INTRODUCTION

As an AWS Select Partner, Aligare is working and strongly pushing its Customers to make job decisions in the cloud. Our experience acquired during these 3 years of Cloud work, have given us the reason in terms of the considerable benefits and improvements that Customers can obtain in the handling of security and availability of their information.

For this SDP, the development is based on two cases of use of Aligare Customers, Customer1 and Customer2. Develop the case for two Customers, it only seeks to show the different logics of use that this product, MTD (Digital Transactional Engine) can develop and analyze through Lambdas and APIs in addition to other AWS services.

MTD is a flexible, secure and transactional workflow that supports high volumes of complex operations (high transactionality) that involve the orchestration of multiple systems.

As an additional definition, MTD allow:

- It allows to execute long-running flows (feasible to integrate with human tasks, through APIs) and short-running typically used for orchestration of complex processes.
- It provides an information model that facilitates the traceability and audit of historical processes, as well as online monitoring to have complete control of its processes.
- To facilitate the definition of the processes, it allows to implement the flows based on definitions made in Bizagui Modeler (Free Workflow Construction IDE).
- This solution can be acquired as a server license or in SaaS mode, allowing elasticity and rapid deployment of the processes.

During the development of this SDP, will be able to know the operation and the different components that the MTD possesses as such and how it is related to the different AWS instances and services.

In respect the problems that our solution comes to solve are relates to the following:

- Online control of orders in process.
- Manage orders with exception.
- Process all orders.
- Comply with dispatch times.
- Early warning.
- Timely information.
- Purchase order statistics.
- Capacity against high demand.
- Avoid crafts.
- Reduce operating costs.

In order to improve the experience of Customer1, MTD expedited the processing and mass management of purchase orders generated from its different sales channels (e-commerce, kiosk terminals, phone purchases) and from its different stores (Paris, Umbrale, Carters).

For the business, MTD solves the life cycle process of a purchase order, handling communication and integration between the different legacy systems existing in CUSTOMER1 (Gift card, EOM, Paperless, Sales Support) and customer-side interactions, such as the generation of purchase tickets / invoices and the

subsequent sending by email, and their operation and traceability is available to CUSTOMER1 executives through a BackOffice Web portal.

Being a cloud-mounted solution, MTD is a highly scalable platform which provides a high level of availability and adaptation to the operating requirements that CUSTOMER1 requires.

CUSTOMER2:
In the context of an important project of digital transformation driven by CUSTOMER2 oriented to the implementation of an E-Commerce Corporate and a new logistics manager. The problem is to manage the back office of sales and integration with different legacies and systems.

Among the back office functions, some fundamental ones are the issuance of the DTEs that correspond to the processes (Ballot / invoice, Dispatch Guide and Credit Note - naming the project as Digital Invoice , FD), the sending of sales from E-commerce to the point of sale (POS) system and the sending of mails with tax documents.

The project considered incorporating a new Cloud platform based on micro services, which allows managing and orchestrating the transactionality of the sales platform from digital channel, integrating with electronic tax document generation systems, with POS, and with delivery system mail, allowing to have the flexibility to scale in functionalities and performance in periods of high demand. In addition, this solution is capable of receiving multi-country orders and has the possibility of connecting to Cloud platforms and back-end systems in CUSTOMER2's internal infrastructure.

