



Desarrollo de aplicaciones en la nube

Grupo 1

Juan Vélez

Presentación de avance de proyecto final

Tianguito



1 de diciembre de 2020

Ricardo Andrés Aldama González | A01337569

Erick David Gil Jackson | A01337675

Jesús Iván Tapia Romero | A01338275

Axel Octavio Menguer Estrada | A01653448

Tabla de Contenidos

La solución que es Tianguito	4
El Objetivo de la App	5
Las historias de Usuario	6
La Arquitectura de alto nivel	9
Servicios Usados	11
Las Redes	18
Security Groups	18
El S3 Bucket con BucketPolicy	19
La Base de datos	21
DynamoDB	21
RDS	22
Conclusiones	24
Referencias	24

La solución que es Tianguito

La pandemia de Covid-19 que azota al mundo es un evento que ha cambiado la forma en la que todos se relacionan y organizan, independientemente de que hubieran querido o no.

Uno de estos sectores forzados al cambio fueron los llamados tianguis, comercios semi-ambulantes de corte de pequeña o micro empresa. Estos puestos se vieron increíblemente perjudicados por la cuarentena y las medidas de confinamiento y distanciamiento social.

Tianguito es una forma para ayudar a este sector económico en la llamada “nueva normalidad.” Una PWA (Progressive Web App) dedicada a hacer uso de servicios y tecnologías de nueva generación para poder darle a los tianguis una forma de operar de acuerdo con las normas de salubridad.

Por eso es que Tianguito ayuda a conectar el concepto de comprar en los tianguis, donde muchas personas lo usan como método predilecto para comprar víveres, con el concepto de mantenerse en casa. Dándole la opción a los clientes de poder hacer pedidos y poder observar todo lo que se encuentra en los puestos de su tianguis de conveniencia desde la comodidad de su casa con sólo la ayuda de su dispositivo móvil. Permitiendo seguir sus pedidos en tiempo real con herramientas como streaming de video y pagos en línea.



El Objetivo de la App

Tianguito nació con la llegada de la pandemia de COVID-19. La llegada de la cuarentena que destruyó la mayoría de las economías del mundo moderno llevó a que muchos negocios e incluso industrias enteras se vieran en dificultades no conocidas desde hace ya casi cien años.

En un país como México, especialmente, donde la mayoría de la población no sólo vive en una situación económica precaria, pero viven de la economía informal, el golpe de la pandemia fue algo muy difícil y adaptarse a la nueva normalidad resulta un reto que todavía hoy en día se ha cobrado los empleos de millones de personas.

Tianguito llega como una forma de poder comenzar una transición a un mundo más allegado a la tecnología y el comienzo de una plataforma que por medio de tecnologías y servicios de última generación sean capaces de ayudar a un sector que se ha visto incapaz de operar en su totalidad como antes del confinamiento podía hacerlo.

Además, Tianguito piensa poder ser mantenible con facilidad y poder ser usado de forma cómoda para los usuarios y mantenimiento sencillo para los desarrolladores con campo de poder expandirse y aumentar su alcance si es que esto es necesario conforme los tiempos cambian, por medio de una alta modularidad.



Las historias de Usuario

Para poder crear una experiencia de usuario correcta y cómoda. El equipo de desarrollo de Tianguito se dio a la tarea de pensar y empatizar con los diversos usuarios que serán parte de la comunidad de la aplicación. De esta manera, creando diversos escenarios hipotéticos que podían dar a relucir las principales funcionalidades y requerimientos críticos necesarios para los usuarios de la aplicación.

Historia de Usuario #1

Juan es una persona no muy afín a la tecnología, por lo que hará uso de alguna clase de método tecnológico para poder comprar en su tianguis local, por lo que desearía que la página principal sea intuitiva y le deje ver rápidamente su tianguis y poder acceder a este desde la página principal.

Historia de Usuario #2

Alberto es un vendedor de tianguis que necesita poder subir a la aplicación cualquier producto que venda, y por las porciones deseadas según su criterio. Además a veces tiene disponibles algunos productos por temporadas por lo que ha de poder gestionar su menú fácilmente.

Historia de Usuario #3

Karla respecto a todo lo de la pandemia preferiría que se pudiese evitar un método de pago que pase por varias manos. Un método electrónico o que sea directo entre ella y el dependiente podría resultar mucho más útil y seguro de acuerdo con los tiempos.

Historia de Usuario #4

Iván es un vendedor de tianguis que desea poder ingresar a la plataforma y poder registrarse con su correo electrónico y una contraseña para poder manejar un perfil individual y poder acceder a la plataforma con esa información.

Historia de Usuario #5

Daniel es un puestero que vende carnes en un tianguis y al asociarse a la plataforma de Tianguito, desearía según sus posibilidades poder dar de alta el mercado donde se encuentra, ya que es la primera en sumarse a la red de Tianguito y que esté disponible para el resto de usuarios mercaderes.

Historia de Usuario #6

Gertrudiz es una señora de la tercera edad que dada la situación actual del riesgo de la pandemia, se le considera como una persona de riesgo y ya no puede salir sin exponerse peligrosamente. Así que se sentiría menos amenazada si sus compras se las llevara a su domicilio un repartidor de vez en cuando de forma que sea más sencillo mantener las distancias sanitarias.

Historia de Usuario #7

David quiere poder indicar con facilidad cuál es el mercado más cercano a su vivienda para poder ir cómodamente. Con un mapa donde pueda, ya sea poner su localización o, una dirección.

