### **Arquitectura General**

### La solución estará basada en microservicios serverless utilizando AWS Lambda y otros servicios de AWS para proporcionar escalabilidad, alta disponibilidad y costos optimizados.

### 

### **Componentes Principales**

### **Cliente (Frontend)**

### Aplicaciones web y móviles consumen las APIs.

### Se entrega mediante Amazon S3 y CloudFront para contenido estático.

### **API Gateway**

### Actúa como punto de entrada para los clientes.

### Maneja autenticación, autorización y rate limiting.

### Conectado a AWS Lambda para ejecutar lógica de negocio.

### **AWS Lambda (Backend en Java)**

### Cada microservicio se implementa como una función Lambda en Java con Spring Boot o Quarkus.

### Se ejecuta solo cuando es necesario, optimizando costos.

### Maneja lógica de negocio específica de cada servicio.

### **DynamoDB (Base de Datos NoSQL)**

### Se usa como almacenamiento para datos estructurados y semiestructurados.

### Integración con Lambda mediante SDK de AWS.

### Modelado basado en acceso para rendimiento óptimo.

### **CloudWatch**

### Monitoreo y logging de Lambda y otros servicios.

### Alarmas y métricas de rendimiento.

### **S3 (Almacenamiento de Archivos)**

### Usado para almacenamiento de imágenes, documentos y otros archivos estáticos.

### Integrado con CloudFront para distribución rápida de contenido.

### **CloudFront**

### Red de distribución de contenido (CDN) para reducir latencia.

### Entrega archivos de S3 y respuestas cacheadas del backend.

### 

### **📌 ¿Qué es un servicio Serverless?**

Un servicio **serverless** (sin servidor) es un modelo de computación en la nube donde el proveedor de la nube **administra automáticamente la infraestructura**, escalando recursos según la demanda y cobrando solo por el tiempo de ejecución.

➡️ **No significa que no hay servidores**, sino que **no necesitas gestionarlos**. AWS, Azure o Google Cloud se encargan de ello.

## **🚀 Características del Serverless**

✅ **Sin gestión de servidores** → No necesitas configurar, escalar ni administrar servidores.  
 ✅ **Escalabilidad automática** → La plataforma ajusta la capacidad según la demanda.  
 ✅ **Pago por uso** → Solo pagas por el tiempo de ejecución del código, no por servidores inactivos.  
 ✅ **Ejecución basada en eventos** → Se activa en respuesta a eventos (HTTP, base de datos, archivos, etc.).

## **⚡ Ejemplos de Servicios Serverless en AWS**

### **🔹 AWS Lambda → Ejecuta código en respuesta a eventos sin servidores dedicados.**

### **🔹 Amazon API Gateway → Crea APIs REST y WebSockets sin gestionar servidores.**

### **🔹 Amazon DynamoDB → Base de datos NoSQL escalable sin administración de instancias.**

### **🔹 Amazon S3 → Almacenamiento de archivos sin necesidad de administrar discos o servidores.**

## 

## 

## **🎯 Diferencia entre Serverless y Microservicios**

| **Característica** | **Serverless** | **Microservicios (con servidores)** |
| --- | --- | --- |
| Gestión de infraestructura | Automática por el proveedor | Necesitas administrar servidores |
| Escalabilidad | Automática y bajo demanda | Manual o con configuración específica |
| Pago | Solo por uso real | Puede haber costos fijos |
| Latencia | Puede tener un pequeño retraso inicial (cold start) | Tiempo de re |

## **Lambdas en AWS: Explicación Completa 🚀**

### **1. ¿Qué es AWS Lambda?**

AWS Lambda es un servicio **serverless** que permite ejecutar código sin necesidad de administrar servidores. Se ejecuta **bajo demanda** en respuesta a eventos como solicitudes HTTP, cambios en bases de datos o cargas de archivos a S3.

### **2. Características Principales**

✅ **Escalabilidad automática**: Se ejecuta cuando es necesario y escala sin intervención manual.  
 ✅ **Pago por uso**: Solo pagas por el tiempo de ejecución del código.  
 ✅ **Integración con otros servicios de AWS**: DynamoDB, API Gateway, S3, SNS, SQS, etc.  
 ✅ **Soporte para múltiples lenguajes**: Java, Python, Node.js, Go, C#, etc.

