



SMU

**SINGAPORE MANAGEMENT
UNIVERSITY**

CS301 – IT Solution Architecture

Project Report

Team Name: Team 6

Hartono Tjakrawinata Jonathan Chow (01339300)

Lixiang Wu (01404879)

Soh Yu Wei (01423316)

Sia Yan Rui (01351538)

Tian Mingze (01368170)

Zhang HongXiang (01366948)

Background and Business Needs	3
Stakeholders	3
Key Use Case	4
Quality Attributes	8
Speed	8
Scalability	8
Ease of Maintenance	8
Security	8
Resilience & Data Recovery	9
Architectural Decisions	9
Development View	13
Solution View	14
AWS Architecture Diagram	14
Deployment Diagram	14
Integration Endpoints	15
Proposed Budgets	16
Development Budget	16
Production Budget	16
Availability View	16
Security View	17
Performance View	19
Performance Test using Python Locust	19
Latency Test using Hey	20
Appendix	21
Screenshots of Hotel Booking Site	21
AWS Architecture Diagram Full	23
Deployment Diagram Full	24
Development Budget Details	24
Production Budget Details	26

Background and Business Needs

A Hotel Search and Booking System will be developed as requested by Ascenda Loyalty and used to power whitelabelled hotel booking platforms on behalf of banks, airlines and loyalty programs. The key functional requirements that will be needed are: **1) *Destination Search***, **2) *Hotel search results***, and **3) *Booking data***. The quality requirements needed are: **1) *Speed***, **2) *Scalability***, **3) *Ease of Maintenance***, **4) *Data security***, and **5) *Resilience and Disaster recovery***.

Stakeholders

Stakeholder	Stakeholder Description	Permissions
Ascenda Loyalty	Client	All resources (read + write)
Hotel Suppliers	Provides pricing data for hotels	N.A.
Users	Customers	N.A.
Frontend Devs	Develop frontend services and an intuitive UI/UX	API Gateway (read + write) CloudWatch (read + write) Amplify (read + write)
Backend Devs	Develop backend services and integrate API into application	Lambda (read + write) API Gateway (read + write) Aurora (read + write) CloudSearch (read + write) CloudWatch (read + write)

Key Use Case

Use Case Title - Enter destination search	
Use Case ID	UC-001
Description	This is a text-based auto-complete search for hotels in the destination. The autocomplete suggestions have to be fast for customers to help them in constructing their search query.
Actors	Customers, Hotel API, CloudSearch
Main Flow of events	<ol style="list-style-type: none">1. Customer enters the query destination in the search bar.2. Amazon CloudSearch ensures low latency and high throughput performance. CloudSearch will recommend autocomplete destinations based on configured options.3. Relevant auto-completed suggestions are listed as the search query changes.
Alternative Flow of events	No suitable suggestions for customer's query
Pre-conditions	Customers have to know what destination(s) they want to search
Post-conditions	Customers can find and select the destination that they're looking for in the suggestions

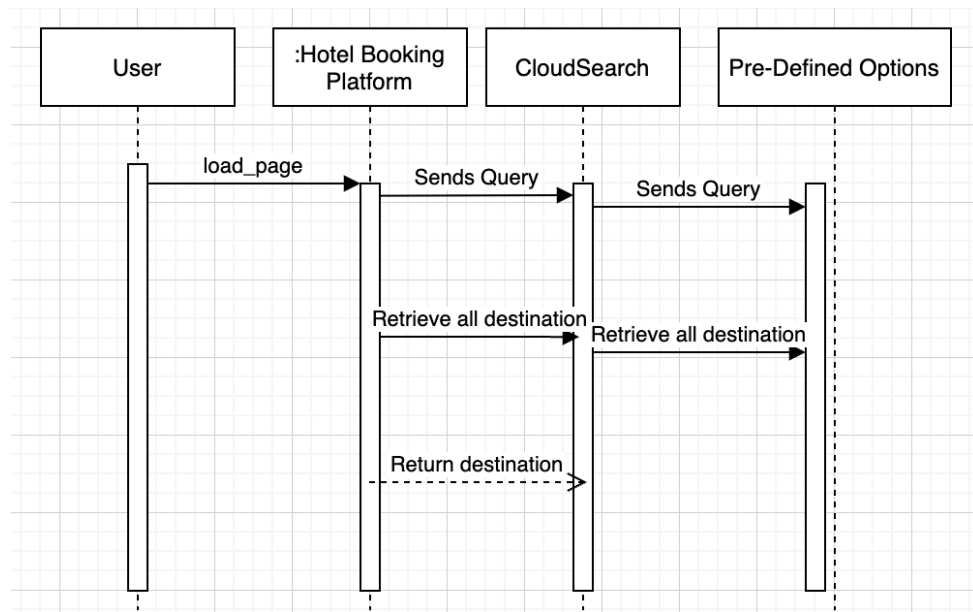


Figure UC1: Sequence Diagram of UC.1

Use Case Title - View and select hotel search results	
Use Case ID	UC-002
Description	For a given destination, dates of stay and number of rooms/guests, a list of matching hotels and the cheapest room for each hotel. The result list is populated as soon as responses are returned from the fastest Hotel API, so customers do not have to wait for the slower responses to fully load. A customer should be able to see the best price among the three endpoints when the slowest responses returned result. (Before the slowest endpoint returns result, the customer is able to see current best price)
Actors	Customers, Hotel API
Main Flow of events	<ol style="list-style-type: none"> 1. Customer clicks on the search button 2. A list of hotels from the customer's searched destination will be displayed, in order of the cheapest room first.