Historia de Usuario #8

Ricardo quiere comprar un pastel, unas velas, unos carritos y una pelota para su hijo Andres por lo que realizará compras en diversos puestos abiertos de forma que se

agrupe todo el pedido en un carrito de compras y se pague en una sola exhibición para el cumpleaños de Andrés.

Historia de Usuario #9

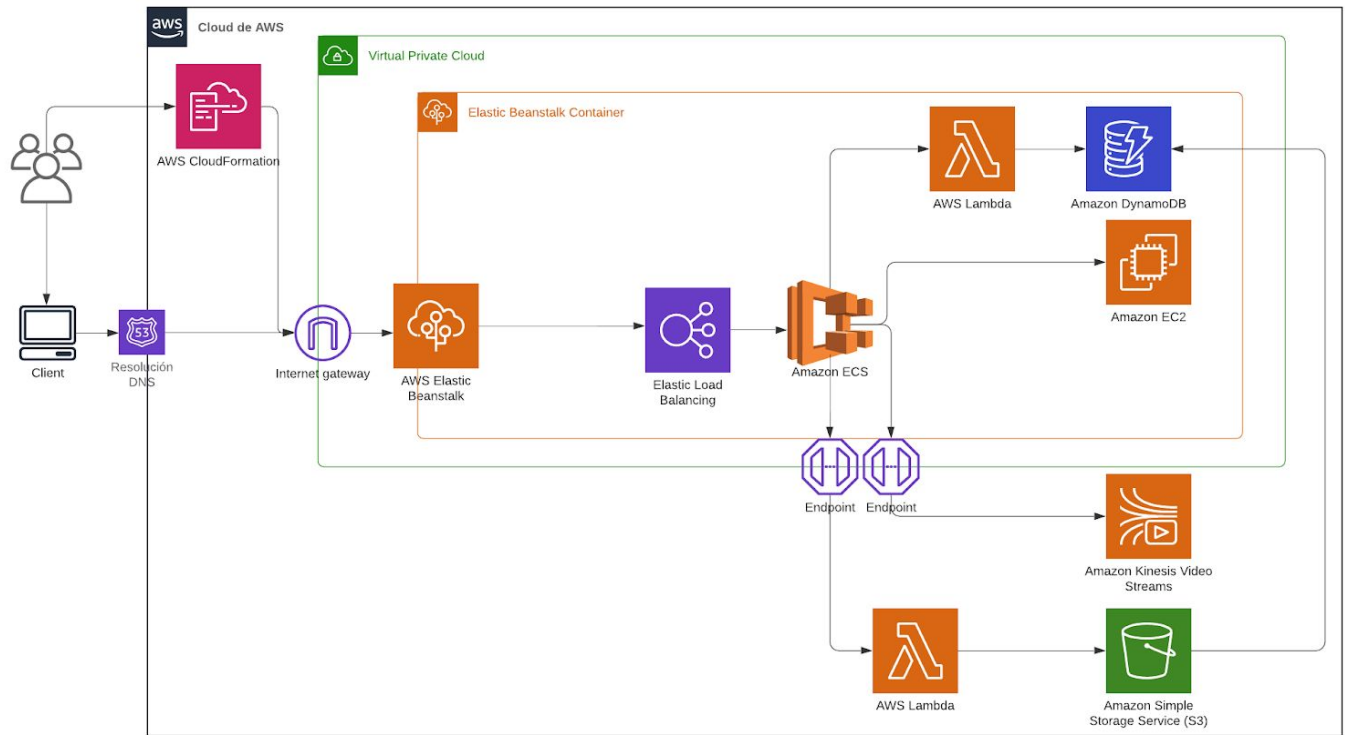
Erick desea cambiar los datos de su cuenta de vendedor ya que se hizo nuevo un correo específico para la app, de extensión gmail, y además planea actualizar su contraseña.

