Spend Transaction Processing

Group 4 Final Presentation

Content

1. Project Context (Business needs, Stakeholders, Key use cases)
2. Solution Demo
3. Architectural Solution Overview
4. Solution View
5. Budget Comparison
6. Maintainability Design
7. Availability Design
8. Security Design
9. Performance Design

Business Needs

Stakeholder	Stakeholder Description	Permissions
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	**Upload** customers transaction details daily	Write access to RDS
Customers (Bank users)	**View** benefits for each eligible transaction	Read permission to CloudFront

Use Case #1: File Upload & Exclusion Processing

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

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.

Use Case #2: Campaigns Management

This use case specifies how the client (Ascenda admin) can manage campaign data.

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.

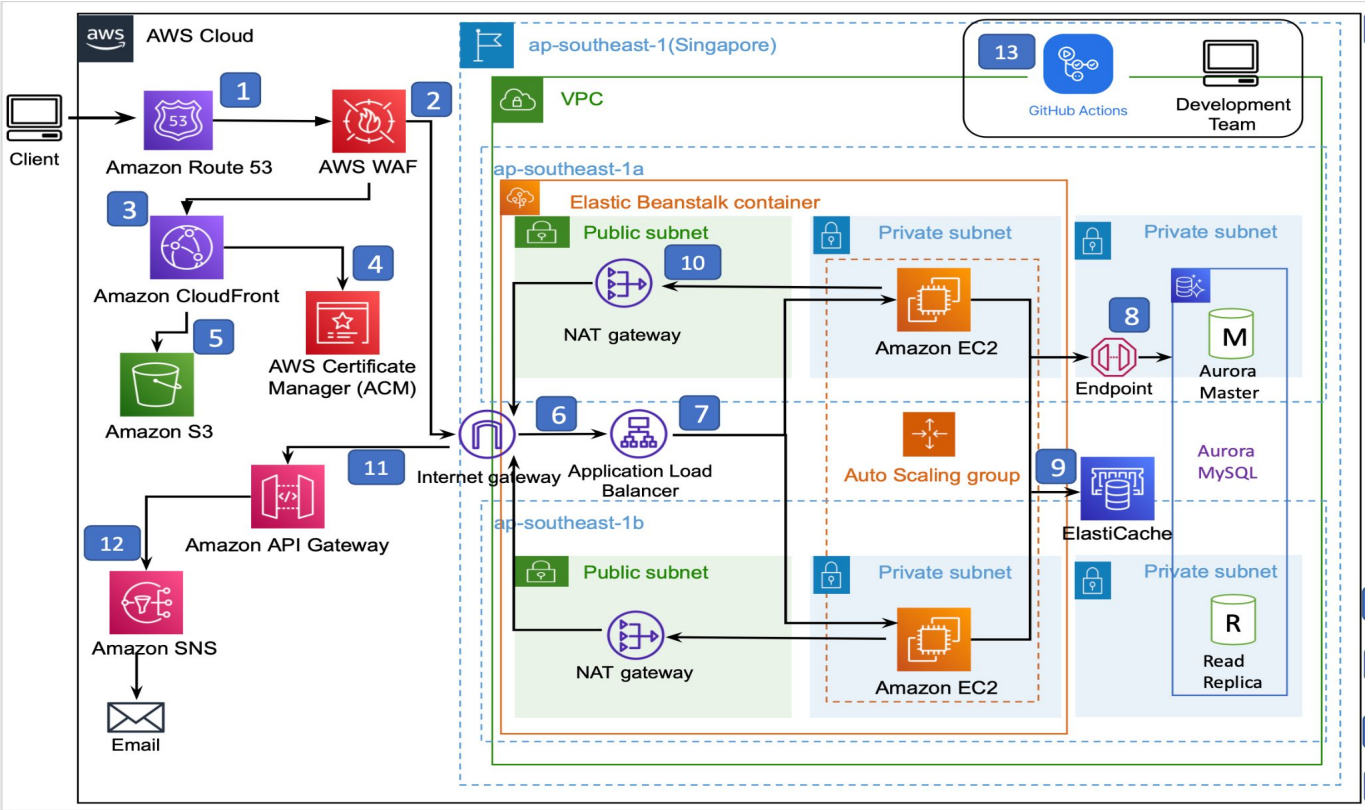
Use Case #3: Client Transaction View

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

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.

Demo

Solution Overview



Solution View - Ease of Maintainability

1. **CI/CD 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. **AWS Elastic BeanStalk**: EBS automatically handles the **deployment**, from capacity provisioning, load balancing, auto-scaling to application health monitoring.

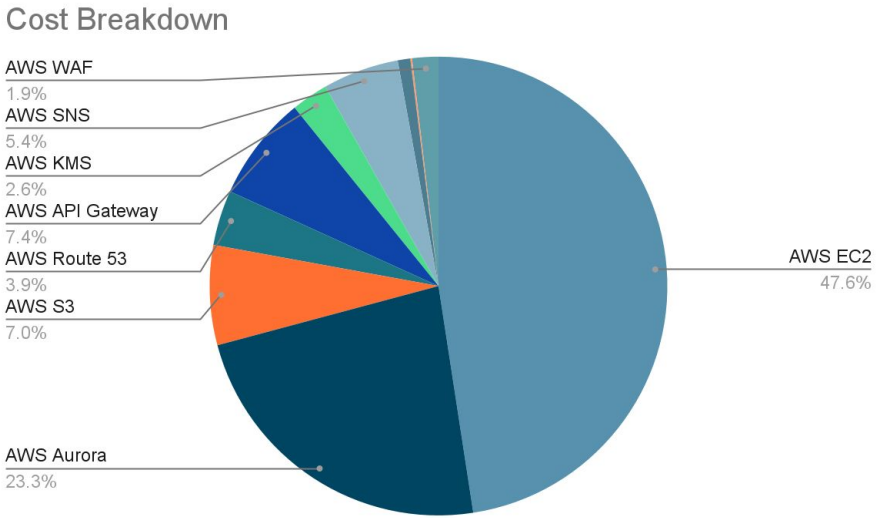
3. **AWS S3**: **Hosts** the static frontend website that is **publicly accessible** and maintains bucket versioning, ACL to grant read/write permissions and provides metrics for analysis purposes

