Práctica 1: API Rest

Universidad de Las Palmas de Gran Canaria
Gorka Eymard Santana Cabrera
Computación en La Nube
5 - 11 - 2025

1. Introducción

El objetivo de la practica es diseñar e implementar dos arquitecturas distintas en Amazon Web Services que proporcionen un servicio de API Rest.

- Arquitectura Acoplada: esta arquitectura estará basada en el uso de ECS FARGATE de modo que es una arquitectura orientada a servidor.
- Arquitectura Desacoplada: está otra arquitectura por contraparte será una arquitectura serverless que se basará en el uso de AWS Lambda, donde cada Lambda se corresponde con un endpoint del CRUD

2. Diseño arquitecturas

2.1. Acoplada

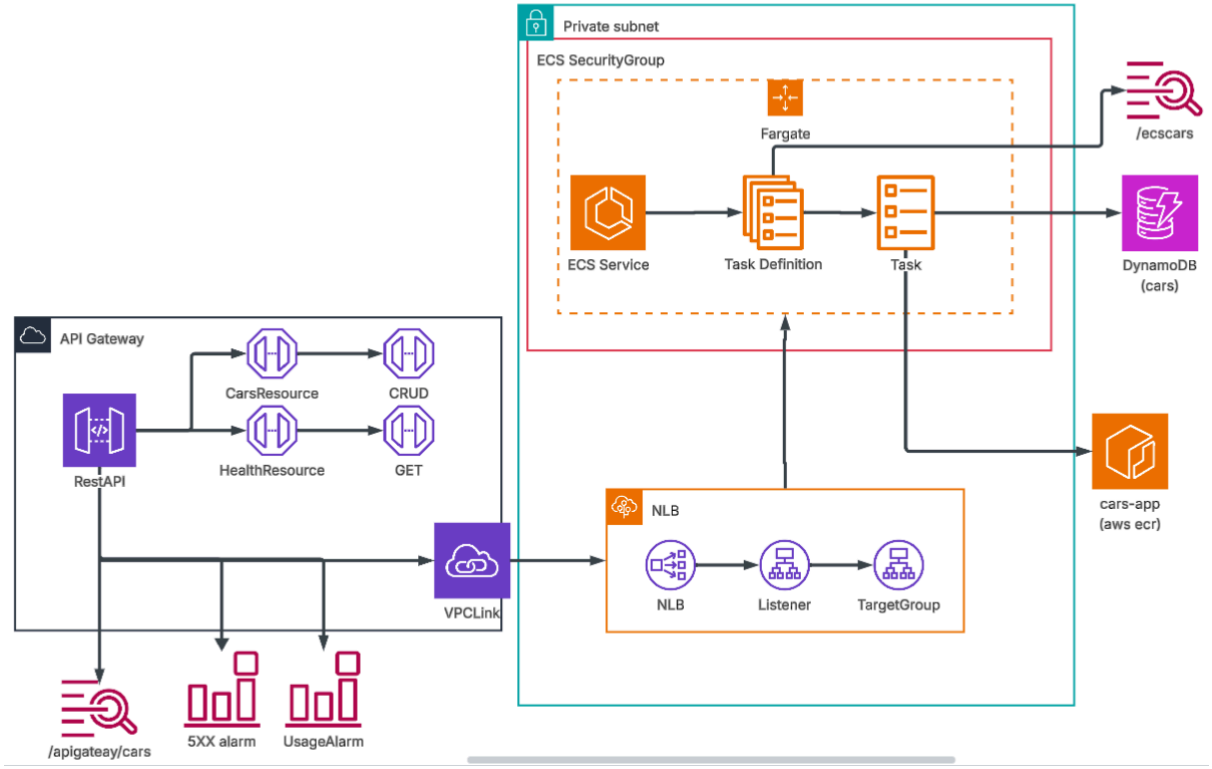

Figura 1: Esquema arquitectura acoplada

En esta primera implementación se han definido una serie de recursos que despliegan una arquitectura basada en **"fargate"** en un grupo de seguridad que solo permite tráfico de ingreso desde el VPC Link:

- **NLB:** Network Load Balancer de tipo interno, en las subnets proporcionadas. Expone el servicio internamente.
- **TargetGroup:** grupo objetivo en puerto 8080, tipo target = ip (para tareas Fargate), health check en /health (HTTP).
- **Listener:** escucha en el NLB puerto 8080 y reenvía al TargetGroup.