Historia de Usuario #10

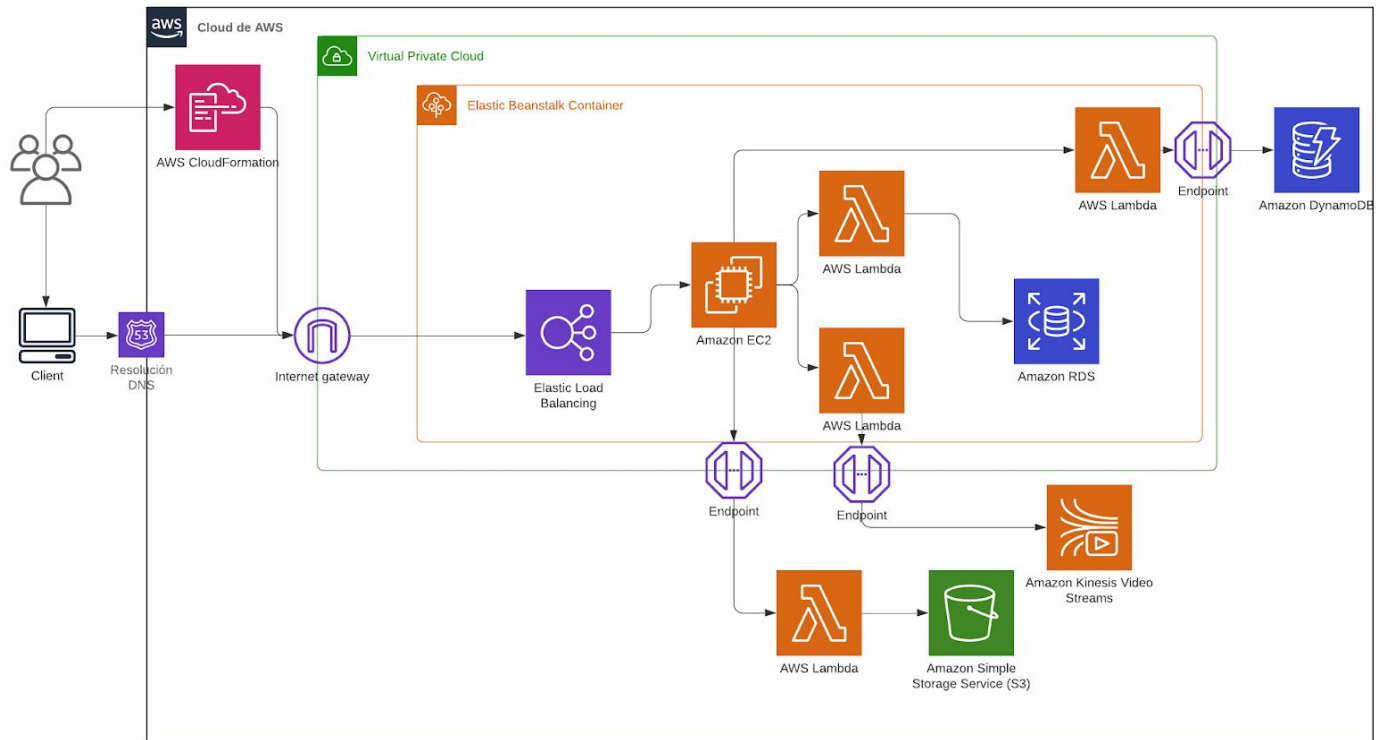
Rodrigo es fan del queso Oaxaca y suele tomarse su tiempo para elegir y filtrar qué queso comprará, por lo que le gustaría poder ver en tiempo real los quesos y elegirlos según su aspecto, de forma que no se de la situación donde el repartidor le lleve un queso de no tan buen calidad según su criterio.

La Arquitectura de alto nivel

El siguiente es el modelo original, de la **arquitectura de alto nivel propuesta**:



El siguiente es el modelo real, de la **arquitectura de alto nivel** realizada:



Servicios Usados

La arquitectura de Tianguito se encuentra construida casi en su totalidad con las herramientas ofrecidas por Amazon Web Services, explotando al máximo sus capacidades de integración y de compatibilidad entre las mismas para poder entregar un proyecto sólido y que es altamente escalable y capaz de poder funcionar sin problemas con cada uno de sus diferentes componentes.

A continuación se muestra una lista completa de todos los servicios usados en la creación de la aplicación de Tianguito tanto en su etapa de planificación como en el producto final.

Virtual Private Cloud (VPC)

Se usa para definir una sección en la nube en AWS que se encuentra aislada de forma lógica y es capaz de lanzar servicios AWS en una red virtual (con acceso a recursos y apps de forma segura y sencilla), en ella es posible definir un rango de direcciones IP y subredes, así como configurar tablas de routing y gateways. Por otro lado permite funcionalidad de sistemas backend dentro de subredes privadas (sin acceso a internet), y posee grupos de seguridad y listas de control de acceso, para mejor manejo de instancias en EC2.

En el producto final se cumple todo lo propuesto con la creación y el uso de la VPC que crea una unidad aislada adecuada en la cual poder trabajar e integrar todas las funciones y componentes de Tianguito.

AWS CloudFormation

CloudFormation tendrá la principal función de crear todo el ambiente desarrollado para la aplicación dentro del ambiente de AWS. Para esto hará uso de archivos .yaml para poder recibir las instrucciones y crearlo.

En la construcción final de la aplicación, esto igualmente se cumple, ya que todo el ambiente de AWS para el proyecto se puede reconstruir desde cualquier otra cuenta con todos los recursos necesarios.

AWS Elastic Beanstalk

Elastic Beanstalk es la principal herramienta para la creación de todo el ambiente de AWS una vez formada la VPC que contendrá todos los servicios de AWS. Beanstalk se encarga de configurar todo lo necesario para poder formar la conexión entre servicios, subredes y balanceador de cargas. Se encuentra conformado por otros servicios que configura de forma automática para el usuario.

Sin embargo, en el producto final el equipo de desarrollo decidió que el uso de AWS Elastic Beanstalk al final podía omitirse, ya que el resto de sus componentes podía ser creado de forma manual y podía ser trabajado de mejor manera si se creaban como entidades separadas y se creaban cada una en el archivo de CloudFormation.

Elastic Load Balancing

Su principal función se encuentra en formar un balanceador de cargas capaz de direccionar el flujo de la aplicación a donde haya más disponibilidad en los servidores dentro del ambiente.

Esta funcionalidad de igual manera se integró a la aplicación final para poder mantener ese control sobre la disponibilidad y la dirección de flujo que reciba la aplicación.

Amazon ECS

Se conoce como Amazon Elastic Container Service, se encarga de administrar contenedores, acepta clientes como duolingo, samsung, ge, y más. Posee

escalabilidad, seguridad y fiabilidad. Se usa para ejecutar contenedores y puede funcionar con EC2, a la vez que controla tráfico, brinda capacidad de observación y funciones de seguridad especializadas.