Solution View - Integration Endpoints

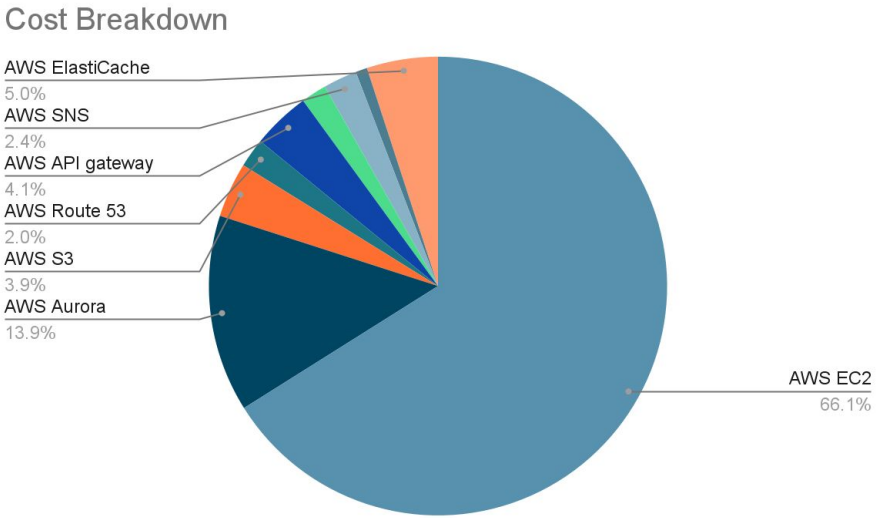
Source System	Destination System	Protocol	Format	Communication Mode
Python Flask	React	HTTPS	JSON	Asynchronous
Web Browser	Route 53	HTTPS	JSON	Synchronous
Route53	CloudFront	HTTPS	JSON	Synchronous
CloudFront	S3	HTTPS	JSON	Synchronous
API Gateway	AWS SNS	HTTPS	JSON	Asynchronous
Aurora	Read replica	HTTPS	MySQL	Asynchronous
Aurora	Write	HTTPS	MySQL	Asynchronous
AWS SNS	Email subscriber	HTTPS	JSON	Asynchronous

Budget Comparison

Proposed Total Cost: ~ 268.92 USD/ month



Actual Total Cost: ~ 610.11USD/ month

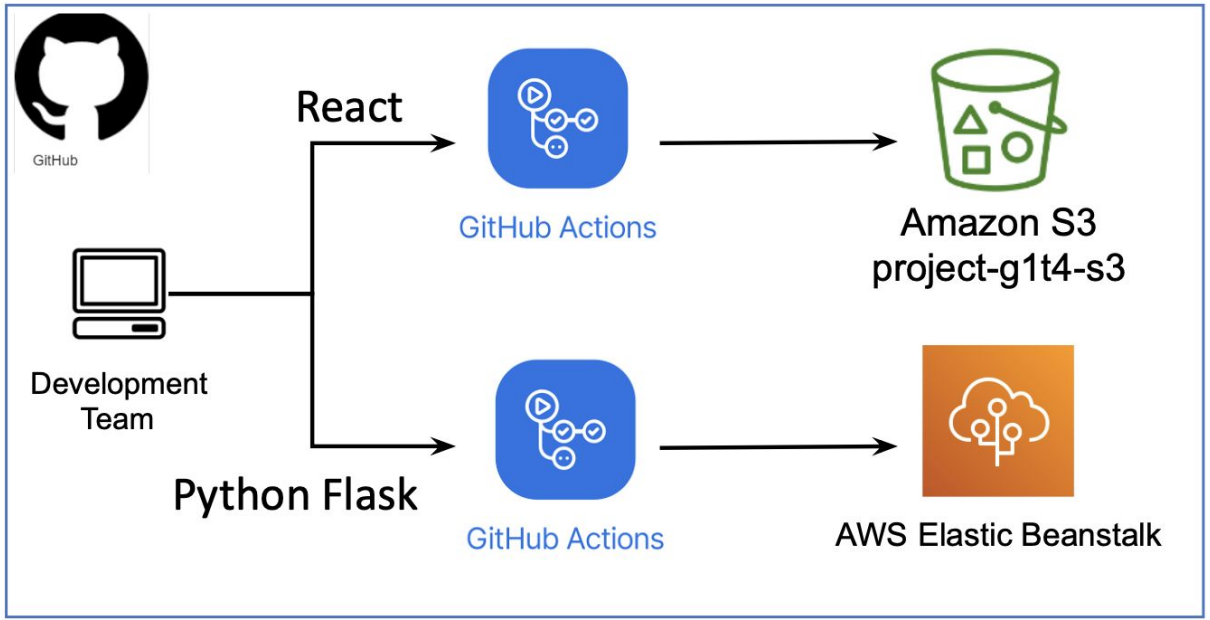


Main Differences

Services	Description	Final Cost	Proposed Cost
AWS EC2	2 x **t2.2xlarge*** instances with VPC and ALB in 2 different AZs	~402.13 USD/month	~128.41USD/month
AWS Aurora MySQL-serverless	1 ACU with 80 GB database storage and backup storage	~84.61 USD/month	~62.41 USD/month
AWS S3	S3 Standard 20GB/month, S3 Glacier 5TB/month	~23.65 USD/month	~18.99 USD/month
AWS ElastiCache	1 standard node, reserved for 1 year, t2.medium cache	~30.66 USD/month	NA
AWS Codepipeline & Codebuild & Codedeploy	-	NA	~5 USD/month

