AMAZON

# WEB SERVICES

## Guia Completa de Estudio

De cero a experto — Todos los servicios explicados

Computo | Redes | Storage | DevOps

2026

ZERIONIS Study Docs

# Contenido

# 1. Fundamentos de AWS

## 1.1. Que es AWS

Amazon Web Services es la plataforma de cloud computing mas grande del mundo. Ofrece **200+ servicios** que van desde servidores virtuales hasta inteligencia artificial. En lugar de comprar y mantener hardware fisico, rentas recursos por demanda y pagas solo lo que usas.

**El modelo mental**

Imagina que en lugar de comprar un servidor, una base de datos, un firewall, un CDN, y un sistema de backups… **rentas cada uno por hora/mes** a Amazon, y ellos se encargan de que funcione, este actualizado, y tenga redundancia. Eso es AWS.

## 1.2. Los 3 modelos de nube

| Modelo | Que significa | Ejemplo |
|---|---|---|
| **IaaS** | Infrastructure as a Service. Te dan la maquina, tu instalas todo. Maximo control. | EC2, VPC |
| **PaaS** | Platform as a Service. Te dan la plataforma lista, tu solo subes codigo. | Elastic Beanstalk, App Runner |
| **SaaS** | Software as a Service. Todo listo, solo lo usas. | WorkSpaces, Chime |

## 1.3. Regiones y Zonas de Disponibilidad

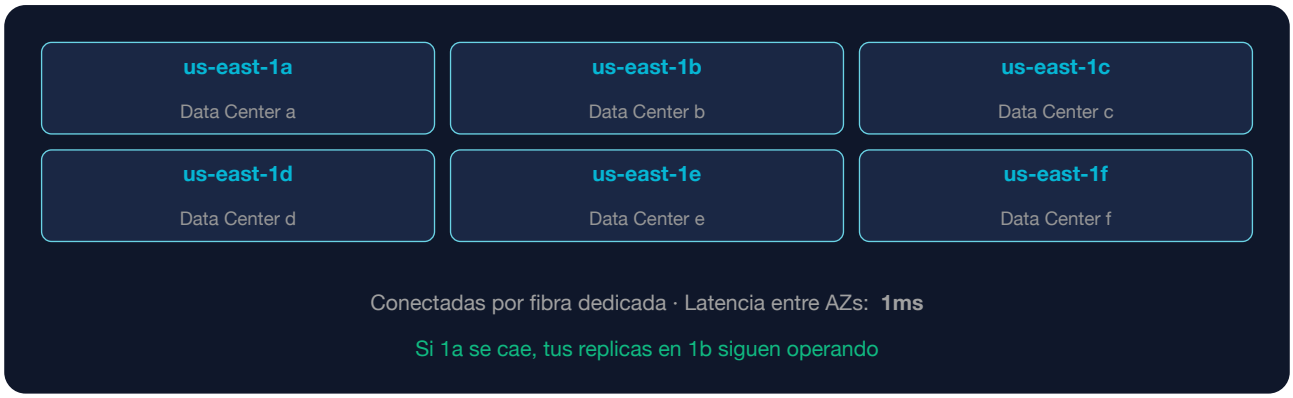
**Region**

Area geografica con multiples data centers. Ejemplo: `us-east-1` (Virginia), `eu-west-1` (Irlanda), `sa-east-1` (Sao Paulo). Cada region es **totalmente independiente** — los datos no se replican entre regiones automaticamente. Tu eliges donde viven tus recursos.

**Availability Zone (AZ)**

Cada region tiene 2-6 AZs. Cada AZ es un data center fisicamente separado (diferente edificio, diferente red electrica). Si un AZ se cae (incendio, inundacion), los otros siguen funcionando. Multi-AZ = alta disponibilidad.