	<p>3. More hotels will continue to be loaded as the responses are returned from the Hotel API</p> <p>4. Customer clicks “View Details” on their desired hotel to be redirected to the room booking screen.</p>
Alternative Flow of events	No hotels available for the chosen destination
Pre-conditions	Hotel pricing and static data APIs are available
Post-conditions	Customers are able to find a hotel that suits their needs

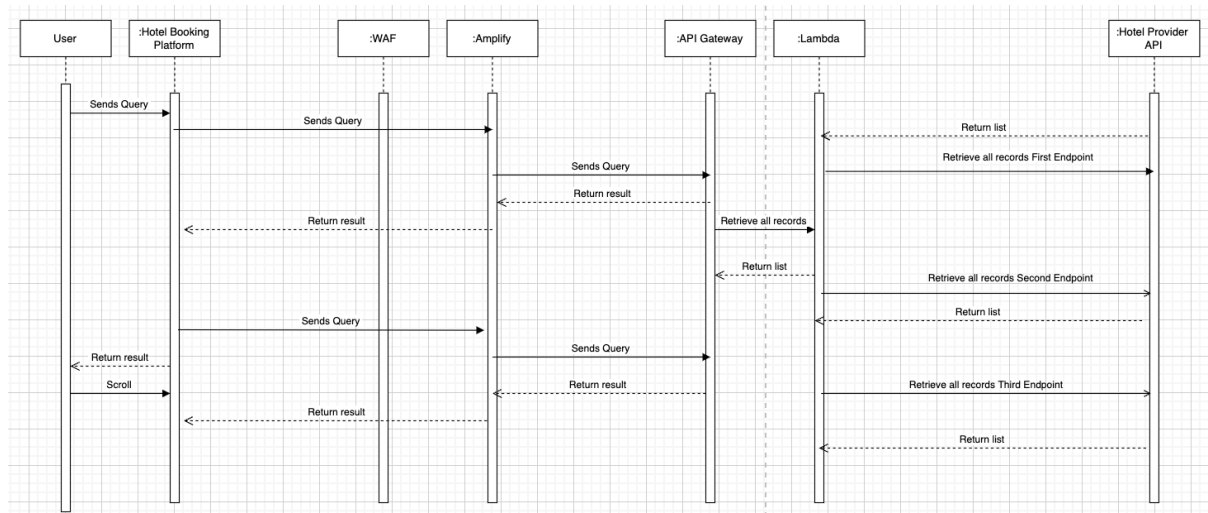


Figure UC2: Sequence Diagram of UC.2

Use Case Title - Confirm booking and make payment	
Use Case ID	UC-003
Description	A customer will proceed to pay for the reserved hotel room.
Actors	Customers, Payment Gateway, Booking Lambda, Payment Lambda
Main Flow of events	<ol style="list-style-type: none"> 1. Customer enters their payment details and clicks ‘Book Now’ 2. A booking confirmation page will be shown if payment is successful 3. Room booking status will be changed to “Paid”
Alternative Flow of events	<ol style="list-style-type: none"> 1. Payment is unsuccessful and customers will be prompted to try again. 2. Hotel room will be unreserved after 1 hour if payment has not been completed
Pre-conditions	Customer is able to reserve the room without any reservation conflicts with other customers
Post-conditions	Payment has been made successfully and customers have received their booking confirmation

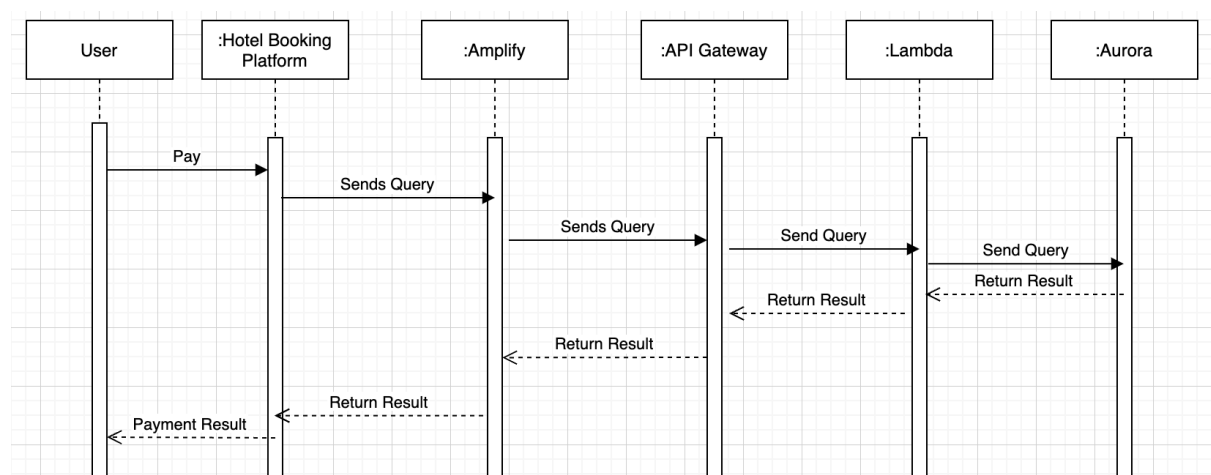


Figure UC3: Sequence Diagram of UC.3

Quality Attributes

Speed

AWS Amplify leverages the Amazon CloudFront Global Edge Network to distribute the web app to users globally with low-latency and high speed transfer of data.

Amazon CloudSearch ensures low latency and high throughput performance, even at large scale through automatic sharding and horizontal and vertical autoscaling.

Amazon Aurora creates a combination of top-tier speed of an in-memory cache with the reliability and flexibility of a relational database.

Scalability

AWS Lambda would also support scalability as it automatically scales when the number of events increases, Lambda will route those events to available instances and create new instances if needed. When the number of events decreases, Lambda will stop unused instances to free up scaling capacity for other functions. It is also able to do so without the need for configuration or deployment due to its serverless nature.

Amazon CloudSearch offers powerful autoscaling for all search domains. Amazon CloudSearch can scale the search domain's resources up or down as needed.

Amazon Aurora features a self-healing storage system that auto-scales up to 128TB per database instance.

Ease of Maintenance

AWS API Gateway allows for easy maintenance of the many APIs used within our application as it is a fully managed service by AWS, allowing the concurrent handling and processing of many API calls. A CI/CD pipeline (consisting of AWS CloudFormation and GitHub Actions) will be implemented to automate the deployment of application changes to reduce manual errors as well as time spent on deployment and testing.

