INTRODUCTION
Management Tools: AWS CloudFormation

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

For this PRACTICE, the development is based on some of ours Customers use case and where CloudFormation has been implemented to streamline the setting of work environments and also generate integration and continuous deployment with lambdas through generated templates.

The function that it fulfills is to deploy the architecture quickly (all the components of the Digital Transaction Engine "MTD"), allowing to interact as little as possible with the AWS console, defining in CloudFormation all the parameters of the services.

During the development PRACTICE, the operation and the different components implemented “in” and “with” CloudFormation will be explained in addition to its relationship with the Aligare MTD.

CONTEXT

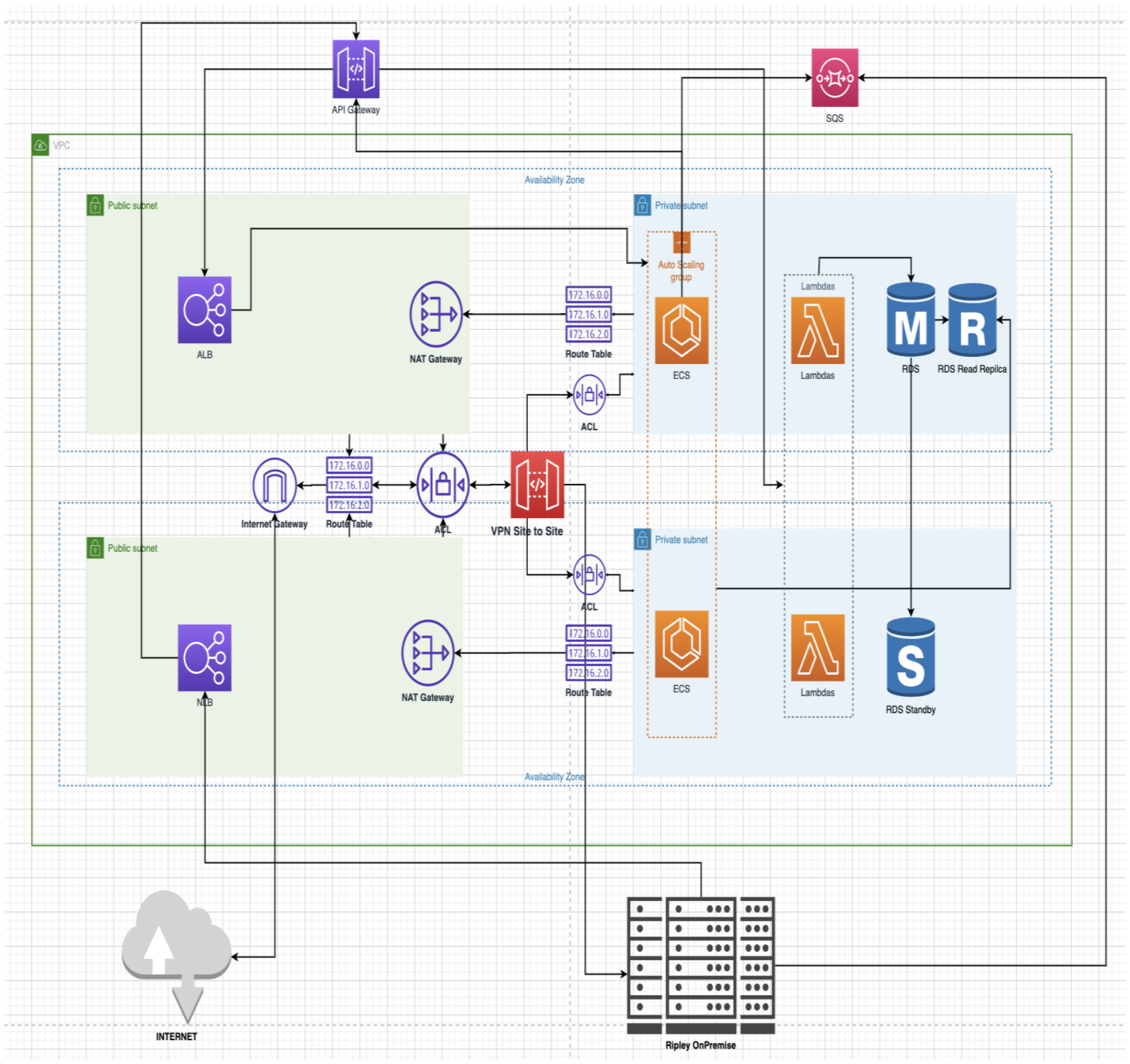
“Streamline the setting of the environments through a generic template which allows you to display the Development, QA and production environments”