**Region: us-east-1 (N. Virginia)**




## 1.4. El modelo de precios

**REGLA DE ORO DEL COSTO**

En AWS pagas por **uso**: horas de computo, GB almacenados, GB transferidos, requests procesados. Lo mas caro casi siempre es el **data transfer (egress)** — sacar datos de AWS hacia internet. El ingress (meter datos) generalmente es gratis.

| Modelo | Como funciona | Ideal para |
|---|---|---|
| **On-Demand** | Pagas por hora/segundo sin compromiso. Prendes, pagas; apagas, dejas de pagar. | Desarrollo, cargas impredecibles |
| **Reserved** | Compromiso 1-3 años. Descuento 30-72% vs on-demand. | Produccion estable |
| **Spot** | Capacidad sobrante de AWS con 60-90% descuento. AWS puede quitartela con 2 min de aviso. | Batch processing, CI/CD runners |
| **Free Tier** | 12 meses gratis de ciertos servicios (750h EC2 t2.micro, 5GB S3, etc). | Aprender y prototipar |

# 2. IAM: Identidad y Acceso

**Identity and Access Management**
Controla QUIEN puede hacer QUE en tu cuenta AWS. Es el servicio mas importante.

IAM es el **sistema nervioso de seguridad** de AWS. Cada accion en AWS (crear una instancia, leer un archivo de S3, apagar una base de datos) pasa por IAM para verificar si tienes permiso.

## 2.1. Los 4 conceptos fundamentales

### Users (Usuarios)

Una persona o aplicacion con credenciales propias. Tiene username + password (para consola) y/o access key + secret key (para CLI/API). **Nunca uses el root user para trabajo diario.**

### Groups (Grupos)

Coleccion de usuarios. Asignas permisos al grupo y todos los miembros los heredan. Ejemplo: grupo “Developers” puede leer S3 y hacer deploy a ECS. Grupo “Admins” puede todo.

### Roles

Identidad **temporal** que un servicio o usuario puede “asumir”. Una Lambda asume un rol para leer S3. Una EC2 asume un rol para escribir a CloudWatch. No tiene credenciales fijas — son temporales via STS.

### Policies (Politicas)

Documento JSON que define permisos. Dice: “Allow/Deny” + “Action” (s3:GetObject) + “Resource” (arn:aws:s3:::mi-bucket/*). Se adjuntan a users, groups o roles.