- **VPCLink:** vínculo usado por API Gateway para enrutar tráfico hacia el NLB dentro de la VPC (TargetArns = NLB).
- **ECSSecurityGroup:** SG para las tareas ECS que permite entrada TCP 8080 desde el CIDR de la VPC (regla ingress).
- **ECSCluster:** clúster ECS (nombre cars-cluster).
- **TaskDefinition:** definición de tarea Fargate:
    - networkMode awsvpc, CPU 256, MEM 512.
    - ExecutionRoleArn / TaskRoleArn apuntan a rol LabRole en la cuenta.
    - Contenedor cars-container: imagen construida desde ECR (cuenta+región+ImageName), expone puerto 8080, configura awslogs (/ecs/cars) y establece variable de entorno DB_DYNAMONAME.
- **ECSService:** servicio Fargate, DesiredCount 1, usa awsvpc con AssignPublicIp ENABLED, subnets y SG definidos; registra el contenedor en el TargetGroup del NLB. Dependencia en Listener.
- **RestAPI :** definición del API REST (cars-api).
- **CarsResource**: recurso /cars en la API.
- **PostCarsMethod**: POST /cars
    - AuthorizationType NONE, ApiKeyRequired true.
    - Integración HTTP_PROXY vía VPC_LINK al NLB: http://${NLB.DNSName}:8080/cars (mapea petición directamente al servicio).
- **GetCarsMethod**: GET /cars — misma configuración que POST (ApiKeyRequired true).
- **OptionsCarsMethod**: OPTIONS /cars
    - Tipo MOCK que responde con cabeceras CORS (Allow-Headers, Allow-Methods, Allow-Origin). Método usado para CORS preflight.
- **CarResource**: recurso hijo /cars/{id} (path parameter).
- **GetCarMethod**: GET /cars/{id}
    - ApiKeyRequired true; RequestParameters habilita method.request.path.id; integración HTTP_PROXY a /cars/{id} y mapea path parameter al integration.request.
- **PutCarMethod**: PUT /cars/{id} — similar a GET pero con método PUT.
- **DeleteCarMethod**: DELETE /cars/{id} — similar a PUT/GET.
- **OptionsCarMethod**: OPTIONS /cars/{id} — similar a /cars
- **HealthResource**: recurso /health.
- **HealthGetMethod**: GET /health
- **OptionsHealthMethod**: OPTIONS /health — similar a /cars
- **UsagePlan**: plan de uso cars-usage-plan aplicado al stage prod con límites (RateLimit 10 rps, Burst 200).

- **Api5xxAlarm**: alarma que vigila 5XXError en API Gateway para la API y stage prod; umbral >=1 en período 5 min.
- **ApiHighUsageAlarm**: alarma sobre Count de peticiones (umbral >=1000 en 5 min) para detectar picos.
- **APIGatewayLogGroup**: grupo de logs /aws/apigateway/cars con retención 7 días (usado por APIStage).
- **ApiGatewayAccount**: configura CloudWatchRoleArn (rol LabRole) para permitir a API Gateway escribir logs en CloudWatch.

