Azeem Mumtaz · Follow

Jul 4, 2021 · 14 min read

Case Study: Cloud Adoption of a Shipping Company in Sri Lanka

Background

ABC (i.e., codename) is a family-owned shipping company that provides various services to ships (i.e., chandlers, bondsmen, duty-free shops at Colombo port, etc.). Their primary business is Ship chandelling. They supply locally sourced vegetables, fruits, poultry and meats, water and drinks, equipment, clothing, medical requirements, and many more products to Ships transitioning through ports in Sri Lanka.

They have a basic eCommerce monolith application written in Java using servlets and JSPs with a MySQL database.

A typical business flow looks like the following.

- Captains of ships submit their orders via emails and WhatsApp.
- A customer support representative logs into the company's intranet opens the application via a web browser, enters their credentials, and input the order.
- Then fulfilment officers receive email notifications about new orders.
- Validate orders against inventory, and create required purchasing orders with vendors.
- Deliver and load orders to Ships at the exact scheduled time with approval from the port authorities.

at an optimal level with the growth, which has a potential impact on their reputation for being a quality and on-time service provider.

Therefore, the company's management has decided to evaluate and change its IT capabilities proactively before any impact to their customers and business partners. Further, the company wants to remove manual data entry by allowing ship captains to place orders directly and check their order statuses without needing to contact the company.

Current Problems

Following are some of the problems faced by the company.

- During typical busy days, working with the eCommerce application is frustrating when it becomes unresponsive and takes time to respond to user actions (i.e., increased latency)
- The credentials to the application are stored in the database and can be accessed by the employees within the intranet.
- There is no secure way to expose the application publicly for captains to place orders.
- The application server and the database running on a single hardware server. Current specifications are 2.9 GHz CPU (Intel Xeon W-2102), 8 GB memory, and 240 GB HDD (magnetic) with Ubuntu 16.04.7 LTS server operating system.
- Other business applications are running on the same server. However, the server virtualization support has not been used, and it cannot handle high loads.
- The MySQL database runs on the same server as the application with one instance. There are no replications and fail-over mechanism and does not support end-to-end encryption.
- Upgrading existing network components and outdated hardware is costly.



- The company currently has invested up-front capital with procurements and installations and recurring costs for maintenance and support to mitigate risks to their business operations.
- Servers are installed within the company's headquarters.
- ABC does not have to pay an up-front capital but only require paying what is actually used.

B — Economies of Scale

- There are millions of customers who have already adopted Cloud.
- This gives new adopters like this Shipping company the benefit of the Economies of Scale, which means the higher number of customers reflects lower prices (Baron, et al., 2017).

C — Scale On-Demand

- The company does not need to pre-plan the capacity required to run its application ecosystem in the cloud.
- With the cloud, the company can scale out or scale in or scale up or scale down at the right time, based on the application and service analytics like transaction per second (TPS), latency, CPU usage, memory usage, active connections to the database, database usage metrics, etc.

D — Focus on Business Priorities

- Cloud computing allows organizations to focus on their business priorities instead of managing servers (Baron, et al., 2017).
- This Shipping company can save money from spending on internal IT capabilities to the cloud and focus on the business capabilities and operational outcome.



- It is important to provide a reliable service to ship captains with the expansion while ensuring their applications work with low latency with increasing demand.
- Moving to the cloud will enable ABC to scale their business globally without even owning any data centres or internal services at the premises in a highly available setup.

F — Speed and Agility

- ABC will continue to improve its core shipping applications and provide more features over the next few months to years.
- Therefore, ABC needs to build features for customers faster.
- The cloud will give ABC a way to deploy features faster without impacting its customers in a controlled way, like using strategies like blue/green deployment and A/B testing.

Cloud Adoption: Lift and Shift Strategy

As a solution, a proposal was made to shift the company's application servers to the AWS cloud, one of the pioneers in cloud computing and provides a wide range of technologies that solve problems faced by ABC. Further, this architecture will allow ABC to migrate to the Cloud with minimal changes to the application code and minimal effort.

The following proposed architecture allows ABC to migrate to the Cloud with minimal changes to the application code and minimal effort.