Este fue otro componente que se decidió era mejor prescindir ya que la creación de la aplicación no necesitaba del uso de múltiples contenedores, pero es algo que sin duda se puede ampliar en un proyecto a futuro, ya que permitir que la aplicación funcione en un sistema de contenedores la haría no sólo más modular y completa.

Amazon EC2

Conocido como Amazon Elastic Compute Cloud, es un servicio que proporciona capacidad informática modificable en la nube (instancias), es decir, máquinas virtuales a la medida según las necesidades web. Sus instancias poseen claves, etiquetas de recursos, direcciones, firewalls y volúmenes (almacenamiento), en caso de ser manejadas en grupos pueden integrarse en subredes de VPC.

El servicio de Amazon EC2 también fue implementado de manera correcta para poder desplegar la aplicación de Tianguito en un servidor que le permitiera ser accesada desde cualquier punto al momento de ya encontrarse en producción.

Amazon RDS

Ambos servicios de bases de datos de AWS forman parte de la aplicación. Tanto la no relacional como la tradicional, en este caso RDS que se encarga de mantener tablas entidad-relación de algunos aspectos vitales y que necesitan alta estandarización para ser adecuados.

Los servicios de Amazon RDS de igual forma fueron integrados como parte del sistema para guardar información sobre los vendedores y los tianguis que forman parte de la red de la aplicación. Gracias a sus fuertes relaciones al tratarse de bases de datos relacionales tipo SQL, fueron de gran utilidad para este tipo de elementos.

Amazon DynamoDB

Para la base de datos de la aplicación se hará uso de la propia base de datos no relacional de Amazon, conocida como DynamoDB ya que cuenta con una buena integración al resto de la arquitectura dentro del ambiente de AWS.

La base de datos que más se usó en la aplicación fue el segundo servicio de base de datos que Amazon Web Services ofrece. Este segundo servicio es mucho más cómodo y moderno a su contraparte relacional, además de ofrecer un servicio *serverless* e independiente al resto de la arquitectura de la VPN. DynamoDB fue usado para guardar en forma de documentos la mayoría de la información que es usada por la aplicación.

Amazon Kinesis Video Streams

Dentro de los servicios que se usarán de AWS el Kinesis Video Streams será de especial uso para el servicio de video streaming que usará la aplicación, donde los dueños de un negocio pueden mostrar sus productos en tiempo real.

La funcionalidad de Amazon Kinesis Video Streams aunque originalmente fue la planeada para la aplicación y se llevó hasta la entrega final se encontró un último problema al momento de su implementación. Ya que Amazon Kinesis Video Streams aún no se encuentra optimizado para poder funcionar con backend programado por medio de Javascript en el entorno de NodeJS, por lo que su integración incluso con Lambda Functions sería bastante complejo ya que se

buscaría la integración de dos lenguajes de programación distintos. Además de tener que ser adaptados y aprendidos por el equipo de desarrolladores en último momento.

Amazon Simple Storage Service (S3)

El servicio de S3 funcionará junto con DynamoDB para todo lo relacionado a carga de información de la aplicación, en el caso de S3 se especializa en la carga de los archivos e imágenes especialmente para el uso de la aplicación.

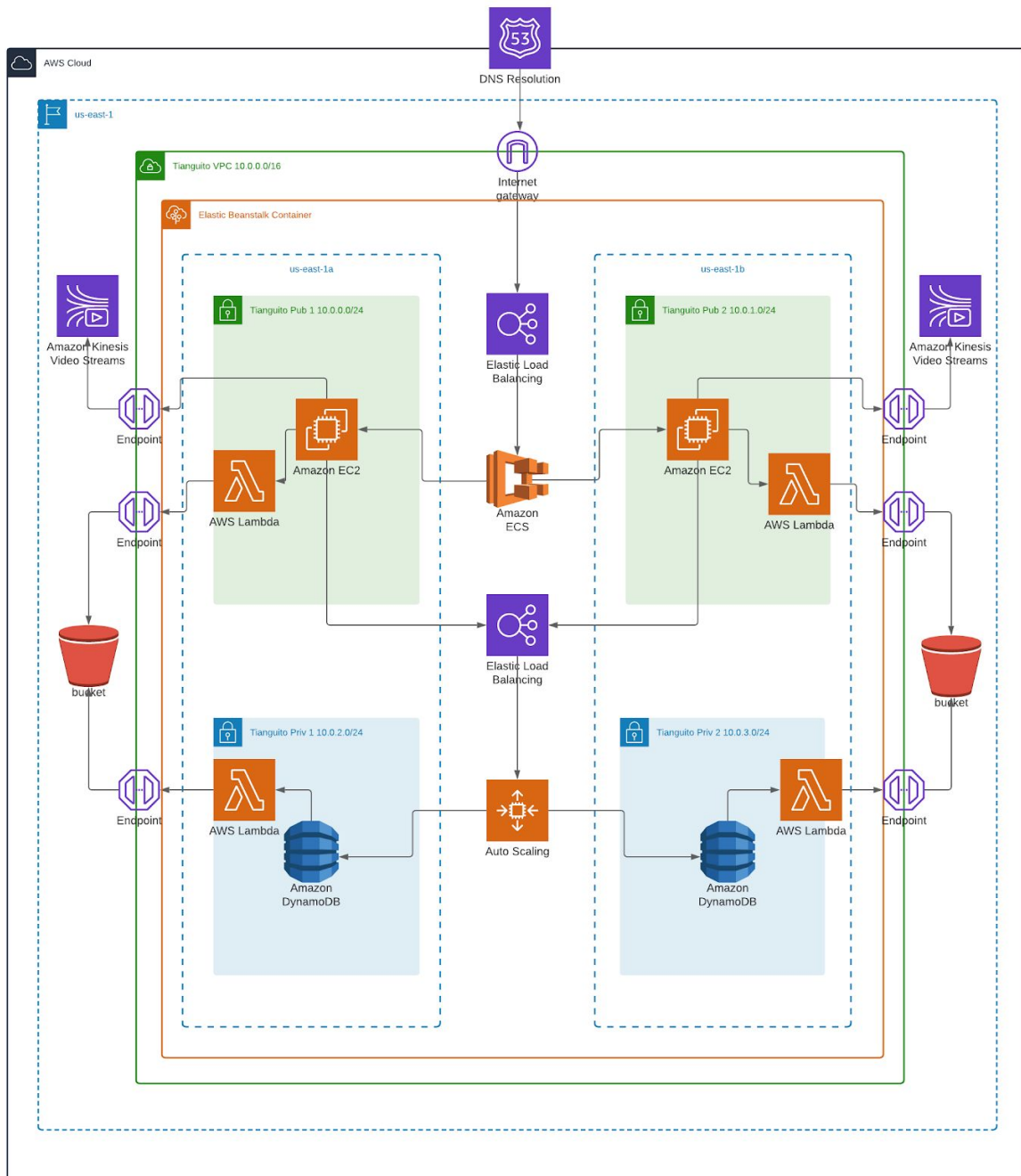
El funcionamiento implementado para el Amazon Simple Storage Service, conocido como S3 fue la integración de los buckets para guardar información muy concreta de la aplicación y su funcionamiento no necesariamente relacionada de forma estrecha con los usuarios. Gracias a su formato más libre de almacenamiento se pueden guardar cosas como archivos entre otras cosas en vez de nada más datos como en los servicios antes mencionados de bases de datos.