## **3. ¿Cómo funciona AWS Lambda?**

### **🔹 1. Evento desencadenador**

AWS Lambda se activa en respuesta a eventos como:

* Una **solicitud HTTP** a **API Gateway**.
* Un nuevo archivo subido a **S3**.
* Un cambio en una tabla **DynamoDB**.
* Un mensaje recibido en **SQS** o **SNS**.

### **🔹 2. AWS Lambda ejecuta el código**

Lambda **carga el código**, lo ejecuta y devuelve un resultado.

### **🔹 3. Devuelve la respuesta**

Si se usó **API Gateway**, responde con un JSON al cliente.  
 Si se usó **DynamoDB Streams**, puede activar otra Lambda.

## **4. Arquitectura de AWS Lambda**

📌 **Ejemplo: API REST con AWS Lambda + API Gateway + DynamoDB**

1️⃣ Un cliente realiza una petición **HTTP** a API Gateway.  
 2️⃣ **API Gateway** invoca una función AWS Lambda.  
 3️⃣ **AWS Lambda** obtiene datos de **DynamoDB** y los procesa.  
 4️⃣ **Lambda** responde con un JSON a API Gateway.  
 5️⃣ **API Gateway** devuelve la respuesta al cliente.

## **📌 ¿Qué es API Gateway?**

**Amazon API Gateway** es un servicio completamente administrado de AWS que te permite **crear, publicar, mantener, asegurar y monitorear APIs** a gran escala. Sirve como un **punto de entrada** para aplicaciones que interactúan con servicios backend, como AWS Lambda, EC2, DynamoDB, o incluso APIs externas.

➡️ **Esencial para arquitecturas serverless y microservicios**, ya que gestiona el tráfico, la seguridad y la escalabilidad de las APIs.

## **🚀 Características Clave de API Gateway**

✅ **Crea APIs REST, HTTP y WebSockets** → Compatible con diferentes tipos de APIs.  
 ✅ **Integración con AWS Lambda, EC2, DynamoDB, S3, SNS, SQS** → Se conecta fácilmente con otros servicios AWS.  
 ✅ **Gestión de tráfico** → Maneja **cientos de miles de solicitudes** por segundo.  
 ✅ **Autenticación y Autorización** → Soporte para **JWT, Cognito, IAM y API Keys**.  
 ✅ **Rate Limiting y Throttling** → Controla el número de peticiones por segundo.  
 ✅ **Monitoreo con AWS CloudWatch** → Registra logs y métricas de uso.  
 ✅ **Caching** → Reduce la carga de los backends almacenando respuestas en caché.

## **🛠️ Tipos de API en API Gateway**

🔹 **REST API** → Permite crear APIs RESTful con validaciones, autenticación y escalabilidad.  
 🔹 **HTTP API** → Más liviana y rápida que REST API, ideal para integraciones directas con AWS Lambda.  
 🔹 **WebSocket API** → Permite conexiones en tiempo real, útil para **chats, juegos, notificaciones**.

## **⚡ Arquitectura de API Gateway con AWS Lambda**

1️⃣ Un **cliente** (web/móvil) envía una solicitud HTTP a **API Gateway**.  
 2️⃣ API Gateway **válida la autenticación** y aplica reglas de seguridad.  
 3️⃣ API Gateway **invoca AWS Lambda** o cualquier otro backend.  
 4️⃣ **AWS Lambda** procesa la solicitud y obtiene datos de DynamoDB u otro servicio.  
 5️⃣ Lambda **devuelve la respuesta** a API Gateway, que la envía al cliente.

➡️ **Beneficio**: Backend **100% serverless**, sin necesidad de gestionar servidores.

## **📌 Conclusión**

Amazon API Gateway es un componente clave para arquitecturas **serverless y microservicios**, permitiendo construir APIs seguras, escalables y sin servidores. Su integración con **AWS Lambda** lo convierte en una de las mejores opciones para exponer APIs REST y WebSockets en la nube.