SMU
SINGAPORE MANAGEMENT UNIVERSITY

CS301: IT Solution Architecture

AY 2021/2022 Term 2

Group 4

Project Final Report: Project B

Spend Transaction Processing

project-2021-22t2-g1-project-2021-22t2-g1-team4

project-2021-22t2-g1-team4-fe

Team Members:

Charmaine Tan	rxtan.2018@scis.smu.edu.sg	01374766
Chen Sujuan	sujuan.chen.2018@scis.smu.edu.sg	01370038
Er Soon Hao	soonhao.er.2019@scis.smu.edu.sg	01364060
Nicolas Wijaya	nicolasw.2018@scis.smu.edu.sg	01332027
Sim Sheng Qin	sqsim.2018@scis.smu.edu.sg	01340058
Renata Dharma	renatad.2018@scis.smu.edu.sg	01372220

Table Of Contents

Stakeholders

List the key business and IT stakeholders for this application. Refer to your proposal deliverable. Your list can differ from the proposal deliverable. Keep this short.

Stakeholder	**Stakeholder Description**	**Permissions (if not applicable, write N.A.)**
Ascenda Loyalty	Provides banks with a turn-key solution for a loyalty points processing platform.	Read/Write permission to API Gateway Write access to RDS
Banks	Submit customers transaction details daily	Write access to RDS
Customers (Bank users)	Spend and receive benefits for each eligible transaction	Read permission to CloudFront

Key Use Cases

Use Case #1 Name: File Upload & Exclusion Processing

Description: This use case specifies how the client (Ascenda admin) can upload different file sizes on our systems.

Steps:

1. Upload transaction and user files
    a. We assume that files can be dragged and dropped into a central dropbox or posted via API (documented in appendix). The process should operate in the background. We expect a file of 1 million records that will be processed by the end of the day.

       As new customers join the platform and more customers spend more over time, we expect that the file processing will receive files periodically over a day. Thus, files can be asynchronously uploaded.
2. Convert Currency
    a. If the transaction's currency is not in SGD, convert to SGD.
3. Apply exclusion under a card program

a. All transactions will be first checked against exclusions by using each transaction's category codes (MCC). Certain spend types will be excluded from the card program.

Alternate flow of events: If uploaded files contains invalid

Post-Conditions: Validation is valid and transactions are written into the database.

Use Case #2 Name: Campaigns Management

Description: This use case specifies how the client (Ascenda admin) can manage data, that row's data will be deemed invalid and skipped. The other rows of valid data will be inserted.

Pre-Conditions: Customers have a valid card, the page is only available only upon submitting valid admin credentials. campaign data.

Steps:

1. Create Campaigns
    a. The bank merchant managers will create new campaigns by inputting the details of the campaign. If inputs are invalid, an error message will appear.
2. Apply campaigns
    a. The backend system will compile to see if each transaction under the customer's card will fulfil a campaign and issue the rewards to the card.
3. Display/ Update existing campaigns
    a. The campaigns will be displayed and can be dynamically switched on and off.

Alternate flow of events: If creation of campaign files contains invalid data, that row's data will be deemed invalid and skipped. The other rows of valid data will be inserted.

Pre-Conditions: Campaign inserted is valid, and the page is only available only upon submitting valid admin credentials.

Post-Conditions: Validation is valid and campaigns are written into the database. All inputs, processed tasks outcome and errors should be displayed to the users. All errors should be logged into the system for audit purposes.

Use Case #3 Name: Client Transaction View

Description: This use case explains how the bank customers and Ascenda can access a view of all their transactions eligible for rewards.

Steps:

1. Input customer ID
    a. The unique customer ID will then be used to identify the transactions they have on all their cards. The transactions are pre-fetched according to the page they are on to reduce the number of API calls.
2. View transactions
    a. To view more transactions, the next button can be clicked. When accessing previous pages of transactions, there will be no API calls as the transactions have already been called and stored.

Alternative flow of events: If no transactions are found for a customer, the system will show an alert.

Pre-Conditions: Customer ID is valid and the page is only available only upon submitting valid admin or customer credentials.

Post-Conditions: Validation is valid and transactions are displayed. All errors should be displayed to the users and logged into the system for audit purposes.

Proposed Budgets

Development Budget

Activity Name	Description	Cost
Analysis and Design	Project analysis and architecture design	Man-hours: 5
Creating of UI	Creating UI for customers to view transaction and reward	Man-hours: 3
	Creating UI for banks to upload transaction file	Man-hours: 3
	Creating UI for client (Ascenda) to manage campaign data	Man-hours: 3
Backend logic development	Developing logic for file and transaction processing	Man-hours: 30
Implement service logic for each use case	Creating endpoints for the frontend access	Man-hours: 20

Production environment configuration	Services setup in AWS	Man-hours: 20
	Implementing application to AWS	Man-hours:20
System Testing	Performance testing and security testing	Man-hours: 10

Production Budget[1]

Hardware/Software/ Service	Description	Cost
AWS EC2 T2.2xlarge with VPC and ALB	2 EC2 for 2 AZs	~ 403.12 USD/month
AWS Aurora MySQL - serverless	1 ACU with 80 GB database storage and backup storage	~ 84.68 USD/month
AWS S3	S3 Standard 20GB/month, S3 Glacier 5TB/month	~ 23.65 USD/month
Amazon CloudFront	Global CDN to deliver website and API call at high speed with low latency	1TB Free Tier for data transfer to internet and 10 million http/https requests
AWS Route 53	Cloud DNS web service, 1 hosted zone	~ 12.50 USD/month
AWS Certificate Manager	Public SSL/TLS certificates provisioned through AWS Certificate Manager are free	No cost
AWS API Gateway	30 million API call per month, including free tier	~ 25 USD/month
AWS Key Management	Securely Generate and Manage AWS Encryption Keys	~ 11 USD/ month
AWS SNS	Notification service with estimated 30 million request/month	~ 14.50 USD/month
AWS Web Application Firewall (WAF)	Number of Web Access Control Lists (Web ACLs) utilized (1 per month)	~ 5 USD/ month
AWS ElastiCache	1 standard node, reserved for 1 year, t2.medium cache	316 USD/yr 30.66 USD/month

[1] https://calculator.aws/#/estimate?id=d7a496c0fc35b2ec4840511b44f3b879d45e2804

AWS Systems Manager	On-premise instance management No additional charge 1000 per account per region	No cost
Total per month		610.11 USD/month

Key Architectural Decisions

Architectural Decision - GitHub Action in CI/CD	
ID	AD1
Issue	It is not efficient to manually deploy the backend code to AWS EBS and upload the frontend static contents to AWS S3 after making necessary changes.
Architectural Decision	The team utilised Github Action to automate the upload and deployment process to our application. This was done mainly by specifying all the necessary configurations in the frontend.yaml and backend.yaml files located in frontend and backend repositories. If there is any change that has been pushed onto the repository in the frontend app or backend app folder, github action will start the build and deployment action.
Assumptions	NIL
Alternatives	Codepipeline with codeBuild and codeDeploy
Justification	Our team has 2 repositories, frontend and backend, needed to deploy into different locations. If we use 2 codepipeline, it will cost more. And it is harder to maintain.

Architectural Decision - Microservice Architecture	
ID	AD2
Issue	Logic for Application Backend

Architectural Decision	Each function relating to the features requirements should be separated and independent of that with the other. The independence of the function will decouple the services of others. With a new function added, the new microservice will not affect the existing functions.
Assumptions	NIL
Alternatives	End to End Connections
Justification	In terms of maintenance, this option will be viable in the long run compared to maintaining point of point spaghetti architecture. With each function separated, there is no need to change the whole webpage and functions in totality and there is no need to do a regression test of the whole system whenever maintenance is performed.

Architectural Decision - Active-Passive Failover Config in Multiple AZ	
ID	AD3
Issue	Scalability and High Availability
Architectural Decision	EC2 instances are constructed in two different regions - Singapore and Seoul, with the Singapore region being the main branch. In the event the main instance is down, Route 53 will detect this ping heartbeat failure and will route the request to the secondary server. DNS Failover.
Assumptions	NIL
Alternatives	Active-Active Failover Configuration
Justification	In terms of cost of the service, it is cheaper compared to having the Active-Active configuration. As for the purpose, the service was intended for the standby EC2 instance for disaster recovery process when the failure is detected in

	the main instance. Hence there is no need for the Active-Active failover configuration for the current purpose.

Architectural Decision - Aurora Serverless (V2) Horizontal and Vertical	
ID	AD4
Issue	Millions of records and Workloads may be unpredictable at times.
Architectural Decision	Aurora Serverless v2 has the flexibility to scale both in & out on demand hence allowing millions of records to be stored with massive cost saving when batch transactions are not in use. Therefore suitable for the unpredictable and intermittent use case of uploading of batch transactions.
Assumptions	NIL
Alternatives	RDS Instance
Justification	Aurora Serverless v2 is able to scale both vertically and horizontally on demand peak period does not occur on a regular basis helping to reduce time in the configuration made using RDS.

Architectural Decision - Elastic Caching	
ID	AD5
Issue	Slow performance and the need to reduce the number of calls to Aurora and EBS.
Architectural Decision	By using AWS ElastiCache to store cache, this allows for the faster retrieval of data without increasing the number of calls to the database.
Assumptions	NIL

Alternatives	Pre-Fetch
Justification	In terms of performance, this will lead to faster performance and will improve the latency of the request. With caching it also means that there will be fewer calls, meaning that it will also cost less and reduce the stress on the server due to the many requests to the server.

Architectural Decision - Web Application Network Security Service	
ID	AD6
Issue	Web exploits and network attacks which may affect the availability of the system.
Architectural Decision	By using web application firewall into the architecture, it helps us to protect against web attacks by filtering traffic based on the rules that we have set up with. It also helps to filter out web requests to common attack patterns such as Cross Site Scripting and SQL injection.
Assumptions	Protection layer at incoming layer of the architecture but does not cover EC2 instances directly. With this in place, this allows web security to be placed in multiple points of the development process chain. This also allows more visibility and control for network security to our application.
Alternatives	AWS Shield Advanced
Justification	As the Route 53 DNS sends requests to the CloudFront edge location, best suited to serving the request. This is usually the nearest in terms of latency. AWS WAF will then inspect incoming requests according to your configured web ACL rules at the edge location. These rules might detect and block requests to protect against SQL injection, and many others