The solution seeks to generate a system that provides to unify all the ***purchase orders or invoices*** generated by the different sales channels. The architecture is made up of an engine which is in charge of managing the life flows of the orders, the engine is generated through the ECS service, where a gateway API is used to make requests, the different order flows are generated by lambdas, which send the order data to an RDS PostgreSQL instance depending on the corresponding flow.

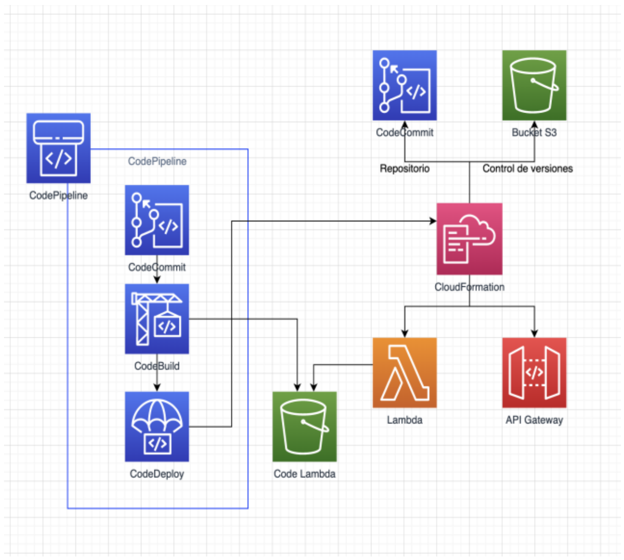
SQS is used to receive purchase orders or invoices where there is a lambda which is responsible for obtaining the data and sending it to the engine.

There is a WEB service in which you can see the status of each order, this allows you to have greater control over the history of each order, speeding up and facilitating solutions to problems in the flows of each order.

DIAGRAM SOLUTION (general):



LIFE CYCLE DIAGRAM CLOUDFORMATION LAMBDAS (general)



The life cycle of a CloudFormation template specifically for the lambda works as follows, the developer generates a push in the Code Commit repository, automatically enters the process of the Code Pipeline pipe, where the Build of the source code starts and the compiled file is stored in an S3 bucket. Later the CodeDeploy service sends the deployment through CloudFormation which generates the deployment of the lambda and the API gateway, after the deploy, the template is used and it is sent to a repository where all the templates are deployed and simultaneously sent to an S3 bucket to generate version control of the templates.

VALIDATION CHECK LIST (AWS CLOUDFORMATION).

The environment has been deployed in a mixed architecture (On premise client and AWS) CloudFormation was used to generate the deployments of the lambdas and API gateway, the realization of the environment was carried out in the Virginia region and uses the following resources:

- VPC
- Subnets
- Nat Gateway
- Internet Gateway
- VPN Connection
- API Gateway
- Lambdas
- SQS
- ECS
- ECR
- ALB
- NLB
- RDS

The implementation of CloudFormation templates are done through the AWS console, however, the implementation of the lambdas is done through the AWS ci / cd tools (Codepipeline, Codebuild, CodeDeploy and Codecommit), which allow updates and quickly deploy to lambdas.

CUSTOMERS PRACTICE DEPLOYMENT AND WORKLOAD PATTERNS:

DEPLOYMENT:

The objective of using CloudFormation in the deployment of this solution is to be able to streamline the continuous integration of the releases that make up the old and new lambdas resources.

The templates have the necessary components to be able to deploy a lambda with the required configurations, these configurations allow us to implement a new or existing function through an artifact generated with the codecommit and codebuild services, this artifact maintains all the lambda data which is stored in an S3 bucket where it is subsequently used to display the function code, when generating the function, the template generates an API gateway configuration where the connection between the services is enabled, allowing the necessary permissions for communication.