### 2.1.1. Ejemplo de Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::mi-bucket/*"
    },
    {
      "Effect": "Deny",
      "Action": "s3:DeleteBucket",
      "Resource": "*"
    }
  ]
}
```

Esta policy dice: “puedes leer y subir archivos a `mi-bucket`, pero **nunca** puedes borrar ningun bucket”. Deny siempre gana sobre Allow.

### 2.1.2. Best practices

**REGLAS SAGRADAS DE IAM**

1. Nunca uses el root user. Crea un IAM user con MFA.
2. Principio de minimo privilegio: da solo los permisos necesarios.
3. Usa Roles en lugar de access keys cuando sea posible.
4. Rota access keys regularmente (90 dias maximo).
5. Activa MFA (autenticacion multifactor) en todas las cuentas.
6. Usa AWS Organizations para multiples cuentas (dev, staging, prod).

# 3. VPC: Tu red privada en la nube

**Virtual Private Cloud**
Red virtual aislada donde viven todos tus recursos AWS. Tu defines las IPs, subnets, y reglas de trafico.

Una VPC es como tener tu propia red de oficina pero en la nube. Defines el rango de IPs, creas subredes, decides que puede hablar con que, y controlas todo el flujo de trafico.

## 3.1. Componentes de una VPC

### 3.1.1. CIDR Block

El rango de IPs privadas de tu VPC. Ejemplo: `10.0.0.0/16` te da 65,536 direcciones IP (10.0.0.0 a 10.0.255.255). Eliges el tamaño al crear la VPC y **no se puede cambiar despues**.

**NOTACION CIDR**

`/16` = 65,536 IPs. `/24` = 256 IPs. `/28` = 16 IPs. Entre menor el numero despues del `/`, mas IPs. Para VPC usa `/16`. Para subnets usa `/24` (256 IPs por subnet es suficiente para la mayoria de casos).

### 3.1.2. Subnets (Subredes)

Subdivisiones de tu VPC. Cada subnet **vive en una sola AZ**. Hay dos tipos:

**Public Subnet**

**Tiene ruta al Internet Gateway.** Los recursos aqui pueden recibir trafico de internet y enviar trafico a internet directamente.

**Aqui va:** ALB, NAT Gateway, bastion hosts, instancias con IP publica.

Ejemplo: `10.0.1.0/24` en us-east-1a

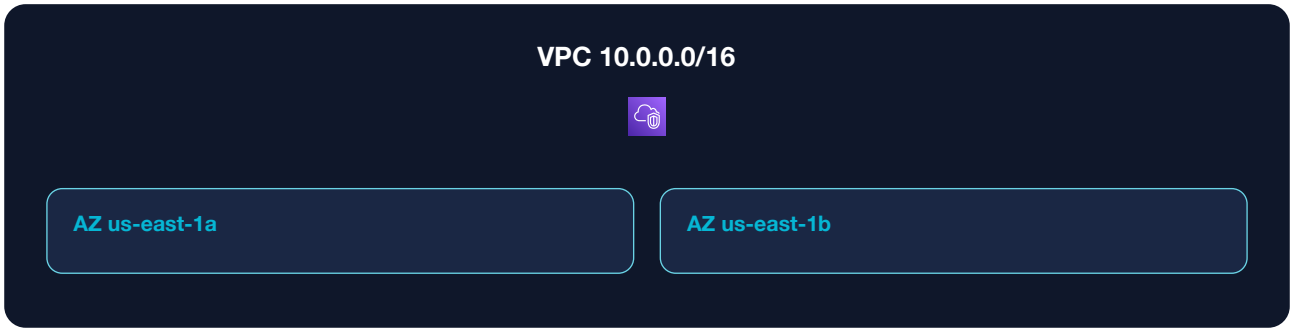
**Private Subnet**

**No tiene ruta directa a internet.** Los recursos aqui estan aislados. Para salir a internet (ej: descargar updates) necesitan un NAT Gateway.

**Aqui va:** EC2 de aplicacion, RDS, ElastiCache, ECS tasks, Lambda.

Ejemplo: `10.0.10.0/24` en us-east-1a

### 3.1.3. Diagrama de VPC tipica




### 3.1.4. Componentes de routing

| Componente | Que hace |
|---|---|
| **Internet Gateway (IGW)** | Puerta de entrada/salida a internet para la VPC. Lo conectas a la VPC y las public subnets rutean trafico a traves de el. |
| **NAT Gateway** | Permite que recursos en **private subnets** salgan a internet (para updates, llamadas API externas) sin ser accesibles desde internet. Trafico de salida si, entrada no. |
| **Route Table** | Tabla de rutas que cada subnet tiene. Define a donde va el trafico. Public subnet: `0.0.0.0/0 → IGW`. Private subnet: `0.0.0.0/0 → NAT Gateway`. |
| **VPC Peering** | Conexion entre dos VPCs para que sus recursos se comuniquen como si estuvieran en la misma red. Util para conectar VPC de dev con VPC de prod. |
| **VPC Endpoints** | Acceso a servicios AWS (S3, DynamoDB) desde private subnets **sin pasar por internet**. Trafico va por la red interna de AWS. Mas seguro y barato. |

### 3.1.5. Security Groups vs NACLs

**Security Groups**

Firewall a nivel de **instancia/recurso**. Stateful: si permites entrada, la respuesta sale automaticamente. Solo reglas **Allow** (no puedes denegar explicitamente).

Ejemplo: SG del ALB permite entrada 443. SG del ECS permite entrada 8080 **solo desde el SG del ALB**. SG