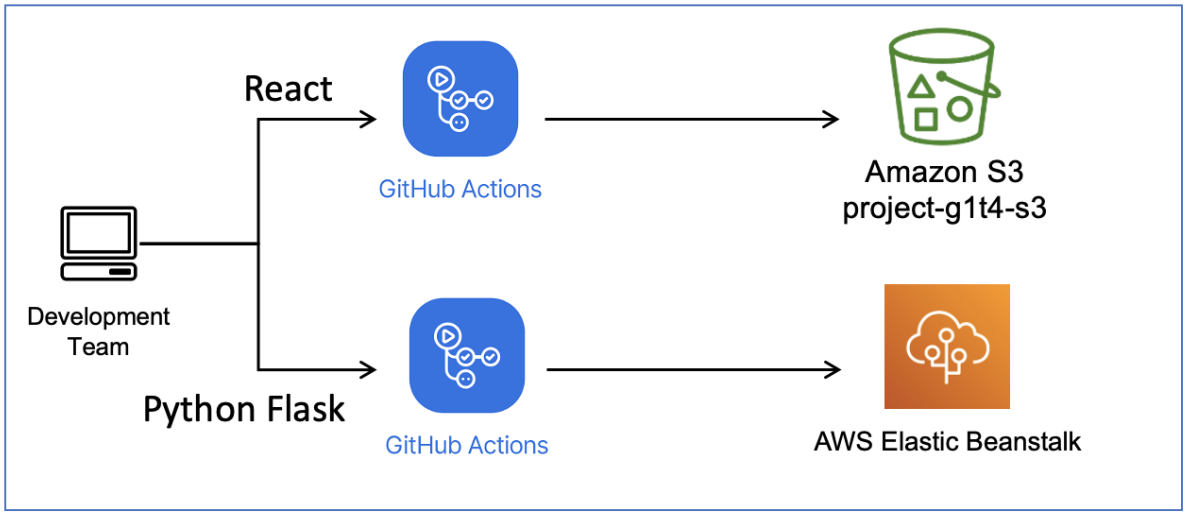
Development View

Deployment Strategy

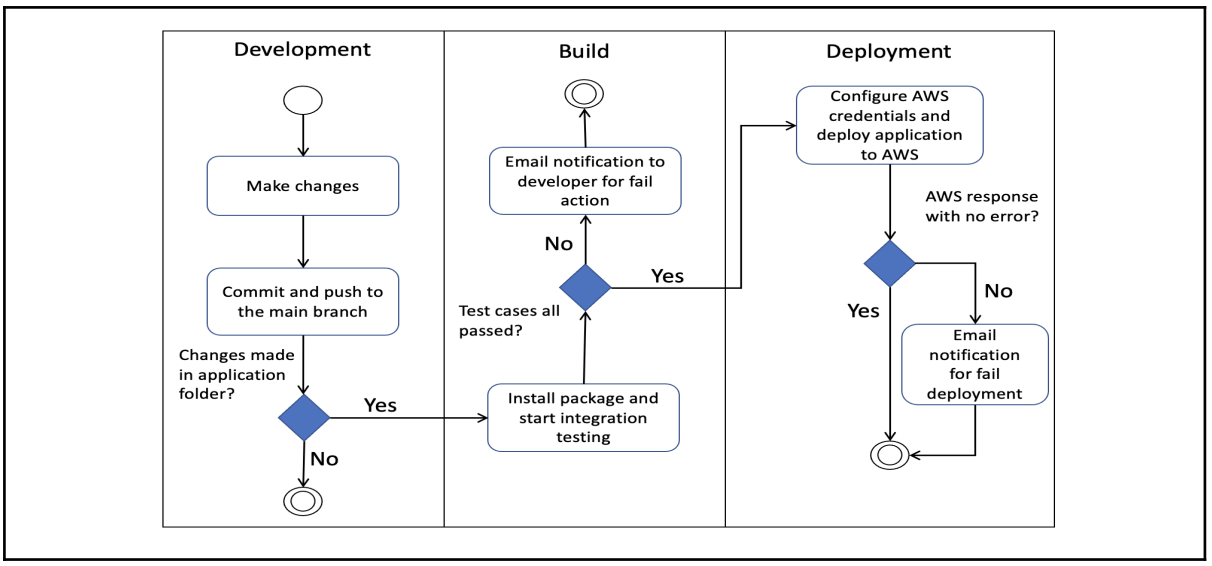
We created 2 repositories, one for the frontend website (**React**) and another for the backend server (**Python Flask**). We have an automated **GitHub workflow** for each repository. Workflow contains 3 steps, development, building and testing in Github, and deployment to AWS services. Once the code changes are made in the respective application folder and pushed to the main branch, workflow will be triggered. The Deployment step can only be executed when all the test cases are passed.

For the server, it will install all the necessary python libraries in requirements.txt and run test cases using pytest automatically to ensure that the main functionalities are working well and bug free. If all test cases are passed successfully, it will deploy the application on **Elastic Beanstalk**. For the client, it will automatically install the requirements in package.json and run automated Cypress tests for integration testing. Once successful, it will deploy the static websites on **S3**. If there is any failure in the building and deployment step, it will send email notification for the failure action to developers.

Using these continuous integration, deployment and testing, it will ensure a higher maintainability as the development pipeline can be accelerated. Also, bugs in the application can be caught early by the automated test cases before they become bigger problems in the future.



Development View:



Solution View (Maintainability)

Ease of Maintainability:

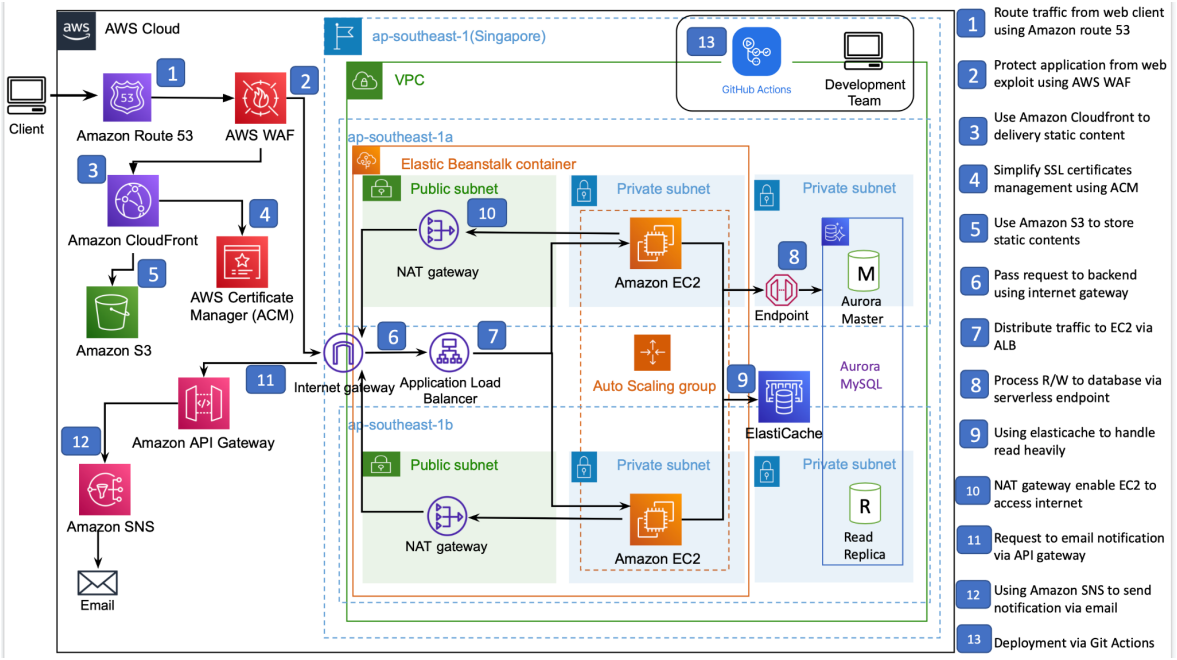
1. CICD with Github Action - Any changes to the frontend and backend application will go through integration testing and automatically deploy to AWS. This would reduce manual work.
2. Elastic Beanstalk - Allow easy deployment by uploading the code and Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, auto-scaling to application health monitoring.
3. S3 - Hosts the static frontend website that is publicly accessible and maintains bucket versioning, ACL to grant read/write permissions and provides metrics for analysis purposes

Integration Endpoints:

Source System	Destination System	Protocol	Format	Communication Mode
Python Flask	React	HTTPS	JSON	Asynchronous
Web Browser	Route 53	HTTPS	JSON	Synchronous
Route53	CloudFront	HTTPS	JSON	Synchronous

CloudFront	S3	HTTPS	JSON	Synchronous
API Gateway	AWS SNS	HTTPS	JSON	Asynchronous
Aurora	Read replica	HTTPS	MySQL	Asynchronous
Aurora	Write	HTTPS	MySQL	Asynchronous
AWS SNS	Email subscriber	HTTPS	JSON	Asynchronous

Solution View:



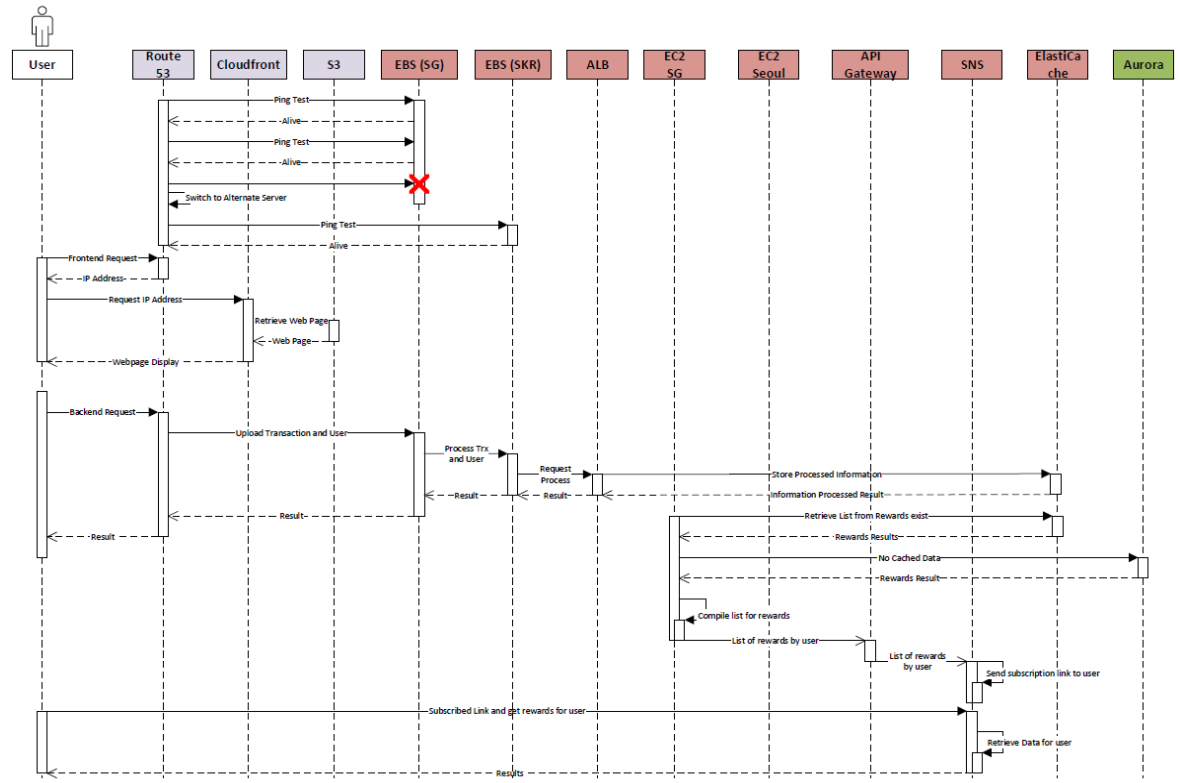
Note: The Architecture Diagram represents Singapore (ap-southeast-1) but it is multi-regional with deployment in Seoul (ap-northeast-2) as well

Availability View

Node	Redundancy	Clustering			Replication (if applicable)			
		Node config.	Failure detection	Failover	Replication type	Session state storage	DB repl. config	Repl. mode

Aurora	Horizontal scaling	Active-Passive	Serverless	Automatic multi-AZ failover				
EC2	Horizontal scaling	Active-Active	Ping	Load balancer				

Availability View:



Frontend Components (Route 53, CloudFront, S3)

The frontend website with **Route53** domain name services with **CloudFront** distribution using CNAME records distribution to service static websites from **S3** with low network latency and high transfer speeds. With this set up, The Frontend web page is cached and served with **CloudFront** with all edge locations to provide high availability for users.

Server Components (Elastic Beanstalk, Application Load Balancer, Multi-Region)

A multi-region deployment strategy in **ap-southeast-1** and **ap-northeast-2** to achieve high availability in the unlikely event of a regional failure. In each region, **Elastic Beanstalk** web tier environment to handle specifically HTTPS requests from users that is load balanced with an application load balancer within each **Elastic Beanstalk** environment distributes traffic to minimally two availability zones in both regions. EC2 Instance is automatically scaled based on traffic demand with an auto scaling group.

Database Components (Aurora Serverless & AWS Backup)

Aurora is a fully managed database that is designed for speed, availability and scalability of high-end databases with the simplicity and cost of open source databases. **Aurora** provides a distributed**,** fault-tolerant and self-healing system that is decoupled from compute resources which can be auto scaled up to 128TB per database instance on demand.

The downside to this was the immense cost that comes with it as Aurora (Server) cost 300% more than traditional MySQL databases and 20% more than RDS however this is circumvented by using **Aurora Serverless**.