For the deployment of lambdas services, an AWS ci / cd platform is used, which fulfills the following function:

- **Repository:** It is used as version control of the source code of each application.
- **Compilation:** Through the ci / cd stages, compilation of the source code is carried out, which undergoes unit tests to validate the correct operation
- **Deployment:** The file generated in the previous step is stored in an S3 bucket where, subsequently, the continuous deployment configuration generates a CloudFormation template. The latter is where the settings and the location of the compilation in S3 are specified, generating a new stack for the deployment of the new lambdas.

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

WORKLOAD:

The AWS ci/cd services are used to deploy lambdas and gateway API functions, the way the deployment is performed is as follows:

- In a code commit repository are the source codes of a lambda function, this repository contains a .yaml file which contains the CloudFormation template that will be used in the deployment of the function.
- Through the Code pipeline service, a pipe is made to generate the integration and deployment
- A code build configuration is created which compiles the source code, generates a SAM artifact and sends it to an S3 bucket (Contains the compilation file)
- With the Code Deploy service, the CloudFormation destination is used to deploy the lambda function, where the parameters of the resources (API and Lambda) and the code of the function are indicated, which is stored in an S3 bucket

The other components of the solution are made using a defined CloudFormation template, with the need for it to be reused for the generation of new project environments, the generated templates are deployed through the AWS console, this except for the ECS service, where only one cluster is generated, this because task definitions, services and task are generated using AWS ci / cd tools, where it is not necessary to define the infrastructure or a template to carry out the deployments.

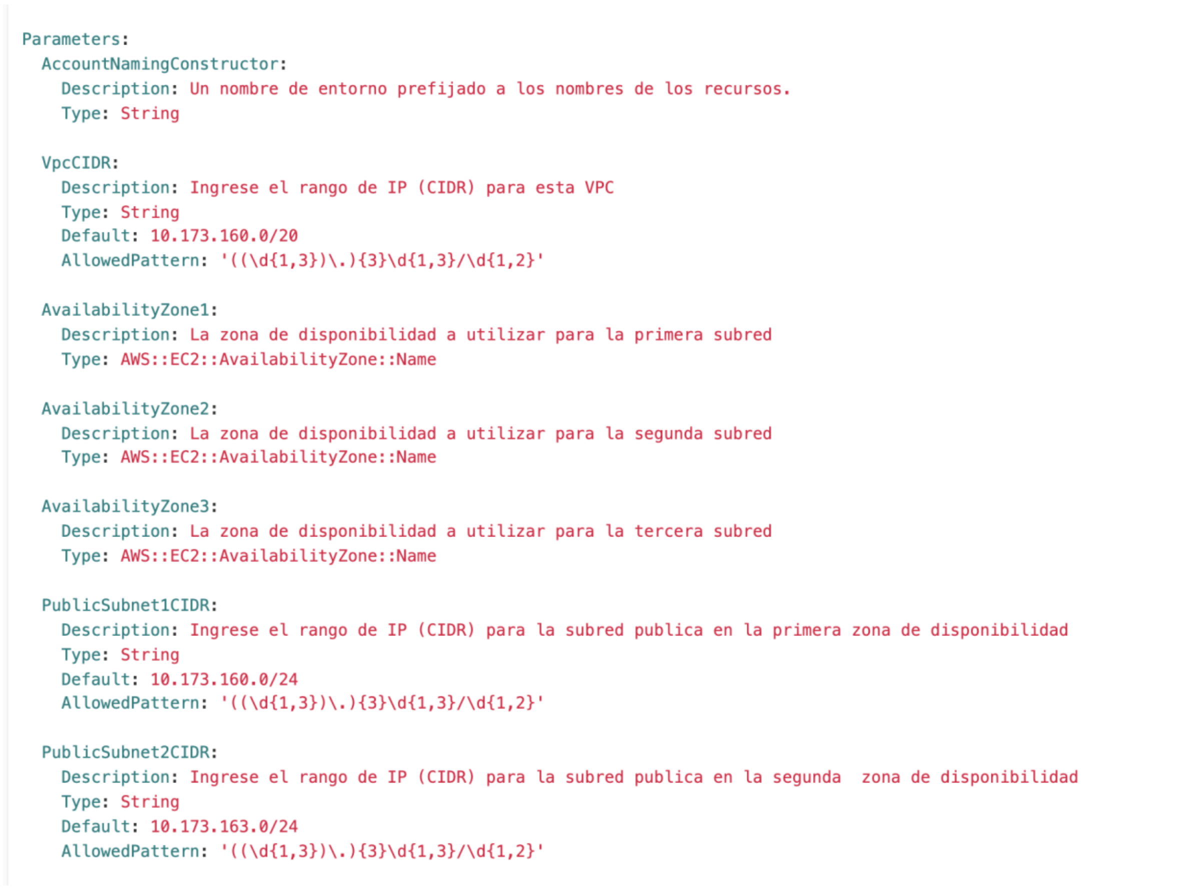
One of the considerations through which the solution templates were generated was to rely on the well architected framework in order to ensure the security and privacy of all the components generated. All the resources generated in the templates have a compliance format with the security regulations of the company, where backup copies and coding of the templates are used, as well as the encryption of all the resources generated in these in order to safeguard security and have control of the entire environment.