Security

AWS Amplify creates a flexible, layered security perimeter against multiple types of attacks. As a managed service, AWS Amplify is protected by the AWS global network security procedures

All of these services co-reside at the AWS edge and provide a scalable, reliable, and high-performance security perimeter for applications and content. All data would be encrypted at rest in Amazon Aurora using industry standard AES-256 encryption algorithm and keys that are managed under AWS Key Management Service.

AWS IAM would allow fine-grained access control over the different services and resources used by different stakeholders, and access can be analysed across our entire AWS environment to further enforce the principle of least privilege.

Resilience & Data Recovery

AWS Lambda offers several features to help support your data resiliency and backup needs. With a hot-standby, databases can be recovered quickly without affecting the user experience.

AWS Aurora not only ensures high availability for data but also ensures high availability for the DB instance. It can tolerate a failure of an Availability Zone without any loss of data and only a brief interruption of service.

Besides, AWS API Gateway is also acting as a load balancer by directing requests to specific resources based on the endpoints being requested, ensuring the system to be highly available.

Architectural Decisions

Architectural Decision - Use CloudSearch instead of In-Memory Cache.	
ID	AD-001
Issue	In-Memory Cache does not have auto scaling features and it is time consuming to create the In-Memory Cache.
Architectural Decision	We decide to use AWS's CloudSearch service. CloudSearch provides powerful autoscaling features for all search domains. CloudSearch also provides automatic monitoring and recovery. Lastly, CloudSearch can ensure low latency and high throughput performance compared to In-Memory cache.
Assumptions	None
Alternatives	Elasticsearch
Justification	CloudSearch is a built-in SaaS service provided by AWS. It is very easy and convenient to set up and it does not require us to re-upload our data when there is a change in our configurations. Most importantly, CloudSearch is a cost-effective service. We only pay low hourly rates,

	and only for the resources we are using.
--	--

Architectural Decision - Microservices using AWS Lambda (Serverless)	
ID	AD-002
Issue	A typical monolithic application represents a single point of failure as all functions and services are tightly coupled. When a service is unavailable, the entire application will be down. Tightly coupled services will also be challenging to develop and test as several members work on the application at the same time.
Architectural Decision	<p>We developed our application with several loosely coupled microservices running on AWS Lambda that communicate with one another through RESTful API and HTTPs API.</p> <p>With this, changes to any services will not affect other services. Every developer can focus on building their services without understanding the underlying implementation of the services. Overall, this achieves ease of maintainability, development, and deployment.</p> <p>AWS Lambda serverless model would also allow the Ascendas to save on costs as its pricing is based on resource usage for serverless service and not resources provisioned.</p>
Assumptions	None
Alternatives	AWS Fargate
Justification	The AWS Lambda function is responsible for making API calls, Invoke database injection or reading data from the database. As most of the requests only take 1-2 seconds to run which lambda would allow us to pay less compared to Fargate. In addition, AWS Fargate is Amazon's solution to run docker containers without managing any servers for container orchestration which does not align with our intention in this project's architecture.

Architectural Decision - Continuous Integration & Continuous Delivery	
ID	AD-003
Issue	Manual deployment of application code is time-consuming and

	introduces human-error when deploying such as wrong file uploads, wrong region etc.
Architectural Decision	<p>From the CI/CD tools available in the market, we decided to use GitHub Actions. The pipeline was set up such that when the Lambda function codes are pushed onto our git repository, these changes will be deployed onto the corresponding lambda function hosted on the AWS.</p> <p>Additionally, we host our frontend on AWS Amplify, in which changes made to our frontend on Github will be deployed to Amplify. Amplify also has its own built-in CI/CD pipeline to install all new dependencies that are required in our new changes.</p> <p>This allows for the code changes and updates to be reflected to the users with minimal downtime or delay.</p>
Assumptions	Github Actions service is live.
Alternatives	Jenkins
Justification	Github Actions is much easier to set up because it operates in the cloud and is a managed service. The individual actions in a Github Actions workflow are isolated by default, making it possible to use them in different computing environments and easier to debug.

Architectural Decision - Database using Aurora Serverless	
ID	AD-004
Issue	A database is required to store user's booking information.
Architectural Decision	<p>Using Aurora Serverless, we are able to reduce cost by its pay-as-we-use model, as Aurora Serverless pricing is based on usage. In addition, it automatically scales up and down based on the application needs.</p> <p>For failover, Aurora Serverless supports automatic multi-AZ failover where if the DB instance for an DB cluster becomes unavailable or the Availability Zone (AZ) it is in fails, Aurora recreates the DB instance in a different AZ.</p>
Assumptions	None
Alternatives	AWS RDS

Justification	Aurora Serverless can handle resource capacity more flexibly. In Amazon RDS, a deployed database server is not scaled to a larger instance type unless its configuration is explicitly updated to a different size, which can result in up to five minutes of downtime. RDS does offer auto scaling for read replicas, but the process takes a few minutes to execute since it has to deploy additional RDS instances. Aurora Serverless' automatic scaling results in much faster deployment times, typically within 30 seconds.
----------------------	---