The implemented solution uses **AWS Aurora Serverless** that is suitable for transactions for unpredictable spikes in workloads such as uploading large amounts of data. **Aurora Serverless** is configured with auto-scaling set to a minimum instance when usage for the database is low for example in the solution view, the minimum instance is set to 1 while the maximum instance is currently set at 10. This provides charge on on demand usage, auto-scaling, and high-availability.

The solution uses a MySQL database on **Aurora Serverless** The solution prioritises data consistency & integrity over a non-relational database such as DynamoDB. Using a non-relational database such as DynamoDB and using Dynamo transactions to provide ACID properties to transactional data requires additional complexity to the solution. Additionally, **AWS Backup** is used to ensure data is backed up (currently set to monthly) in the unlikely event of downtime since it is a fully managed serverless solution.

Aurora Serverless v2 Global Tables

Requests like view card transaction and campaign management are highly performant, especially during peak periods. Global Tables can help to serve these requests, achieving horizontal scaling and improving performance

Security View

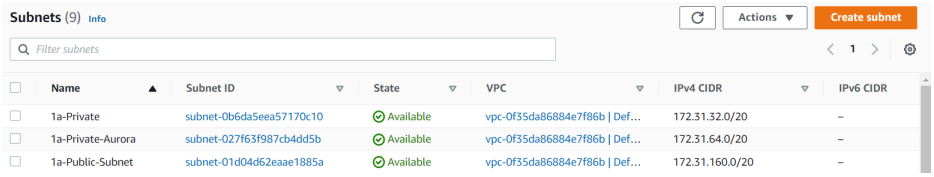
No	Asset/Asset	Potential Threat/Vulnerability pair	Possible Mitigation Controls
1	Amazon S3, Amazon EC2	Packet sniffing / Confidentiality Packets sent over through HTTP can be easily sniffed as they are not encrypted. This allows attackers to sniff for user login details through network packets.	Using HTTPS protocol automatically encrypts network packets which thwarts sniffing attacks.
2	AWS Aurora	SQL Injection / Authorisation Attackers are able to gain access to the database by exploiting an SQL injection attack. This may cause unauthorised access to both user and transaction data.	AWS WAF protects against SQL injection attacks by using SQL injection match conditions. If requests contain malicious SQL code, they will be automatically blocked.
3	Amazon S3	DDoS / Availability Since the front page is exposed to the public, anyone can connect to it to initiate requests. This can allow attackers to perform DDOS attacks by flooding the network with requests.	AWS Shield provides always-on detection and automatic inline mitigations to prevent DDoS attack.
4	Amazon S3, Amazon EC2	SSL strip / Confidentiality SSL stripping allows attackers to force a response from the server by hijacking packets and sending them to the unprotected HTTP endpoint to bypass HTTPS. As information sent over HTTP is unencrypted, sensitive data can be leaked this way.	HTTPS redirection forces all connections to the server to connect only using HTTPS which encrypts all data in transit by default. As such, any data sent to the server via HTTP port will be

			redirected to HTTPS, and the information sent back will be encrypted.
5	Amazon EC2	Unauthorised access / Weak authorisation Without the use of API keys or proper access controls, any unauthorised individual will be able to access and modify the data stored in the database through the unprotected API endpoints.	Using security groups, access control list and API keys allows only authorised users to access the data retrieved by the API.

Security View:

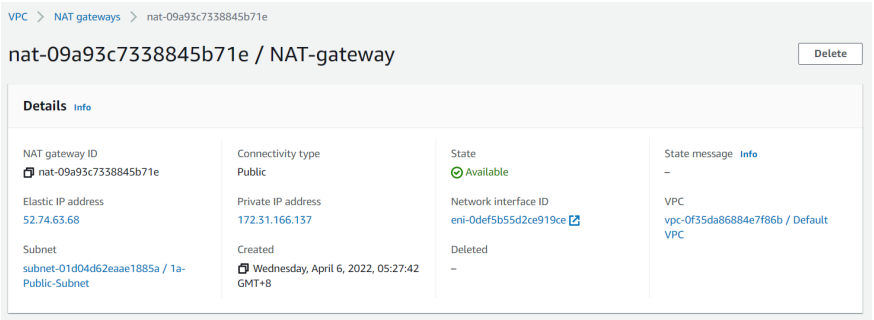
<u>Subnets</u>

Subnet Configuration



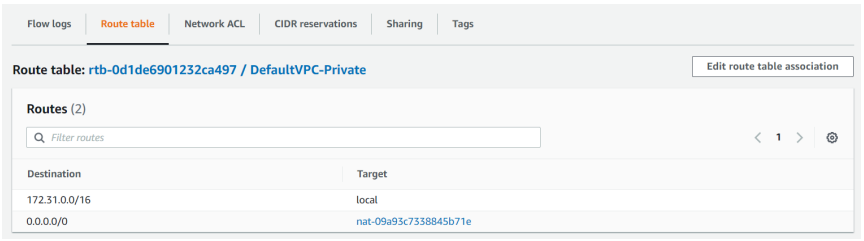
1a-Private contains the EC2 instance while 1a-Private-Aurora contains the RDS Aurora instance. Both of these subnets are private subnets. The 1a-Public-Subnet contains the NAT gateway that will route to the Internet gateway.

NAT Gateway



These are the configurations for the NAT gateway used in 1a-Public-Subnet.

Private Subnet



This is the route table configuration for 1a-Private and 1a-Private-Aurora.

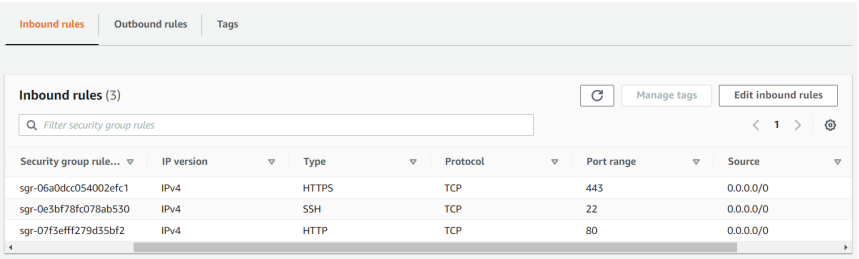
Public Subnet

This is the route table configuration for 1a-Public-Subnet.

<u>Security Groups</u>

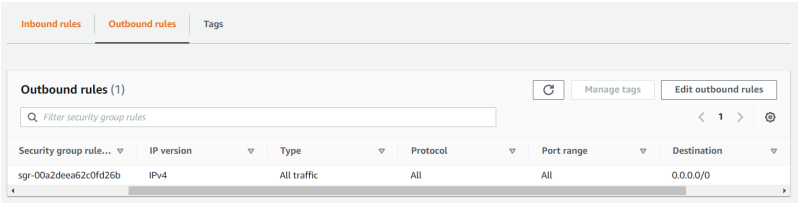
The below Security Group configuration was used for our EC2 instance and Aurora.

Inbound



All HTTP, HTTPS and SSH connections are allowed for inbound connections.

Outbound

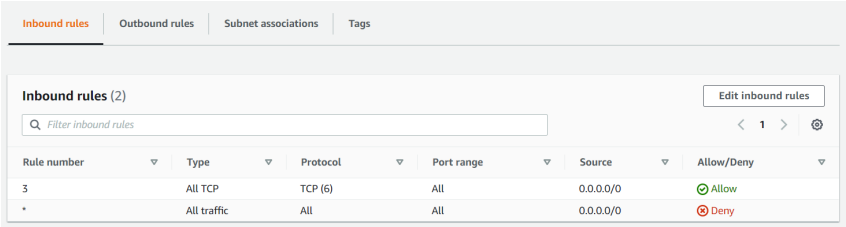


All traffic is allowed for outbound connections.

<u>Access Control Lists (ACL)</u>

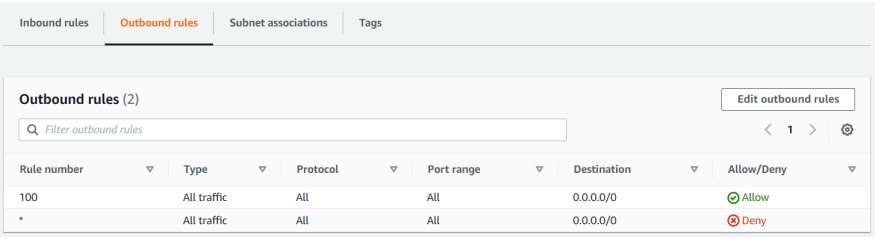
The same ACL is used for both the EC2 instance and Aurora

Inbound



We allowed TCP connections to enable both Route 53 to connect via port 53(TCP) and Aurora to connect. We deny all other requests to ensure security

Outbound



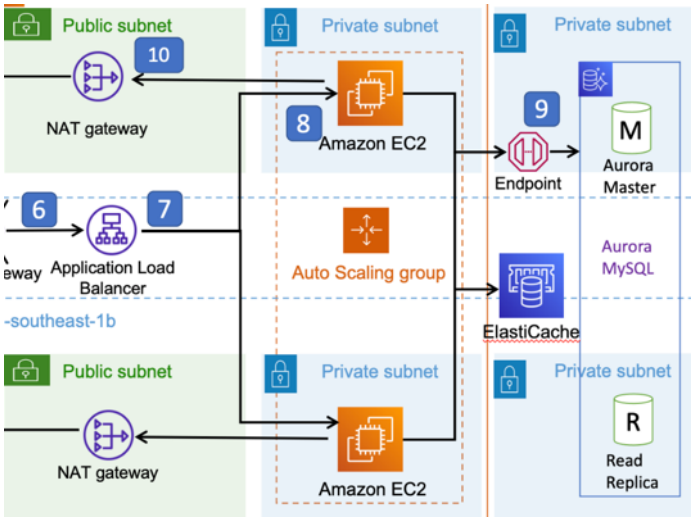
All traffic is allowed for outbound connections.

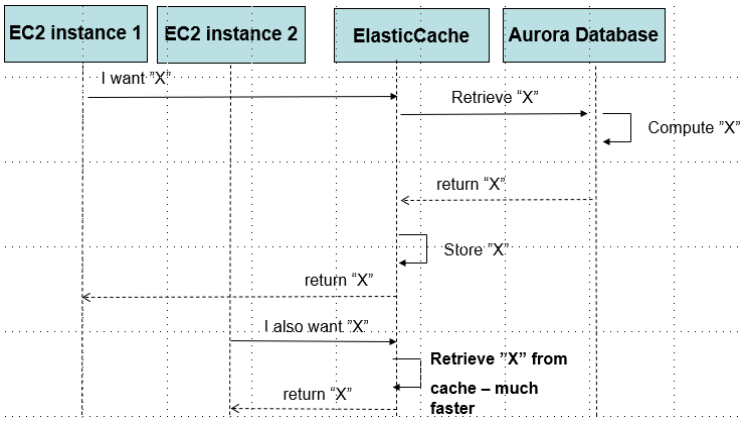
Performance View

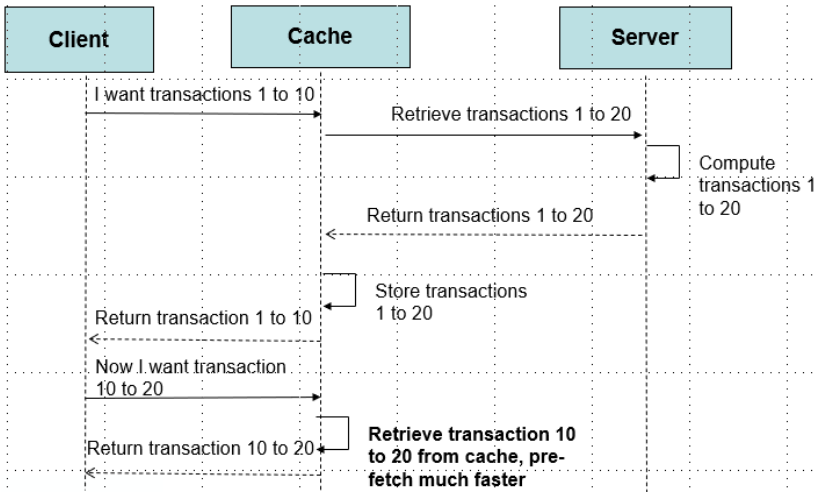
No	Description of the Strategy	Justification
1	Aurora read replica	Requests like view card transaction and campaign management are highly performant, especially during peak periods. Read replicas can help to serve these requests, achieving horizontal scaling and improving performance
2	Caching is used for the backend server's API endpoints	For repeated requests of the same set of data, the result can be retrieved from the cache instead of the server. This reduces the number of calls to the server and the database. This improves their performance, especially during periods with a large number of requests
3	EC2 autoscaling with Elastic load balancing	Elastic Load Balancing scales the application load balancer to match the changing traffic to our website over time. It is able to scale to the vast majority of workloads automatically. For instance, during periods where there are many requests to the server, the load balancer will

		automatically register new instances using auto-scaling to handle an increased number of requests.
4	Least outstanding requests routing algorithm using an Application Load balancer	We chose to use the least outstanding requests routing algorithm over the default round-robin algorithm as the requests for our website vary in complexity. For example, the requests to process the transactions and users' files will take a significantly longer time than simply requesting the processed records. Using this algorithm, as new requests arrive, the load balancer will route them to the target with the least number of outstanding requests. Targets processing long-standing requests are not burdened with more requests and the load is evenly spread across targets.
5	The user file and transactions file can be uploaded and processed using parallel execution	The user file and transaction file do not have dependencies on each other so they can be uploaded simultaneously and processed using parallel execution. This saves a substantial amount of time as it usually takes around 5 mins to upload each file.
6	When users retrieve their transactions together with their rewards earned, the server will pre-fetch 20 transactions when users requests for 10 transactions	This increases the speed of viewing the transactions as users can immediately go to the next page without waiting for another request to the server

