** roles and policies. It's worth noting that most of the other services that were proposed were successfully integrated into the architecture of the BudgetWise solution. However, certain services were dropped (AWS QuickSight and Forecast) due to finding better alternatives, which will be covered in detail in the next section



c. Changes/Pivots – Why?

The changes made to the solution architecture were primarily driven by several factors, including cost considerations, integration challenges, and comparisons with alternative options. Cost played a significant role in the decision-making process, prompting me to explore more cost-effective alternatives to certain services. By prioritizing these factors, I aimed to ensure the efficiency, effectiveness, and sustainability of the BudgetWise solution architecture. The major changes/pivots in the architecture were:

- **Pivot from AWS Forecast to Python function**: After a spike, I concluded that implementing a forecasting solution with Forecast was not feasible due to the high cost and time that was required for training the data. Hence, I decided to pivot to a Lambda hosted Python method that implemented basic forecasting functionality.

- **Pivot from AWS QuickSight to HighCharts**: Due to QuickSight's limited 30-day free trial, it was difficult to maintain continuous development through the course of the project. Noticing that Highcharts provided similar capabilities and analytics (while also being open-source), I decided to switch to Highcharts.

- **Pivot from AWS Amplify to EC2**: AWS Amplify required an open GitHub repository to provision the frontend. I opted for using EC2, hosting a docker image of the application,

D. Benefits of cloud technologies

Utilizing cloud infrastructure for the BudgetWise project offers many benefits that significantly enhance its functionality and efficiency. Firstly, scalability is paramount, allowing the application to seamlessly accommodate fluctuating user demands without compromising performance. All of the services are configured to auto-scale based on traffic. Additionally, leveraging tools like Terraform simplified deployment and management processes, ensuring consistency and reproducibility across environments. The robust security features provided by cloud ensured the protection of sensitive financial data. Furthermore, the seamless data synchronization capabilities ensured by cloud platforms facilitated real-time updates and collaboration, enhancing user experience and productivity. The overall automation of processes minimized the need for manual intervention, reducing human error and streamlining operations. Hence, using cloud empowers BudgetWise to deliver a reliable, scalable, secure, and user-friendly solution for effective personal finance management.

E. Estimated Cost:

I approached the solution with a keen awareness of cost considerations that came about from leveraging cloud services. Although most of the services fell under "free tier" during the development phase, the below table shows the estimated cost and resources if one were to run this application at a production level with real-time users:

Service	**Assumption**	**Monthly Cost***
EC2 (t4g.nano)	24/7 usage	$3.024
Lambda	100 requests/day	$0.006
DynamoDB	100GB	$25

Cognito		$0.0055
Textract	5 receipts/day	$0.225
CloudWatch		$0.00001
SNS	10 triggers/month	$0.1 per user

** These costs are roughly calculated and will increase if there are more users due to provisioning of bigger compute power.*

The overall cost is estimated to be a monthly fixed cost of about $28 (using basic EC2) and a per user monthly cost of about $0.4.

F. Recommendations:

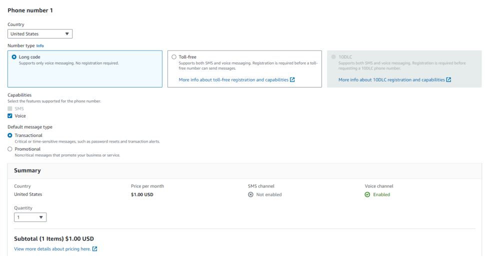
In light of the project experience, I recommend adopting a structured approach to Terraform file organization, emphasizing the separation of files based on specific services. This modularization enhances maintainability and scalability while facilitating easier troubleshooting and updates. Additionally, doubling down on Infrastructure as Code (IaC) automation minimizes manual intervention, ensuring consistency and reliability across deployments. Lastly, consideration of offloading compute power by utilizing Docker images for the frontend, enables efficient resource utilization and enhances performance scalability. These recommendations collectively contribute to a more streamlined, scalable, and cost-effective deployment process for the BudgetWise application.