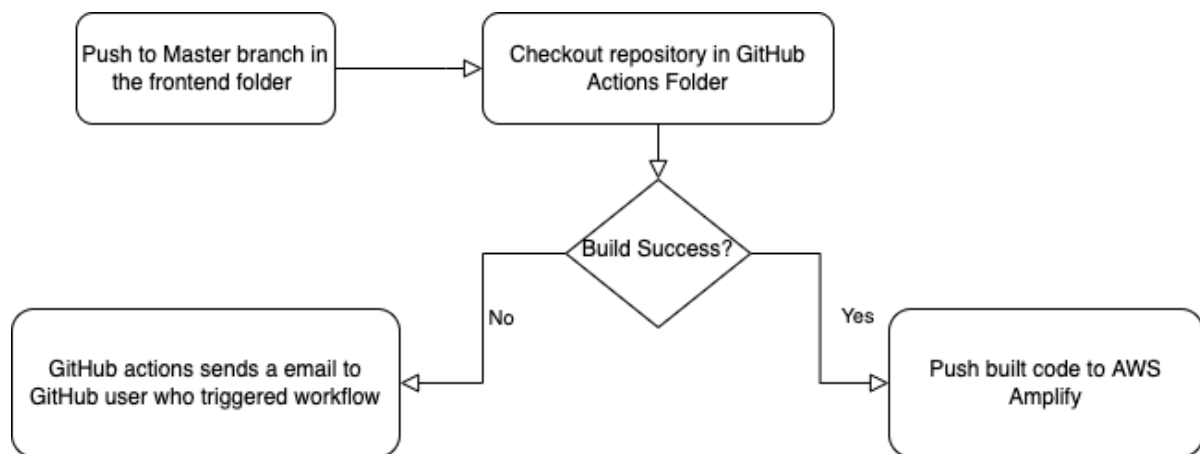
Architectural Decision - Facade using API Gateway and authentication with Authorizers	
ID	AD-005
Issue	Due to the use of Microservices architecture, services can become more tedious to manage, such as duplicate implementations for authentication, access control, and rate-limiting. The services also expose more points of entry for a malicious attacker.
Architectural Decision	<p>The use of Amazon API Gateway serves as a single point of entry for our microservices. This has several benefits:</p> <ol style="list-style-type: none"> 1. API Gateway can perform authentication for all microservices. 2. By obfuscating the microservices endpoint, we prevent them from being targeted by a DDoS attack. 3. Changes to the microservices code only require potential configuration change on the API Gateway, which makes the backend loosely coupled from the frontend. 4. Amazon API Gateway also supports caching for the responses from the microservices, which reduces latency and improves performance for the hotel booking website for the customers.
Assumptions	API Gateway is properly configured to handle all the microservices routes and authentication is done correctly.
Alternatives	Direct frontend-backend communication
Justification	API Gateway offers way too much benefits without any downsides.

Development View

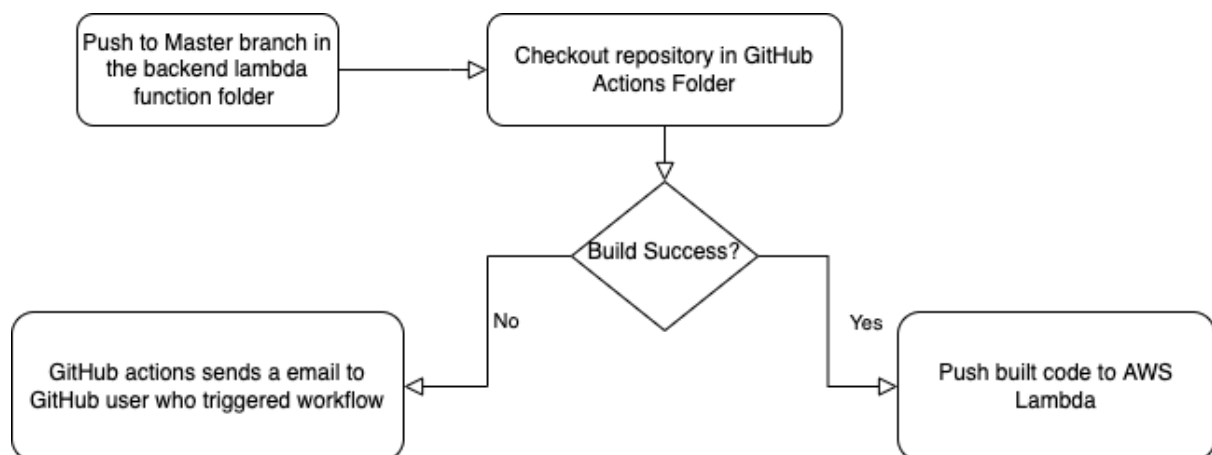
All application code is developed and pushed to our team's Github repository. For development, changes are first made locally. After local testing has been done, working code will be pushed into the master branch.

Continuous Deployment pipeline is created using Github Actions.

For the frontend code, Github Actions will monitor for any push to master branch and automatically build and deploy to the AWS Amplify and update the website in the master environment. When working code that has been tested is merged into the master branch, the same process happens but in the production environment.

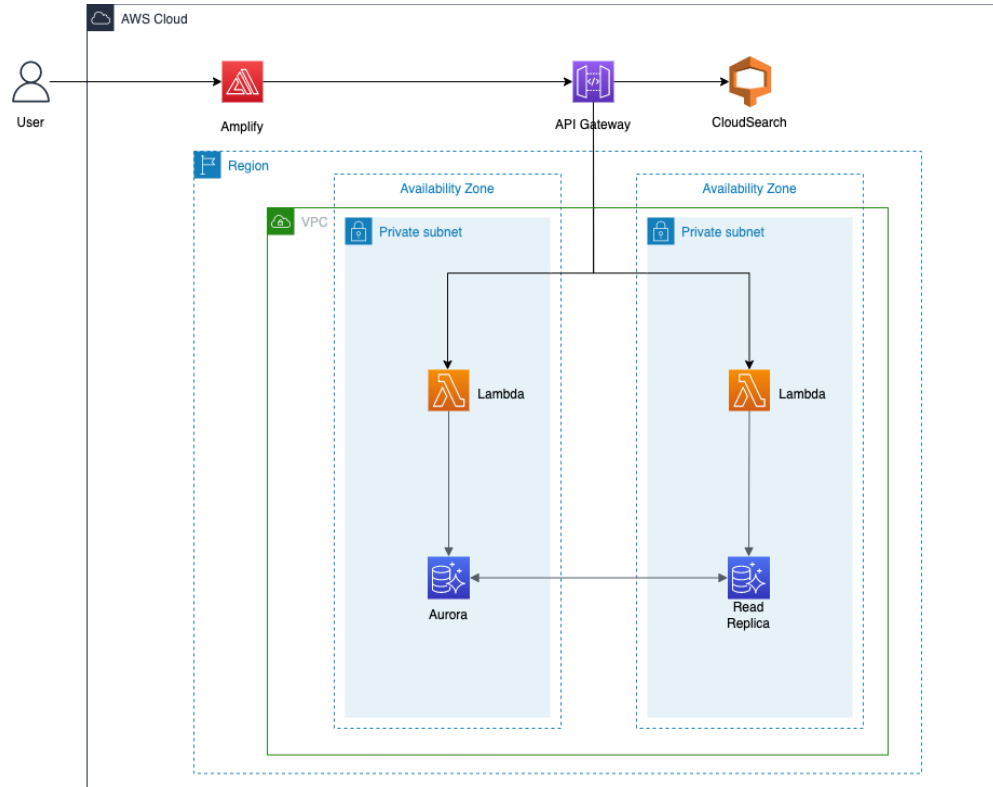


For the backend code, Github Actions will monitor for pushes of Lambda Python files on the repository and automatically deploy to AWS Lambda.