AWS Lambda

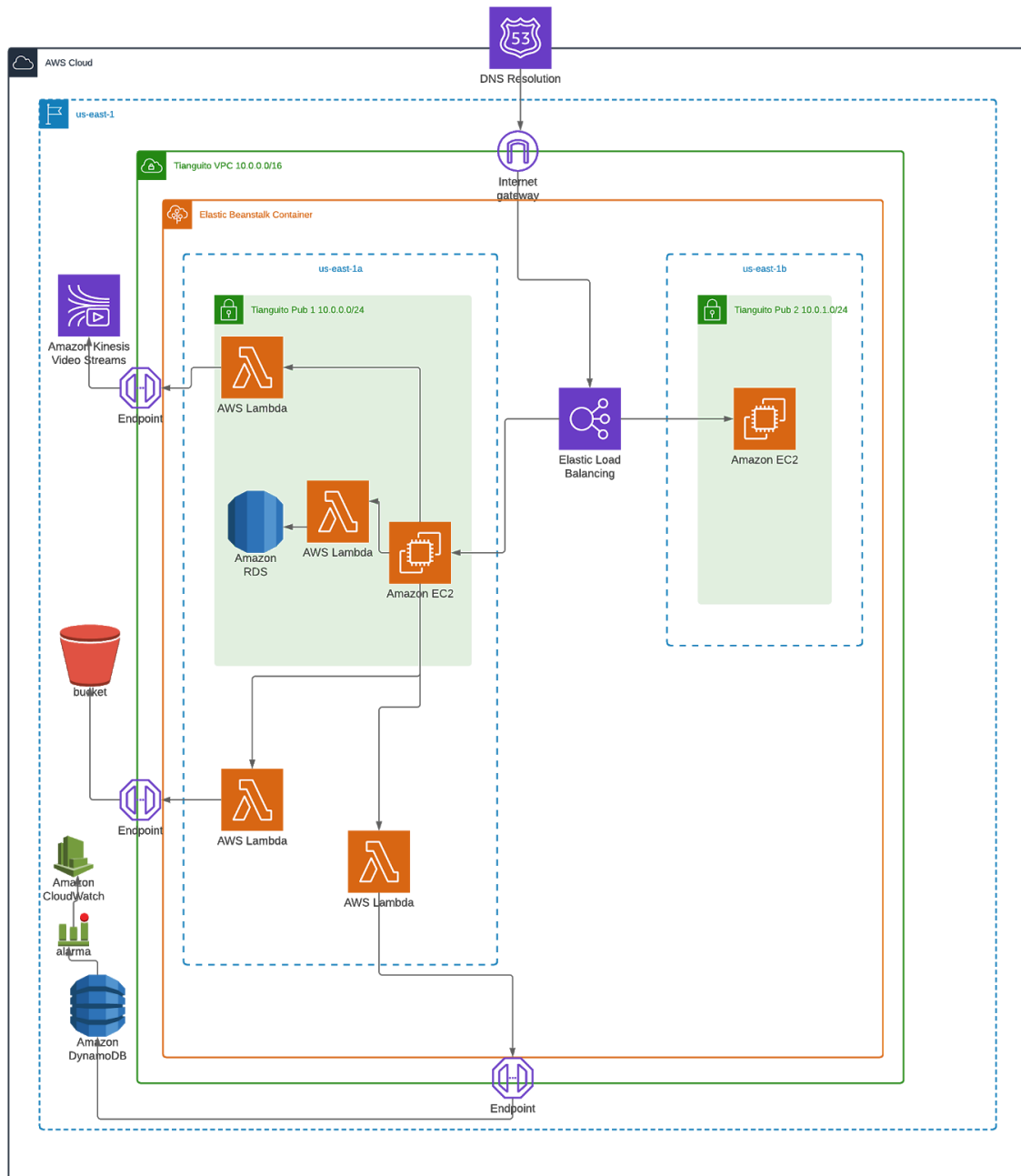
AWS Lambda será utilizado para la comunicación entre varios servicios de AWS, por ejemplo, se tendrá una función en AWS Lambda que sea activada cuando la aplicación indique que quiere guardar un dato en la base de datos; esta función recibirá los datos a guardar y tendrá el código necesario para pasar los datos a DynamoDB y asegurarse de que se guarden de manera correcta. Adicionalmente, se tendrán funciones Lambda para S3 y Amazon Kinesis, para guardar archivos y manejar streams de video, respectivamente.