Performance View:

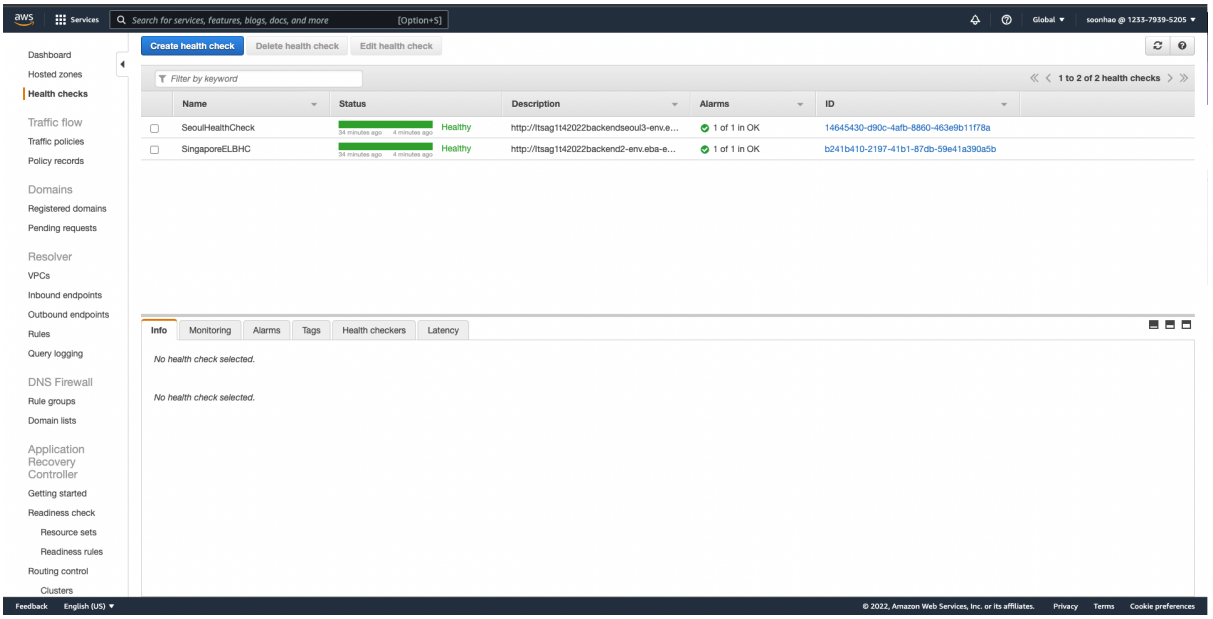

Public subnet
Private subnet
Private subnet
10
NAT gateway
8
Amazon EC2
9
Endpoint
M
Aurora Master
6
7
Application Load Balancer
Auto Scaling group
Aurora MySQL
-southeast-1b
ElastiCache
Public subnet
Private subnet
Private subnet
NAT gateway
Amazon EC2
R
Read Replica

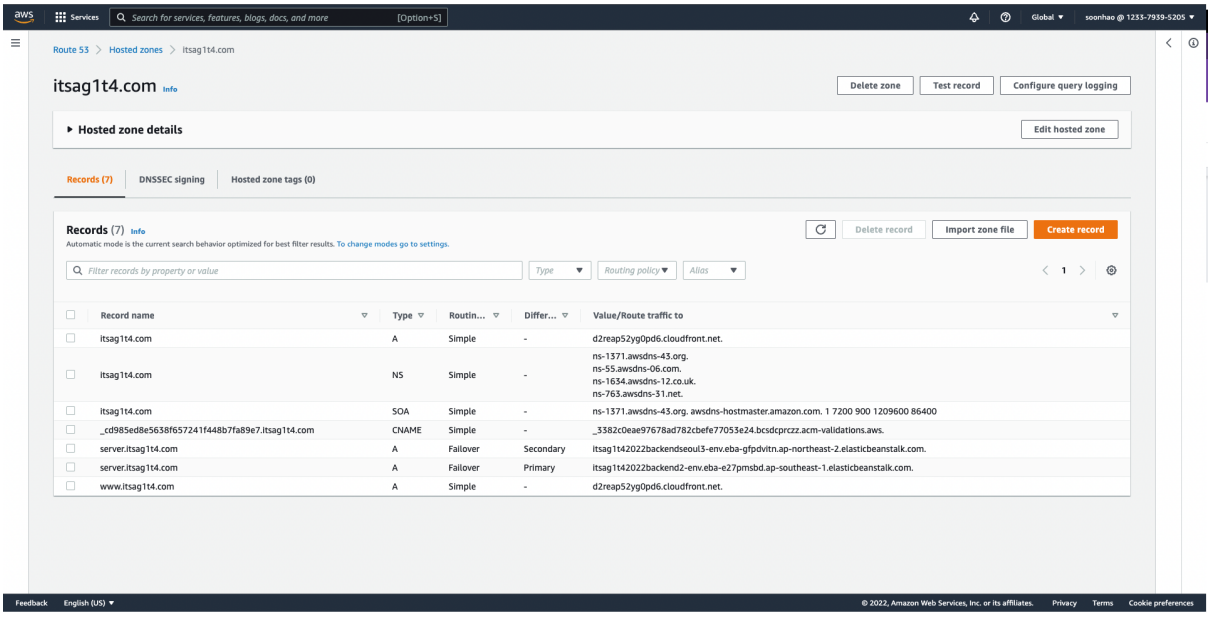

EC2 instance 1
EC2 instance 2
ElasticCache
Aurora Database
I want "X"
Retrieve "X"
Compute "X"
return "X"
Store "X"
return "X"
I also want "X"
Retrieve "X" from cache – much faster
return "X"


Client
Cache
Server
I want transactions 1 to 10
Retrieve transactions 1 to 20
Compute transactions 1 to 20
Return transactions 1 to 20
Store transactions 1 to 20
Return transaction 1 to 10
Now I want transaction 10 to 20
Return transaction 10 to 20
Retrieve transaction 10 to 20 from cache, pre-fetch much faster

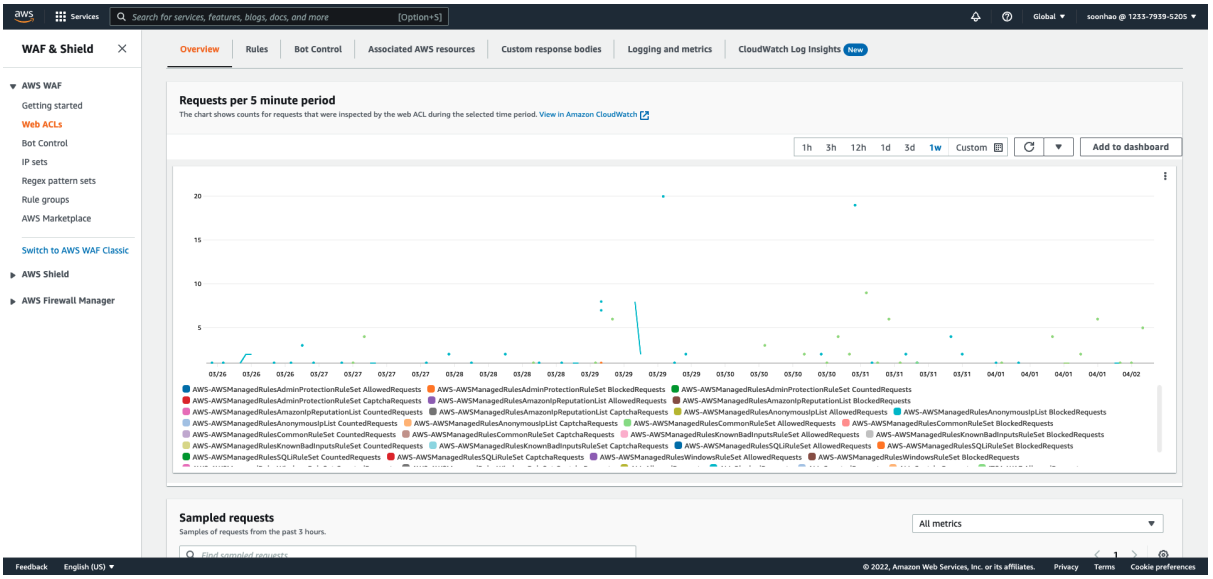
Appendix

AWS Route53 (Health Checks & Regional Failover)