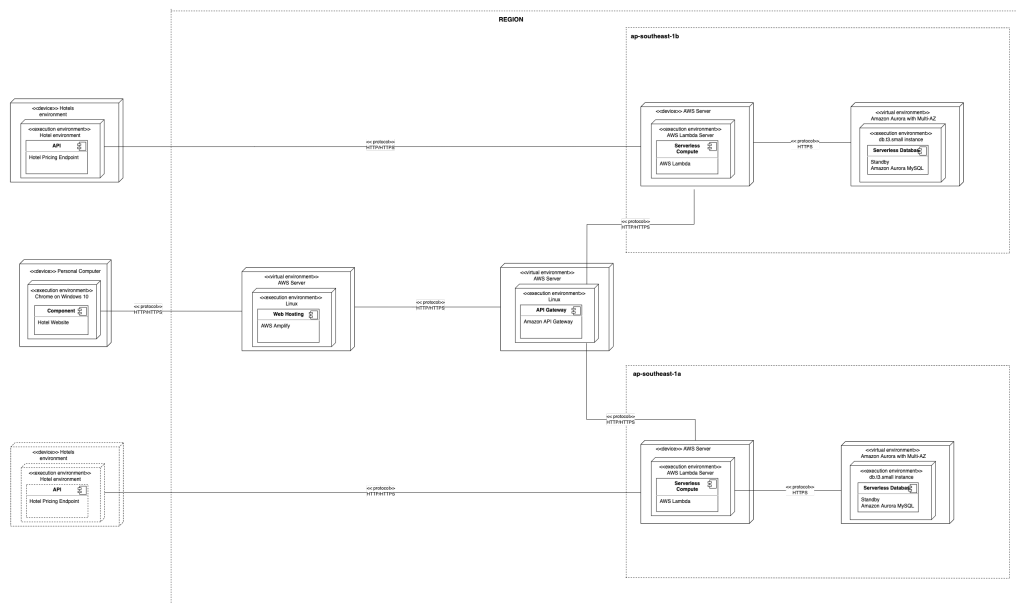


Solution View

AWS Architecture Diagram



Deployment Diagram



Integration Endpoints

Source System	Destination System	Protocol	Return Format	Communication Mode
Hotel Booking Website	Amazon API Gateway	HTTPS	JSON	Synchronous
Amazon API Gateway	Amazon CloudSearch (/destinationSearch)	HTTPS (GET)	JSON	Synchronous
Amazon API Gateway	Hotel Search Lambda (/getHotel/)	HTTPS (GET)	JSON	Synchronous
Amazon API Gateway	Room Details Lambda (/getRoomDetails)	HTTPS (GET)	JSON	Synchronous
Amazon API Gateway	Get Payment Lambda (/deployment/getAllPayment Details)	REST (GET)	JSON	Synchronous
Amazon API Gateway	Create Payment Lambda (deployment/insertBookingP ayments)	REST (PUT)	JSON	Synchronous

Proposed Budgets

Development Budget

Activity / Hardware / Software / Service	Total Cost
Development of solution, Cloud Watch, Lambda, Aurora, API Gateway, CloudSearch, Amplify	Total Monthly Cost: 86.87 USD

Production Budget

Activity / Hardware / Software / Service	Total Cost
Development of solution, Cloud Watch, Lambda, Aurora, API Gateway, CloudSearch, Amplify	Total Monthly Cost: 521.9 USD

Availability View

Node	Redundancy	Clustering			Replication (if applicable)			
		Node Config.	Failure Detection	Fail-over	Repl. Type	Session State Storage	DB Repl. Config.	Repl. Mode
AWS CloudSearch	Horizontal & Vertical Scaling	Active-Active	Heartbeat	DNS	N/A			
Amazon Amplify	Horizontal Scaling	Active-Active	Heartbeat	DNS	N/A			

Amazon API Gateway	Horizontal Scaling	Active-Active	Heartbeat	DNS	N/A
AWS Lambda (serverless)	Horizontal Scaling	Active-Passive	Ping	DNS	N/A
AWS Aurora (serverless)	Horizontal Scaling	Active-Passive	Heartbeat	DNS	N/A, Aurora Serverless handles automatic multi-AZ database replication.

Sequence diagrams are not applicable as AWS Lambda and Aurora are serverless, and their failover process is handled automatically by AWS.









Security View

No.	Asset/Asset Group	Potential Threat/Vulnerability Plan	Possible Mitigation Controls
1	Amplify Application	Distributed Denial of Services (DDoS) attacks capable of impacting availability of backend microservices	We use AWS Shield to protect against the most common, frequently occurring DDoS attacks.
2	Aurora Database	SQL Injection to gain access to confidential information	We used prepared statements so that the attackers will be unable to change the intent of the query. Data is also encrypted at rest with the key managed by AWS Secrets Manager.
3	User data in transfer	Man-in-the-middle (MITM) attack to intercept confidential information and compromise data confidentiality and integrity	Amplify protects communication between users and Amplify and between Amplify and the API gateway using TLS connections that are signed using the Signature Version 4 signing process.

Security Testing:

We conducted penetration testing on our frontend site using OWASP's Zed Attack Proxy. As seen in the alerts below, we have two vulnerabilities that should be given higher priority to be

patched. Configuring AWS Amplify to set the Content Security Policy(CSP) header would solve both issues as the CSP helps to mitigate a wide range of clickjacking attacks against our frontend site. However, the CSP header is currently not configured.