2.2. Desacoplada

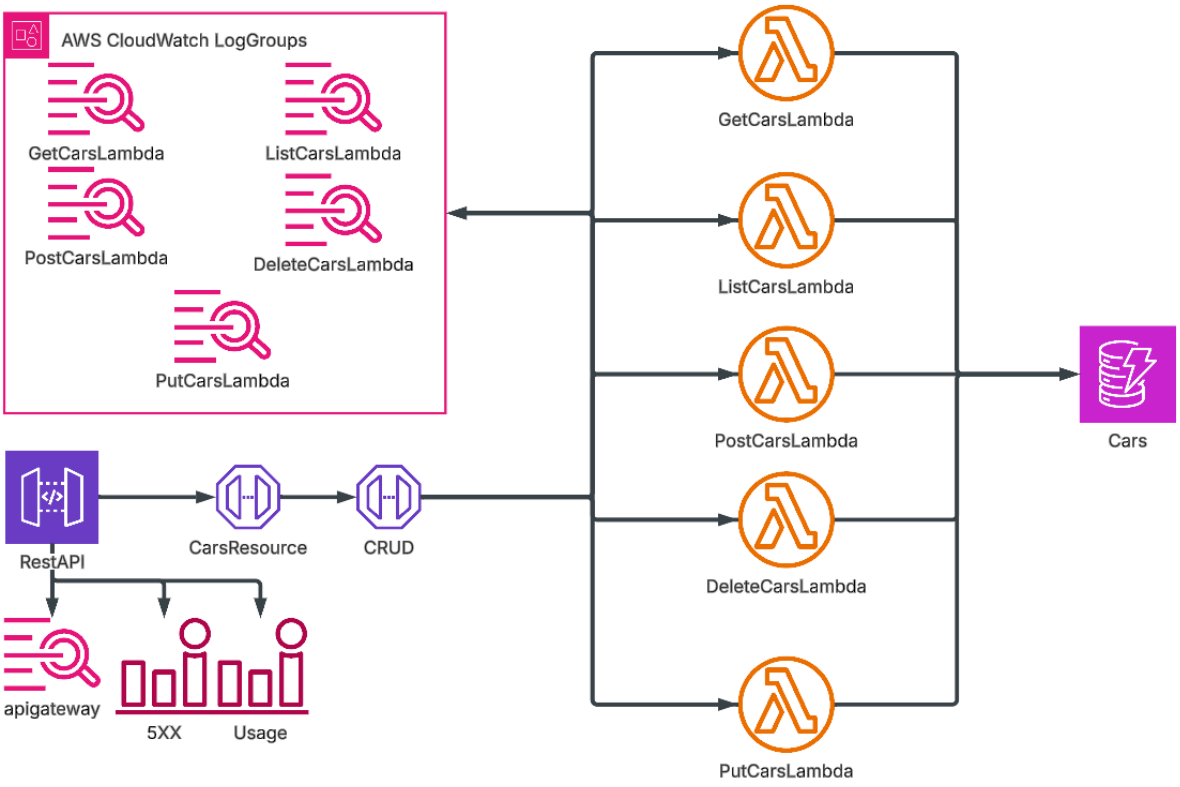

Figura 2: Esquema desacoplada

Esta implementación consiste en una arquitectura AWS Serverless en la que se definen

- **GetCarsLambda / ListCarsLambda / PostCarsLambda / PutCarsLambda / DeleteCarsLambda:**
  - Funciones empaquetadas como imágenes ECR (PackageType: Image) apuntando a cars-service:latest en la cuenta/región.
  - Role: rol IAM hardcodeado LabRole.
  - Timeout 30s, Memory 128, arquitectura x86_64.
  - Variable de entorno HANDLER_NAME para distinguir el handler (get, list, create, update, delete).