In order to identify and have control of all the services generated with CloudFormation templates, CloudTrail is used, which provides us with information on all the modifications generated in the CloudFormation service.

AWS SERVICE REQUIREMENTS.

SOLUTION CHARACTERISTICS: TEMPLATE AUTHORING & COMPOSITION.

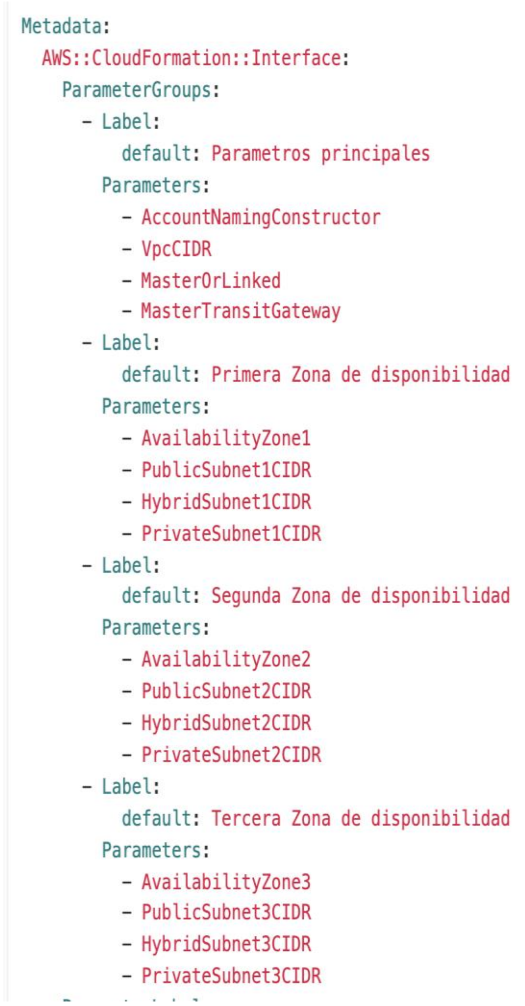
The generated template should be useful for all the necessary environments, that is why it allows entering parameters which must be modified depending on each environment. For the deployment of services the parameters pre-defined by AWS are used, where we use the parameter ***AWS :: EC2 :: AvailabilityZone :: Name*** to identify the available zones within a region, mainly it is used to identify the AZs where the Existing subnets, in order to distinguish public and private subnets, for lambdas functions and their environment variables we use the (ssm) which are entered as parameters.



```
Parameters:
  AccountNamingConstructor:
    Description: Un nombre de entorno prefijado a los nombres de los recursos.
    Type: String

  VpcCIDR:
    Description: Ingrese el rango de IP (CIDR) para esta VPC
    Type: String
    Default: 10.173.160.0/20
    AllowedPattern: '((\d{1,3})\.){3}\d{1,3}/\d{1,2}'

  AvailabilityZone1:
    Description: La zona de disponibilidad a utilizar para la primera subred
    Type: AWS::EC2::AvailabilityZone::Name

  AvailabilityZone2:
    Description: La zona de disponibilidad a utilizar para la segunda subred
    Type: AWS::EC2::AvailabilityZone::Name

  AvailabilityZone3:
    Description: La zona de disponibilidad a utilizar para la tercera subred
    Type: AWS::EC2::AvailabilityZone::Name

  PublicSubnet1CIDR:
    Description: Ingrese el rango de IP (CIDR) para la subred publica en la primera zona de disponibilidad
    Type: String
    Default: 10.173.160.0/24
    AllowedPattern: '((\d{1,3})\.){3}\d{1,3}/\d{1,2}'

  PublicSubnet2CIDR:
    Description: Ingrese el rango de IP (CIDR) para la subred publica en la segunda  zona de disponibilidad
    Type: String
    Default: 10.173.163.0/24
    AllowedPattern: '((\d{1,3})\.){3}\d{1,3}/\d{1,2}'
```