- ▼  Alerts (7)
- >  Content Security Policy (CSP) Header Not Set (2)
 - >  Missing Anti-clickjacking Header (2)
 - >  Cross-Domain JavaScript Source File Inclusion (2)
 - >  Timestamp Disclosure - Unix (15)
 - >  X-Content-Type-Options Header Missing (6)
 - >  Information Disclosure - Suspicious Comments
 - >  Re-examine Cache-control Directives (3)

Other Security Components:

The Aurora database and all Lambda functions are deployed in a private VPC which can only be accessed through the API gateway. IAM policies are used to ensure that the Cloudsearch domain is only accessible through the API gateway to prevent unauthorised direct access.

IAM policies are also used to ensure that other AWS resources can only perform the functions for which they are responsible for all Lambda functions.

Performance View

No.	Description of the Strategy	Justification
1	Use CloudSearch to ensure the performance of destination search	Amazon CloudSearch ensures low latency and high throughput performance, even at large scale through automatic sharding and horizontal and vertical autoscaling. Amazon CloudSearch can scale the search domain's resources up or down as needed.
2	Aurora to host customers' data	5X faster than a standard MySQL database. Amazon Aurora automatically grows storage as needed, up to 128TB per database instance. Aurora also provides up to 15 low latency read replicas for better performance.
3	Use Lambda to host backend services.	AWS Lambda is suited to rapidly launch as many copies of the function when the new hotel & room search requests come in. Lambda runs the code within milliseconds of an event. Since Lambda scales automatically, the performance remains consistently high as the event frequency increases.
4	Use AWS Amplify to host Front-end Application	Improved app performance with built-in support for AWS backend management

Performance Test using Python Locust



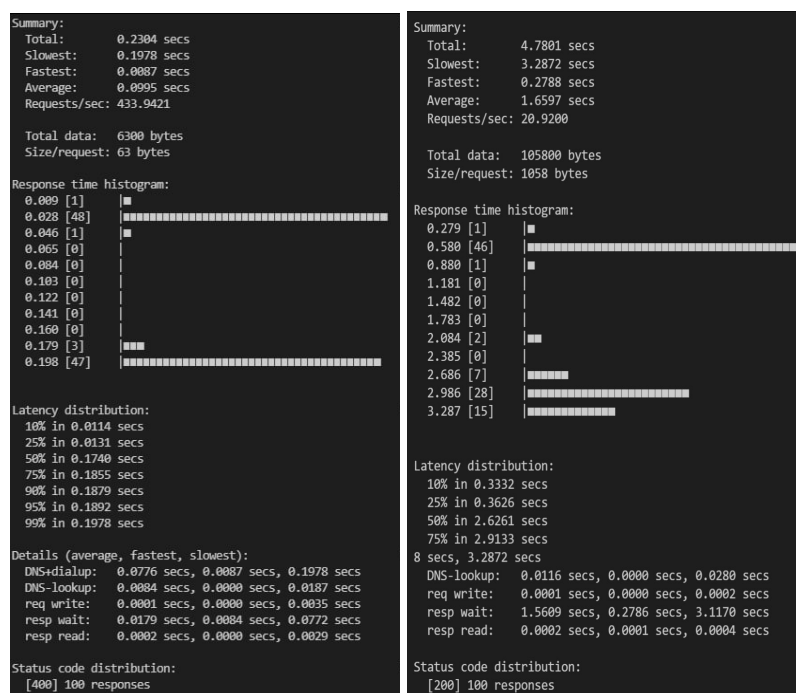
(Destination Search)

(Get Payment Details)

We performed a load test on our software architecture using Python Locust. The test was done on our API Gateway with a peak concurrency of 500 users and a spawn rate of 100 users started per second, for about 1 minute. The requests per second slowly increased from 100 to about 430 over 20 seconds, with 0 failed requests throughout. While the median response time of the test stayed at about 1000ms throughout the test, the 95% percentile response time steadily increased for 20 seconds before falling to about 1100ms.

When a lambda function receives a request while it's processing a previous request, Lambda launches another instance of the lambda function to handle the increased load. Lambda automatically scales to handle 1,000 concurrent executions per Region.

Latency Test using Hey



(Destination Search)

(Hotel Room Search)

By monitoring the latency of the Lambda functions such as Destination Search and Hotel Room Search, we are able to get an accurate understanding of time needed for the functions to complete and thus discover potential bottlenecks that we can resolve.

Appendix

Screenshots of Hotel Booking Site

Hotel Scanner

HOTEL SCANNER

Destination

sir

'S Hertogenbosch, Netherlands
S Horta, Felanitx, Spain
S'agaro, Spain
S'Argamassa, Santa Eulalia del Rio, Spain
S'illot, Spain
S'Iscale, Italy
Sa Caleta, Ciutadella de Menorca, Spain
Sa Coma, Spain
Sa Kao, Thailand
Sa Pobla, Spain

language

en_US

currency

SGD

guests

1







Start Date

to

End

Submit

Hotel Scanner

HOTELS: WDOM		
	<div>Value Hotel Thomson</div> <div>592 Balestier Road</div> <div>Rating: 3 / 5</div> <div>Price: \$2846.95</div>	<div>Book</div>
	<div>YWCA Fort Canning</div> <div>6 Fort Canning Road</div> <div>Rating: 4 / 5</div> <div>Price: \$3352.68</div>	<div>Book</div>
	<div>Summer View</div> <div>173 Bencoolen Street</div> <div>Rating: 3 / 5</div> <div>Price: \$3699.27</div>	<div>Book</div>
	<div>Hotel Grand Central</div> <div>22 Cavenagh Road</div> <div>Rating: 4 / 5</div> <div>Price: \$3858.41</div>	<div>Book</div>
	<div>Hotel Royal</div> <div>36 Newton Road</div> <div>Rating: 3.5 / 5</div> <div>Price: \$4084.76</div>	<div>Book</div>
	<div>Village Hotel Albert Court</div> <div>180 Albert Street</div> <div>Rating: 4 / 5</div>	<div></div>

CHECK OUT OUR FANCY ROOMS!!!