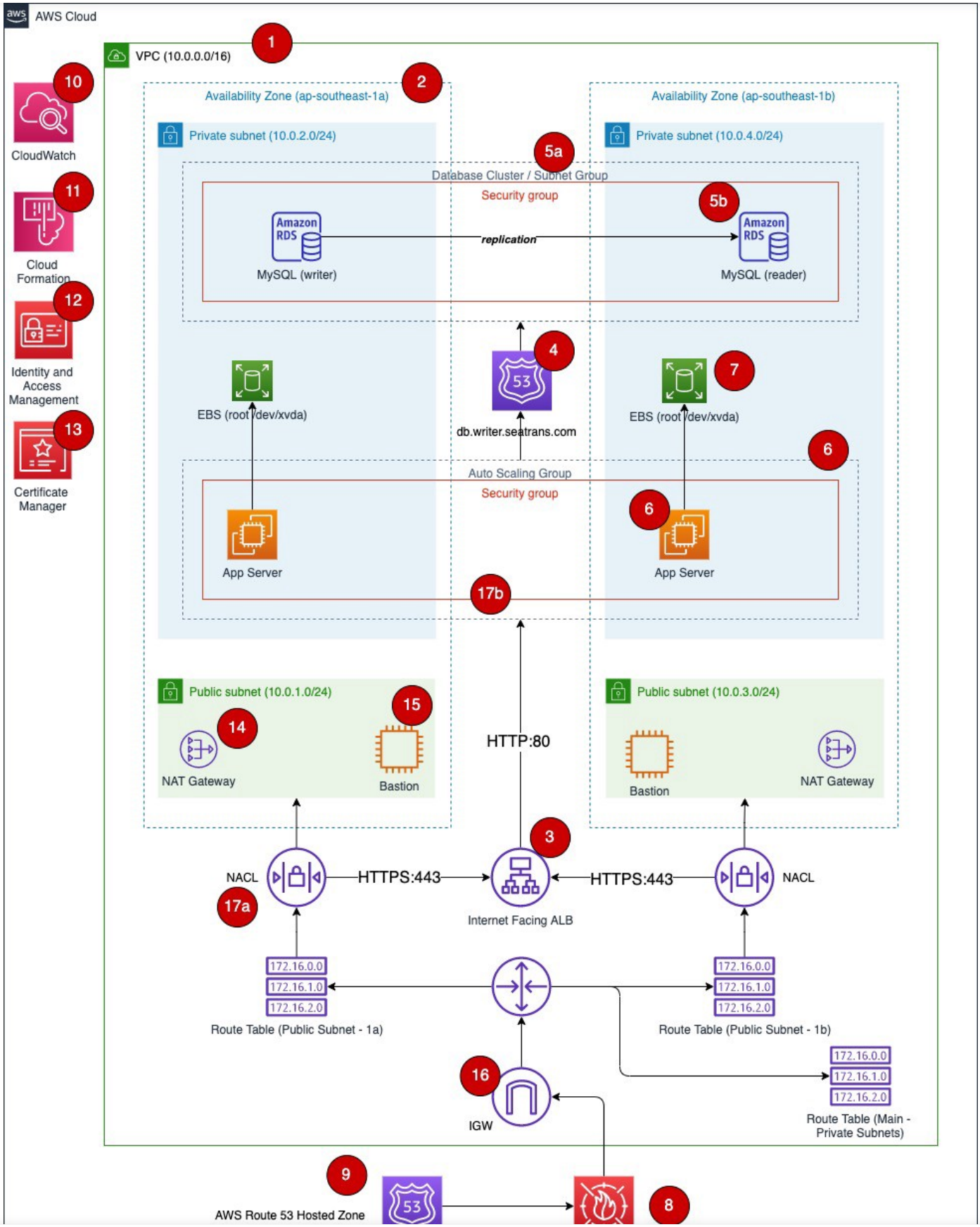

Figure 1 — Lift and Shift Architecture

Refer to https://blog.azeemmumtaz.com/details-of-proposed-lift-and-shift-architecture-for-seatrans-9e6169c8dc27 for a detailed explanation of each numbered component in this architecture.

Migrating the Internal Workload to the AWS Cloud

ABC can use the following two services to migrate its current workload to the cloud.