**NACLs (Network ACLs)**

Firewall a nivel de **subnet**. Stateless: debes permitir entrada Y salida explicitamente. Tiene reglas Allow **y** Deny con prioridad numerica.

del RDS permite entrada 5432 **solo desde el SG del ECS**.

**Es el firewall que mas usaras.**

Ejemplo: NACL de private subnet deniega todo trafico SSH (puerto 22) entrante como capa extra de protec-cion.

**Segunda linea de defensa. Usalo para bloquear rangos de IP.**

# 4. Computo: Donde corre tu codigo

## 4.1. EC2 (Elastic Compute Cloud)

**Elastic Compute Cloud**

Servidores virtuales en la nube. Tu eliges CPU, RAM, disco, OS. Control total.

EC2 es el servicio original de AWS. Es una maquina virtual que tu administras: instalas el OS, configuras el software, manejas updates, y escalas manualmente (o con Auto Scaling Groups).

### 4.1.1. Tipos de instancia

| Familia | Optimizada para | Ejemplo | Caso de uso |
|---|---|---|---|
| **T3/T4g** | General purpose (burstable) | `t3.micro` (1 vCPU, 1GB) | Web servers, dev, APIs pequeñas |
| **M5/M6i** | General purpose (fija) | `m6i.large` (2 vCPU, 8GB) | Apps de produccion, backends |
| **C5/C6i** | Compute optimized | `c6i.xlarge` (4 vCPU, 8GB) | Procesamiento intensivo, batch |
| **R5/R6i** | Memory optimized | `r6i.large` (2 vCPU, 16GB) | Bases de datos in-memory, caches |
| **G4/P4** | GPU | `g4dn.xlarge` | Machine learning, rendering |
| **I3/I4i** | Storage optimized | `i4i.large` | Bases de datos de alto I/O |

**COMO LEER EL NOMBRE**

`t3.large`: **t** = familia (burstable), **3** = generacion, **large** = tamaño. Las generaciones mas nuevas son mas baratas y rapidas. Siempre elige la generacion mas reciente disponible.

### 4.1.2. AMI, EBS, User Data

**AMI**

Amazon Machine Image. Template del disco con el OS preinstalado. Puedes usar AMIs oficiales (Ubuntu, Amazon Linux) o crear las tuyas custom.

**EBS**

Elastic Block Store. Disco duro virtual. Se conecta a la EC2. Persiste cuando apagas la instancia. Tipos: gp3 (SSD general), io2 (alto IOPS), st1 (HDD barato).

**User Data**

Script bash que se ejecuta al primer boot. Automatiza instalacion de software. Equivalente a nuestro `setup.sh` pero integrado en EC2.

### 4.1.3. Auto Scaling Groups (ASG)

Un ASG monitorea metricas y automaticamente ajusta el numero de instancias EC2.

**Configuracion tipica:**

- Min: 2 instancias (siempre hay al menos 2 corriendo)
- Max: 10 instancias (nunca mas de 10)
- Desired: 3 (estado normal)
- Scale up: si CPU > 70% por 5 min → +2 instancias
- Scale down: si CPU < 30% por 10 min → −1 instancia
- Health check: si una instancia no responde, la reemplaza automaticamente

## 4.2. Lambda

**AWS Lambda**

Funciones serverless. Subes codigo, AWS lo ejecuta cuando hay un evento. Pagas solo por ejecucion.

Lambda es el servicio **serverless** por excelencia. No administras servidores, no pagas por tiempo idle, y escala automaticamente de 0 a miles de ejecuciones concurrentes.

### 4.2.1. Como funciona

**1.** Subes tu codigo (Python, Node.js, Java, Go, etc) como un ZIP o imagen Docker
**2.** Configuras un **trigger**: peticion HTTP (via API Gateway), mensaje SQS, archivo subido a S3, cron (EventBridge), etc
**3.** Cuando el trigger se activa, AWS ejecuta tu funcion en un container efimero
**4.** La funcion corre, retorna resultado, y el container se destruye (o se reutiliza si hay mas peticiones)
**5.** Pagas por: numero de requests + duracion (ms) + memoria asignada


### 4.2.2. Limites importantes

| Limite | Valor | Implicacion |