Standard Double

Price: \$3237.25

Points: 80925

Amenities: Room size (sqm), ...

Reserve



Superior Double

Price: \$3546.86

Points: 88650

Amenities: Room size (sqm), ...

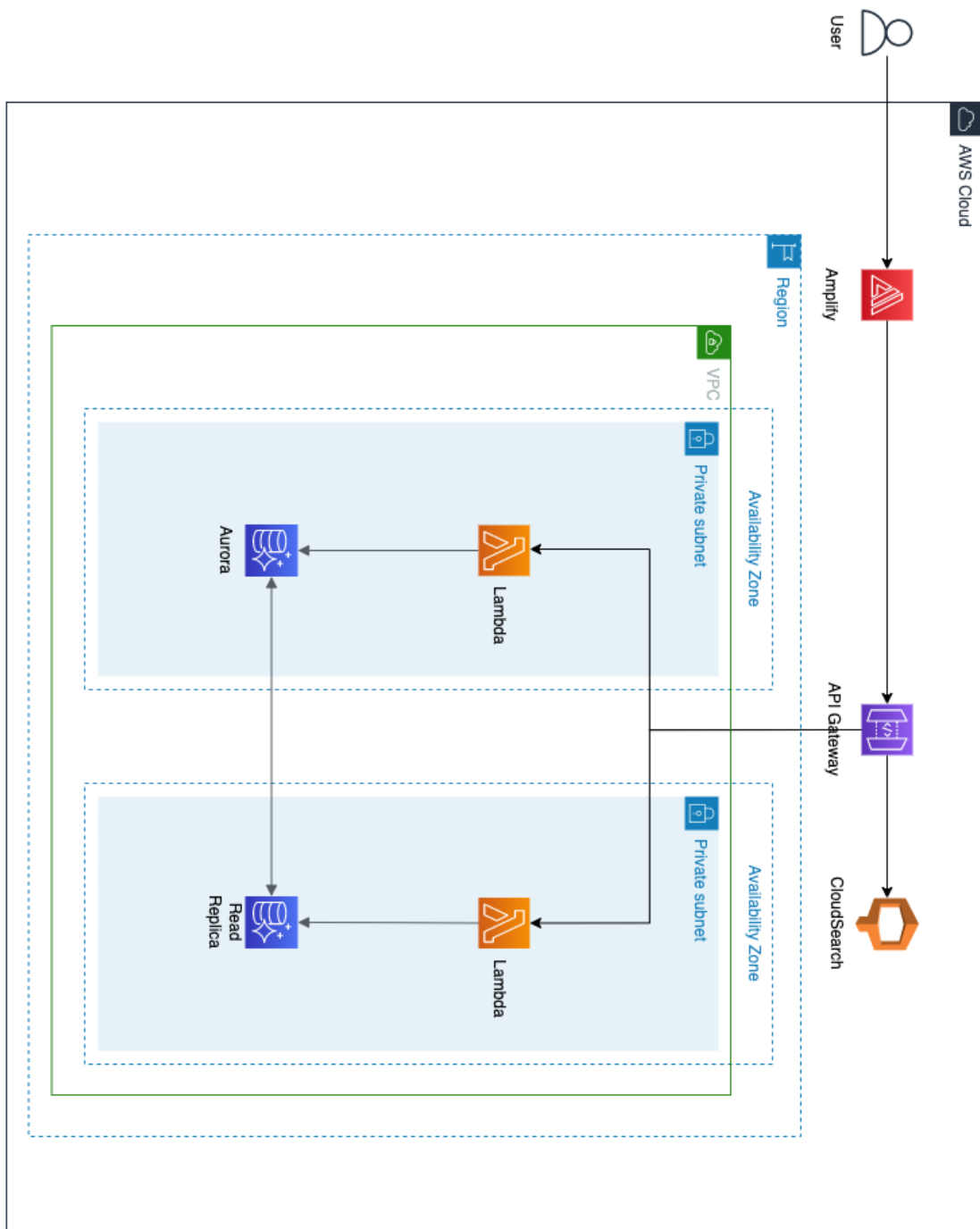
Reserve

Hotel Scanner

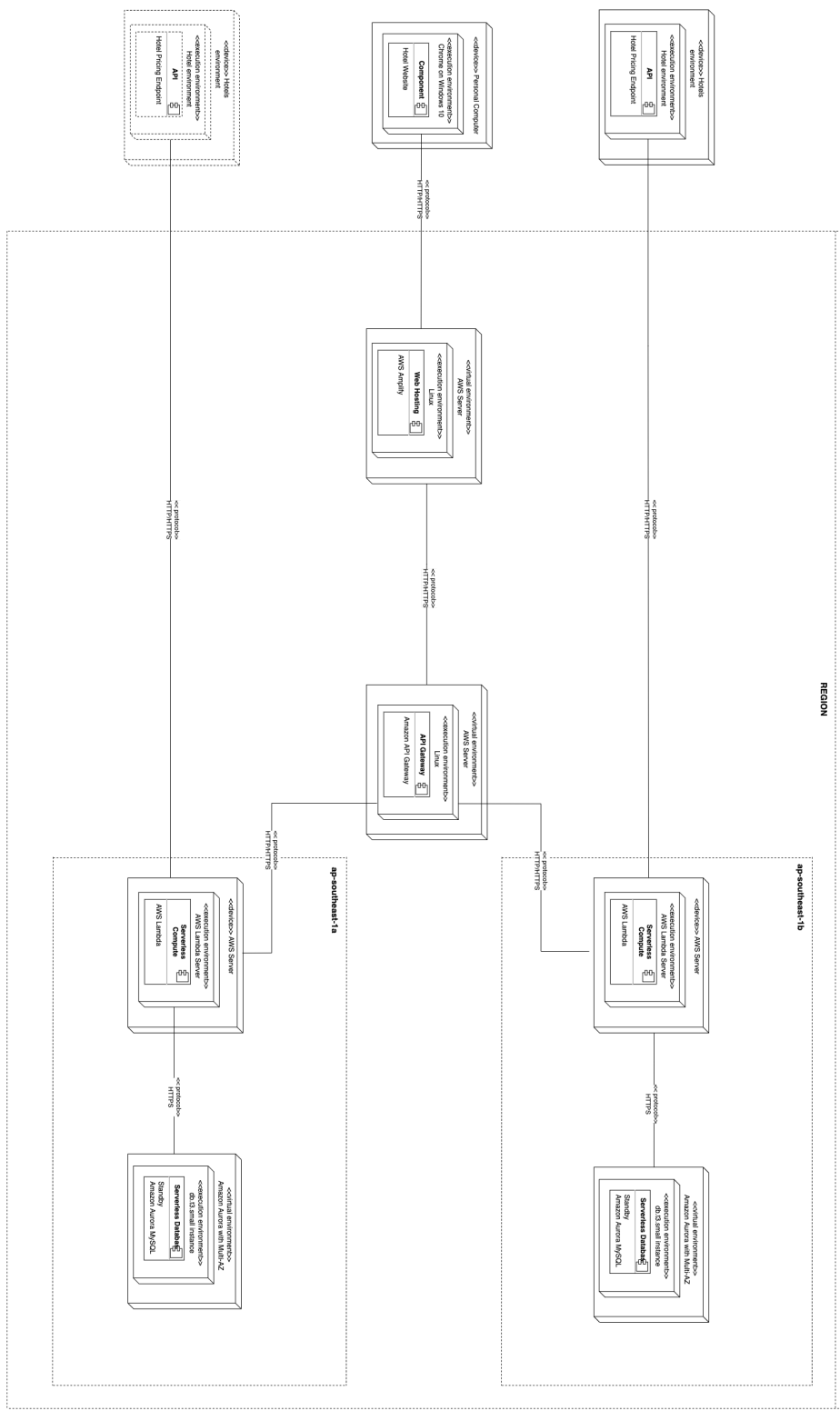
Email	Card Number
<input type="text" value="johnsmith@gmail.com"/>	<input type="text" value="1234123412341234"/>
Salutation	Name on Card
<input type="text" value="Mr"/>	<input type="text" value="John Smith"/>
First Name	Card Expiry Date
<input type="text" value="John"/>	<input type="text" value="11"/>
Last Name	CVV/CVC
<input type="text" value="Smith"/>	<input type="text" value="..."/>

Submit

AWS Architecture Diagram Full



Deployment Diagram Full



Development Budget Details

Activity / Hardware / Software / Service	Description	Cost
Development of Solution	Implementation of frontend and backend logic; Setting up of AWS services; Testing	420 man-hours for a 6-man team
AWS Lambda	1 million requests per month and 400,000 GB-seconds of compute time per month, usable for functions powered by both x86, and Graviton2 processors.	Free Tier https://calculator.aws/#/createCalculator/Lambda
Amazon Aurora	1 x db.t3.small instance	1 instance(s) x 0.041 USD hourly x 730 hours in a month = 29.9300 USD Amazon Aurora MySQL Compatible cost (monthly): 29.93 USD Amazon Aurora MySQL Compatible cost (upfront): 0.00 USD https://calculator.aws/#/createCalculator/AuroraMySQL
Amazon CloudSearch	1 x search.large	1 instance(s) x 0.078 USD hourly x 730 hours in a month = 56.9400 USD Amazon Aurora MySQL Compatible cost (monthly): 56.94 USD

		Amazon Aurora MySQL Compatible cost (upfront): 0.00 USD
Amazon API Gateway	1 million API calls received for REST APIs 1 million API calls received for HTTP APIs 1 million messages 750,000 connection minutes for WebSocket APIs per month Free tier - up to 12 months	Free Tier https://calculator.aws/#/createCalculator/APIGateway