1. AWS Application Migration Service (AWS MGN)
2. AWS Database Migration Service (AWS DMS)

Application Migration

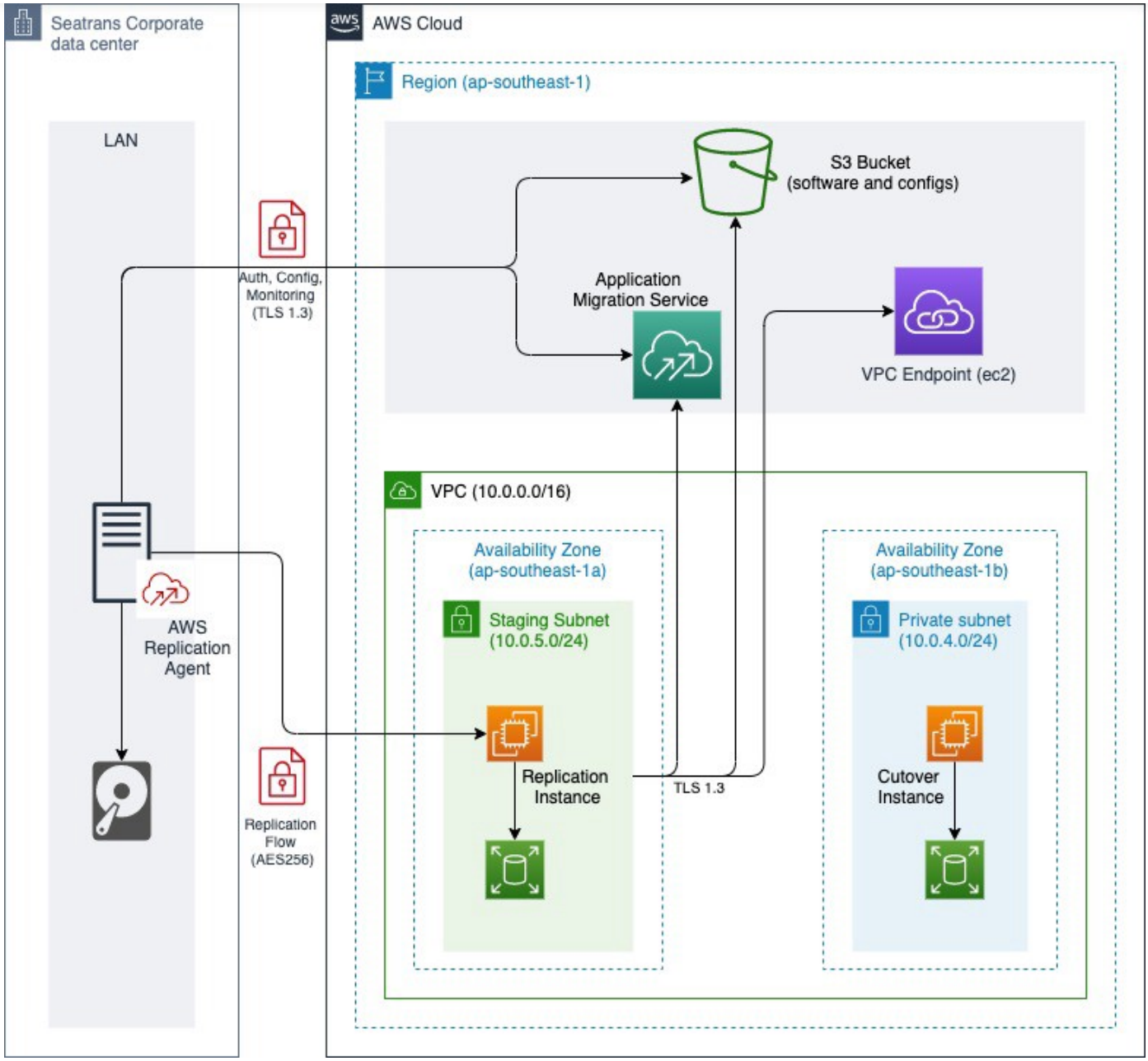

Figure 2 — Application migration using MGN

AWS MGN allows ABC to migrate its application workload from its server within the LAN and the contents in the HDD (i.e., images, log files) to the proposed AWS cloud in a simple, continuous and secure way AWS replication agent in the local server. Once it establishes the successful connection with the MGN, the replication setup in a designated staging subnet will be created.