OUR PROPOSED

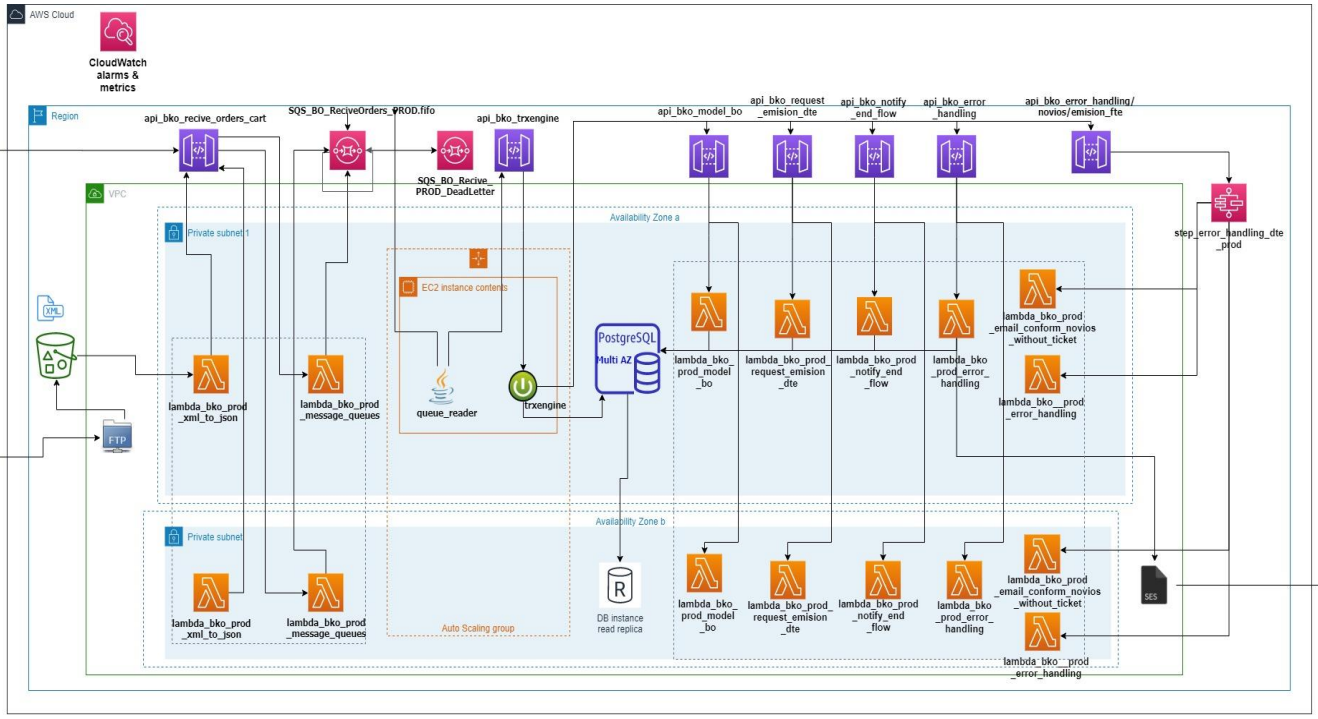
Our proposal is based on a solution that allows to collect, manage and orchestrate purchase orders or transactions incorporating business rules, monitoring elements and others. An example of its implementation is in the management of Retail Purchase Orders where it manages its flow considering business rules, state management, exception control, mailings and integration with the different legacies among other things. This solution is our Digital Transactional Engine (MTD), mounted on an AWS environment.

This solution proposes in addition to being able to eliminate or reduce to the maximum the Customer problematic (defined in previous point "Problem statement / definition") and delivery considerable improvements like:

- Operation and sales 100 % on line (increase the commercial actions in time).
- Early alarms for business exceptions (Anticipate problems for sale (prices, discounts, etc.).
- Fast diagnosis with exceptions (increase platform availability to 99.9%).
- Time - to - Market: In incorporation of new requirements (incorporation of business rules).
- Online Business Intelligence (Comparative Online with Historical).
- Reduction of Operating costs and delivery of Operational Efficiency (cost reduction of 30% "operational").
- Scalable solution (Saas): Elasticity on demand.
- Increase Customer Satisfaction in After Sales (delivery of early information for improvements in care).

ARCHITECTURE DIAGRAMS.

CUSTOMER1:



For this solution, the services used by AWS Cloud are lambdas, API Gateway, bucket S3, SQS, postgres database, read-only copy of the database, SES messaging, metrics and CloudWatch alarms.

Its logical operation is developed as follows:

When entering a purchase order, it is stored in an S3 bucket in xml format. This activates a lambda that transforms the order to json format and sends the command to a SQS queue and this SQS sends the command to a queue reader. At this point there is a condition that, if the queue reader does not read the order in 10 attempts, it goes to an SQS where the order will be pending. In case the reader correctly reads the order, it will go to an API connected to the database engine.

When the order arrives at the database, it is verified in what state the order is and depending on this, it will be redirected to the corresponding API with the status of the order, launching a lambda and saving the order in the database.