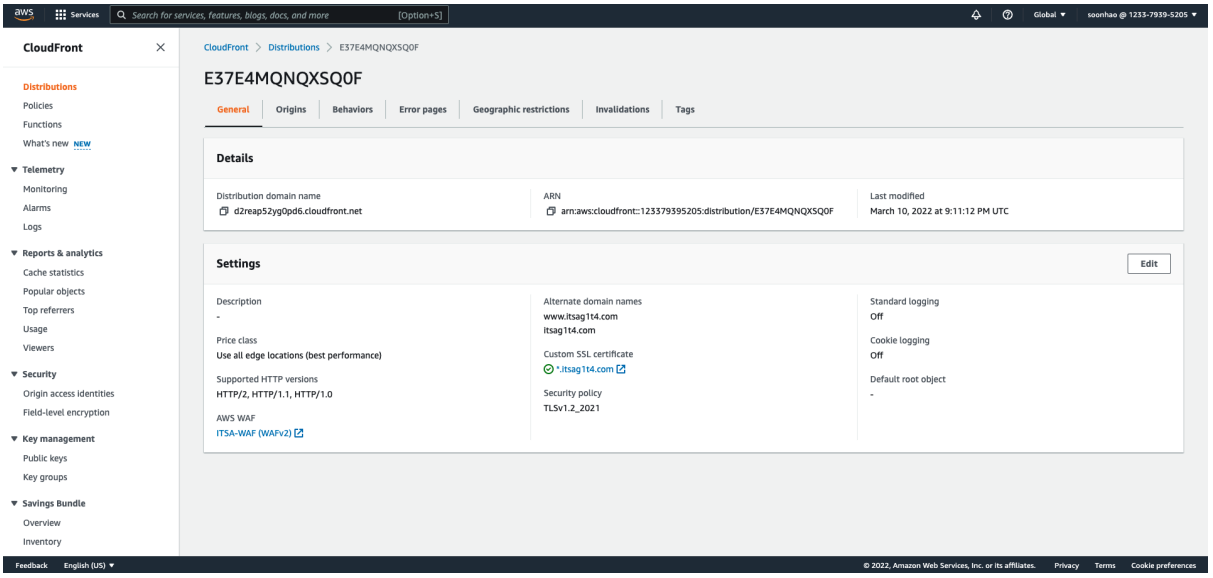




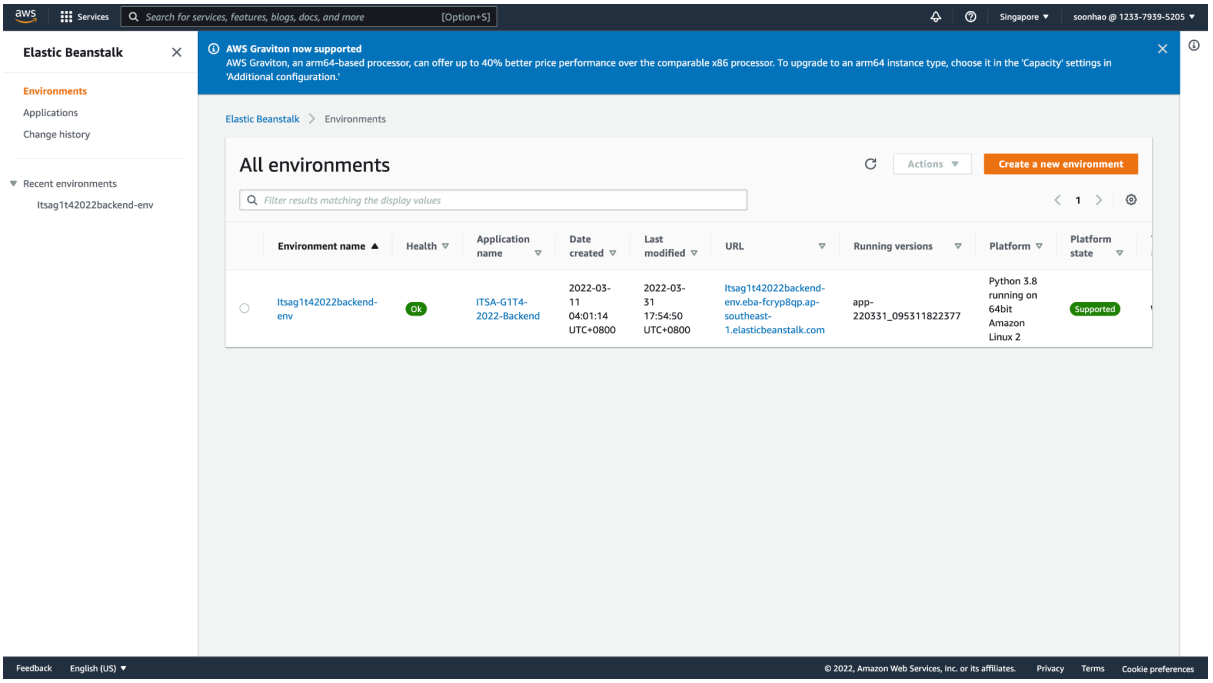
AWS WAF

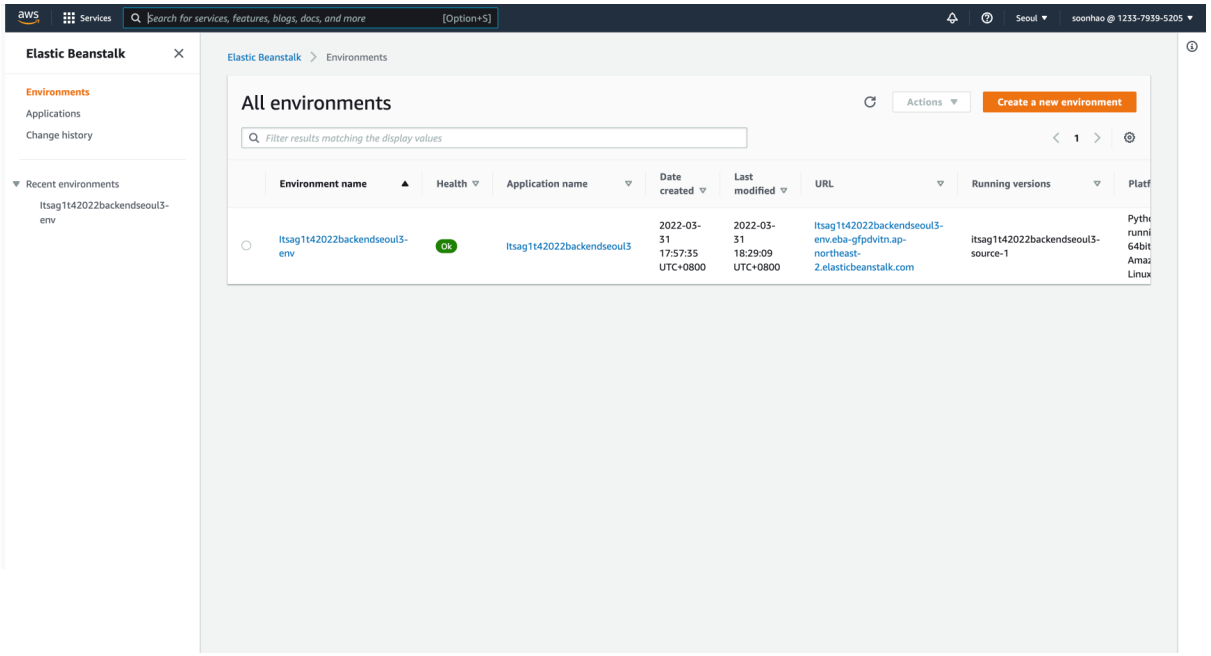


AWS S3

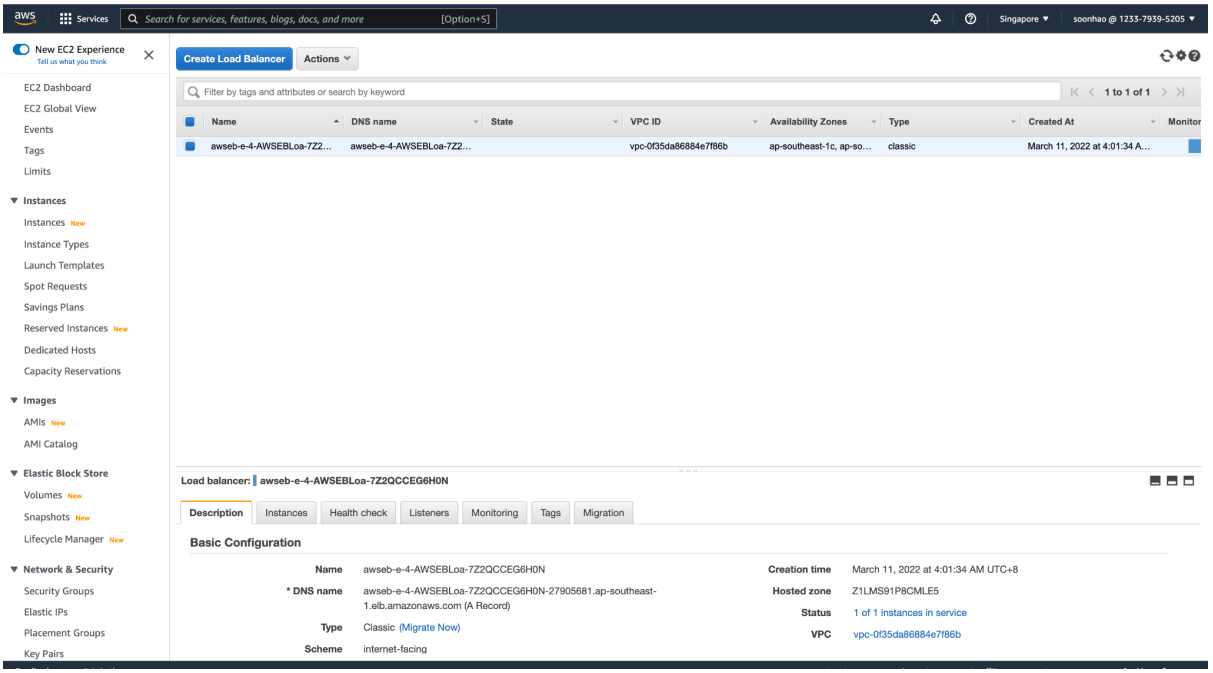


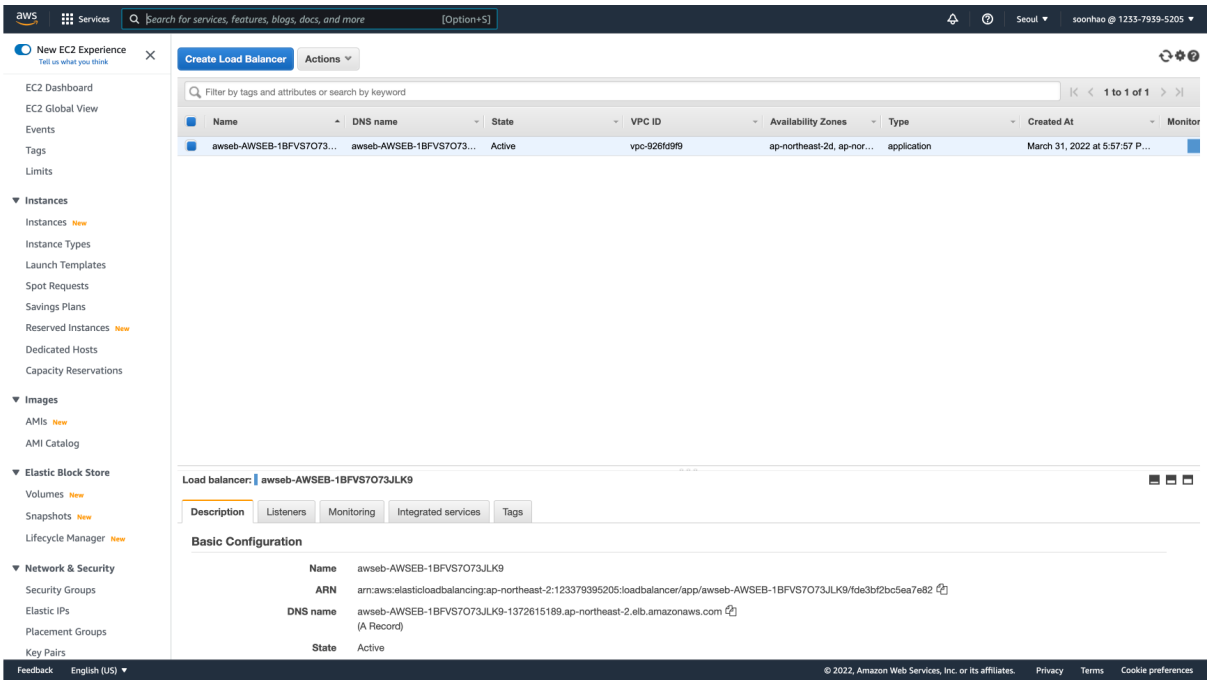
AWS Elastic Beanstalk (ap-southeast01 & ap-northeast-2)



