CASH (Cash And Savings Helper)

A0087087R Leong Zhong Wei Nicholas
A0094501M Loo Choon Boon
A0228569M Ngan Zisheng Nigel
A0225203X Tan Hui Min

Executive Summary

When asked to make crucial life decisions, we are often advised to think what a future version of yourself 10, 30 or even 50 years down the road would have done.

The CASH web app aims to help Singaporeans visualize what the future version of yourself would look like financially. Coupled with high levels of interactivity and dashboard capabilities, this web app aims to empower Singaporeans to forecast their financials and realize their financial goals.

As this is a cloud computing module, we will also explore the various cloud computing tools we used to make CASH a reality.

1. Business Case Identification

1.1 Background

As the saying goes, “It’s never too early to plan for retirement”. According to a survey conducted during the “circuit breaker”, two in three working Singaporeans do not have sufficient savings to last them beyond half a year if they were to lose their jobs (OCBC Bank, 2020). A study showed that 4 in 5 Singaporeans underestimated the amount they require for retirement by 32% (OCBC Bank, 2021) in spite of Singapore’s lauded Central Provident Fund system which contributes significantly to retirement planning. Thankfully, research has shown how more can be done to address this problem.

In an early examination of the relationship between the future self and intertemporal choice, researchers measured the extent to which adolescents were able to imbue future events in their lives with a sense of reality by asking them to list a number of typical life events in chronological order. The more these participants perceived their futures to follow a realistic and vivid course, the more likely they were to delay gratification on a subsequent choice task. (Klineberg SL,1968).

We took the problem statement of Singaporeans not having enough to retire and solutioned it with promising research of allowing individuals to be more attuned to their future self leading

to good decisions. We then embarked on a solution based on helping Singaporeans visualize what their financial future would materialize.

1.2 Competitors

There are a few similar existing products that cater to one's personal finance.

Personal Finance	**Pros compared to CASH**	**Cons compared to CASH**
SGFinDex, Seedly App	There are direct data integrations into one's bank accounts, CPF for an overall view of your financial health.	Rubbish in, rubbish out. Not all financial data points are captured leading to wrong recommendations. For e.g. money in robo advisors, insurance or niche markets like cryptocurrencies. The sign-up process is tedious and time consuming requiring lots of setup to get started. Recommendations are based on historical input data without the interactive ability to model based on various user choices.
CPF Retirement Calculator	It is extremely comprehensive and suitable for users that already have all the available information on hand.	The interface is not as user friendly as the user takes up to 10 minutes filling numerous questions which may be irrelevant to get an output. Outputs are also text based which is not intuitive to users.
Financial Advisors	This option is suitable if the user has enough wealth to outsource your personal finance needs to an experienced, professional third party.	This option is not suitable for the majority of people without high net worth. There are costs associated with hiring a financial advisor. Furthermore they may not be incentivized to provide the best recommendations since they earn commissions.
D.I.Y (Do it Yourself)	This is a suitable option if users are naturally interested	Not everyone has the financial knowledge, time and excel know-hows to keep up to date with their personal finances.

In essence, we see CASH as more of a complementary application to the existing ecosystem. For example for users that D.I.Y and already track their budgeting very closely on excel, they can input their data easily into CASH for more insights. Because of CASH's ease of use and quick results, it may be used as a starting point to get people incentivized to be vested in their financial interest and then begin to use other products or services depending on their needs.

2. Business Model

2.1 Business Model Canvas

We have analysed and summarised our business model according to the business model canvas below:

<table>
<tr>
<td rowspan="2">Key Partners

1. Investment providers: trading brokerages, roboinvestors, etc.
2. Expenditure providers: wedding services, interior design services, etc.
3. Savings providers: banks, digital banks, etc.</td>
<td>Key Activities

1. Product development: Developing features which are useful and valuable to our users
2. Business development and partnerships: Attracting and onboarding partners onto our platform to sell their services</td>
<td rowspan="2">Value Propositions

To our users: easy to use interface which will help to improve their financial literacy and allow them to make better financial decisions

To our partners: Match buyers to relevant services/products</td>
<td>Customer Relationship

1. Self-service: providing customers with easy-to-use tools to manage their personal finances
2. Protection their personal data</td>
<td rowspan="2">Customer Segments

Mass market working adults in Singapore</td>
</tr>
<tr>
<td>Key Resources

1. Human: product, engineering and BD talent
2. Intellectual property: SaaS product</td>
<td>Channels

1. Own website
2. Partner channels</td>
</tr>
<tr>
<td colspan="2">Cost Structure

1. Staff cost
2. Cloud computing cost (See 4. Economic factors)</td>
<td colspan="3">Revenue Streams

1. Charging a fee to partners for successful sales
2. Ads</td>
</tr>
</table>

2.2 Cloud vs traditional on-premise IT resources / Key considerations for using cloud

We compared the pros and cons of cloud vs on-premise based on agility and scalability, and cost.