|---|---|---|
| Timeout maximo | 15 minutos | No sirve para procesos largos |
| Memoria maxima | 10 GB | CPU escala proporcional a la memoria |

| | | |
|---|---|---|
| Tamaño del deploy | 250 MB (ZIP) / 10 GB (Docker) | No para apps monoliticas enormes |
| Concurrencia | 1000 simultaneas (default, aumentable) | Cuidado con cold starts a escala |
| Almacenamiento efimero | 512 MB en /tmp (hasta 10 GB) | Para archivos temporales |

**CUANDO USARLO**

- APIs ligeras con poco trafico
- Procesamiento de eventos (S3, SQS)
- Cron jobs (EventBridge schedule)
- Webhooks y notificaciones
- Backend de apps mobile/web simples
- Cuando el trafico es impredecible (escala a 0)

**CUANDO NO**

- Apps con trafico constante (pagas mas que EC2/Fargate)
- Procesos que tardan >15 min
- Apps que necesitan estado persistente
- Cold starts afectan latencia (~100-500ms)
- Debugging es mas dificil que en EC2

## 4.3. ECS y Fargate

**Elastic Container Service**

Orquestador de contenedores Docker. Corre tus containers en la nube con auto-scaling y health checks.

ECS es como Docker Compose pero managed y distribuido. Defines “task definitions” (que containers correr), “services” (cuantas copias mantener), y ECS se encarga del scheduling, health checks, y auto-scaling.

### 4.3.1. ECS en EC2 vs Fargate

**ECS en EC2**

**Tu administras las instancias EC2** donde corren los containers. Tu decides cuantas, de que tipo, y manejas patches/updates.

**Ventaja**: Control total, mas barato para cargas predecibles. Puedes usar Spot instances.
**Desventaja**: Mas trabajo operativo.

**Fargate (serverless)**

**AWS administra la infraestructura**. Tu solo defines CPU y memoria por task. No ves ni tocas instancias EC2.

**Ventaja**: Zero infraestructura que administrar. Escalado perfecto.
**Desventaja**: Mas caro por hora. Menos control.

### 4.3.2. Conceptos de ECS

| Concepto | Que es |
|---|---|
| **Cluster** | Grupo logico de recursos donde corren tus tasks. Puede tener EC2 instances o ser Fargate-only. |

| | |
|---|---|
| **Task Definition** | Template que define tus containers: imagen Docker, CPU, memoria, puertos, variables de entorno, volumes. Es el equivalente al `docker-compose.yml`. |
| **Task** | Una instancia ejecutandose de un Task Definition. Es como un container corriendo. |
| **Service** | Mantiene N tasks corriendo. Si una muere, lanza otra. Conecta con ALB para distribuir trafico. Es el “desired state”. |

## 4.4. App Runner

**AWS App Runner**
La forma mas simple de correr containers en AWS. Push de imagen o codigo, y App Runner hace todo lo demas.

App Runner es **la opcion mas simple** para deploy de containers. No necesitas configurar VPC, ALB, ECS, ni Task Definitions. Le das una imagen Docker (o codigo fuente) y el crea todo automaticamente.

| Aspecto | App Runner | ECS Fargate | Lambda |
|---|---|---|---|
| Complejidad | Minima | Media-Alta | Baja |
| Control | Poco | Total | Medio |
| Auto-scaling | Automatico | Configurable | Automatico |
| VPC/Networking | Opcional | Obligatorio | Opcional |
| Load Balancer | Incluido | Tu lo configuras (ALB) | API Gateway |
| Deploy | Push imagen → deploy | Task definition → service update | Upload ZIP/Docker |
| Costo minimo | ~$5/mes | ~$10/mes | Pay per request |
| Ideal para | APIs simples | Microservicios complejos | Event-driven |

# 5. Almacenamiento

## 5.1. S3 (Simple Storage Service)

**Simple Storage Service**
Object storage infinito. Archivos, backups, static websites, data lakes. El servicio mas versatil de AWS.

