Prueba AWS Acreditta

Realizado por: Juan José Monsalve
Fecha: 2024-08-02

1.

> Se tiene una aplicación backend contenerizada en aws, esta solución tiene JWT como sistema de autenticación, debido a una nueva integración con

Inicialmente mediante esta definición no se especifica exactamente la arquitectura de la cual se encuentra compuesta el servicio Backend existente, pero sin embargo se puede intuir que este se puede encontrar alojado en una instancia de **EC2** o en un orquestador de contenedores como lo es **ECR** mediante **ECS** y **AWS Fargate**, para este caso por simplicidad se tomará el Backend como un servicio alojado en ECS.

2.

> generación de JWT porque la solución final hará que a estas empresas se les entregue APIKEYS para el llamado de esta API, estas APIKEYS tienen que tener un manejo de rotación y seguridad. Tienen que existir trazas de las peticiones

En este caso, la sugerencia más recomendable sería utilizar un servicio como **AWS KMS** que nos permite controlar y administrar claves criptográficas en las aplicaciones.

3.

> un manejo de rotación y seguridad. Tienen que existir trazas de las peticiones realizadas para poder diagnosticar y analizar diferentes tipos de fallas que pueden pasar. Adicional a eso se debería poder tener métricas por empresa para procesos de cobros por consumo de API. Por ultimo muchos de los

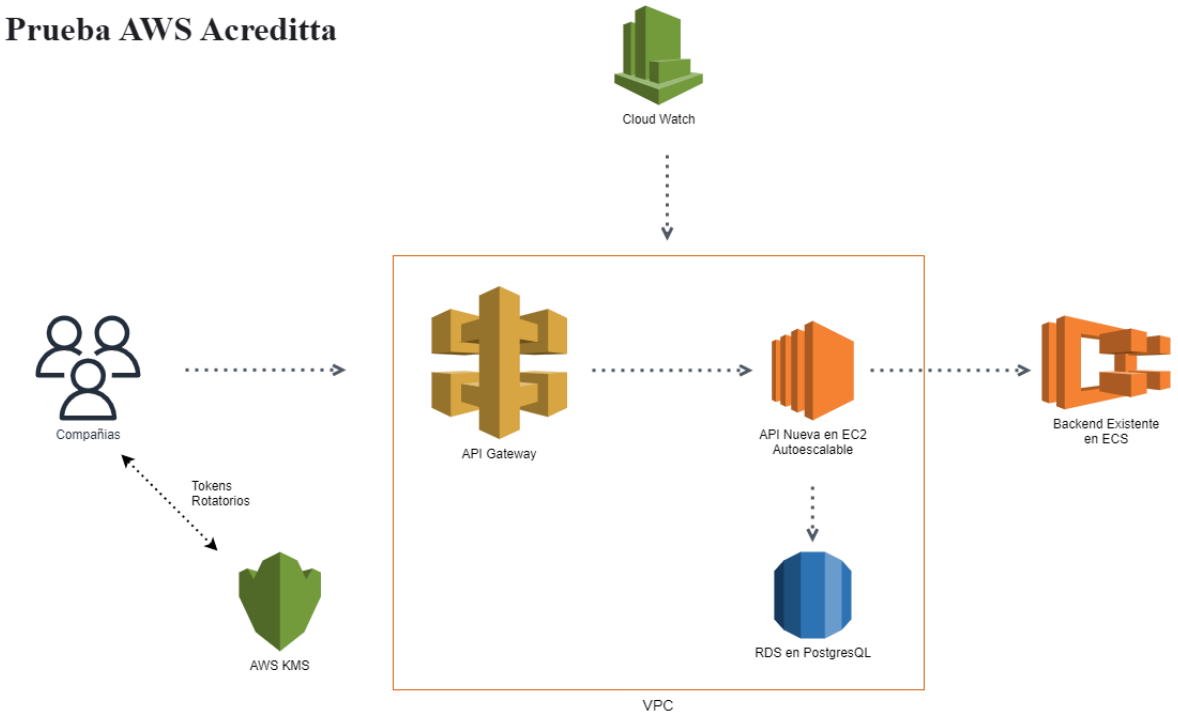
Estos 2 requerimientos juntos nos llevan a sugerir la utilización de un servicio como lo es **AWS CloudWatch** el cual nos permite capturar errores, monitorear y observar métricas constantes sobre los recursos de AWS. Esta solución se podría complementar mediante el uso de alguna herramienta del framework utilizado en Backend, como lo es en el caso de Django Rest Framework https://www.django-rest-framework.org/api-guide/throttling/ - el cual brinda la posibilidad de administrar politicas **ThrottleCount** y así poder administrar los request que pueden ser ejecutados por los usuarios de la API.

4.

> endpoints del backend contienen información tanto en el request como en el response que no son importante para la integración así que también se quiere abstraer esa transformación de las peticiones.

Y finalmente la necesidad de transformar los datos de las peticiones (request) nos lleva a sugerir un **API Gateway**, el cual pensando en una posterior integración con otras API'S y una mayor escalabilidad, es sugerible realizarlo utilizando la tecnología **GraphQL** ya que da la libertad a los consumidores de formatear las respuestas de acuerdo a sus necesidades.

Un posible diagrama de la arquitectura y flujo de la integración sería la siguiente (realizado mediante *https://draw.io/*).



La API se podría alojar en un servicio de EC2 con unas reglas de autoscaling previamente definidas, para lo cual se podría replicar en una AZ cercana a la región de consumo del cliente, la base de datos se sugiere con un motor de PostgreSQL debido a que es la que brinda mayor compatibilidad con el backend en Django, aunque cualquier otro sistema relacional también sería válido. Adicionalmente se sugiere el montaje de un servicio ya sea On premise o el plan pago de Sentry el cual es una herramienta que permite realizar monitoreo sobre los errores ocurridos en la API, generando notificaciones y alertas a los administradores del sistema.