Agility and scalability

As a new digital native business, we do not have the burden of legacy applications hosted on-premise and time-to-market to launch our products and services is a priority.

Platform-as-a-service (PaaS) affords us agility and the ability to develop and test iterations of prototypes quickly, with their packaged databases, middleware and frameworks. In addition, launch costs on the cloud "are generally lower than those that can be achieved with comparable on-premise architecture" (Boston Consulting Group, 2019). However, we also run the risk of vendor lock-in as PaaS vendors have different configuration requirements, making it very difficult for us to switch to another vendor in the future.

As a start-up only in the prototype phase, scalability is also very important. Thanks to virtualization, IaaS, allows us to increase and decrease our IT resources as needed quickly without the need to purchase additional hardware. However, the downside is that we have to accept less privacy compared to on-premise.

Also, by adopting cloud computing, we do not need to hire specialists to maintain on-premise infrastructure. This allows us to have a leaner team and allows our development team to focus on creating value-adding products to serve our customers.

Cost

First, we do not have to incur significant upfront costs when we utilise cloud computing, as opposed to traditional on-premise resources. This is important for a start-up like us as we will be able to redeploy the upfront costs saved to other functions such as marketing and staff costs.

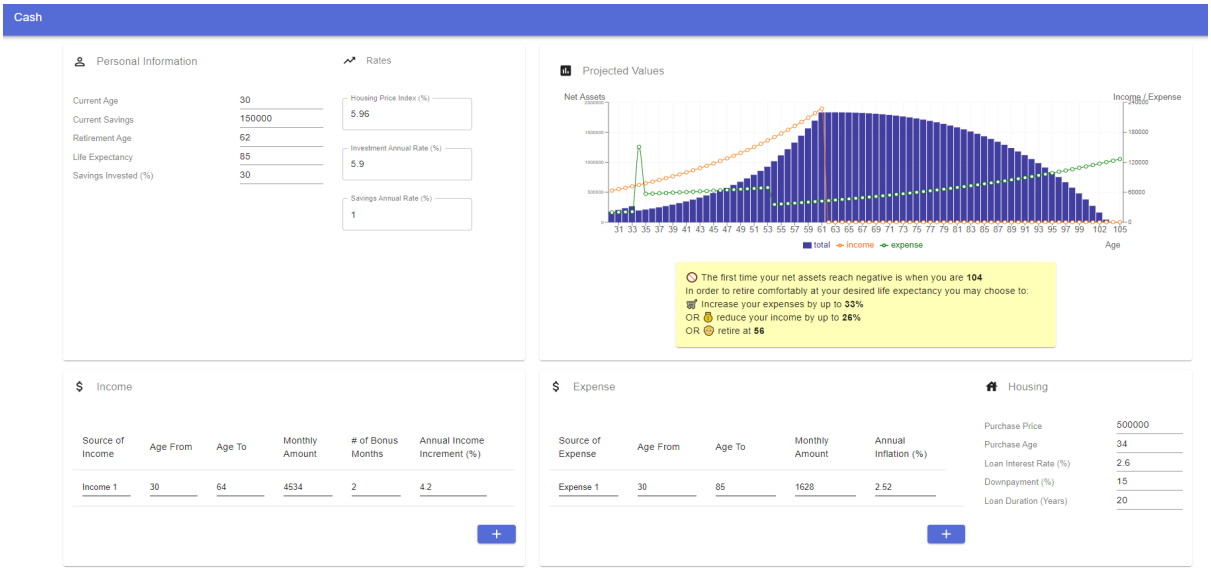
Second, cloud computing gives us transparency of our costs while companies tend to lose track of on-premise resources purchased as time passes. Cloud providers such as AWS allow us to track the costs incurred by individuals, groups and projects, allowing for greater transparency. In addition, we can make use of the available cost analytics and management tools to manage our costs better.