Because the solution has existing resources nested stacks are used, they provide us with the availability of linking the contents of one stack with another, it is mainly used to generate new lambdas, which require the use of subnets, security groups and roles generated in previous stacks.

Using good practices for generating CloudFormation templates, these are created with groups of parameters, in order to differentiate the resources and services generated. In addition to the parameter groups, the templates come with the output of the main resource data, such as the APIs id, balancers dns, lambdas arn, etc.



```
Metadata:
  AWS::CloudFormation::Interface:
    ParameterGroups:
      - Label:
          default: Parametros principales
        Parameters:
          - AccountNamingConstructor
          - VpcCIDR
          - MasterOrLinked
          - MasterTransitGateway
      - Label:
          default: Primera Zona de disponibilidad
        Parameters:
          - AvailabilityZone1
          - PublicSubnet1CIDR
          - HybridSubnet1CIDR
          - PrivateSubnet1CIDR
      - Label:
          default: Segunda Zona de disponibilidad
        Parameters:
          - AvailabilityZone2
          - PublicSubnet2CIDR
          - HybridSubnet2CIDR
          - PrivateSubnet2CIDR
      - Label:
          default: Tercera Zona de disponibilidad
        Parameters:
          - AvailabilityZone3
          - PublicSubnet3CIDR
          - HybridSubnet3CIDR
          - PrivateSubnet3CIDR
```

Generated templates provide ease of deployment or reuse for other environments streamlining the creation of resources and also facilitating the administration and maintenance of the workforce and general environment.

STACK MANAGEMENT:

The version controls of the templates are stored in Code Commit repositories, branchs are created in the repository according to modifications and updates, in the same way and in order to protect the information and content of the templates, they are stored in an S3 bucket, where it is used versioned to keep the content updated without losing the previous modifications.

In order to streamline the process of modifying resources, mainly the templates of lambdas functions, when they are modified and go through the process of continuous integration and deployment, they automatically update the corresponding stack, maintaining the ARN and integration. with other services, modifying only the variables and code, generating a new version of the function and this modification in the template is stored in the code commit and bucket S3 repository.
Before generating the templates, names are defined for some resources which after creation cannot be modified, this in order to avoid recreating an existing and working resource, one of these services is the name of the S3 buckets, since these are global and cannot be repeated, the name of the resource is defined before the creation of a template, avoiding subsequent modifications.

During the generation of the resources, in order to maintain the previously generated stacks, Update rollback is performed, this is used in the event that the releases to the environments contemplate an error, allows the elimination of specific resources generated in the stack, preventing the elimination from other resources. Each stack has the Stack termination protection option active to avoid the elimination of highly productive resources, in addition to maintaining control of the resources generated and which are essential for the operation of the solution.

AWS config is used to control resources, which helps to monitor stack updates. When a productive stack undergoes a modification, SNS alarms are sent, obtaining a history of modifications in the Qa and prod environments. Additionally, personalized resources are used in lambdas, and with CloudFormation templates function updates are generated, mainly, they are used to enable new functionalities in the micro services.

URL templates.

http://ec2-44-231-83-193.us-west-2.compute.amazonaws.com/cloudfromationpublicos/sdp-cloudformations/tree/master