S3 es almacenamiento de **objetos** (archivos). No es un filesystem — no hay directorios reales (las "carpetas" son solo prefijos en el nombre). Cada objeto puede ser de hasta 5 TB. Durabilidad: 99.999999999% (11 nueves).

### 5.1.1. Conceptos clave

| Concepto | Que es |
|---|---|
| **Bucket** | Contenedor de objetos. Nombre globalmente unico. Ejemplo: `mi-app-backups-prod`. |
| **Object** | Un archivo + metadata. Key = nombre/path, Value = contenido, Version ID. |
| **Prefix** | Simula directorios. `backups/2026/03/dump.sql.gz` tiene prefix `backups/2026/03/`. |
| **Versioning** | Guarda todas las versiones de un objeto. Si sobreescribes, la version anterior sigue ahi. |
| **Lifecycle Rules** | Automaticamente mueve objetos entre clases de almacenamiento o los borra despues de N dias. |
| **Presigned URL** | URL temporal (ej: valida 1 hora) que permite descargar/subir un archivo privado sin credenciales AWS. |
| **Bucket Policy** | Politica JSON adjunta al bucket. Controla acceso. Similar a IAM Policy pero especifica al bucket. |

### 5.1.2. Clases de almacenamiento

| Clase | Costo/GB | Latencia | Caso de uso |
|---|---|---|---|
| **S3 Standard** | $0.023 | ms | Acceso frecuente (archivos activos, assets web) |
| **S3 Infrequent Access** | $0.0125 | ms | Backups recientes, logs mensuales |
| **S3 Glacier Instant** | $0.004 | ms | Archivos legales, auditorias |

| | | | |
|---|---|---|---|
| **S3 Glacier Flexible** | $0.0036 | 1-12 horas | Archivos de compliance, rare access |
| **S3 Glacier Deep Archive** | $0.00099 | 12-48 horas | Retencion a largo plazo (7+ años) |
| **S3 Intelligent-Tiering** | Variable | ms | Cuando no sabes el patron de acceso |

**LIFECYCLE EXAMPLE**

Puedes crear una regla: *"despues de 30 dias, mover a Infrequent Access. Despues de 90 dias, mover a Glacier. Despues de 365 dias, borrar."* Asi optimizas costo automaticamente sin tocar nada.

# 6. Bases de datos

## 6.1. RDS (Relational Database Service)

**Relational Database Service**
Bases de datos relacionales managed. Soporta PostgreSQL, MySQL, MariaDB, Oracle, SQL Server, Aurora.

RDS te quita el trabajo operativo de una base de datos: backups automaticos, patching, Multi-AZ failover, y monitoring. Tu solo te preocupas por el schema y los queries.

### 6.1.1. Que incluye RDS managed

**Backups automaticos**

Snapshots diarios. Retencion 1-35 dias. Point-in-time recovery hasta 5 min atras. **No necesitas pg_dump**.

**Multi-AZ**

Replica sincrona en otra AZ. Si el primary cae, failover automatico en ~60s. El endpoint DNS no cambia. Tu app no se entera.

**Read Replicas**

Copias async de lectura. Distribuyes queries SELECT. Hasta 5 replicas. Cross-region posible. Reduce carga del primary.

### 6.1.2. RDS vs Aurora

| Aspecto | RDS PostgreSQL | Aurora PostgreSQL |
|---|---|---|
| **Performance** | Standard | 3-5x mas rapido que RDS standard |
| **Storage** | EBS (tu defines tamaño) | Auto-scaling hasta 128 TB |
| **Replicas** | Hasta 5 | Hasta 15, failover mas rapido |
| **Costo** | ~$15/mes (db.t3.micro) | ~20% mas que RDS |
| **Serverless** | No | Si (Aurora Serverless v2, escala a 0) |

**CUANDO USAR AURORA**

Usa Aurora cuando necesites **mejor performance sin cambiar codigo** (es compatible con PostgreSQL/MySQL). Aurora Serverless v2 es ideal para apps con trafico variable — escala CPU y memoria automaticamente y puede bajar casi a cero en momentos sin trafico.