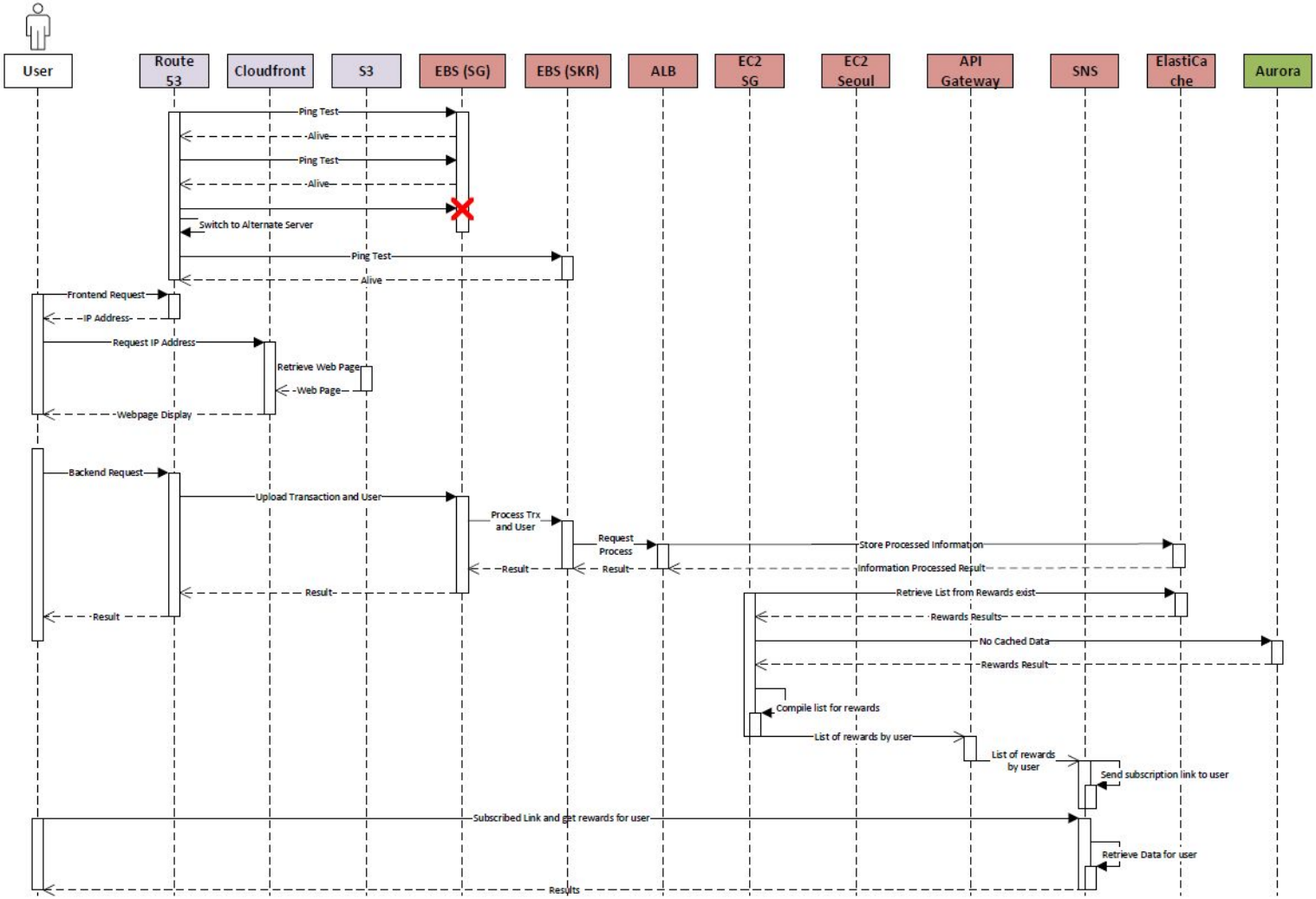
Maintainability Design

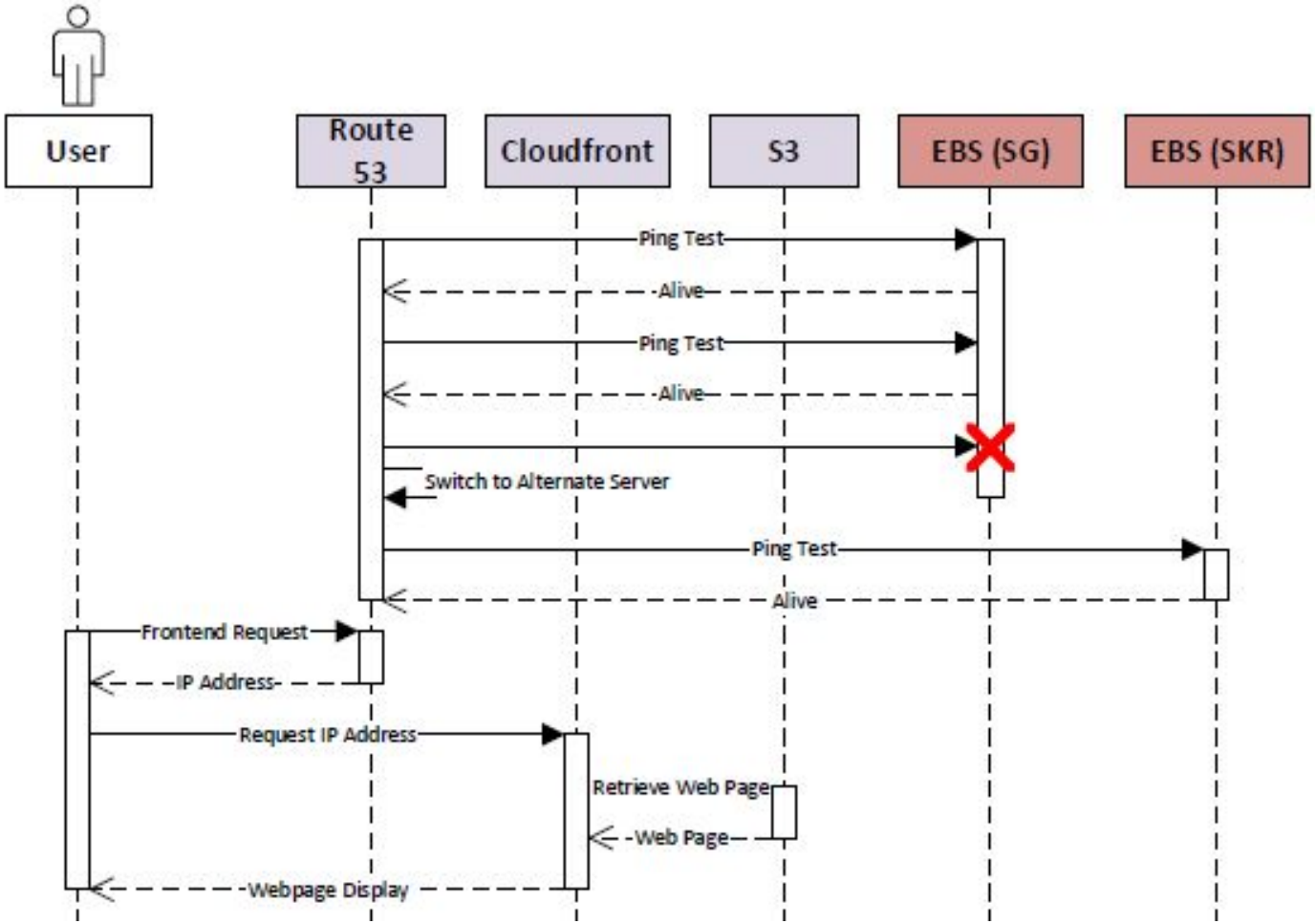
Development Pipeline (CI/CD)