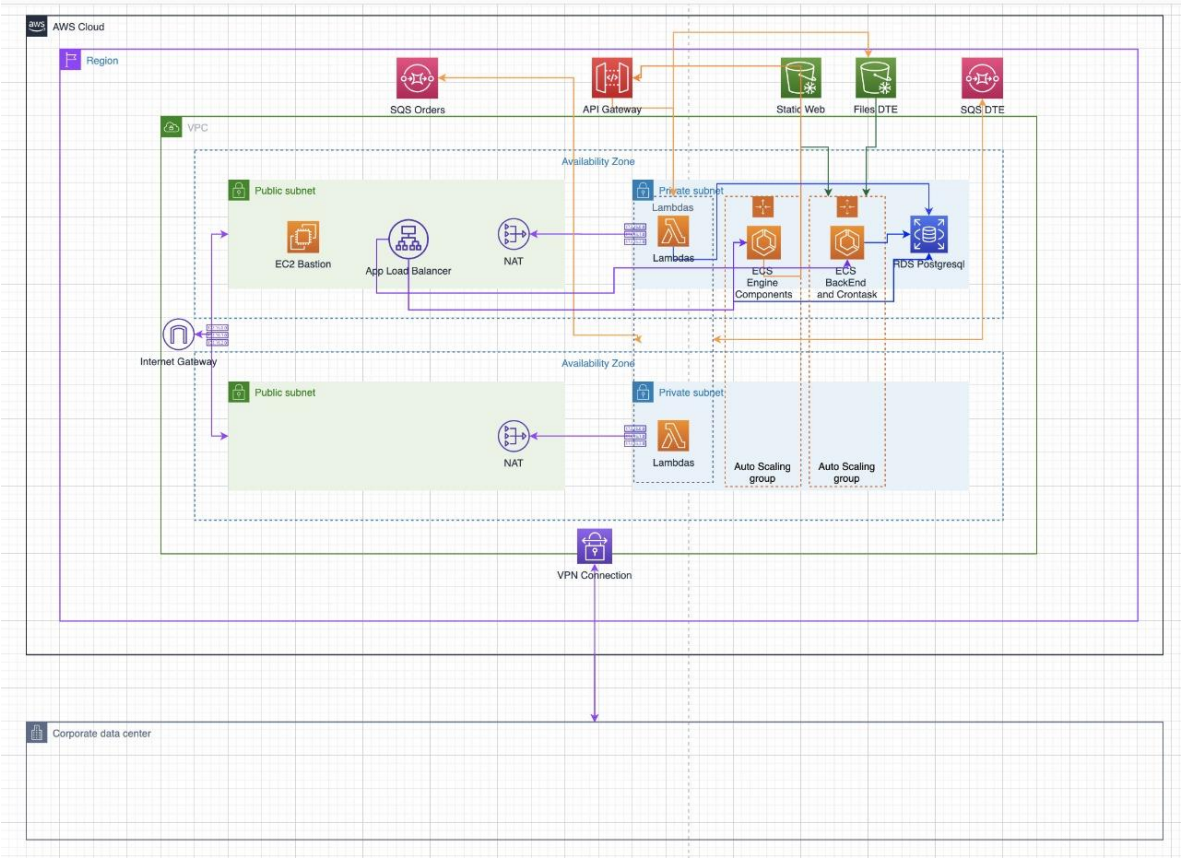
In the event that an order fails when trying to enter the state of broadcast_dte, it will be sent to a step function that will trigger two lambdas where the error will be controlled.

To control errors that occur in the remaining stages, these will be directed to an API that controls these errors by triggering the corresponding lambda by storing and updating the order in the database.

For the second stage of handling the order status, the SES mail service is used, which, at the time of the order, passes through the issuance_dte successfully and is updated in the database, sends a legacy through of an email to the customer indicating that the order began to be processed.

The entire workflow is controlled with CloudWatch metrics and alarms for the different services indicated in the scheme.

CUSTOMER2:



The environment is deployed in a mixed On-Premise and AWS architecture and deployment using CloudFormation where the following services are used:

VPC, Subnets, Nat Gateway, Internet Gateway, Transit Gateway, API Gateway, Lambdas, SQS, SES, SNS, ECS, ECR, ALB, EC2, RDS, S3.

Its logical functioning is developed as follows:

For CUSTOMER2 solution, CloudFormation templates were used using the AWS console. CloudFormation allows to generate all the resources and for most deployments we use the AWS console which provides generation of template stacks.

The goal of using CloudFormation in this solution is to streamline the management and creation of resources with the least human impact in service management.

For lambda deployment the GitLab ci/cd versioning tool was used allowing you to quickly update and deploy lambdas. In addition it complies with repository, compiler, and deployment functions.