Las AWS Lambda se implementaron de numerosas maneras para poder conectar de forma exitosa múltiples servicios de Amazon Web Services de forma sencilla y rápida utilizando pocos recursos y líneas de código separadas e independientes para cada caso y servicio diferente.

La siguiente es la **arquitectura que se planeó** en primer lugar



Ahora se presenta el **modelo de arquitectura actualizado** y realizado:



Las Redes

Para este proyecto, utilizaremos una VPC con la dirección 10.0.0.0/16, con dos pares de subnets (una pública y una privada) en dos diferentes zonas de disponibilidad. Para las subredes, utilizaremos las direcciones 10.0.0.0/24 y 10.0.1.0/24 en el caso de las públicas, y en el caso de las privadas, 10.0.2.0/24 y 10.0.3.0/24. Utilizaremos las zonas de disponibilidad us-east-1a para la primera pareja de redes y us-east-1b para la segunda pareja.

En cuanto a la tabla de ruteo, la VPC tendrá su tabla, con la respectiva asociación de subred para cada una de las subredes antes mencionadas. Adicionalmente, se agrega una regla a la tabla de ruteo que permita salida al internet a través del Internet Gateway que será adherido a la VPC.

Se requerirán VPC endpoints para los servicios de Amazon Kinesis Video Streams y las Buckets de Amazon S3.

Security Groups

Para las Network ACL's, se permite cualquier tráfico de tipo HTTP o HTTPS. Adicionalmente, se permite el tráfico UDP y TCP en el puerto 53 para la resolución DNS. Finalmente, se permite el tráfico SSH y RDP para que sea posible conectarse con EC2.

Para el diseño del security group se deben definir las reglas de tráfico entrante y de tráfico saliente, para las de tráfico entrante se utilizaron 3 reglas: HTTPS y HTTP con protocolo TCP, para estas dos reglas la fuente permitida se dejó como “de cualquier lado” esto para permitir que los usuarios entren a la aplicación, además se agregó una tercer regla de Custom TCP con el puerto 8111 ya que este puerto es necesario para la comunicación con DynamoDB.

El S3 Bucket con BucketPolicy

```
Resources:
  S3Bucket: #Configuracion principal del bucket PARA TIANGUITO PRIMER PARCIAL
    Type: AWS::S3::Bucket
    Properties:
      BucketName: tianguito-bucket3 #Nombre del bucket = ID fisico
      AccessControl: PublicRead #Medidas de acceso al bucket, tipo de lectura o privado
      WebsiteConfiguration: #Configuracion para hosteo de website
      IndexDocument: index.html #index
      ErrorDocument: error.html #error default
      RoutingRules: #Reglas de ruteo
      - RoutingRuleCondition: #Condicion de ruteo
        HttpStatusCodeReturnedEquals: '404' #code error
        KeyPrefixEquals: app/ #prefijo de las rutas a redireccion
      RedirectRule: #reglas de redireccion
        HostName: tianguito-bucket.us-east-1.amazonaws.com #hostname no puede tener /
        ReplaceKeyPrefixWith: report-404/
      DeletionPolicy: Retain #politica del delete del cubo (retain, evita que el cubo se borre
        #si tiene algo en su stack)
  BucketPolicy: #Declaracion de una politica del bucket, solo se puede colocar 1 por bucket
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket:
        Ref: "S3Bucket" #(Bucket donde se aplica) id logico para iniciializacion de cloudformation*
      PolicyDocument: #Formato de una politica//ejemplo para lectura de objetos (files) cargados
        Id: MyPolicy #Name physical id
        Statement: #Declaracion de una politica, solo 1 por bucket
          - #Politica de GET abierta a todas las requests (full access)
            Action:
              - 's3:GetObject' #aplica al type y Bucket -> Ref
            Effect: Allow #parametro
            Principal: '*'
            Resource: #construccion de la ruta
              Fn::Join:
                - ""
                -
                  - "arn:aws:s3:::" #prefix
                  -
                    Ref: "S3Bucket" #logical id*
                    - "/*"

#NOTA: Cuando se aplica esta plantilla en cloudformation, se crea un bucket que
#guarda el .yaml pero no le aplica las politicas el bucket que importa es que se nombra al
#inicio de este recurso "BucketName" a ese se le aplican las politicas
```

En el esquema anterior se muestra un borrador preliminar de la configuración .yaml y creación de un bucket que se haría con un template de Cloudformation de S3, por cómo funcionan los buckets de S3 sabemos que con declararlo y configurar el control de acceso de los archivos ya se pueden compartir y leer archivos. (Es más sencillo manejarlo en el dashboard de S3 o de AWS). Por otro lado en esta configuración además definimos una política (**PolicyDocument**) de acceso de **GET requests** con la que se accedería a los archivos que se han subido, desde la ruta dada, se recomienda esta política para sitios web ya que da full access. Los objetos en un bucket son equivalentes a los archivos que se cargan en la nube, el bucket de S3 es escalable y acepta archivos de diversos volúmenes, después estos mismos pueden descargarse siguiendo una url y configurar su acceso a grupos.