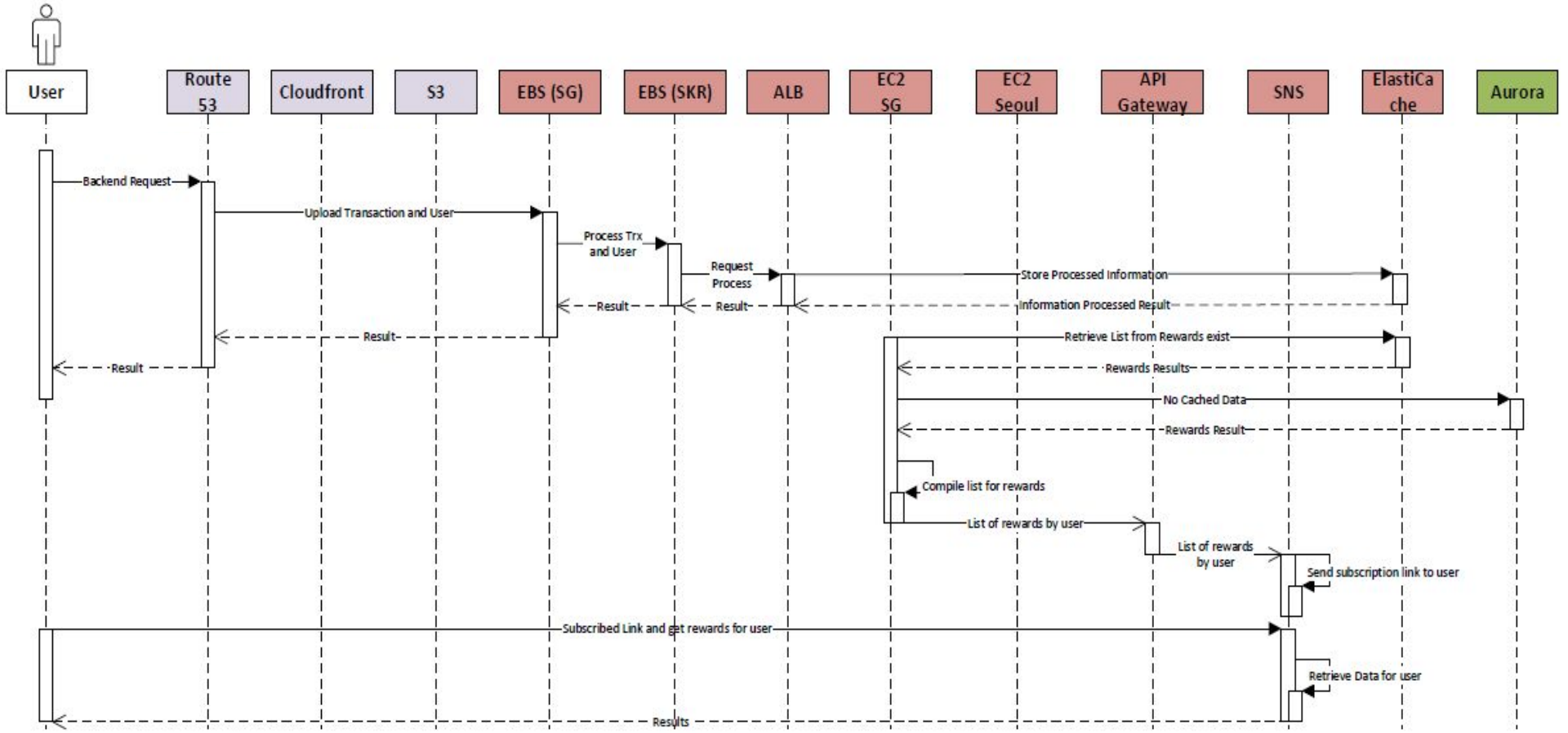


- Cypress used for end–to-end testing
- Pytest used for function unit testing
- Email notification to developer if any failure action e.g testing, deployment

Availability Design

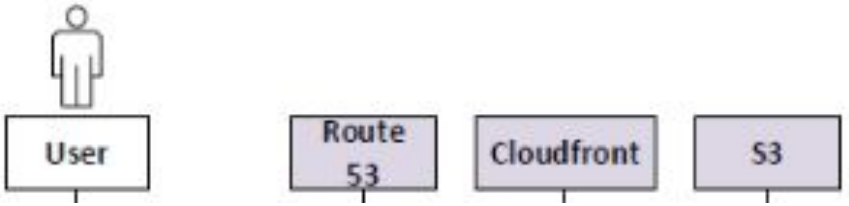

User
Route 53
Cloudfront
S3
EBS (SG)
EBS (SKR)
ALB
EC2 SG
EC2 Seoul
API Gateway
SNS
ElastiCache
Aurora
Ping Test
Alive
Ping Test
Alive
Switch to Alternate Server
Ping Test
Alive
Frontend Request
IP Address
Request IP Address
Retrieve Web Page
Web Page
Webpage Display
Backend Request
Upload Transaction and User
Process Trx and User
Request Process
Store Processed Information
Result
Result
Information Processed Result
Result
Result
Retrieve List from Rewards exist
Rewards Results
No Cached Data
Rewards Result
Compile list for rewards
List of rewards by user
List of rewards by user
Send subscription link to user
Subscribed Link and get rewards for user
Retrieve Data for user
Results


User
Route 53
Cloudfront
S3
EBS (SG)
EBS (SKR)
Ping Test
Alive
Ping Test
Alive
Switch to Alternate Server
Ping Test
Alive
Frontend Request
IP Address
Request IP Address
Retrieve Web Page
Web Page
Webpage Display


User
Route 53
Cloudfront
S3
EBS (SG)
EBS (SKR)
ALB
EC2 SG
EC2 Seoul
API Gateway
SNS
ElastiCache
Aurora
Backend Request
Upload Transaction and User
Process Trx and User
Request Process
Store Processed Information
Result
Result
Information Processed Result
Result
Result
Retrieve List from Rewards exist
Rewards Results
No Cached Data
Rewards Result
Compile list for rewards
List of rewards by user
List of rewards by user
Send subscription link to user
Subscribed Link and get rewards for user
Retrieve Data for user
Results

Frontend Components (Availability)

- Users are redirected by Route53 to CloudFront.
- Cloudfront will retrieve web page display from S3 and will display to user.
- Cached and High Availability.

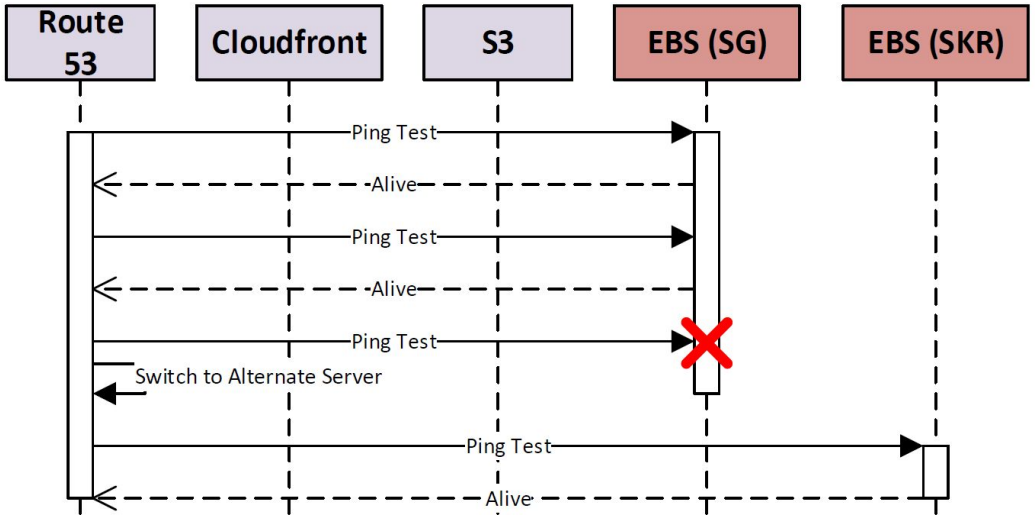


Server Components (Availability)