When deploying a lambda function and then updating this function the ci/cd configuration in GITLAB allows you to perform a rolling-update of the stack, allowing at the time of upgrade, to modify the environment variables, handlers, and function code.

AWS LAMBDA VALIDATION CHECK LIST.

CUSTOMER1:

AWS lambda is used for processing orders, for example, in the first step, a file is taken from an S3 bucket in format XML, which, is converted to json format and is sent to the SQS queue.

With lambda, the customer is given the processing of purchase orders stored in Amazon S3 buckets from when an order is entered until the order ends its flow, being stored in the PostgreSQL database.

Around 25 lambdas functions are used for the present solution, of which the most relevant are added to the diagram. All these are worked in the us-west-2 region of Oregon.

For each lambda function, executions per month may vary. Depending on the function, some reach 30,000 monthly executions approx., as there are others that can reach 190,000 approx., depending on the type of task you complete. All these executions are controlled with CloudWatch with control metrics and alarms, in addition to having constancy in CloudWatch logs.

The challenge that the Customers proposed to develop the solution was to improve the flow times of a purchase order, which was satisfactorily covered with AWS lambda and additional services, thus increasing the quality and improvements of the system development so that the calls the lambdas will take no more than 5 milliseconds. With this form of work, in which Amazon takes care of the servers, it allowed developers to improve the infrastructure and work of the application.

CUSTOMER2:

Using GITLAB ci/cd Serverless applications were deployed using SAM format to deploy and update stack in CloudFormation. These are formatted for creating lambda functions and each with its corresponding API Gateway.

Using the AWS console, you create a CloudFormation template in yaml format, where the stacks contain the other services in the solution, such as an ECS cluster and task definitions configured pointing to existing ECR repositories.

Additionally, the same template provides the generation of all other resources, which have a specific configuration, and which perform the creation of a task for each micro service used in the solution.

Additionally, use of the CloudTrail service is additionally applied, which allows you to identify the accounts that make changes to each resource.

The challenge that Customers proposed to develop the solution was to improve response times and increase security in the services developed and implemented, for which it was achieved using templates generated in the solution which ensure the privacy of all generated components, using the Best Practices Framework provided by AWS. All resources generated in the templates have a compliance format with the company's security regulations, where backups and encoding of templates are used, as well as the encryption of all the

resources generated in these for the purpose of safeguarding security and having control of the entire environment.

DEPLOYMENT & WORKLOAD PATTERNS

For the presented scheme the “all at once” implementation pattern is used with the eight lambdas functions presented so that all the work is updated and passes to the new version at the same time, without interrupting the flow of the orders.

For the solution, the SAM (Serverless Application Model) implementation application model was used as an architecture framework due to its adaptability with other AWS services such as API, databases, etc., and therefore focus on the application and its development

Every lambda function connects to AWS services, including APIs, SQS services, database, SES mail service and AWS step functions, in addition to being all controlled with CloudWatch.

For the main processing and data flow of the orders within the implemented scheme, the lambdas functions are used due to their adaptation with the form of “Serverless” work without, with this, the developers and support focus on the development of the application and comply the requirements that the Customers requests. In addition, you can better manage the databases used in the solution, due to the feasibility of AWS managing the servers.

The system was designed based on the requirements requested by our Customers and based on the initial architecture provided by the architecture team, achieving a satisfactory work architecture, both for the Customers and for the development and support team.

AWS SERVICE REQUIREMENT / SOLUTION CHARACTERISTICS.

CUSTOMER1:

For the stress tests carried out, periods of high flow of the Customers were taken into consideration, for this case, in a productive environment, the flow was started with 14,551 orders, which all came to complete the workflow satisfactorily. At the same time, emails sent to customers arrived effectively. Also, during the development of the test, 400 more orders were added, which did not diminish the performance. The tests were carried out in a period approx. 1 hour.

For the second iteration of stress tests, the IOPS of the RDS machine were increased to 4000 and the thread number was modified to 200 which allowed that when entering more than 400 orders every 3 minutes, the RDS maintained a constant balanced load of CPU With this modification, the peak of orders per minute is matched by presenting in the first iteration supporting the volumetric.

For each lambda function an IAM role is used where the “AWS Lambda Full Access”, “AWS Lambda VPC Access Execution Role” and “oneClick_lambda_basic_execution_1515077751336” permission is granted. In addition, the duration of a session is controlled, and the date and time of the last activity that was taken.