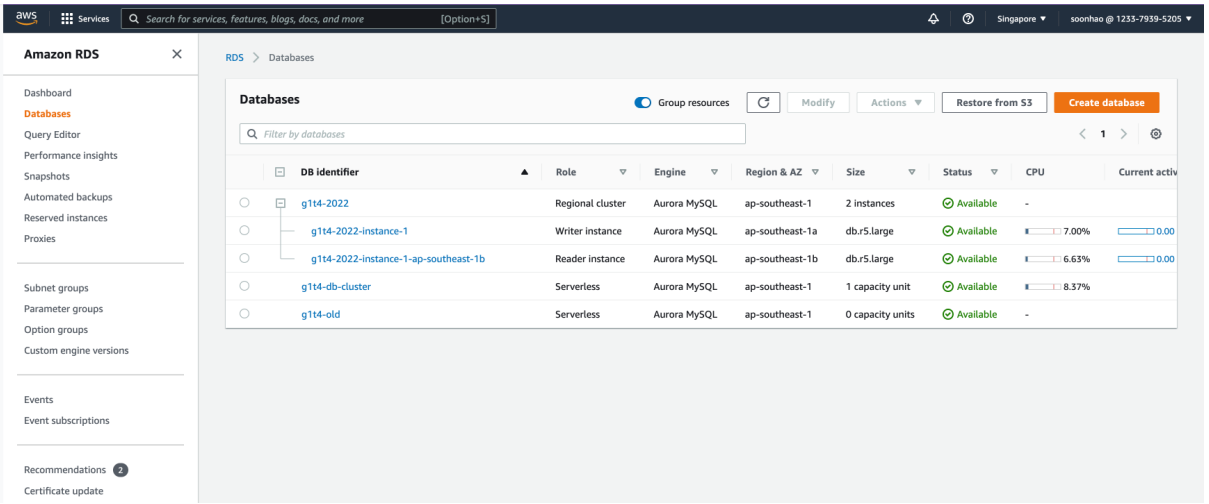


Elastic Load Balancing (Classic & Application)

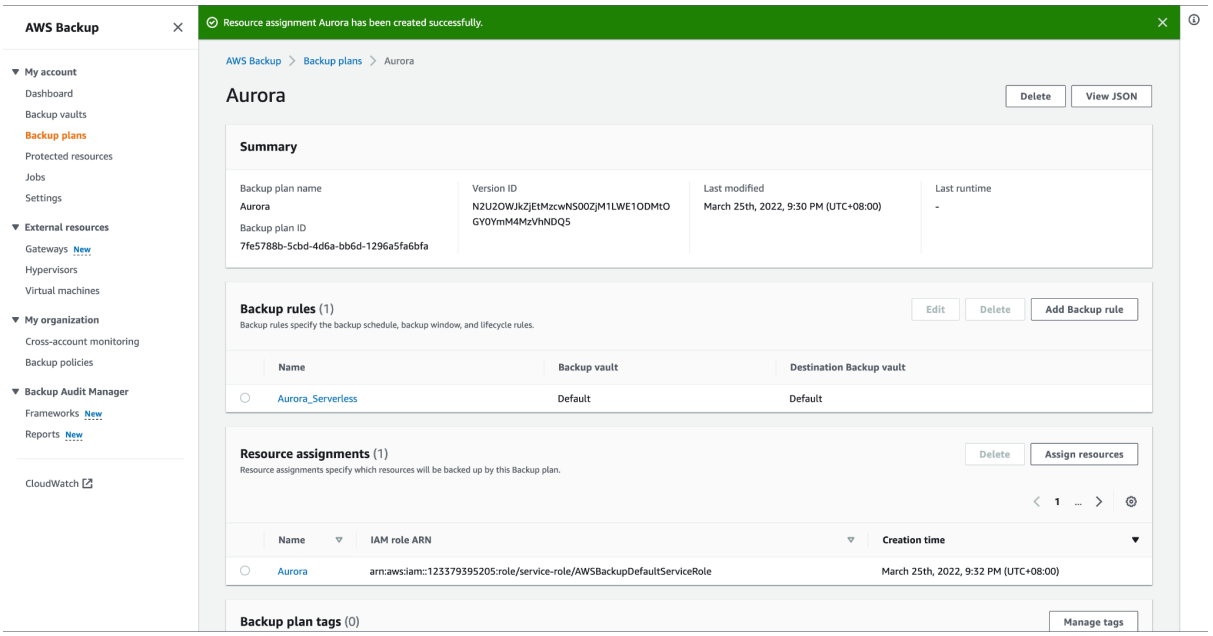




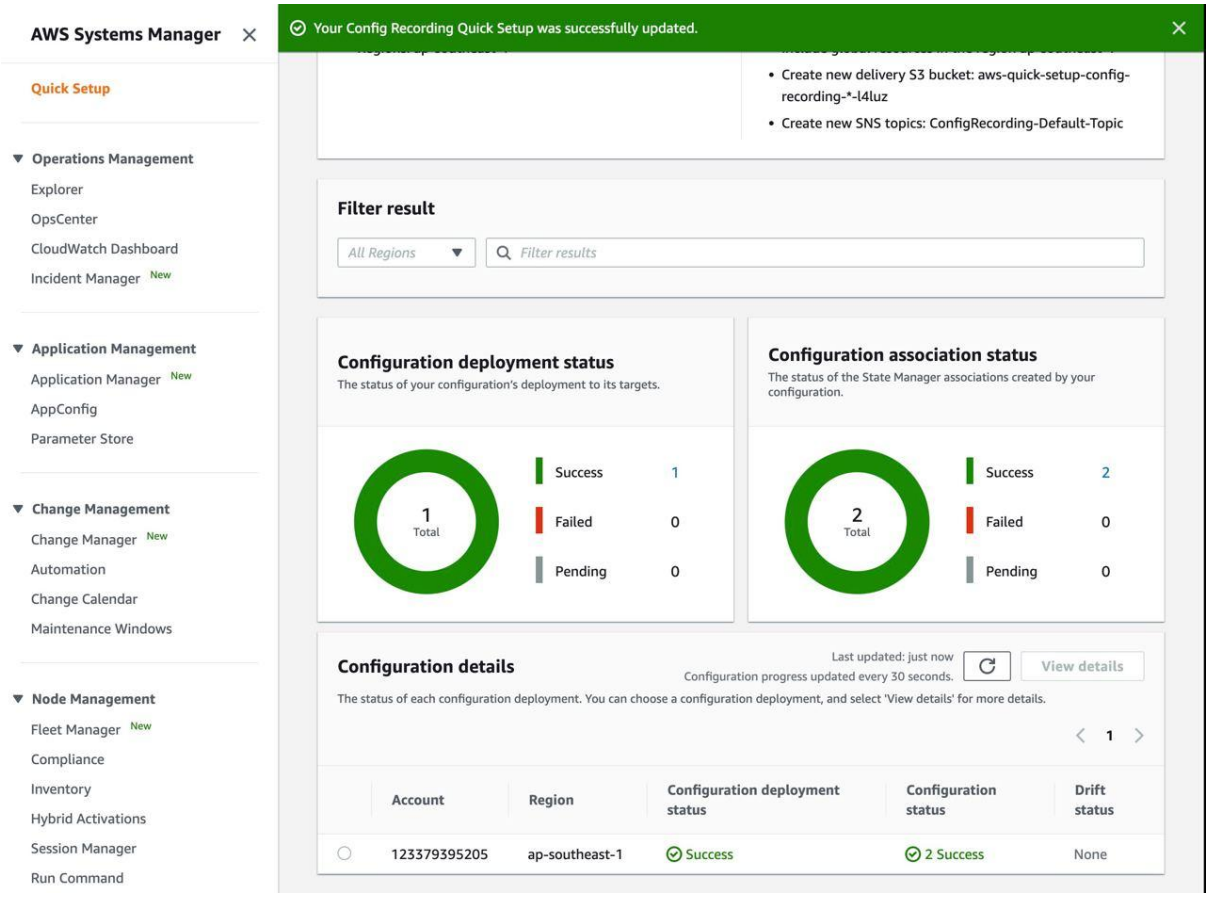
AWS Aurora (Server & Serverless)

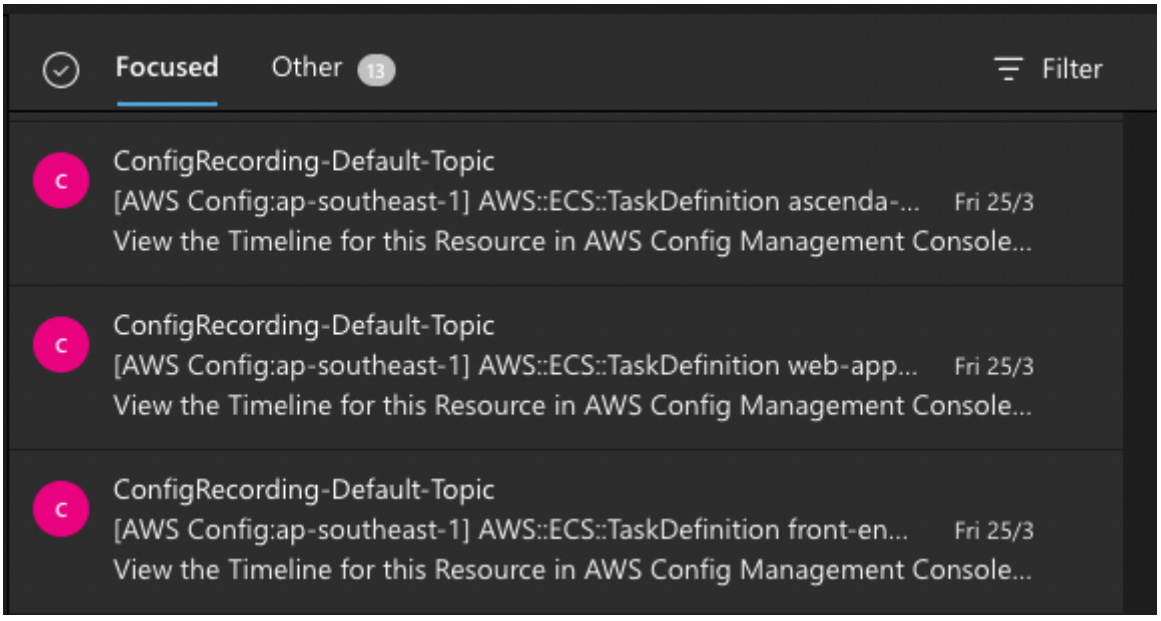


AWS Backup (inclusive of all resources)