- Elastic Beanstalk environments (Managed Service)
  - **Ap-southeast-1**
  - **Ap-northeast-2**
- Distributes traffic to minimally two availability zones
  - Auto Scaling of EC2 Instances
  - Routing with Application Load Balancer
  - Health Monitoring in the event of failure
- Recommendations:
  - ElastiCache (Redis) configured for high availability
    - Multi-AZ
    - Automatically fails over to replicas

Regional Failover

- **Ap-southeast-1 (Singapore)**
  - Primary Region
- **Ap-northeast-2 (Seoul)**
  - Secondary Region
- **Route53 (DNS Service)**
  - Health Checks to monitor Elastic Beanstalk
  - Failover to secondary region by routing traffic

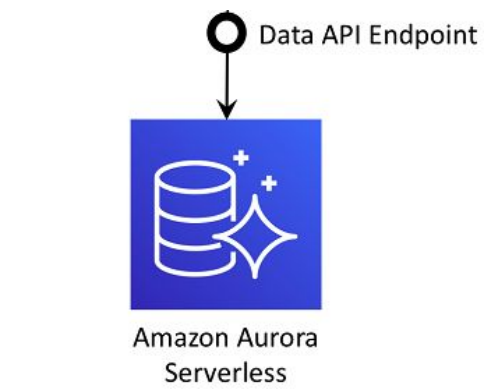


SeoulHealthCheck	a day ago — now	Healthy	http://ltsag1t42022backendseoul3-env.e…	1 of 1 in OK	14645430-d90c-4afb-8860-463e9b11f78a	
SingaporeELBHC	a day ago — now	Healthy	http://ltsag1t42022backend2-env.eba-e…	1 of 1 in OK	b241b410-2197-41b1-87db-59e41a390a5b	

server.itsag1t4.com	A	Failover	Secondary	itsag1t42022backendseoul3-env.eba-gfpdvitn.ap-northeast-2.elasticbeanstalk.com.	
server.itsag1t4.com	A	Failover	Primary	itsag1t42022backend2-env.eba-e27pmsbd.ap-southeast-1.elasticbeanstalk.com.	

Database Components (Availability)

- Fully Managed Service Designed for speed, availability and scalability
- High Availability achieved through Global Database
  - Auto-scaling
  - Distributed
  - Fault-tolerant
  - Self-healing system
- Implemented both Aurora & Serverless
  - Recommendations: **Serverless v2**
- **AWS Backup** In unlikely event of downtime



Security Design

Confidentiality

Potential Threat: **Packet Sniffing**
Packets sent via HTTP are not encrypted; Attackers can easily sniff for sensitive data transmitted in these packets.

Assets: **Amazon S3**, **Amazon EC2**

Mitigation: Using **HTTPS protocol** for automatic encryption of network packets to thwart sniffing attacks

Confidentiality

Potential Threat: **SSL Strip**
SSL stripping allows attackers to force a unencrypted response from the server by hijacking packets and sending them to its respective HTTP endpoint. As packets sent over HTTP are not encrypted, sensitive data can be leaked this way.

Assets: **Amazon S3**, **Amazon EC2**

Mitigation: Automatic HTTPS redirection forces all connections to the server to only be HTTPS connections. Since data transmitted through HTTPS is encrypted by default, attackers will not be able to view the data being transmitted.

Availability

Potential Threat: **Distributed Denial of Service (DDoS)**
As the API gateway is exposed to the internet, attackers can flood the endpoints with request to overload the servers, causing a denial of service.

Assets: **Amazon S3**

Mitigation: **AWS Shield** provides always-on detection and automatic inline mitigations against DDoS attacks.

Authenticity

Potential Threat: **SQL Injection**
Allowing SQL injection attacks will leave the database vulnerable to unauthorised access. Attackers who do not have the credentials to the database are able to view and even modify the data in the database using this attack.

Assets: **Amazon S3**

Mitigation: **AWS WAF** protects against SQL injection attacks by using SQL injection match conditions. If a request contains malicious SQL code, the request will be automatically blocked.

Authenticity

Potential Threat: **Unauthorised access**
Without the use of API keys or implementing proper access controls, any unauthorised individual will be able to access and modify the data stored in the database through the unprotected API endpoints.

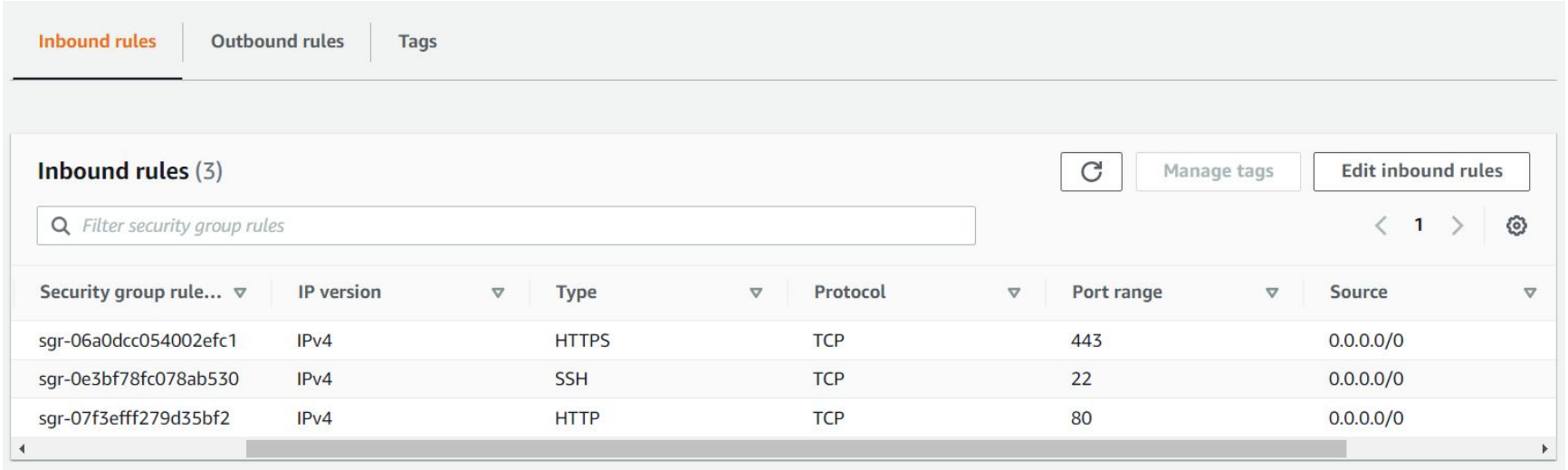
Assets: **Amazon S3**

Mitigation: Configuring **Security Groups, Access Control Lists** in AWS and using **API Keys** will allow only authorised users to access and modify the data retrieved from the API.