- **GetCarsLambdaLogGroup, ListCarsLambdaLogGroup, PostCarsLambdaLogGroup, DeleteCarsLambdaLogGroup, PutCarsLambdaLogGroup:** Grupos de logs /aws/lambda/${Lambda} con retención 7 días.
- **CarsRestAPI:** API regional llamada CarsAPI.
- **CarsApiKey** API Key habilitada y asociada al stage prod (StageKeys).
- **CarsUsagePlan:** Plan de uso que referencia la API/stage prod; throttling RateLimit 10, BurstLimit 200.
- **CarsUsagePlanKey :** Asocia la API Key al Usage Plan.
- **CarsResource:** Recurso /cars.
- **CarsIdResource :** Recurso /cars/{id} (path parameter).
- **ListCarsLambdaPermission, GetCarsLambdaPermission, PostCarsLambdaPermission, PutCarsLambdaPermission, DeleteCarsLambdaPermission:** Permiten a apigateway.amazonaws.com invocar cada Lambda; SourceArn restringe invocación a la API creada.
- **OptionsCarsMockMethod:** OPTIONS /cars como MOCK que responde cabeceras CORS (Allow-Headers, Allow-Methods, Allow-Origin).
- **ListCarsMethod:** GET /cars con ApiKeyRequired: true y integración AWS_PROXY hacia ListCarsLambda (URI: lambda invocations).
- **PostCarsMethod:** POST /cars -> PostCarsLambda (AWS_PROXY), ApiKeyRequired: true.
- **OptionsCarIdMockMethod:**OPTIONS /cars/{id} MOCK con cabeceras CORS.
- **GetCarMethod:** GET /cars/{id} -> GetCarsLambda (AWS_PROXY); RequestParameters habilita method.request.path.id.
- **PutCarMethod:** PUT /cars/{id} -> PutCarsLambda (AWS_PROXY); path id requerido.
- **DeleteCarMethod:** DELETE /cars/{id} -> DeleteCarsLambda (AWS_PROXY); path id requerido.
- **UsagePlan**: plan de uso cars-usage-plan aplicado al stage prod con límites (RateLimit 10 rps, Burst 200).
- **Api5xxAlarm**: alarma que vigila 5XXError en API Gateway para la API y stage prod; umbral >=1 en período 5 min.
- **ApiHighUsageAlarm**: alarma sobre Count de peticiones (umbral >=1000 en 5 min) para detectar picos.
- **APIGatewayLogGroup**: grupo de logs /aws/apigateway/cars con retención 7 días (usado por APIStage).
- **ApiGatewayAccount**: configura CloudWatchRoleArn (rol LabRole) para permitir a API Gateway escribir logs en CloudWatch.

2.3. Documentación

- **APIDocumentation, CarsResourceDocumentation, GetCarsMethodDocumentation, PostCarsMethodDocumentation, CarResourceDocumentation, GetCarMethodDocumentation, PutCarMethodDocumentation, DeleteCarMethodDocumentation, OptionsCarsMockMethodDocumentation, OptionsCarIdMockMethodDocumentation:** Documentación estructurada (resúmenes, parámetros, esquemas, request/response) para API, recursos y métodos.
- **APIDocumentationVersion:** Versiona la documentación (1.0.0), depende de las partes de doc.

3. Seguridad

3.1. Arquitectura Acoplada

El servidor está encapsulado en un servicio AWS Fargate desplegado dentro de un ECS Cluster, protegido mediante múltiples capas de seguridad:

- El acceso externo se realiza exclusivamente a través de Amazon API Gateway (REST API), que actúa como frontend público y única puerta de entrada. El API Gateway implementa:
    - Autenticación mediante API Key: Todas las peticiones requieren el header `x-api-key` con una clave válida generada en AWS, limitando el acceso no autorizado.
    - Integración VPC Link: API Gateway se conecta al servicio Fargate a través de un Network Load Balancer (NLB) interno mediante AWS Private Link, sin exponer el contenedor directamente a Internet.
    - Throttling (Limitación de Peticiones): Se ha configurado una política de throttling con los siguientes límites:
        - Rate Limit: 10 peticiones por segundo de forma sostenida
        - Burst Limit: 200 peticiones para absorber picos temporales de tráfico

- El ECS Service ejecuta tareas Fargate en subnets privadas con las siguientes características de seguridad(**Security Group**):
    - Tráfico entrante: Únicamente permite conexiones desde el NLB interno (puerto 8080/TCP), restringiendo cualquier acceso directo externo.
    - Tráfico saliente: Permite salida controlada para acceder a servicios de AWS (ECR, DynamoDB, CloudWatch).