If there are failed executions for example when you want to enter the queue reader and it fails in the 10 allowed attempts, it is housed in an SQS queue that is “waiting” and for these orders they must be pushed manually to go through the queue-reader and continue with the normal flow of the scheme until reach the accommodation in the database and the customer receives the email corresponding to the order processed.

For the solution “AWS Step Function” was used for the management of multiple services and in this case to manage the execution of two lambdas for when a failure occurs in the emission stage_DTE, so, the lambda is executed notifying the user that there was an incident with the order that will be processed.

For the designed scheme lambdas with VPC are not used because all the functions are within a public subnet.

The concurrence in which the lambdas can be executed, varies depending on it and for the process in which the order is found. It can vary from one hundred daily executions up to a thousand, depending on the lambda executed with a maximum duration of 11,900 milliseconds with the ratio of errors and satisfaction which usually has 100% satisfaction.

All the lambdas functions with which we work for our Customers are controlled with CloudWatch metrics such as invocations, durations, errors count and success rate, among other conditions.

CUSTOMER2:

For stress tests performed periods of high customer flow were taken into account. For this case in a productive environment the flow with a maximum of 1,000 orders was started, all of which came to finish the workflow satisfactorily. At the same time, emails sent to customers arrived effectively. Also, during the development of the test, 100 more orders were added, which did not decrease performance. The tests were carried out in a period approx. 1 hour.

For the second iteration of stress tests the RDS machine's IOPS is increased to 3000 and the thread number was changed to 200 which allowed more than 400 orders to enter every 3 minutes the RDS maintained a constant balanced CPU load. With this modification the peak of orders per minute presented in the first iteration supporting volumetric is matched.

Each lambda function uses an IAM role where you grant the permission "AWSLambdaFullAccess", "AWSLambdaVPCAccessExcecutionRole", and "one Click lambda basic execution 1515077751336”. In addition you control the duration of a session, and the date and time of the last activity that was taken.

In case of failed executions for example wanting to enter the queue reader and fails in the 10 allowed attempts it is housed in an SQS queue that is left in "wait" and for these orders, they must be pushed manually, so that they return to the flow and follow normally the schematic procedure until you stay in the database and the customer receives the email corresponding to their processed order.

For the designed schema, lambda is used with VPC because the functions are within a private subnet.

The concurrency in which the lambdas can be executed, varies depending on the order entered, which can vary from one hundred daily executions to one thousand, depending on the lambda executed with a maximum duration of 11,900 milliseconds with the error and satisfaction ratio, which usually has 100% satisfaction.

All lambda functions that work for our client are controlled with CloudWatch metrics, such as “Invocations", “Duration”, Error count and success rate", among other conditions

AWS APIs VALIDATION CHECKLIST (both cases).

Implementation of REST-type APIs for the messaging flow between system components and lambda functions. The message format is JSON, through HTTPS secure protocol.

For the expected responses from the APIs, the requests that the client received in its high-flow period before implementing the AWS services were considered. By supplying AWS services, requests averaged between 6 and 8 per second which equals 500 orders per minute.

The CloudWatch metrics used for this solution are the number of calls the API receives, latency, integration latency, 4xx error, and 5xx error.

The implemented APIs are based on HTTP protocol with a regional endpoint, this because the calls that are made are within the same region. The API connected to the database engine, is connected to an Edge endpoint, in addition to having different control metrics with CloudWatch.

The APIs are integrated with lambdas functions located in the us-west-2 region, while the API connected to the database engine, both the GET and POST action, is integrated with HTTP.

Using a usage plan and API clients, client needs were identified to identify performance requirements.

As indicated in the previous point, tests were carried out before implementing AWS services in a period of high order flow, in order to calculate the average number of orders received per second and with this, when incorporating AWS services, having the application estimate.

From this, the so-called "API clients" were generated where the accesses to each API and the amounts of incoming requests in a specific time interval were defined, thus restricting the activation of the APIs by unauthorized users and protecting the operation of the backend.

HOW THE DEPLOYMENT WAS MANAGED TO CREATE REPEATABLE AND VERSIONED DEPLOYMENTS THAT RESULT IN ZERO DOWNTIME THROUGH THE USE OF THE AT LEAST ONE OF FOLLOWING CAPABILITIES ACROSS ALL CASE STUDIES:

CUSTOMER1:

The deployment of the APIs in the different environments was managed through the creation of three stages (one for each environment).