Security View

Security Groups

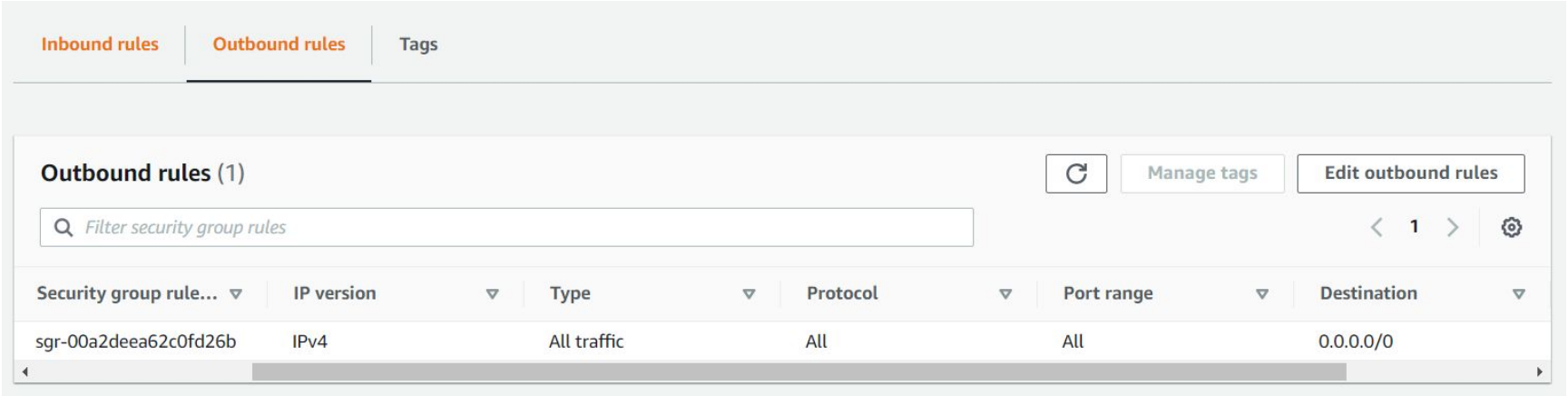
Inbound



Security View

Security Groups

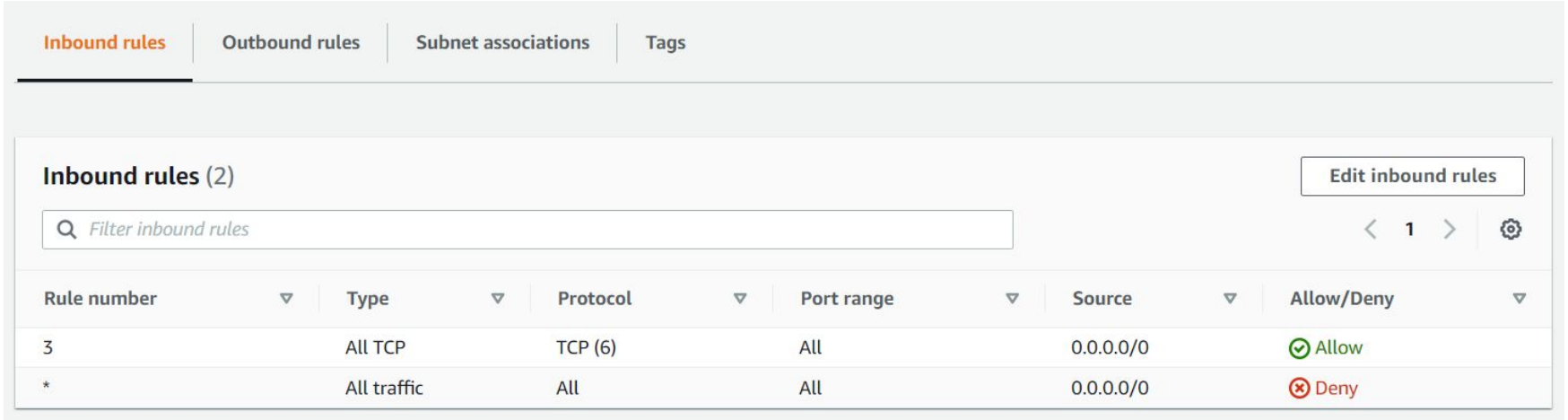
Outbound



Security View

Access Control List

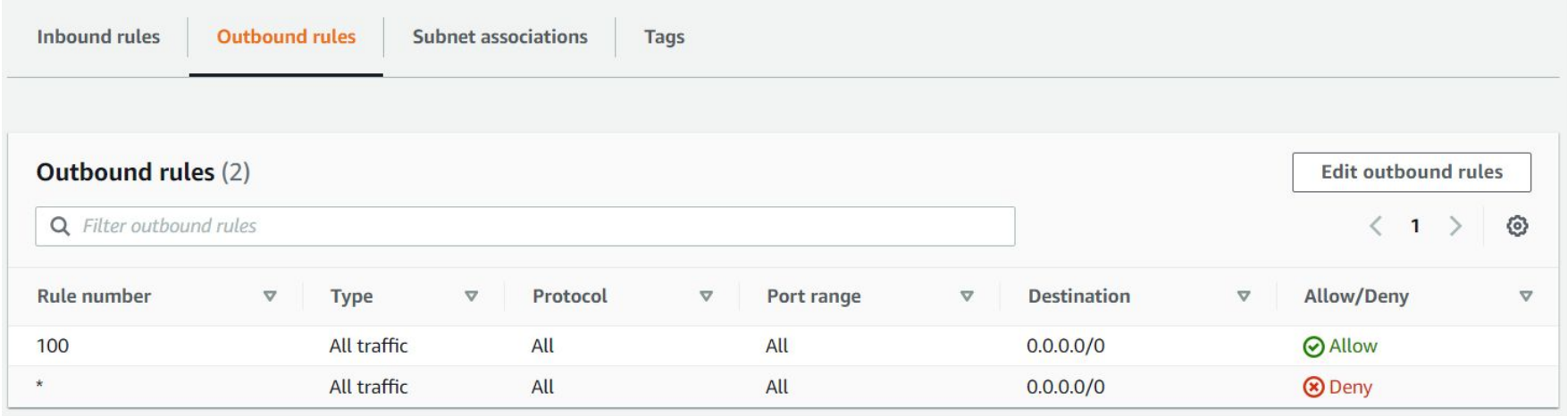
Inbound



Security View

Access Control List

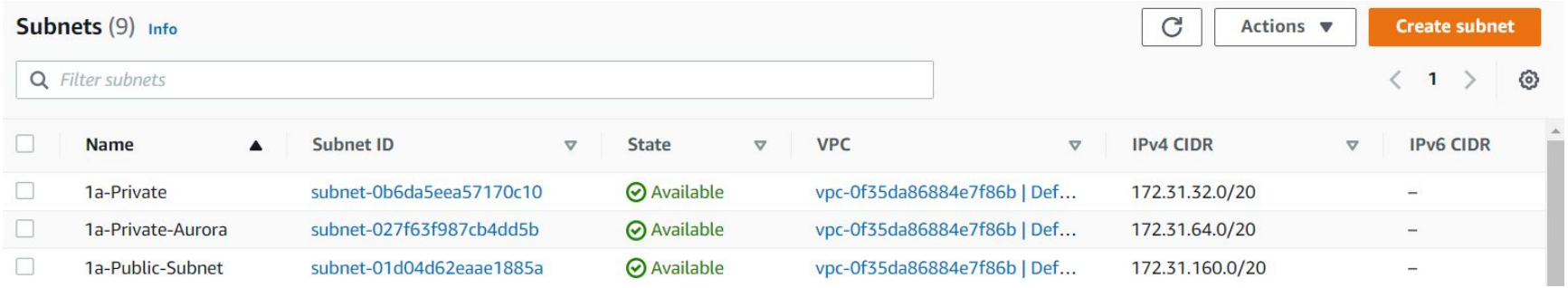
Outbound



Security View

Subnet Configuration

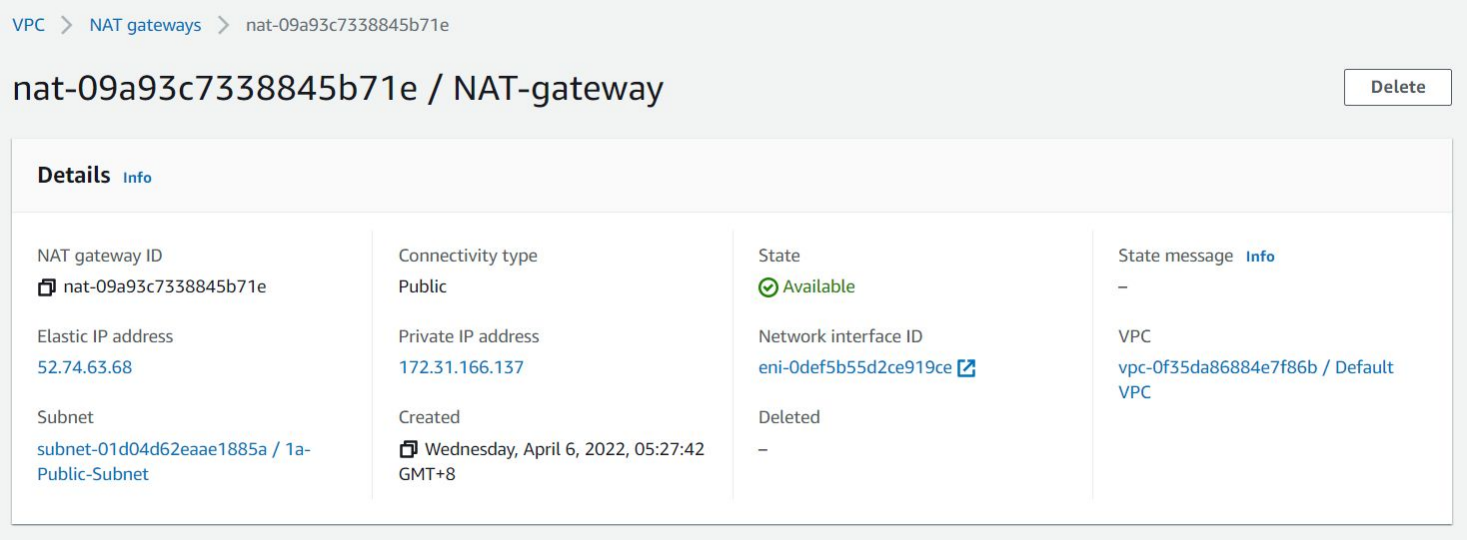
Subnets



Security View

Subnet Configuration

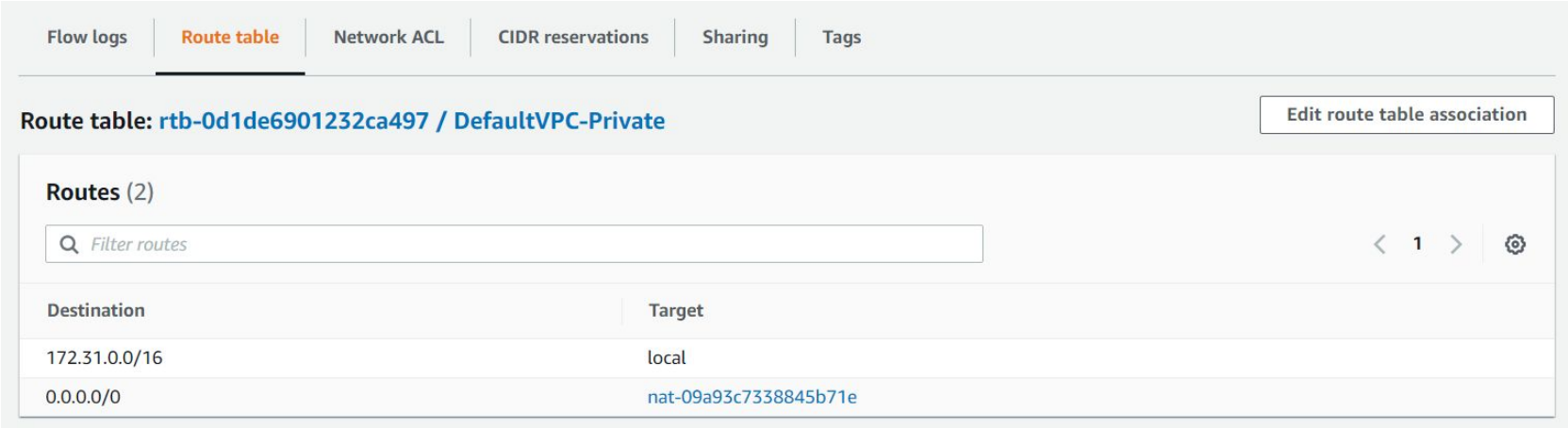