Third, we only need to pay only for what we use when we adopt cloud computing. Capacity planning for on-premise resources is highly challenging. The result of having annual corporate budgets is that many companies rather request for more budget to overprovision, resulting in underutilization of resources. Cloud computing costs are usage-based, reducing the tendency to overprovision. Having said that, cloud computing costs are also much easier to incur, due to the cloud's on-demand self service characteristic, so only a few clicks are required to incur more costs. Proper cost governance should be implemented to ensure judicial spending, such as the use of spending alerts and educating the team on how to work and minimize pay-per-use costs in a cloud environment. Higher spending does not automatically translate to better performance and we need to regularly evaluate the return on investment of our cloud computing costs.

3. SaaS Architecture & Implementation

3.1 Prototype

The prototype is being hosted on https://www.cash.cs5224cash.site/



On the first load, CASH displays default values based on average data taken from various sources. By changing any of the inputs (personal information, rates, income, expense and housing), it causes an immediate change in the dashboard (the graph) and financial insights (the yellow sticky note). The key features of the web application include:

- Emphasizing ‘personal’ in personal finance. Recognizing that everyone’s circumstance might be different, our application allows configurability and customization.
  - Do you have multiple streams of income or do they vary significantly across years?
  - Do you wish to take a gap year and see how it affects your retirement goals?
  - Do you invest significantly better than others or have lower expenses?
- CASH allows users to set parameters of their financial lives dynamically and visualize the results instantly well into their retirement. Once a user supplied their inputs, depending on one’s goals and current situation, they can then:
  - If they have been financially extravagant, see their gaps from their desired financial state, triggering positive behavioural changes to one’s financial decisions. In the future we plan to include recommendations and online resources to help this group of people.
  - If they have been financially prudent, feel more at ease about their financial future. Financial insights computed by CASH allow users to model exactly how much more risk they can take, for example how much they can increase spending without compromising on their financial goals.
- It can be picked up instantly and used by almost anyone with a simple and easy to use interface. The various inputs are already provided by default, if users lack financial knowledge and all the computations are done on the backend without requiring any mathematical knowledge. In the future we plan to add on more inputs and information for each input.
- There are also plans for other features in our web application as well. This includes:
  - a login functionality to save and secure your past computations, as well as customized links and deals to recommended partners based on your financial needs

(e.g. bank loan referral link if you are reaching house buying age, or an online broker/RoboAdvisor referral link should you need to increase your investment)

- ability to compare between two computations you made
- export functionality for exporting your calculations into CSV reports