Further, ***AWS S3*** buckets store required software and configurations for both the agent and the replication server. A ***VPC Endpoint*** on EC2 instance powered by ***AWS PrivateLink*** will allow the replication server to take snapshots of the EBS by invoking the API of the EC2 service.

Database Migration

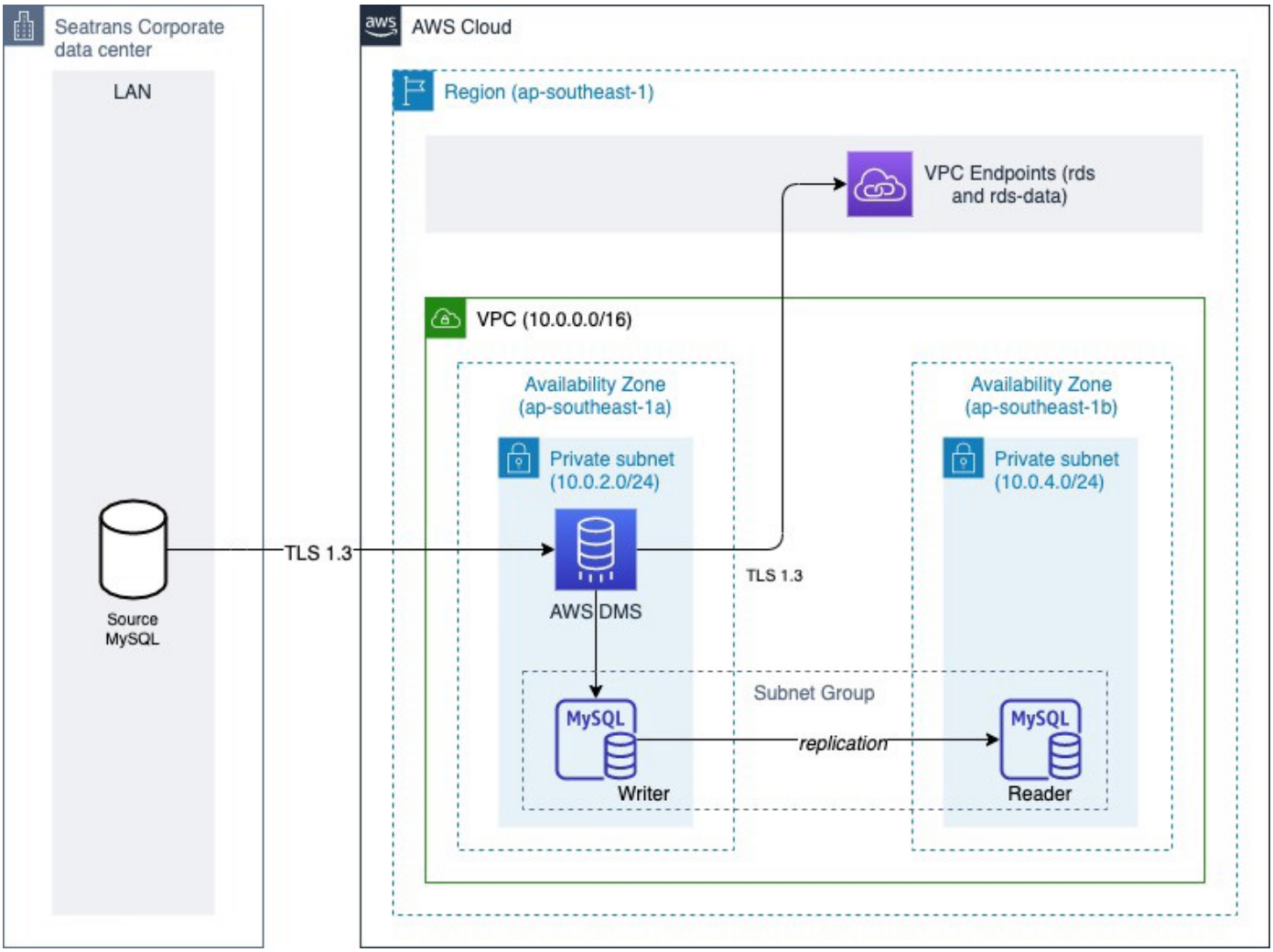

Figure 3 — Data Migration using DMS

AWS DMS service allows ABC to migrate an on-premises database to the ***AWS RDS***, with real-time syncing while ensuring the source database is operational for business continuity. The AWS DMS supports different migration strategies. Since source and the target database engines are compatible, ABC can leverage a homogenous migration

A ***VPC Endpoint*** on RDS and RDS Data powered by ***AWS PrivateLink*** will allow the DMS to securely invoke RDS APIs and replicate data to the target database without needing an IGW or a NATG.