A. Un ejemplo de la implementación de un archivo compartido en S3:

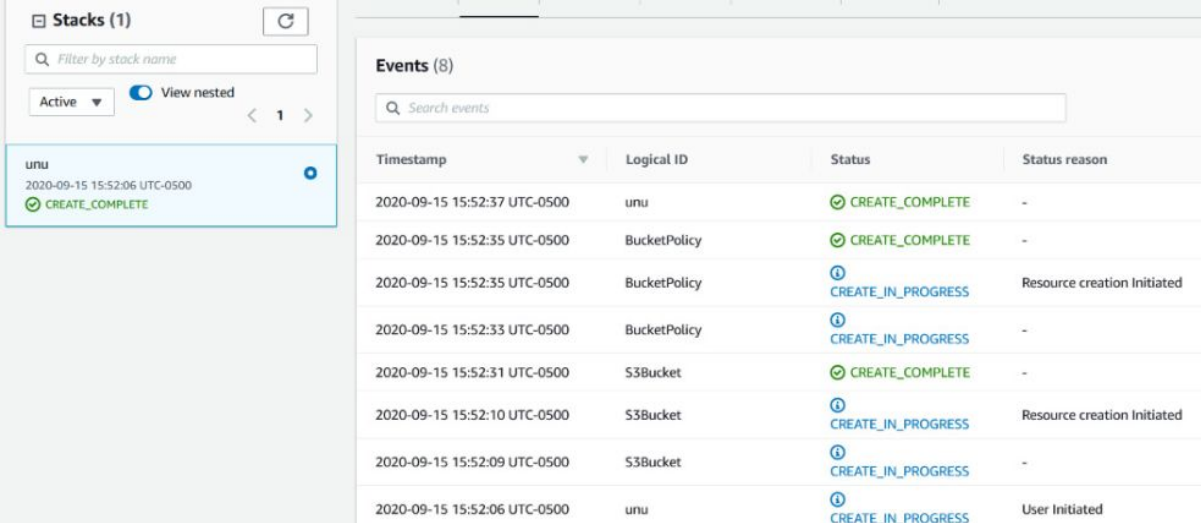
<https://tianguito-bucket2.s3.amazonaws.com/Tianguis.JPG>

B. Un ejemplo de una implementación de un sitio web estático en aws

(imágenes, css, js y htmls)

<https://tianguito-bucket2.s3.amazonaws.com/index.html>

También se han definido algunas propiedades relacionadas al hosteo de un sitio web en él, sea rutas home, rutas de redirección y dominios.



Stacks (1)	
Filter by stack name	
Active	View nested
unu	2020-09-15 15:52:06 UTC-0500 CREATE_COMPLETE

Events (8)				
Search events				
Timestamp	Logical ID	Status	Status reason	
2020-09-15 15:52:37 UTC-0500	unu	CREATE_COMPLETE	-	
2020-09-15 15:52:35 UTC-0500	BucketPolicy	CREATE_COMPLETE	-	
2020-09-15 15:52:35 UTC-0500	BucketPolicy	CREATE_IN_PROGRESS	Resource creation Initiated	
2020-09-15 15:52:33 UTC-0500	BucketPolicy	CREATE_IN_PROGRESS	-	
2020-09-15 15:52:31 UTC-0500	S3Bucket	CREATE_COMPLETE	-	
2020-09-15 15:52:10 UTC-0500	S3Bucket	CREATE_IN_PROGRESS	Resource creation Initiated	
2020-09-15 15:52:09 UTC-0500	S3Bucket	CREATE_IN_PROGRESS	-	
2020-09-15 15:52:06 UTC-0500	unu	CREATE_IN_PROGRESS	User Initiated	

En este esquema se demuestra la implementación del .yaml del bucket de S3 con BucketPolicy de full read access.

La Base de datos

Para el diseño de la base de datos, como ya se ha establecido anteriormente en la arquitectura de alto nivel, se hará uso de otro de los servicios de AWS, en este caso los dos servicios de bases de datos de AWS, **Amazon DynamoDB** y **RDS**.

DynamoDB

Amazon DynamoDB se trata de una base de datos de clave-valor, no relacional y duradera en varias regiones, completamente administrada.

Las ventajas que ofrece DynamoDB es su alto nivel de escalabilidad y su increíble compatibilidad con aplicaciones ya sea nativas o web. Además de tratarse de un servicio con la característica de poder aplicarse dentro del framework de “serverless” con la simple modificación de un archivo .yaml.

Otro punto importante por el que se decidió decantarse por el uso de DynamoDB es por su fácil integración con el resto de servicios de Amazon. Y ya que el modelo de arquitectura en la nube de Tianguito está hecho alrededor del uso de servicios de AWS se vio como la respuesta más obvia a la cuestión de almacenaje de datos de los usuarios de la aplicación.