- Amazon DynamoDB y Amazon ECR: el acceso se realiza mediante IAM Role (LabRole) asignado a la tarea Fargate con permisos `dynamodb:*`.

Además se han configurado 2 alarmas y se han añadido LogGroups para monitorizar el uso de la aplicación.

3.2. Arquitectura Desacoplada

Dado que en esta arquitectura se cuenta con un diseño serverless basado en AWS Lambda, se ha implementado un control de acceso y limitación de peticiones a través de Amazon API Gateway:

- El acceso externo se realiza exclusivamente a través de Amazon API Gateway (REST API), que actúa como punto de entrada único y público. El API Gateway implementa los siguientes mecanismos de seguridad:

- Autenticación mediante API Key: Todas las peticiones requieren el header `x-api-key` con una clave válida generada en AWS, limitando el acceso no autorizado.
- Throttling (Limitación de Peticiones): Se ha configurado una política de throttling con los siguientes límites:
  - Rate Limit: 10 peticiones por segundo de forma sostenida
  - Burst Limit: 200 peticiones para absorber picos temporales de tráfico

Estos límites se aplican tanto a nivel de Stage (CarsAPIStage) como en el Usage Plan (CarsUsagePlan), protegiendo las funciones Lambda de sobrecargas y garantizando un uso controlado de los recursos.

4. Despliegue

Para realizar el despliegue de ambas arquitecturas primeramente se ha de crear las pilas de:

- DynamoDB
- ECR para arquitectura acoplada
- ECR para arquitectura desacoplada

Estas pilas se crean usando las plantillas definidas como **“Ecr-attached.yml”** **“Ecr-detached.yml”** y **“DynamoDB.yml”**.

A continuación usando el script de apoyo “Build-and-push.ps1” se suben las imágenes a sus respectivos ECR.

Por último se crean las últimas pilas:

- Cars-app
- Cars-lambdas

Para estas pilas son las plantillas:

- **“main-attached.yml”**: que tiene como parámetros el VPC, la lista de subnets a usar, el nombre de la DynamoDB, VPCCidr que es la dirección IP del VPC y el nombre de la imagen.
- **“main-lambda.yml”**: para la que no hace falta parámetros dado que ya tiene definidos el nombre de la DynamoDB y el nombre del ECR donde se encuentra la imágen.

5. Pruebas

5.1. POSTMAN

Se han creado las siguientes pruebas para validar cada endpoint:

- GET /cars:
    - Método: GET
    - URL: {{base_url}}/cars
    - Headers:
        - Content-Type: applicaciton/json
        - x-api-key: {{api_key}}
    - Respuesta esperada: 200 OK
    - Validación: Verificar que se devuelve lista
- POST /cars:
    - Método: POST
    - URL: {{base_url}}/cars
    - Headers:
        - Content-Type: applicaciton/json
        - x-api-key: {{api_key}}
        - body(JSON)
            - {"plate": "1234ABC", ”make": "Toyota", "model": "Corolla", "year": 2023,"owner": "John Doe" }
    - Respuesta esperada: 200 OK
    - Validación: Verificar que se devuelve lista

5.2. Interfaz Web

Uso de interfaz web que cuenta con las siguiente funcionalidades:

- **Formulario:** Añadir coches
- **Tabla**: lista todos los coches
- **Búsqueda:** Filtrar por matrícula.
- **Consume la API REST:** internamente, usando fetch/axios con API Key)

Esto proporciona unas pruebas rápidas y una demostración visual del CRUD.