Estimated Cost with Lift and Shift

The architecture will be created in the ***ap-southeast-1 region*** with two ***m5. Large*** instances and two RDS instances for MySQL with ***DB.m5.large*** instance type.

More details of the estimation can be found in the following like.

https://calculator.aws/#/estimate?id=1adf24318ace2d0c992d6a72093ed508eda715ab

Service	Monthly (USD)
Amazon EC2	223.2
Amazon Virtual Private Cloud (VPC)	97.94
Amazon RDS for MySQL	2061.4
Application Load Balancer	48.47
Amazon Route 53	2.2
Amazon CloudWatch	6
AWS Web Application Firewall (WAF)	206.6
Total	2645.81

Re-engineer the Cloud Adoption

Rationale for Re-engineering

The lift and shift architecture will allow ABC to adopt cloud faster, decommission their on-premises servers, and benefit the cloud. However, lit and shift will not allow ABC to build an optimal architecture that is cost-effective. Further, ABC needs to spend approximately **2645.81 USD** monthly, while its compute instances and RDS instances could be underutilized. Therefore, ABC needs to re-engineer its application architecture to achieve operational excellence, performance efficiency, improve reliability and security, and optimize the operational cost on the cloud.

ABC's eCommerce application is a monolith, which ABC aspire to improve and take to an enterprise-level platform to allow Ship captains to submit their orders in real-time. Following are the main functionalities of this application.

1. Customer registration
2. The Product catalogue (search, review, view products)
3. Cart functionalities (add, update, remove products)
4. Order submission, modification, and cancellation
5. Order confirmation emails
6. Delivery notification emails
7. Authentication and Authorization
8. Admin functionalities (customer review and approval)
9. Invoices (view, print invoices)

The product catalogue and order processing functionalities need more resources to provide robust searching and pagination functionalities. On the other hand, authentication and authorization do not need many resources like CPU and memory. This is why in lift-and-shift architecture, an ***m5.large*** instance with 2vCPU cores and 8GB memory was proposed to support the entire application. Furthermore, the current application runs in a single runtime; we cannot scale individual functionalities independently. Moreover, a change in functionality requires executing the entire end-to-end tests and deploy the entire binary, which means it is slow to do continuous deployment.

We can break the eCommerce application into multiple microservices using principles like aggregates and bounded contexts (Newman, 2019). Each of these services becomes loosely coupled with high cohesion (Newman, 2015).



delivered, etc.) which can be grouped into one self-contained unit of work (Newman, 2019). One of many of these aggregates can be represented in a single bounded context with inter-dependencies. Therefore, we can break the application into the following microservices and their data using database decomposition techniques.

1. Order Service → Manage ordering lifecycle
2. Customer Service → Manage customer details
3. User Service → Manage users (i.e., manage profile, authentication, and authorization of ship captains).
4. Admin Service → Admin related functionalities like provisioning customers, adding products to the catalogue, updating inventory, etc.
5. Invoice Service → Multiple orders compose into one invoice.
6. Email Service → Send transactional emails.
7. Product Service → Manage the lifecycle of the entire products catalogue
8. Product Search → Product searching and filtering

Each of these services now can function in its own bounded context with its own database.

Re-engineering the eCommerce Application

We need to follow a methodology when implementing the microservices architecture to get the complete benefits of its offerings. The Twelve-Factor App is such a methodology that will help us develop services with ***maximum portability***, ***clean contract***, ***maximum parity***, ***continuous deployment***, ***scalability***, ***language-agnostic***, and ***cloud-ready*** (Wiggins, 2017).

Codebase: Each service should have its own codebase on Git.

Config: Configurations that are needed to run services must not be a part of the codebase. Each service may need different values for the same configurations key (i.e., DATABASE_URL) when deploying to a different environment. Instead, those configs need to be part of the environment variables and inject into the service at runtime. The ***AWS Systems Manager Parameter Store*** can store such environment variables and pull them at the deployment stage.