3.2 SaaS Architecture

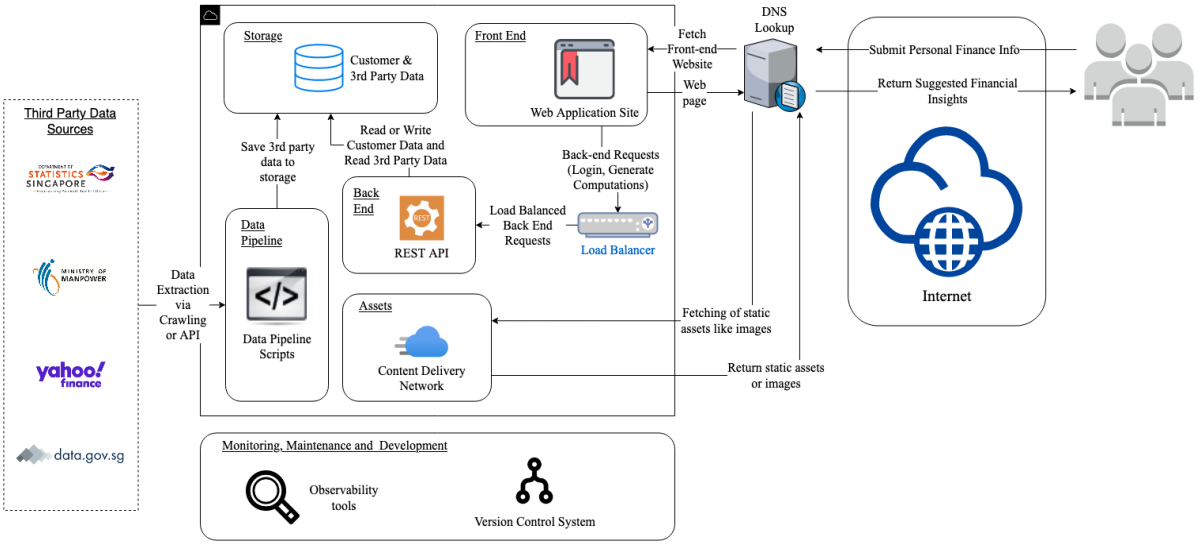


Figure 3.2: High Level Architecture of CASH App

High Level Design and Application Flow

Figure 3.2 outlines the high level view of the components in our application stack as well as the main application flow for our CASH application. When a user first visits our CASH application website through the internet by visiting our website using their own browser to plan his or her finances, the Domain Name System (DNS) will first route the user to our front end application. Our front-end application then serves the CASH app web page to the users' browser where they can fill up their personal financial information. Static images and assets on the website like favicons or banners etc. are stored separately on a content delivery network (CDN), to serve these assets to our users as fast as possible.

There are some use cases for our application, like calculating the financial metrics, exporting csv reports, or login functionalities. These use cases either need high computation power or talk to a storage layer, which may not be suitable for executing via the front end web application in the users' web browser. These tasks are instead handled by our back-end, which consists of REST API endpoints running in servers; as well as a database for persisting the user's computations as well as some of their data. Sitting in front of these servers running REST API endpoints are load balancers, that route the back-end requests to only healthy REST API servers to minimize downtime. The front-end web application can then pass the users' personal financial info to the back-end REST APIs via communicating with the load balancer. The back-end APIs can then do the computations and 1) save the information and results in the database for further use and 2) return the results back to the web application,

which then finally return the users the suggested financial insights as well as financial breakdowns in the form of widgets and charts.

Also as shown from Figure 3.2, our application also relies on data from external sources to enrich the information we provide to our users. They can be in the form of demographics and statistics data or financial figures which we would need to get through web crawling or extracting the data via APIs. There also needs to be a systematic and reliable way to bring these data into our system and keep these data up to date periodically. We do so by building a data pipeline, using a combination of Web Crawlers and Extract Transform Load (ETL) scripts to run on these websites or API endpoints. These scripts are executed periodically and help extract data from these external sources and persist them into our database, which the back-end application can then tap on for data from external sources.

Finally, we also have observability tools to help monitor our application health as well as the uptime, and a version control system to manage our source code and the contributions to the codebase.