In the development stage (DEV), developers self-manage the deployment of APIs according to their needs. In later stages, deployments are performed once all unit tests have been run, according to an internal quality process and the authorization of the release manager.

The next stage is QA, where a version approved by the development area is delivered to the client. In this stage the integration and stress tests necessary to move to productive environments are carried out.

In the final production stage, the APIs were deployed in the production stage.

Starting from the QA stage, it is verified with CloudWatch metrics and alarms, that the operation was as expected, emphasizing response times and safe accesses.

CUSTOMER2:

For API implementation management the first step through the stage of defining the amends where the business capabilities were defined the application validation of the customer and to whom the application is directed. In addition the rules of users who can access and run the deployed APIs were defined.

For the development stage we started with the design to integrate them within the in agree with the API strategy defined in the design stage. With this, you started creating API rest as a set of entities with one

method or multiple methods so that at the test stage you start testing the resource and representing an incoming request sent by the client. For the application response, a "Method Response" resource was created to represent a response to the client request and an "Integration Response" resource to represent the response returned by the backend.

For the testing stage, orders and requests were sent to test the API and its function, where stress tests were performed simulating a time period of alto flow. In addition, the conditions defined next to the HTTP proxy integration were tested to verify that the methods that the client requests are directed to the backend and respond with the defined method

In the final stage of production, the system is deployed to the client and verified with CloudWatch metrics and alarms that it is working correctly and does not throw procedure's, either operational, as well as security.

From the QA stage, CloudWatch metrics and alarms are verified, which the operation was expected, emphasizing response times and secure access.

DETAILS ON AUTOMATING THE CREATION OR UPDATING AMAZON API GATEWAY

CUSTOMER1:

For our client CUSTOMER1, the SAM template was used to perform the updates or creation of the API Gateway, for example, for the invocation of the APIs, the template is controlled by who are the authorized users to execute the APIs and who are not.

CUSTOMER2:

For our CUSTOMER2 client, CloudFormation was used to perform updates or create API Gateway. For API invocation, it is controlled by the SAM template, who are authorized users to run the API, and who are not. It also automates events such as receiving SQS events by lambdas that connect to this. In addition, you also control the number of requests that you can receive in a defined time interval.

ACROSS ALL SUBMITTED CASE STUDIES SHOULD DEMONSTRATE PROFICIENCY (both cases)

HTTP Proxy Integrations

The HTTP proxy integration solution is used so that the methods that the client enters, whether it is POST, PUT, GET, DELETE, are sent directly to the application backend and in this way the same backend parses and returns a response. This allows the API Gateway to handle the authentications and the usage plan, in addition to being able to make mapping requests.

IAM Integration

For the security of AWS resources, each user was integrated with IAM roles specifying the policies of the services that they can see and execute. This controls who is authenticated and authorized to use the services.

AWS Service Proxy Integrations

APIs are built to be able to queue purchase orders in SQS queue. This API is triggered by a lambda function, and allows to deposit the message of the purchase orders in a SQS queue of the FIFO type directly.

SOLUTION CHARACTERISTICS (both cases)

Data Delivery and Display (BI)

For the use cases defined for the client, the hadoop framework and its simple model of storage and

processing of large data sets and redundancy were used to avoid losing information, in addition to allowing it to be integrated with the EMR, EC2 and S3 bucket services.

This framework was chosen because it launches clusters for large data sets and the user does not have to worry about the infrastructure and cluster settings. With this, the work of analysts, developers and people involved in the system, can work together for the analysis and processing of information having a readable and easy-to-use data visualization.

In security matters, when interacting with EMR, in the case of data storage in S3, the service contains two forms of security by the client or the server, in the case of the server, this encrypts the files stored in S3 with a unique key in 256 bit format. Also, in stored data, you have the option to encrypt everything in a bucket using security policies.

Outcome(s) / Results:

The deployments of the lambdas functions made it easier for us to manage the code (generating a lack of concern for the infrastructure). This gives us greater flexibility to make modifications to the functions, reducing downtime when making new deployments and a high availability and speed of execution of the functions, which in their entirety generated in customers (CUSTOMER2 and CUSTOMER1) a high degree of satisfaction.

Lessons Learned:

The use of new Serverless services gave us greater agility in creating new highly complex architectures, as well as having high availability and scalability. The use and complementation of lambdas and api gateway functions helped us avoid creating a highly available infrastructure, as well as saving time in the deployment of the architecture, which helped us to reduce costs proportionally.