G. Challenges

During the development phase of the project, I encountered a few challenges that required careful consideration and strategic adjustments.. Here are some of the challenges which I encountered during the development:

- **Terraform for automation**: The implementation of Terraform brought about a considerable learning curve, particularly regarding access control and configuration management. Navigating the intricacies of access permissions and configurations within Terraform required careful consideration and experimentation. Through this process, I gained valuable expertise in managing access policies and fine-tuning configurations.
- **AMI for EC2**: It is only after I finished building and deploying the frontend docker image did I notice that AWS does not provide an AMI for ARM architecture (encountered by default when built with Mac OS) that comes under free-tier. Hence, I made a collaborative call to go ahead with t4g.nano.

- **AWS SNS for text messages**: This challenge stemmed from a relatively new requirement introduced by AWS, which mandated different provisioning procedures for US-based numbers compared to other regions. Specifically, AWS required the setup of an origination number for long codes, which incurred a cost of $1, while the requirement for a toll-free number demanded a $2 fee. As a result, I conducted the testing with an Indian phone number to mitigate expenses and ensure the smooth functioning of the code within the Sandbox environment. This approach allowed me to validate the functionality of the SNS integration while minimizing costs during the testing phase.



Phone number 1

Country: United States

Number type Info

- [✓ Long code] Supports only voice messaging. No registration required.
- Toll-free: Supports both SMS and voice messaging. Registration is required before a toll-free number can send messages. More info about toll-free registration and capabilities
- 10DLC: Supports both SMS and voice messaging. Registration is required before requesting a 10DLC phone number. More info about 10DLC registration and capabilities

Capabilities
Select the features supported for the phone number.
- SMS
- [✓ Voice]

Default message type
- [✓ Transactional] Critical or time-sensitive messages, such as password resets and transaction alerts.
- Promotional: Noncritical messages that promote your business or service.

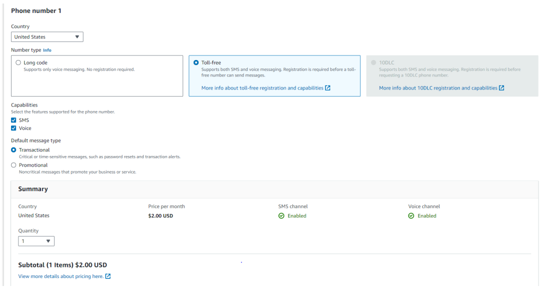
Summary

Country	Price per month	SMS channel	Voice channel
United States	$1.00 USD	Not enabled	Enabled

Quantity: 1

Subtotal (1 Items) $1.00 USD

View more details about pricing here.



Phone number 1

Country: United States

Number type Info

- Long code: Supports only voice messaging. No registration required.
- [✓ Toll-free] Supports both SMS and voice messaging. Registration is required before a toll-free number can send messages. More info about toll-free registration and capabilities
- 10DLC: Supports both SMS and voice messaging. Registration is required before requesting a 10DLC phone number. More info about 10DLC registration and capabilities

Capabilities
Select the features supported for the phone number.
- [✓ SMS]
- [✓ Voice]

Default message type
- [✓ Transactional] Critical or time-sensitive messages, such as password resets and transaction alerts.
- Promotional: Noncritical messages that promote your business or service.

Summary

Country	Price per month	SMS channel	Voice channel
United States	$2.00 USD	Enabled	Enabled

Quantity: 1

Subtotal (1 Items) $2.00 USD

View more details about pricing here.

- **AWS SES for emails**: SES requires users to verify their email addresses before they can send emails, a security measure aimed at preventing misuse of the service. While Terraform provides robust infrastructure as code capabilities for automating the deployment of AWS resources, it does not offer a direct means of bypassing the email verification process in the SES Sandbox environment. This presented a challenge for me as I sought to automate the deployment of SES resources without manual intervention.

H. User Guide

To use the application, I created a comprehensive User Guide which can be found in our code repository. This document highlights the following:

- Prerequisites for Demo
- How to run the demo? (Provisioning)
- How to use the application?
- How to clean-up after the demo? (Teardown)

I. Conclusion

In short, BudgetWise transforms personal finance planning with an easy-to-use system, leveraging the latest AWS advancements. This document showcases the framework and how BudgetWise offers forward-looking financial strategies and instant insights, surpassing traditional budgeting methods.