3.3 Implementation

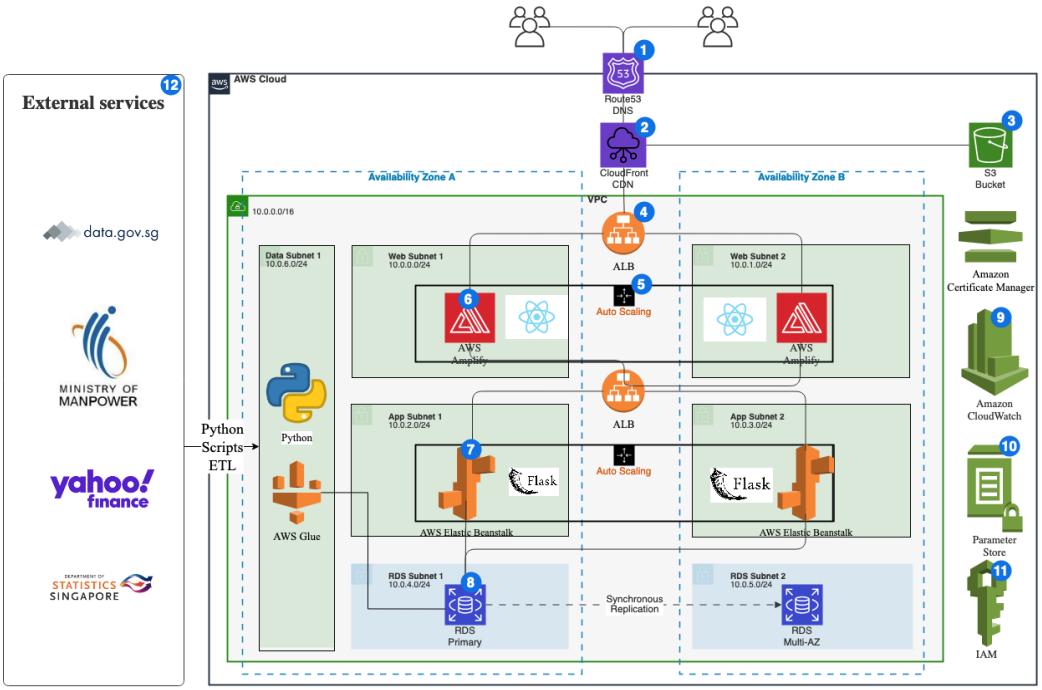


Figure 3.3: AWS Architecture Diagram of CASH App

For our CASH Application's SaaS implementation, we have chosen to implement it on Amazon Web Services (AWS) as AWS provides many Platform as a Service (PaaS) features that can allow us to more easily build our application, as well as having data centers in Singapore to fit our data localization requirements. Figure 3.3 above outlines the actual implementation for our application in AWS based on the design in Figure 3.2. This includes the AWS architecture and network layout, as well as other AWS tools, external data sources and services the application is built on. The subsequent sections describe the different

components of our application and explain the design decision for choosing each of the tools in our architecture.

Front-end Design

React.js, a popular front-end framework, would be used to build the front-end of our application. This includes the main interface, as well as dashboards and widgets to display the financial statistics to the user. We chose React as it is a very mature front-end framework with a huge amount of community support for building single page applications (SPA). All the different user functionalities such as users' financial input box, login page, export report menu, dashboard graph, and insights generation etc. are made into React components inside a single page. The front end will be in charge of handling all the various front end inputs and logic such as showing the graph and insights etc., as well as communications with the back-end, such as for the computation and insights generation, as well as sign-up page and login page etc.

Hosting of the front end will be through AWS Amplify, a PaaS for front-end offered by AWS. It was chosen for its ability to deploy static sites and single page apps with a Git-based workflow, allowing us to build deployment pipelines, and having tools like authentication and admin dashboards out of the box. AWS Amplify also runs on Docker, meaning we could leverage on containerization to ensure portability of our software.

Back-end and Data Pipeline Design

The back-end application would be implemented using Flask, a Python framework for developing web APIs. These APIs would power the main logic of the application's main functionalities, which are collecting the information from the user for their financial planning; doing the financial calculations based on the user's input; and finally generating the values to power the dashboards and insights for the user. The logic for deriving the metrics and insights from a user of our application can be described by a simple formula:

$$Y = f(X, Z)$$

where Y is the set of financial metrics and insights to be generated for the user, X being the set of input values from the user, Z is additional data and insights from 3rd party sources like SingStat, MOM, data.gov.sg and Yahoo Finance (such as demographics data), and f represents a function comprising a custom proprietary algorithm we are using to compute these metrics out of the input data. Currently this function is still a deterministic algorithm using a series of formulas and financial calculations through the numpy library etc. However, we are also exploring the use of machine learning once we gather more data for our application; to enhance the metrics and insights provided by our custom algorithm to be more relevant and insightful to the user. For users who may want to export these metrics for their own offline use, the back-end application also has the ability to generate reports for offline consumption. Our back-end application provides APIs for which the front end can call to export csv reports that are saved in S3, with a pre-signed URL and an expiry date which the user can use to download the reports. The bucket also comes with a life cycle policy to clean up reports after X days to reduce S3 costs of storing the reports.

Privacy is ensured as only a logged in user can see his/her own data, and these user preferences and filters are persisted to a database so that the users would not need to re-enter this information on subsequent visits. The database of choice will be MySQL, a popular open-source RDBMS used in many companies. As the application has a login feature, the database will also be storing sensitive information such as passwords. Proper steps must be made to ensure the security of the application. This includes the use of methods such as salting and bcrypt hashing with a suitable cost factor to ensure the password cannot be easily reverse engineered.

As mentioned in the previous section on application flow, the database also stores data from third party sources like SingStat and MOM to power our features, which we will have to extract and store in our application database. The data pipeline and scripts built to extract these data from the sources into our database will be mainly using Python. Python is a programming language that is commonly used for data extraction and transformation, and they also provide libraries to do so more easily. The scripts can be executed on AWS Glue, a serverless and managed ETL PaaS offered by AWS, that allows us to execute the Python ETL scripts with in-built autoscaling functionalities and without having to manage the underlying infrastructure.