Las integraciones más importantes en este proceso pueden ser su fácil manejo por medio de **AWS Lambda** por medio de **DynamoDB Streams**, que permiten configurar funciones lambda de AWS de tal forma que puedan correr cada vez que hay algún cambio en alguna tabla de DynamoDB.

Adicionalmente también se puede integrar con facilidad con el servicio de **Amazon S3**, permitiendo crear respaldos para cada nuevo registro de DynamoDB hacía S3 por medio de funciones lambda.

Grupos de Subnet en la base de datos

Para los grupos de subnet, DynamoDB hace uso de una herramienta específica para este servicio conocido como **Amazon DynamoDB Accelerator** (DAX) que se encarga específicamente de mantener el acceso desde las subnets privadas donde se encuentran las bases de datos, como mostrado en el diagrama, al resto de instancias de EC2. Siguiendo los parámetros para el desarrollo de estas subnets, se tratan de grupos privados hechos en dos regiones distintas dentro de la misma zona de disponibilidad.

En gran parte la mayoría de las interacciones de DynamoDB se soportará por medio de sus interacciones con el ya mencionado DAX que se encargará de todo tipo de acceso tanto dentro de la red como de acceso provenientes fuera de la VPN.

RDS

El segundo servicio que se usó para la creación de Tianguito es el otro servicio de base de datos que ofrece AWS, el Servicio de bases de datos relacionales (RDS por sus siglas en inglés).

El servicio suministra capacidad rentable y escalable al mismo tiempo que automatiza las arduas tareas administrativas, como el aprovisionamiento de hardware, la configuración de bases de datos, la implementación de parches y la creación de copias de seguridad.

A diferencia de DynamoDB, que opera como lo haría un servicio externo a la arquitectura dentro de la VPN, las instancia de RDS operan dentro de la misma, por lo que pueden ser controladas y escaladas sin necesidad de salir de la VPN. Esto facilita más su control y le permite a la aplicación un uso más controlado de sus datos y de la seguridad de los datos que se guarden en las tablas.

Como se ha de suponer por el nombre, a diferencia de DynamoDB que opera por medio de documentos en un esquema no relacional, Relational Database

Service de AWS opera como lo hacen las bases de datos más tradicionales, por medio de esquemas entidad-relación. En el caso de Tianguito este sistema se hará por medio de una base de datos en estilo SQL que mantiene relaciones a tablas de los datos sobre los tianguis que los usuarios, más en particular los vendedores del propio tianguis, den de alta al sistema para poder formar parte del mismo.

Las aplicaciones web y móviles que se crean para funcionar a gran escala necesitan una base de datos con un gran rendimiento, una enorme escalabilidad de almacenamiento y una disponibilidad alta. Amazon RDS cubre las necesidades de estas exigentes aplicaciones y ofrece el espacio suficiente para que puedan crecer en el futuro. Como no existen restricciones de licencia en Amazon RDS, el servicio se ajusta perfectamente a los patrones de uso variables de estas aplicaciones.

Además de todo esto la alta disponibilidad, confiabilidad y seguridad que ofrece RDS además de la sencilla integración con no sólo los otros servicios, pero con la arquitectura en sí sobre la que se construye la aplicación de Tianguito es una perfecta herramienta para poder construir elementos clave del proyecto.

Conclusiones

Con el ocaso del proyecto llegando al fin, se puede vislumbrar el aprendizaje y lecciones aprendidas con la realización de un proyecto tan denso y robusto como el emprendido por el equipo de desarrollo de Tianguito.



La integración de los servicios de Amazon Web Services es una gran oportunidad y práctica para poder desarrollar aplicaciones cada vez más de más alto calibre y potencia por medio de este tipo de herramientas que ayudan a hacer arquitecturas que puedan tener mayor alcance.

Aprender tecnologías en la nube resulta no sólo de alto nivel de aprendizaje sino que es algo fundamental en un mundo que cada vez se encuentra más en el camino del *serverless*, con funcionalidades cada vez más dependientes de una conexión de internet y de mantener todo fuera de un sistema tradicional de servidores.

A las puertas de la finalización del proyecto se puede ver la entrega final de Tianguito más como un inicio en un conocimiento más profundo e intenso de un campo que será vital en especial en los campos de trabajo que muchos de los miembros del equipo pronto emprenderán.

El equipo de desarrollo de Tianguito

Y sus estudiantes favorito (además de Carlos)

Referencias

- <https://aws.amazon.com/es/products/>
- <https://aws.amazon.com/es/dynamodb/>
- <https://www.serverless.com/dynamodb>
- <https://aws.amazon.com/es/blogs/database/how-to-configure-a-private-network-environment-for-amazon-dynamodb-using-vpc-endpoints/>
- <https://docs.aws.amazon.com/AmazonS3/latest/dev/VirtualHosting.html>
- <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-s3-websiteconfiguration-routingrules-routingrulecondition.html>
- <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-s3-policy.html>
- <https://docs.aws.amazon.com/AmazonS3/latest/dev/example-bucket-policies.html>
- <https://docs.aws.amazon.com/vpc/latest/userguide/vpce-interface.html>
- <https://docs.aws.amazon.com/vpc/latest/userguide/vpc-network-acls.html#n-acl-rules>
- https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Route_Tables.html
- <https://aws.amazon.com/es/rds/>