NAT Gateway



Security View

Subnet Configuration

Private Subnet



Security View

Subnet Configuration

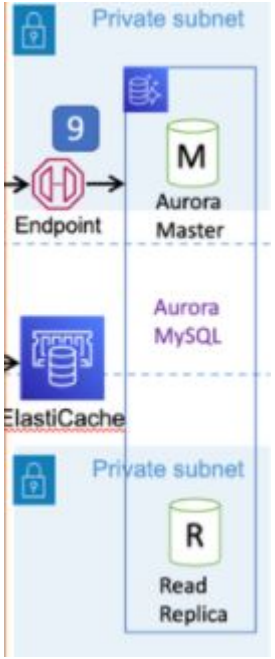
Public Subnet

Performance Design

Performance - Aurora Read Replicas

AWS Aurora Read Replica

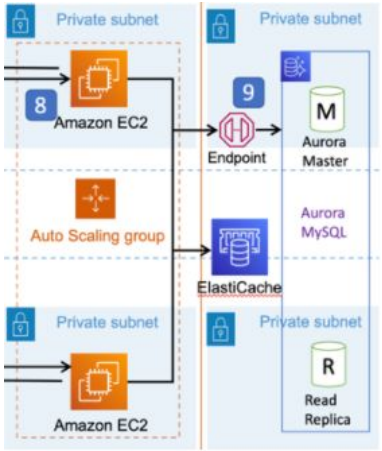
- Requests like view card transaction and campaign management are highly performant, especially during peak periods
- Read replicas can help to serve these requests, achieving horizontal scaling and improving performance

