efRouting

Prueba Técnica: Desarrollador(a) Fullstack

A continuación, se describe una prueba técnica que consiste en construir un sistema que obtenga información de lanzamientos espaciales desde la API pública de SpaceX, procese estos datos mediante una función AWS Lambda escrita en Python, los almacene en Amazon DynamoDB y los muestre a través de una aplicación web desplegada usando Amazon ECS Fargate. Todo este flujo debe estar orquestado y automatizado mediante un pipeline de integración y despliegue continuo (CI/CD) utilizando GitHub Actions.

1. Creación de la base de datos en Amazon DynamoDB

Se requiere la creación de una tabla en Amazon DynamoDB para almacenar la información relevante de los lanzamientos espaciales obtenidos desde la API de SpaceX. Esta tabla deberá tener una clave primaria única y contener campos como el nombre de la misión (mission_name), el nombre del cohete (rocket_name), la fecha de lanzamiento (launch_date) y el estado del lanzamiento (status: success, failed, upcoming). También se recomienda incluir otros atributos relevantes como el nombre de la plataforma de lanzamiento, payloads asociados u otros campos que consideres útiles para su posterior visualización.

Se valorará especialmente si la creación de recursos de infraestructura se realiza mediante infraestructura como código, utilizando herramientas como AWS CDK, CloudFormation o Terraform. La documentación debe incluir instrucciones detalladas para desplegar el entorno de desarrollo desde cero. Se recomienda que, para recursos de aplicación **serverless**, como funciones Lambda, API Gateway o DynamoDB, se utilicen herramientas especializadas como Serverless Framework o AWS SAM (Serverless Application Model), que simplifican la definición, prueba y despliegue de este tipo de servicios**.**

2. Función Lambda en Python para consumir la API de SpaceX

Deberás implementar una función Lambda que se ejecute automáticamente cada seis horas y consuma los endpoints de la API pública de SpaceX (por ejemplo, https://api.spacexdata.com/v4/launches o /upcoming). Esta función debe extraer los datos relevantes de cada lanzamiento, como el nombre de la misión, la fecha, el cohete utilizado, el estado del lanzamiento, entre otros, e insertar o actualizar los registros correspondientes en DynamoDB aplicando una lógica de *upsert*.

Además, la función debe permitir una invocación manual para propósitos de prueba (un endpoint que podamos ejecutar para probar su ejecución), en la que retorne un resumen de los datos procesados, como la cantidad de registros actualizados o una lista de los lanzamientos insertados.

Es obligatorio incluir pruebas unitarias que cubran la lógica de parsing, inserción y manejo de errores. Se considerará un valor agregado definir esta Lambda (incluyendo sus permisos y configuración) mediante infraestructura como código, utilizando herramientas como AWS CDK, CloudFormation o Terraform.

3. Aplicación web con despliegue en Amazon ECS Fargate

Se espera que desarrolles una aplicación web moderna y responsiva que consuma los datos desde DynamoDB (de forma directa o mediante una API intermedia) y los muestre en un formato claro y visualmente atractivo. La interfaz debe permitir explorar los lanzamientos mediante filtros, tablas, líneas de tiempo o gráficos de tendencia.

El uso de bibliotecas como React/Angular para la interfaz, y herramientas como Chart.js, D3.js o Plotly para la visualización de datos, será considerado un plus, especialmente si el postulante tiene un enfoque más orientado al front-end.

La aplicación debe contenerizarse con Docker y desplegarse mediante Amazon ECS con Fargate. Se espera que detalles claramente cómo construir la imagen, cómo subirla a Amazon ECR (u otro repositorio de contenedores) y cómo configurar la tarea y el servicio en ECS para que la aplicación quede accesible públicamente.

4. Pipeline CI/CD con GitHub Actions

Toda la solución debe estar integrada con un flujo de CI/CD automatizado mediante GitHub Actions. El pipeline debe activarse con cada *push* a la rama principal y contemplar los siguientes pasos mínimos:

- Instalación de dependencias y ejecución de pruebas automatizadas (unitarias y/o de integración).
- Construcción de la imagen Docker de la aplicación web.
- Publicación de la imagen en Amazon ECR.
- Despliegue de la aplicación en ECS Fargate, con actualización del servicio existente.
- Despliegue de la lambda y recursos asociados
- Manejo de errores de despliegue y rollback (opcional, pero muy valorado).

Entregables

1. Repositorio en GitHub con:

- Código fuente de la Lambda y la aplicación web.
- Dockerfile completo y funcional.
- Workflows de GitHub Actions.
- Archivos de infraestructura como código.
- Pruebas automatizadas.

- Implementación de Swagger

2. Documentación clara y detallada que explique:

- Cómo desplegar la infraestructura desde cero.
- Cómo correr las pruebas y verificar el funcionamiento localmente.
- Cómo funciona y cómo extender el pipeline CI/CD.
- Cómo interactúan los componentes entre sí.

- Evidencias del proceso llevado

3. Entrega y plazo:

- Desde el momento en que recibas esta prueba, tendrás un plazo de 72 horas para completarla y enviarla.
- La solución debe ser entregada mediante un repositorio en GitHub. Asegúrate de que el repositorio sea público o que nos hayas dado permisos de acceso si es privado.
- **Recuerda incluir en la documentación la URL pública de la aplicación desplegada y URL del swagger, para que podamos acceder y evaluarla correctamente.**

Criterios de evaluación

- Uso correcto de los servicios de AWS involucrados: DynamoDB, Lambda, ECS, IAM, ECR, etc.
- Calidad del código en Python y en el front-end (si aplica).
- Enfoque en pruebas automatizadas: cobertura, claridad y utilidad de las pruebas.
- Implementación correcta del pipeline CI/CD.
- Documentación y claridad en la entrega
- Despliegue continuo de la lambda y sus recursos asociados
- Despliegue continuo del ECS y sus recursos asociados
- Evidencias del proceso llevado en la documentación
- Presentación visual y funcionalidad de la aplicación web.
- Se considerará un plus la incorporación de elementos como observabilidad, separación de entornos, y uso avanzado de herramientas de infraestructura como código.