AWS Amplify	Build & Deploy 1000 build minutes per month Hosting 5 GB stored per month and 15 GB served per month	Free Tier https://calculator.aws/#/createCalculator/Amplify
Total Cost:		Total Monthly Cost: 86.87 USD

Production Budget Details

Activity / Hardware / Software / Service	Description	Cost
---	-------------	------

AWS Lambda	<p>1 million requests per month and 400,000 GB-seconds of compute time per month, usable for functions powered by both x86, and Graviton2 processors.</p> <p>+</p> <p>100 Concurrency + 30 hours per month + 1 million request with provision concurrency + 30000ms Duration + 500mb memory</p>	<p>Free Tier + 164.59 USD Total (monthly): 164.59 USD</p> <p>https://calculator.aws/#/createCalculator/Lambda</p>
Amazon Aurora	<p>2 x db.t2.medium 100 GB Storage 100 GB Backup Storage</p>	<p>Amazon Aurora MySQL Compatible cost (monthly): 119.72 USD</p> <p>Total Storage Cost (monthly): 10.53 USD</p> <p>Additional backup storage cost (monthly): 2.10 USD</p> <p>Total (monthly): 132.35 USD https://calculator.aws/#/createCalculator/AuroraMySQL</p>
Amazon CloudSearch	<p>1 x search.large instance</p>	<p>1 instance(s) x 0.264 USD hourly x 730 hours in a month = 192.72 USD</p> <p>Amazon Aurora MySQL Compatible cost (monthly): 192.72 USD</p>

		Amazon Aurora MySQL Compatible cost (upfront): 0.00 USD
Amazon API Gateway	1 million API calls received for REST APIs 1 million API calls received for HTTP APIs 1 million messages 750,000 connection minutes for WebSocket APIs per month Free tier - up to 12 months After 12 months: 1 million API calls received for REST APIs 1 million API calls received for HTTP APIs	Free Tier + HTTP API request cost (monthly): 1.00 USD REST API cost (monthly): 31.24 USD Total (monthly): 32.24 USD https://calculator.aws/#/createCalculator/APIGateway
AWS Amplify	Build & Deploy 1000 build minutes per month Hosting 5 GB stored per month and 15 GB served per month	Free Tier https://calculator.aws/#/createCalculator/Amplify
Total Cost:		Total Monthly Cost: 521.9 USD