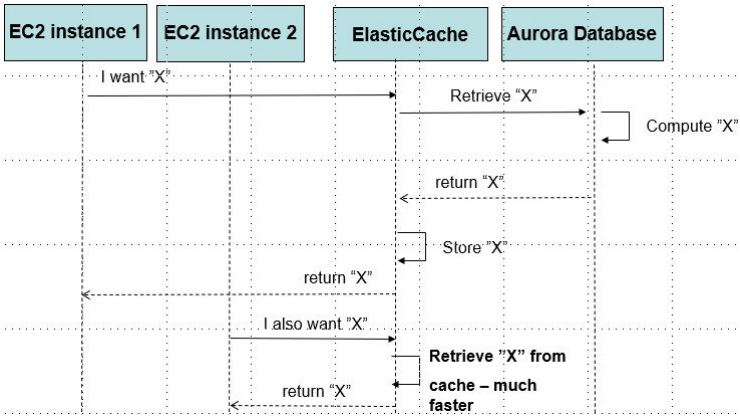


Performance - Caching

AWS ElastiCache

- Caching is used for server api endpoints using AWS ElastiCache
- Retrieve from cache instead of the server for repeated requests for the same set of data
- Reduce number of calls to server and database
- Improves performance, especially during periods of large requests

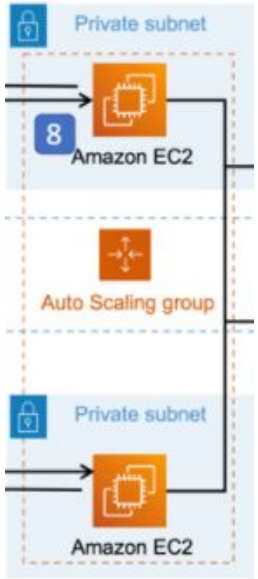


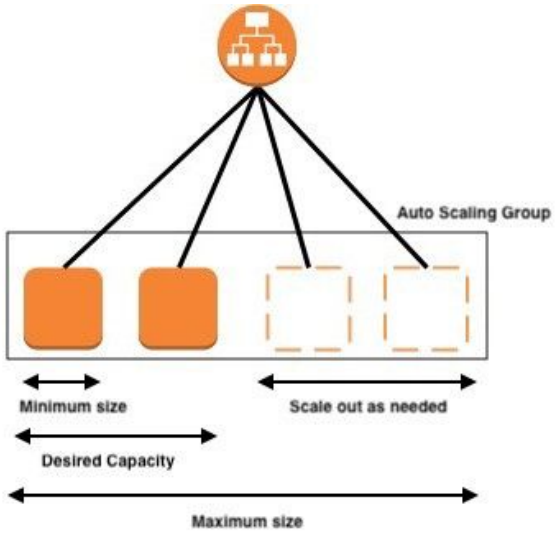


Performance - EC2 autoscaling

AWS Auto Scaling Group

- Elastic Load Balancing scales application load balancer to match the changing traffic to our website over time
- Load balancer automatically register new instances using auto-scaling to handle an increased number of requests

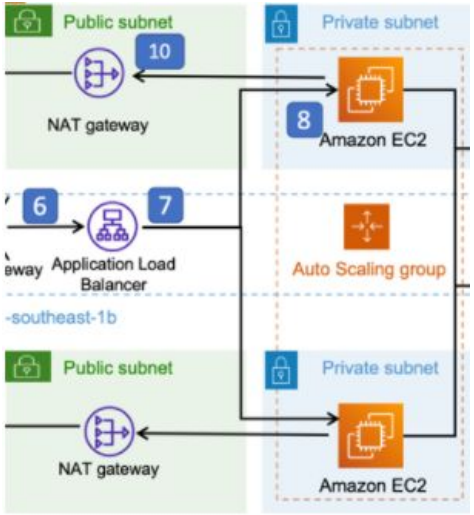


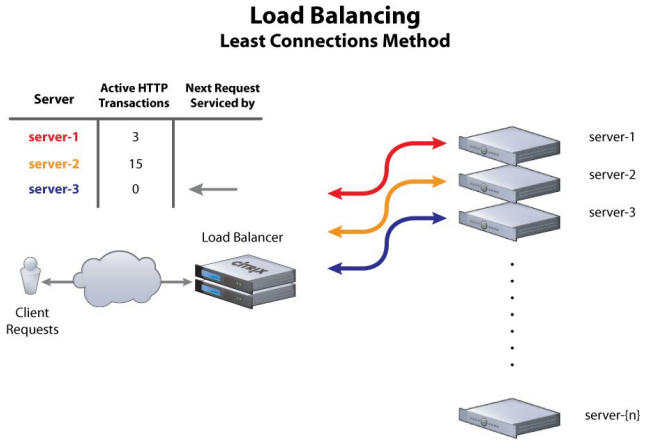


Performance - Least outstanding requests routing

- Requests for our website vary in complexity
- Load balancer route requests to target with least outstanding requests
- Targets processing long requests are not burdened with more requests
- Load is evenly spread across targets

AWS Application Load Balancer





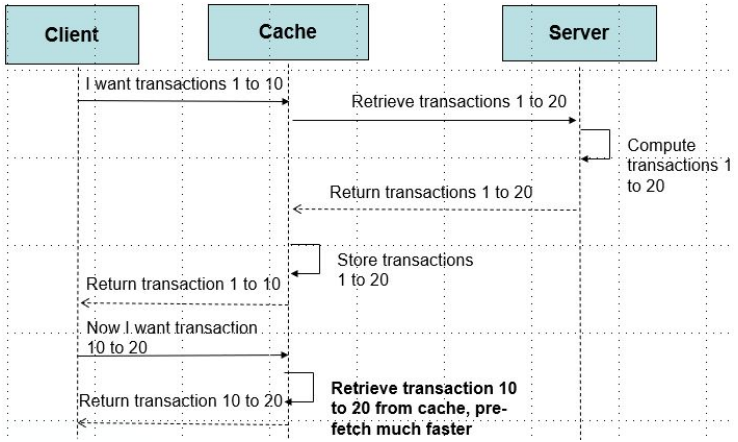
Performance - Parallel transactions

- User file and transactions file can be uploaded and processed using parallel execution
- They do not have dependencies on each other
- Saves a substantial amount of time as it usually takes around 5 mins to upload each file

Performance - Pre-fetch

- When users retrieve 10 transactions, server will pre-fetch 20 transactions
- Users can go to next page without waiting for another request to the server
- Increases speed of viewing the transactions
- Reduce number of calls to server and

increase the performance



Thank you

Group 4

SMU
SINGAPORE MANAGEMENT UNIVERSITY

Load Testing

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/ user File (1 million rows)	500	580,000 ms (9 min 40s)
Get card by user id	1000	10ms
Add transaction	500	9ms