Application Infrastructure

For our back-end infrastructure, the application will be hosted on AWS Elastic Beanstalk. AWS Elastic Beanstalk is selected for hosting the backend APIs as it provides a managed solution for parts of the infrastructure such as load balancers and auto-scaling groups, without requiring us to develop our application in a specific way in order to leverage on fully serverless solutions like AWS Lambda, while still allowing us to leverage on containerization to package our application with Docker runtime. AWS Elastic Beanstalk also comes with other features like swapping of environment url between different Elastic Beanstalk environments, which allows us to leverage on blue-green deployments to minimize deployment risks and reduces downtime. The Elastic Beanstalk for the back-end REST APIs uses t3.medium EC2 instances. We chose T3 burstable CPU instances as we do not always expect users to redo their financial computations, but there might be periods where there might be bursts in CPU demand. There will be policies in place to scale the application instances up or down in the servers based on CPU usage, as these complex financial metrics' computations can be CPU intensive.

Storage

Hosted AWS RDS with MySQL is chosen for the application database due to RDS's supporting automatic backups, multi-AZ support and hot standby for high availability, read replicas and automated upgrades, which otherwise would be tasks we would have to do ourselves if we self-hosted the MySQL database. AWS S3 is also used for storing and serving static assets such as images and generated reports for our users. Due to data protection requirements (PDPA laws in Singapore), both the RDS which contains user input and S3 buckets which contains user generated content (excel exports) will be hosted in ap-southeast-1 (Singapore) as our target customers are mainly Singaporeans. To minimize network latency between our application database and the application, the application EC2

instances would be located in the same region as well. To reduce downtime and keep availability high, our AWS application infrastructure will be configured across multiple availability zones (ap-southeast-1a and ap-southeast-1b) in case one of the availability zones go down. We also encrypt the data in RDS using encryption keys from AWS Key Management Store (KMS) to allow for data encryption at rest so that the data will be more securely stored in the cloud. This is to further minimize any risk of leakage of data by ensuring data is encrypted at rest.

Network and Security

There will be an application load balancer (ALB) sitting in front of the Elastic Beanstalk instances, and it's located in the public subnet. The Elastic Beanstalk instances themselves would be located in the private subnet for security reasons, and we only allow the HTTP ports from the instances to be open to the load balancer. AWS CloudFront which is a CDN service will also be sitting in front of the S3 assets to ensure fast content delivery to users and provide other capabilities such as rate limiting as well. Route 53 would be used for DNS resolution to route the user to the actual application and CloudFront CDN when the user accesses our application through the URL.

Monitoring, Maintenance and Development

Our applications would also need to be monitored for uptime etc. as well, which we will be using AWS CloudWatch Rules for. IAM will be used to control access management for the users, while EC2 Instance Profiles will be used on the Elastic Beanstalk instances to ensure least privileges on the servers, as well to minimize the impact in the unlikely event our servers are compromised. Our website will be protected by SSL with AWS Certificate Manager attached to the ALB that enables SSL auto renewal, while AWS Parameter Store helps to store our application secrets and keeps them safe with encryption using AWS KMS. Finally, the application development is done in GitHub for our version control systems, and CI tools like GitHub Actions can be used to create continuous integration and continuous deployment pipelines as well for the development process.

4. Economic Factors

4.1 Use case

We assumed app size of 5.4MB, 1000 daily active users and an average page size of 300kb to calculate the monthly GB served, GB stored and data transfer for sizing:

Monthly GB served
= Daily active users x average page size x 30 days
= 1,000 x 0.3/1024 x 30
= 8.78 GB

Monthly GB stored
= App size x no. of monthly builds
= 5.4/1024 x 2
= 0.01 GB

Monthly Data transfer
= (Initial Load (app size) + Additional load x no. of loads per person) x daily active users x 30 days
= (5.4/1024 + 2.62/1024x10) x 1000 x 30
= 162.79 GB

4.2 Economic benefits of using cloud

Based on our comparison of cost estimates for on-premise and cloud:

1) Our total cost would be 44% lower for cloud ($10,447) than on-premise ($18,708)
2) Our upfront cost would be 84% lower for cloud ($2,470) even if we only compare it to the cost of servers ($15,000).