Backing services: All supporting services must consider as attached resources, which allow us to replace them at runtime. For example, accidental deletion of a database must not be a reason to update code to connect with a new database instance.

Build, release, run: Separation in build, release, and run stages allow us to roll back releases at any given time and enable us to automate the CI/CD process.

Processes: The identified services are stateless. In the case of stateful services, then we must not store the session in its memory. Instead, the session must be stored in a backing service like **Amazon ElastiCache for Redis**.

Port binding: All services must self-contained, and its service layer must be exposed via a port for other services and clients to interact.

Concurrency: Each service handles requests within its bounded context. And those services can individually scale to handle concurrent requests.

Disposability: All services must be disposable. This means we can scale at any given time without impacting customers. When scaling-in (i.e., removing processors to handle decreasing load), each process must gracefully shut down.

Dev/prod parity: Microservices architecture paves the way to continuously deploy from dev to production in a shorter time window. To achieve this, there has to be maximum possible parity between development to production environments.

Logs: Currently, the monolith stores logs in the HDD. However, with microservices and



Admin processes: Each microservices may have its own database. From time to time, we may need to deploy DDL and DML to the database. Such scripts must be part of the service codebase and run at the appropriate time during the deployment. We can use scripts like Flyway to manage database migration along with the application.

Proposed Solution Architecture

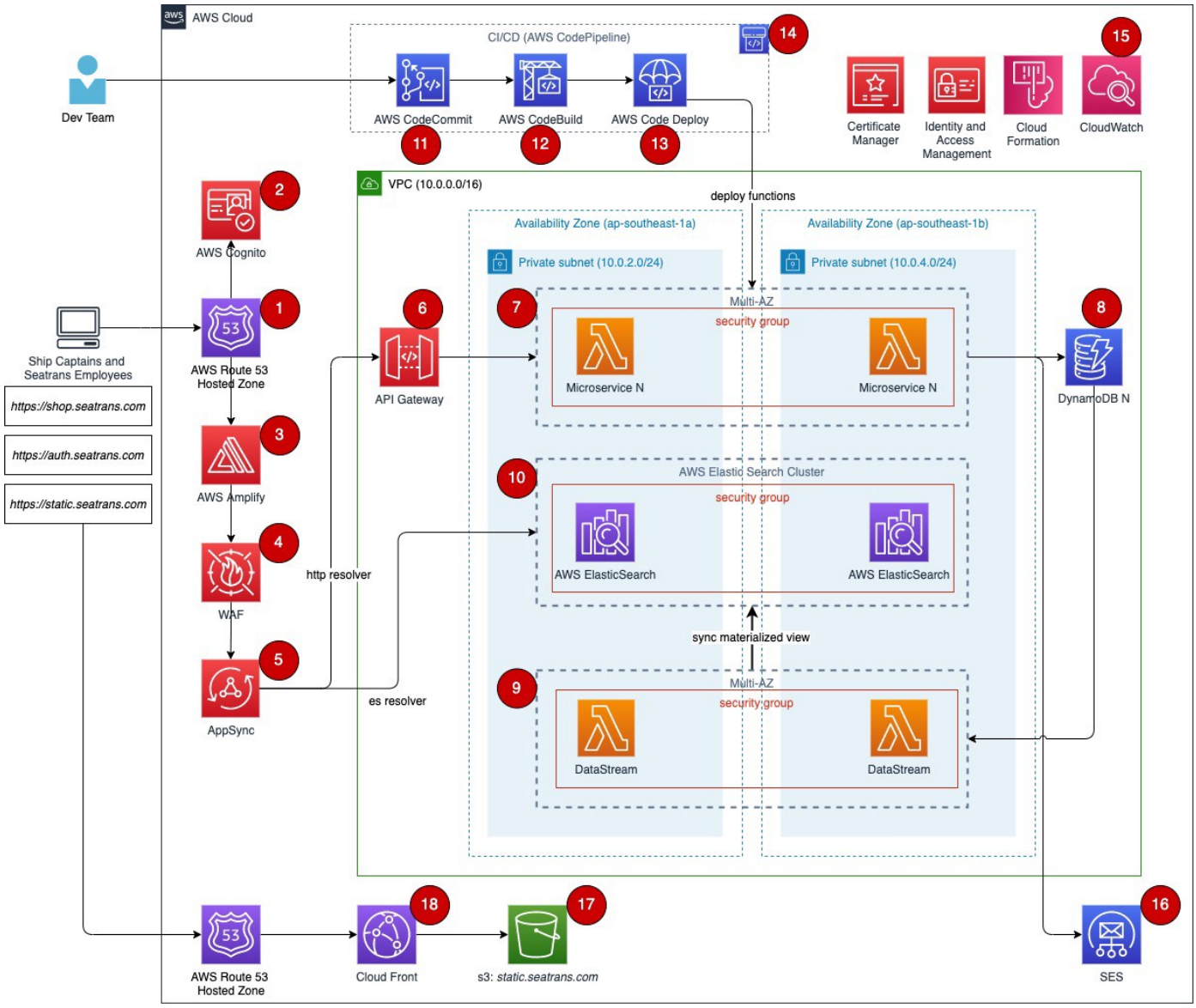

Figure 4 — Re-engineered Architecture

For detailed explanation of each of the ***numbered components*** in this solution

There are 8 identified microservices in this new architecture, which are deployed as follows.