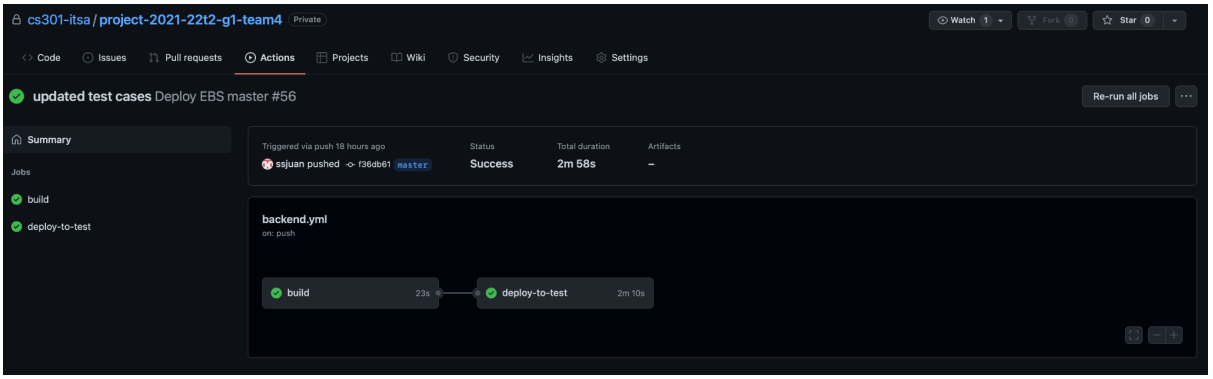


AWS Systems Manager

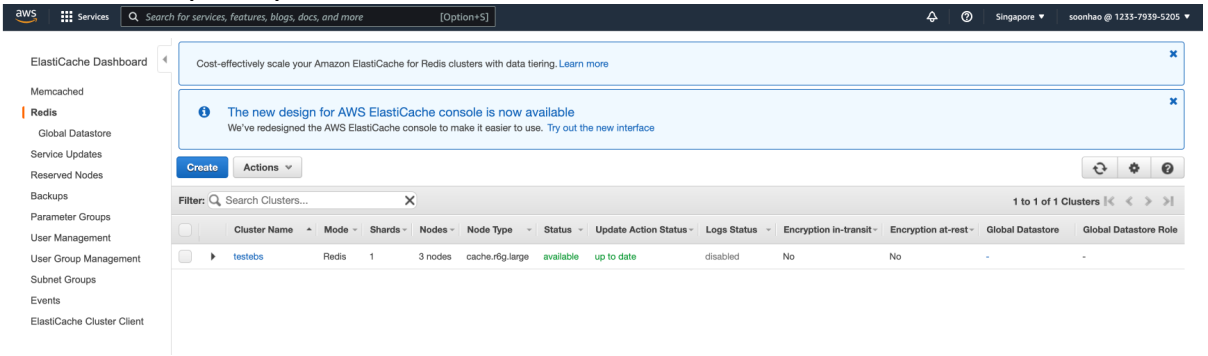


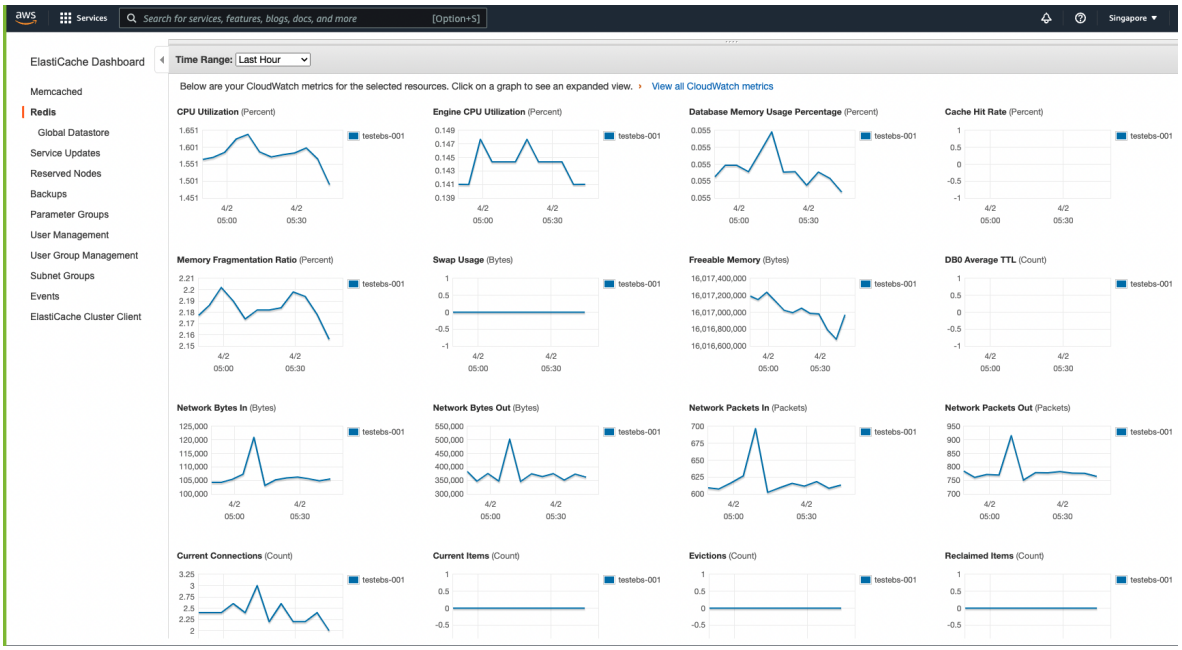


CI/CD using GitHub Action

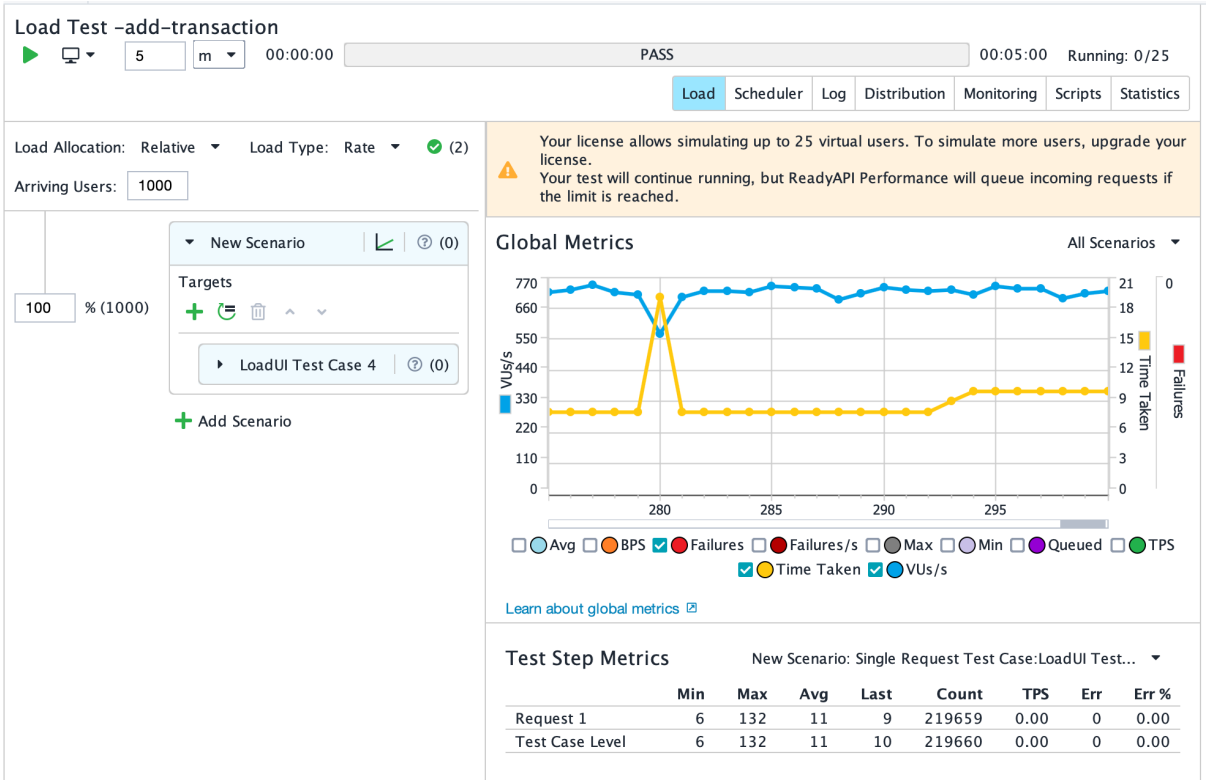


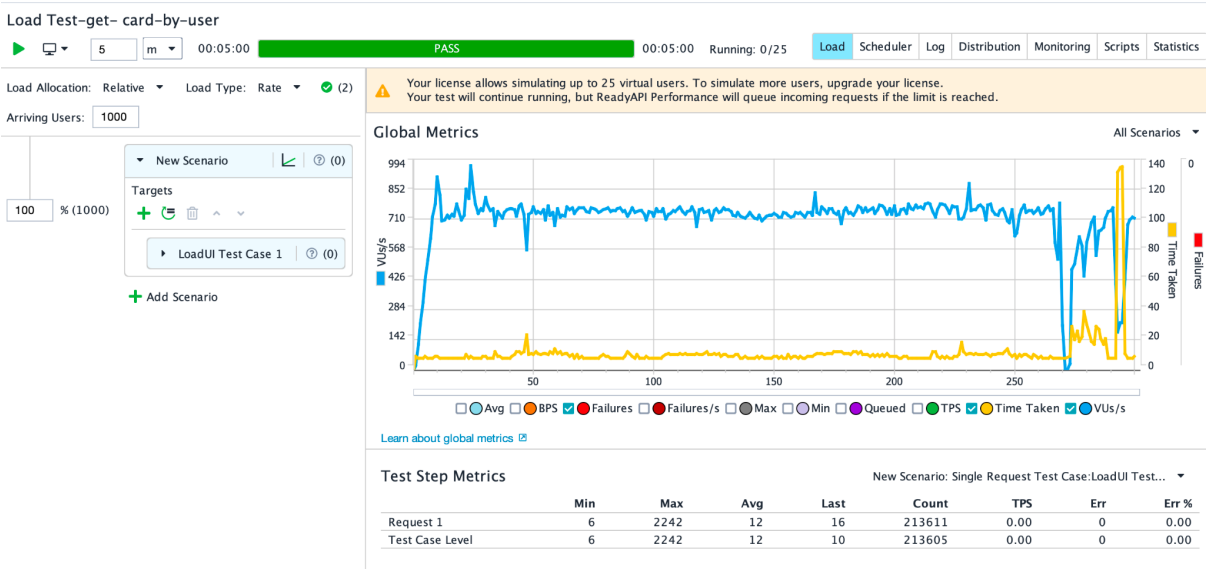
ElasticCache (Redis)