We derived the Total Cost of Ownership for on-premise resources based on the calculator proposed by Barroso et al. (2019). We determined that we would require 3 servers for website, database and backup. We adjusted the parameters with the current cost of electricity in Singapore (Energy Market Authority, 2021) and Singapore corporate interest rate for SMEs (Linkflow Capital, 2021):

Dell PowerEdge FC640			
		Remarks	
cost of electricity ($/kWh)	$0.180	Cost of electricity in Singapore for Q2 2021	
interest rate	6%	Singapore corporate rates for SME	
DC capex ($/W)	10	Barroso et al., 2019	
DC amortization period (years)	20	Barroso et al., 2019	
DC opex ($/kW/mo)	$0.04	Barroso et al., 2019	
PUE	1.5	Barroso et al., 2019	
server capex	$5,000	Barroso et al., 2019	
server lifetime (years)	3	Barroso et al., 2019	
server W	340	Barroso et al., 2019	
server opex	5%	Barroso et al., 2019	
server avg power relative to peak	75%	Barroso et al., 2019	
$/W per month			
DC amortization	$0.025	4.9%	16.3%
DC interest	$0.018	3.5%	
DC opex	$0.040	7.9%	
server amortization	$0.245	48.1%	55.1%
server interest	$0.023	4.6%	
server opex	$0.012	2.4%	
server power	$0.097	19.1%	28.6%
PUE overhead	$0.049	9.5%	
total	$0.509	100%	
3-yr TCO for 1 server	$6,235.91		
3-yr TCO for 3 servers	**$18,707.72**		

Figure 3.2: Cost estimate for on-premise based on Barroso et al. (2019) calculator

We derived the cost estimate for cloud by using the AWS Pricing Calculator. In our deployment, we optimized for cost in our selection of pricing models. For Amazon RDS for MySQL, we opted for a reserved instance, with a 3 year partial upfront. For our EC2 instances, we opted for the Savings Plan with 3 years commitment:

Region	Service	Upfront	Monthly	Configuration summary
Asia Pacific (Singapore)	Amazon RDS for MySQL	$ 2,470.00	$ 96.22	Storage for each RDS instance (General Purpose SSD (gp2)), Storage amount (100 GB), Quantity (1), Instance type (db.m5.large), Deployment option (Multi-AZ), Pricing strategy (Reserved 3yr Partial Upfront)
Asia Pacific (Singapore)	Amazon EC2	$ -	$ 19.95	Operating system (Linux), Quantity (1), Pricing strategy (EC2 Instance Savings Plans 3 Year No UpFront), Storage amount (30 GB), Instance type (t3.medium)
Asia Pacific (Singapore)	S3 Standard	$ -	$ 2.85	S3 Standard storage (100 GB per month)
Asia Pacific (Singapore)	Amazon EC2 for AWS Batch	$ -	$ 41.56	Operating system (Linux), Quantity (1), Pricing strategy (EC2 Instance Savings Plans 3 Year No UpFront), Storage amount (30 GB), Instance type (m5.large)
Asia Pacific (Singapore)	AWS Data Transfer	$ -	$ 59.88	DT Inbound: Not selected (0 TB per month), DT Outbound: Internet (500 GB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (0 TB per month)
Asia Pacific (Singapore)	Amazon Route 53	$ -	$ 1.13	Hosted Zones (2)
Total cost over 3 years		**$10,447.24**		

Figure 3.2: Cost estimate for AWS based on AWS Pricing Calculator

5. Conclusion

The importance of personal finance cannot be overstated. Our mission at CASH is to help working adults by providing them with actionable insights on how to manage their money in terms of expenditure, investments, and savings, considering various life events and risks. As a new digital native business, adopting cloud computing affords us many benefits such as scalability, agility, and cost. This project has served as proof-of-concept of what can be achieved with AWS. We endeavour to improve on our prototype and help Singaporeans in realising their financial goals.

Bibliography

- Barroso, L. A., Holzle, U., & Ranganathan, P. (2019). *The Datacenter as a Computer: Designing Warehouse-Scale Machines* (3rd ed.). Morgan & Claypool Publishers.
- OCBC Bank. (2020). OCBC Financial Impact Survey for COVID-19. OCBC Bank. https://www.ocbc.com/group/covid19-support/survey.html
- OCBC Bank. (2021). *OCBC Financial Wellness Index*. OCBC Bank. https://www.ocbc.com/simplyspoton/financial-wellness-index.html#retirement
- Gilbert DT, et al. Immune neglect: a source of durability bias in affective forecasting. J. Pers. Soc. Psychol. 1998;75:617–638
- Gilbert DT, Wilson TD. Prospection: experiencing the future. Science. 2007;317:1351–1354.
- Klineberg SL. Future time perspective and preference for delayed reward. J. Pers. Soc. Psychol. 1968;8:253–257
- Linkflow Capital Pte Ltd. (n.d.). *SME Business Loan Interest Rate in Singapore*. Linkflow Capital. https://smeloan.sg/business-loan-interest-rate/