1. Order Service → Lambda Function (with DynamoDB)
2. Customer Service → Lambda Function (with DynamoDB)
3. Admin Service → Lambda Function (with DynamoDB)
4. Invoice Service → Lambda Function (with DynamoDB)
5. Product Service → Lambda Function (with DynamoDB)
6. Email Service → AWS SES
7. User Service → AWS Cognito
8. Product Search → AWS ElasticSearch Cluster with a Lambda function to ingest data from DynamoDB to create and maintain the materialized view for searching, filter, and pagination.

Estimated Cost with Re-engineered Architecture

It was assumed that each Lambda would process 1 million requests per month, and each lambda will process 50 GB of data in DynamoDB, where the average item size is 3 KB. The architecture will be created in the ***ap-southeast-1 region***.

Detailed pricing can be found in the following link.

https://calculator.aws/#/estimate?id=55acefbc5567a4456130e1654ba148b81dcef703

Service	Monthly (USD)
Amazon API Gateway	6.25
Amazon Cognito	25
AWS Web Application Firewall (WAF)	206
AWS Lambda	20.52

```
| AWS CodePipeline                    |            9 |
| Amazon CloudWatch                   |      52.1035 |
| Amazon Simple Email Service (SES)   |        10.12 |
| Amazon Route 53                     |         2.5  |
| S3 Standard                         |         4    |
| Amazon Virtual Private Cloud (VPC)  |        97.94 |
| Total                               |       858.26 |
+-------------------------------------+--------------+
```

Disaster Recovery (DR) Strategy

The suitable DR strategy for ABC, at this stage, is the **"Backup and Restore"**, which is a passive recovery strategy that is initiated after an event of a catastrophe (i.e., connectivity issues within a region). Since the architecture follows infrastructure-as-a-code, takes frequent and automated backups (i.e., DynamoDB backup), and provides a Multi-AZ deployment, the overhead to migrate the entire workload to another region is reduced. However, the biggest advantage in this strategy is the cost, which is lower than other types of advanced strategies like "Active/Active".