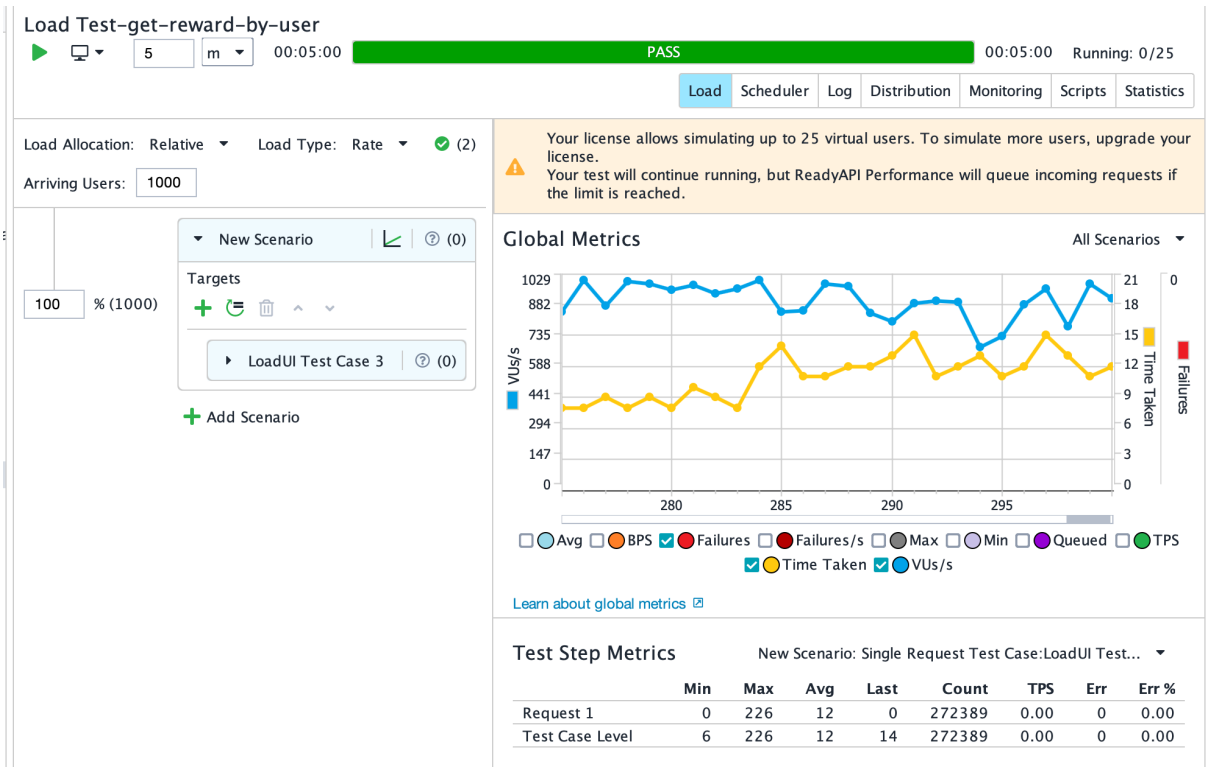


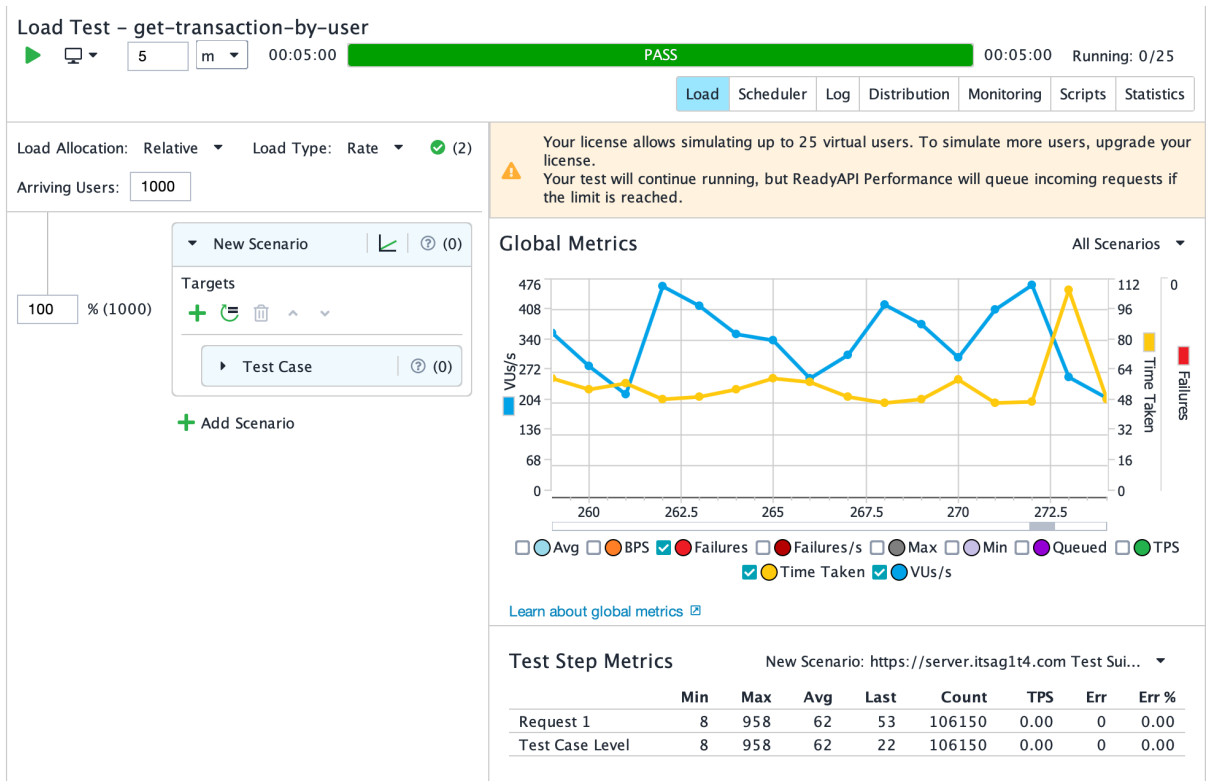


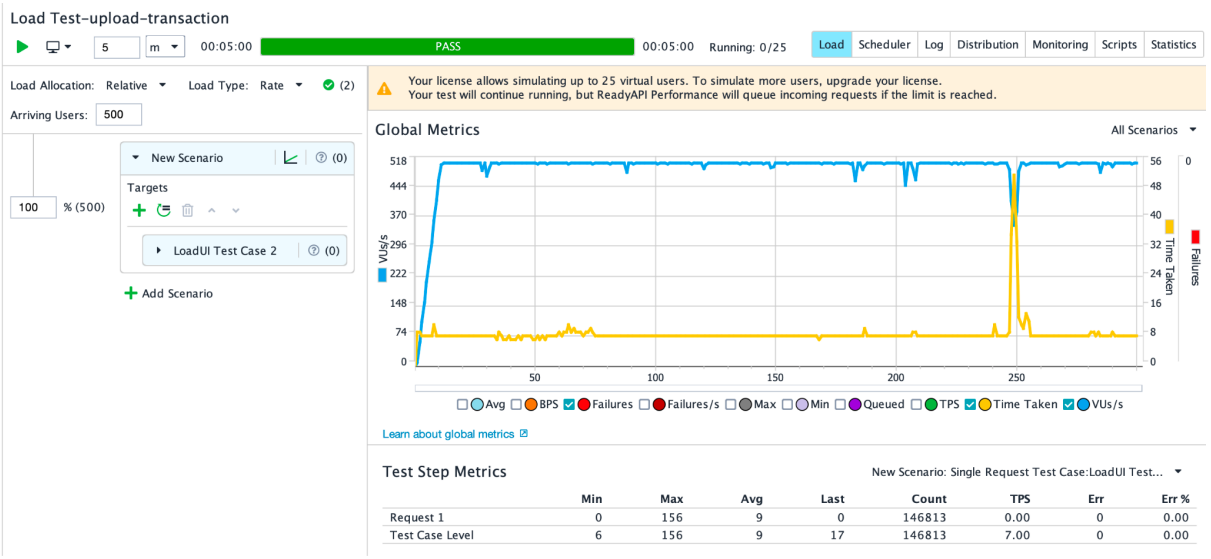
Load Testing using Smartbear for Backend server


Load Test -add-transaction
5
m
00:00:00
PASS
00:05:00
Running: 0/25
Load
Scheduler
Log
Distribution
Monitoring
Scripts
Statistics
Load Allocation: Relative
Load Type: Rate
(2)
Arriving Users: 1000
Your license allows simulating up to 25 virtual users. To simulate more users, upgrade your license.
Your test will continue running, but ReadyAPI Performance will queue incoming requests if the limit is reached.
100
% (1000)
New Scenario
(0)
Targets
LoadUI Test Case 4
(0)
Add Scenario
Global Metrics
All Scenarios
VUs/s
Time Taken
Failures
Avg
BPS
Failures
Failures/s
Max
Min
Queued
TPS
Time Taken
VUs/s
Learn about global metrics
Test Step Metrics
New Scenario: Single Request Test Case:LoadUI Test...
Min Max Avg Last Count TPS Err Err %
Request 1 6 132 11 9 219659 0.00 0 0.00
Test Case Level 6 132 11 10 219660 0.00 0 0.00

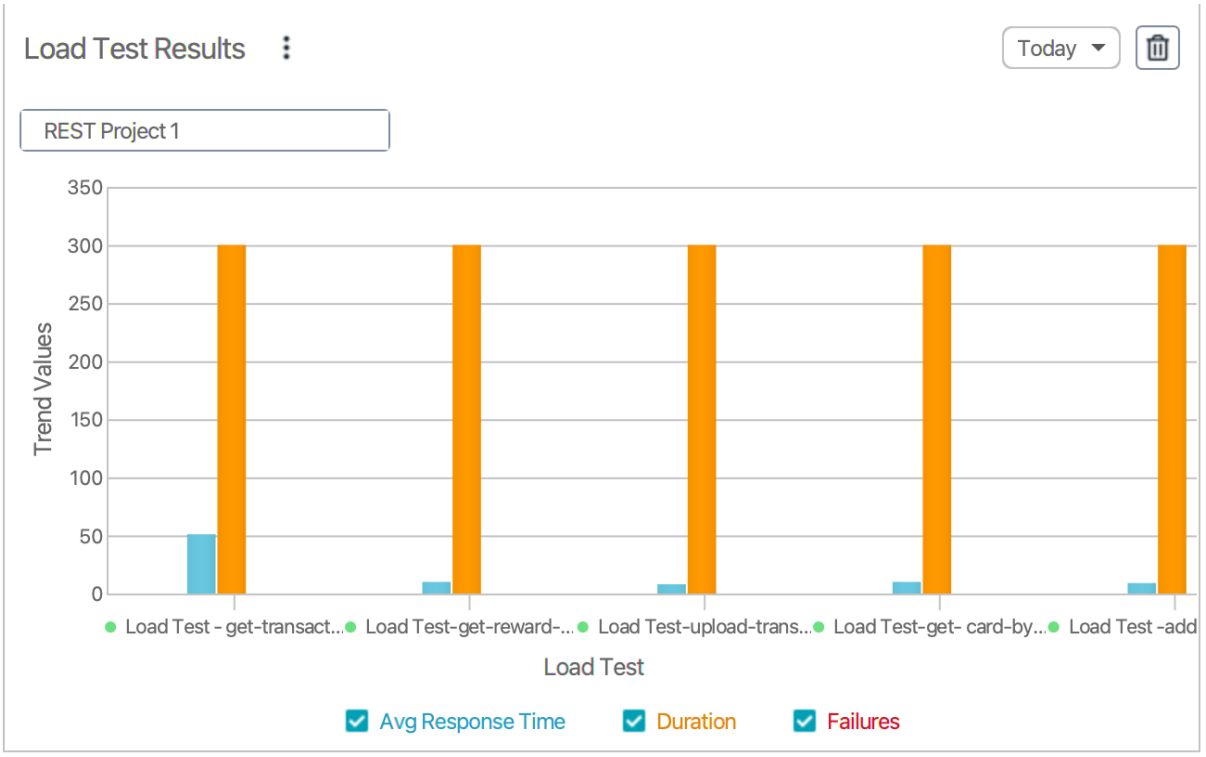

Load Test-get- card-by-user
5
m
00:05:00
PASS
00:05:00
Running: 0/25
Load
Scheduler
Log
Distribution
Monitoring
Scripts
Statistics
Load Allocation: Relative
Load Type: Rate
(2)
Arriving Users: 1000
Your license allows simulating up to 25 virtual users. To simulate more users, upgrade your license.
Your test will continue running, but ReadyAPI Performance will queue incoming requests if the limit is reached.
100
% (1000)
New Scenario
(0)
Targets
LoadUI Test Case 1
(0)
Add Scenario
Global Metrics
All Scenarios
VUs/s
Time Taken
Failures
Avg
BPS
Failures
Failures/s
Max
Min
Queued
TPS
Time Taken
VUs/s
Learn about global metrics
Test Step Metrics
New Scenario: Single Request Test Case:LoadUI Test...
Min Max Avg Last Count TPS Err Err %
Request 1 6 2242 12 16 213611 0.00 0 0.00
Test Case Level 6 2242 12 10 213605 0.00 0 0.00


Load Test-get-reward-by-user
5
m
00:05:00
PASS
00:05:00
Running: 0/25
Load
Scheduler
Log
Distribution
Monitoring
Scripts
Statistics
Load Allocation: Relative
Load Type: Rate
(2)
Arriving Users: 1000
100
% (1000)
New Scenario
(0)
Targets
LoadUI Test Case 3
(0)
Add Scenario
Your license allows simulating up to 25 virtual users. To simulate more users, upgrade your license.
Your test will continue running, but ReadyAPI Performance will queue incoming requests if the limit is reached.
Global Metrics
All Scenarios
VUs/s
Time Taken
Failures
Avg
BPS
Failures
Failures/s
Max
Min
Queued
TPS
Time Taken
VUs/s
Learn about global metrics
Test Step Metrics
New Scenario: Single Request Test Case:LoadUI Test...
Min Max Avg Last Count TPS Err Err %
Request 1 0 226 12 0 272389 0.00 0 0.00
Test Case Level 6 226 12 14 272389 0.00 0 0.00


Load Test - get-transaction-by-user
5
m
00:05:00
PASS
00:05:00
Running: 0/25
Load
Scheduler
Log
Distribution
Monitoring
Scripts
Statistics
Load Allocation: Relative
Load Type: Rate
(2)
Arriving Users: 1000
100
% (1000)
New Scenario
(0)
Targets
Test Case
(0)
Add Scenario
Your license allows simulating up to 25 virtual users. To simulate more users, upgrade your license.
Your test will continue running, but ReadyAPI Performance will queue incoming requests if the limit is reached.
Global Metrics
All Scenarios
VUs/s
Time Taken
Failures
Avg
BPS
Failures
Failures/s
Max
Min
Queued
TPS
Time Taken
VUs/s
Learn about global metrics
Test Step Metrics
New Scenario: https://server.itsag1t4.com Test Sui...
Min Max Avg Last Count TPS Err Err %
Request 1 8 958 62 53 106150 0.00 0 0.00
Test Case Level 8 958 62 22 106150 0.00 0 0.00



Blue line: Number of virtual user sending request per second
Yellow line: Time take to finish one test case



Average response time for different requests to backend server

Test	No.of virtual user per second	Average response time
Get transaction by user id	1000	51ms
Get reward by user id	1000	10ms
Upload transaction File	500	10ms

Get card by user id	1000	10ms
Add transaction	500	9ms