6. Costes

6.1. Acoplada

Recurso	Configuración	Precio Unitario	Uso Mensual	Precio Mensual
API Gateway	REST API	$3.50 / millón requests	500,000 requests	$1.75
ECS Fargate	0.5 vCPU	$0.04048 / vCPU-hora	0.25 / 730 horas	$7.39
	1 GB RAM	$0.004445 / GB-hora	0.5 / 730 horas	$1.62
	Ephemeral Storage	$0.000111 /GB-Hora	20 / 730 horas	$~0.00
	Subtotal Fargate			$9.01
Network Load Balancer	NLB + processed bytes	$0.0225 / hora + $0.006 / GB	730 horas + 0.1 GB	$16.44
VPC Link	API Gateway ↔ NLB	$0.02 / hora	730 horas	$14.60
DynamoDB	Storage	$0.25 / GB-mes	0.5 GB - 1KB/petición	$0.28
ECR	Storage	$0.10 / GB-mes	61.97 MB	$0.01
CloudWatch	Log Ingestion	$0.50 / GB	0.1 GB	$0.1
	Standard Alarms	$ 0.10	2	$0.20
	Subtotal CloudWatch			$0.30
Subtotal Mensual				$42.39
Subtotal Anual				$508.68

6.2. Desacoplada

Recurso	Configuración	Precio Unitario	Uso Mensual	Precio Mensual
API Gateway	REST API	$3.50 / millón requests	500,000 requests	$1.75
Lambdas Functions				
list-cars-lambda	128 MB, 150ms	$0.0000166667 / GB-seg	150k×0.15s×0.25GB	$0.08
get-cars-lambda	128 MB, 500ms	$0.0000166667 / GB-seg	150k×0.5s×0.25GB	$0.19
post-cars-lambda	128 MB, 500ms	$0.0000166667 / GB-seg	100k×0.5s×0.25GB	$0.12
put-cars-lambda	128 MB, 150ms	$0.0000166667 / GB-seg	75k×0.15s×0.25GB	$0.03
delete-cars-lambda	128 MB, 200ms	$0.0000166667 / GB-seg	25k×0.2s×0.25GB	$0.02
	Subtotal lambdas:			$0.44
DynamoDB	Storage	$0.25 / GB-mes	0.5 GB - 1KB/petición	$0.28
ECR	Storage	$0.10 / GB-mes	122.31 MB	$0.01
CloudWatch	Log Ingestion	$0.50 / GB	0.5 GB	$0.25
	Standard Alarms	Gratis (primeras 10)	2 alarmas	$0.20
	Subtotal CloudWatch			$0.45
Subtotal Mensual				$2.93
Subtotal Anual				$35.16

7. Conclusión

Finalmente se puede destacar:

- Ambas arquitecturas exponen la misma API con API Gateway y observabilidad por CloudWatch; Lambda es preferible para cargas esporádicas o con picos por su modelo de pago por invocación y escalado rápido, mientras que Fargate (con NLB y VPC Link) es mejor para cargas sostenidas, control de red y dependencias de sistema que requieran un runtime más estable.
- En seguridad, Fargate ofrece aislamiento en VPC, control con Security Groups y ruta interna mediante VPC Link; Lambda reduce la superficie de gestión al ser serverless pero requiere cuidado con permisos IAM y ENIs si se conecta a VPC. En ambos casos hay riesgo por roles hardcodeados y se recomienda aplicar least privilege, usar Secrets Manager/KMS, WAF/Shield y cifrado en tránsito y en reposo.
- En costes y escalabilidad, Lambda es más económico para baja frecuencia y picos, pero puede ser costoso en ejecuciones largas o muy frecuentes y sufre cold starts; Fargate tiene coste por vCPU/mem y costes fijos del NLB pero suele ser más eficiente para tráfico constante y largo. API Gateway escala en ambos diseños; la elección debe basarse en el patrón de uso esperado y en requisitos de latencia, red y operación.

8. Enlaces externos

Repositorio Github: https://github.com/gorkaftv1/Practica-1-entregable-CN