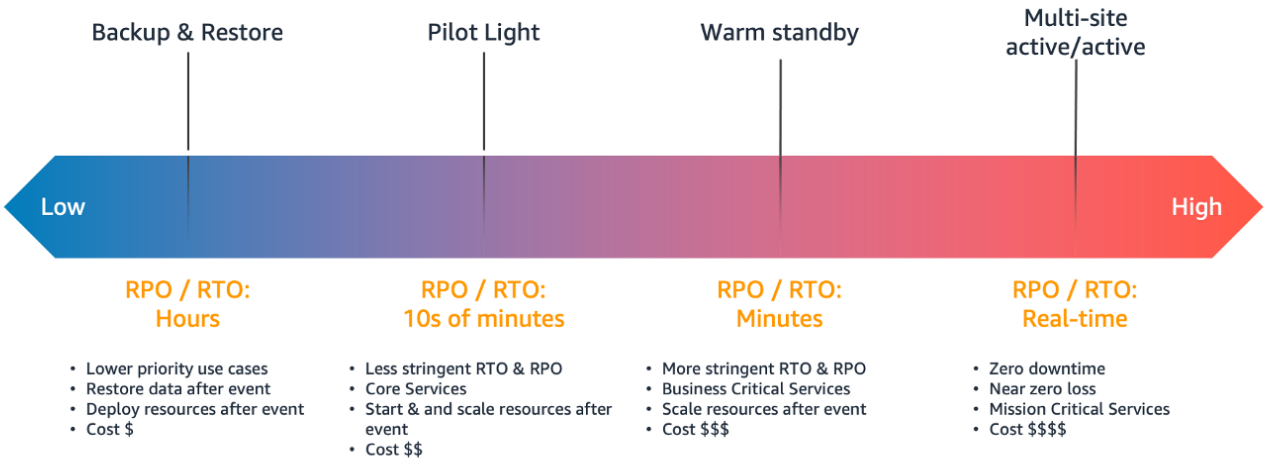

Figure 5 — Disaster Recovery Option in Cloud by AWS (source:

Disaster recovery options in the cloud

Disaster recovery strategies available to you within AWS can be broadly categorized into four approaches, ranging from...

docs.aws.amazon.com

Evaluating the Re-engineered Architecture for Benefits and Risks

The following table evaluates the proposed architecture using the well-architected framework to understand the benefits and risks (Santana, et al., 2021).

Operational Excellence

1. The proposed architecture uses ***AWS CloudFormation*** to build the entire infrastructure as a code. This is in line with the twelve-factor principles as well.
2. The ***microservices*** and ***serverless architecture*** with CI/CD, ABC developers, can do frequent and small releases to production.
3. The architecture supports the "Backup and Restore" DR strategy with Multi-AZ deployment to mitigate smaller scale failures and resume operations faster.
4. Dashboards and metrics in CloudWatch will allow ABC to evaluate the cloud infrastructure and take necessary actions like adding more resources and plan cost optimizations.

Security

1. The architecture uses ***IAM*** to control access to the cloud.
2. The architecture has applied security at every layer possible (i.e., ***NACL***, ***SG***, ***WAF***, and ***IAM Roles and Policies***).
3. All public APIs are secured via ***TLS 1.3*** protocol.

1. Services and other components are horizontally scalable at any given time (i.e., Lambda can be deployed into multiple availability zones).

2. Scaling-out/in happens on-demand without pre-estimating the load.

Performance Efficiency

1. The architecture enables ABC to expand its business from the current scope at any given time.

2. Uses the serverless architecture to handle application and data workload with Multi-AZ setup for improving performance.

Cost Optimization:

1. Rather than provisioning scale, the architecture provides an on-demand scaling based on the load, so that ABC pay just for what is being used. However, cost optimization is a recurring task, and ABC must review its cloud spending more frequently.

2. The lift-and-shift architecture will cost **2645.81 USD** monthly, while the re-engineered architecture will cost around **858.26 USD**. This optimization achieved with re-engineering will enable ABC to save approximately **67.6%** monthly.

Reference and Further Reading

Shawi, M. A., 2020. *Architecting for Reliable Scalability.* [Online] Available at: https://aws.amazon.com/blogs/architecture/architecting-for-reliable-scalability/ [Accessed 9 May 2020].

Baron, J. et al., 2017. *AWS Certified Solutions Architect Official Study Guide: Associate Exam.* 1st Edition ed. s.l.:John Wiley & Sons, Inc.

AWS Well-Architected, 2021. *AWS Well-Architected: Learn, measure, and build using architectural best practices.* [Online] Available at: https://aws.amazon.com/architecture/well-architected [Accessed 5 June 2021].



Newman, S., 2019. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith.* 1st Edition ed. Sebastopol: O'Reilly Media.

Chauhan, S., Devine, J., Halachmi, A. & Leh, M., 2018. *AWS Certified Advanced Networking Official Study Guide: Specialty Exam.* 1st Edition ed. Indianapolis: John Wiley & Sons, Inc.

Santana, G., Neto, M., Sapata, F. & Morais, T., 2021. *AWS Certified Security Study Guide: Specialty (SCS-C01) Exam.* 1st Edition ed. Indianapolis: John Wiley & Sons, Inc.

AWS DR, 2021. *AWS Disaster recovery options in the cloud.* [Online] https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/disaster-recovery-options-in-the-cloud.html [Accessed 9 May 2020].