## 6.2. ElastiCache

### Amazon ElastiCache

Redis o Memcached managed. Cache en memoria para acelerar apps y reducir carga en la DB.

**Redis (ElastiCache)**

Estructuras de datos avanzadas (strings, hashes, lists, sets, sorted sets). Persistencia opcional. Pub/Sub. Lua scripting. Cluster mode con sharding.

**Memcached (ElastiCache)**

Solo key-value simple. Multi-threaded (mejor para cache puro). Sin persistencia. Sin replicacion. Mas simple, un poco mas rapido para cache puro.

# 7. Networking: Load Balancing y API Gateway

## 7.1. ALB (Application Load Balancer)

**Application Load Balancer**
Distribuye trafico HTTP/HTTPS entre multiples targets. Routing por Host, path, headers. TLS termination con ACM.

### 7.1.1. Tipos de Load Balancer

| Tipo | Que hace | Caso de uso |
|---|---|---|
| **ALB** | Capa 7 (HTTP). Routing por host, path, headers, query string. WebSocket support. gRPC support. | APIs, web apps, microservicios |
| **NLB** | Capa 4 (TCP/UDP). Ultra baja latencia (~100us). Millones de requests/s. IP estatica. | Gaming, IoT, alta performance |
| **CLB** | Classic Load Balancer. Legacy. No usar en proyectos nuevos. | Nada — esta deprecado |
| **GWLB** | Gateway LB para appliances de red (firewalls, IDS/IPS). | Seguridad de red avanzada |

### 7.1.2. Componentes del ALB

**Listener**: Escucha en un puerto (443 HTTPS). Define reglas de routing.
**Rule**: Si Host = `api.miapp.com` Y path = `/v2/*` → forward a Target Group B.
**Target Group**: Grupo de destinos (EC2, ECS tasks, IPs, Lambdas). ALB hace health checks al grupo.
**Health Check**: Pings periodicos (`GET /health`, espera 200). Si un target falla, deja de recibir trafico.

## 7.2. API Gateway

**Amazon API Gateway**
Puerta de entrada managed para APIs REST, HTTP y WebSocket. Rate limiting, auth, transformaciones, y monetizacion.

API Gateway se sienta **entre el cliente y tu backend**. Maneja autenticacion, throttling, caching, y puede transformar requests/responses. Es el front-door para tus APIs.

### 7.2.1. REST API vs HTTP API

**REST API**

API completa con todas las features: caching, request/response transformation, API keys, usage plans, WAF integration, custom authorizers.

**Precio**: $3.50 por millon de requests.

**Ideal para**: APIs publicas con monetizacion, APIs que necesitan transformar payloads.

**HTTP API**

Version ligera y mas barata. Soporta JWT auth nativo, CORS automatico, integracion con Lambda y ALB.

**Precio**: $1.00 por millon de requests (71% mas barato).

**Ideal para**: La mayoria de APIs modernas. Si no necesitas features avanzadas de REST API, usa esta.

### 7.2.2. Cuando usar API Gateway vs ALB

| Aspecto | API Gateway | ALB |
|---|---|---|
| **Backend** | Lambda, HTTP endpoints, AWS services | EC2, ECS, IPs |
| **Auth nativo** | Cognito, Lambda authorizer, JWT | No (tu app maneja auth) |
| **Rate limiting** | Si (por API key, por ruta) | No nativo (usa WAF) |
| **Caching** | Si (response caching) | No |
| **WebSocket** | Si | Si |
| **Costo** | Pay per request ($1-3.50/M) | Fijo (~$22/mes) + LCU |
| **Latencia** | Mayor (~10-30ms overhead) | Menor (~1-5ms) |

# 8. Mensajeria y Eventos

## 8.1. SQS (Simple Queue Service)

**Simple Queue Service**
Cola de mensajes managed. Desacopla servicios para que procesen trabajo de forma asincrona.

SQS es una **cola de mensajes**. Un servicio pone mensajes en la cola (productor), otro los lee y procesa (consumidor). Si el consumidor esta ocupado o caido, los mensajes se quedan esperando. Esto **desacopla** tus servicios.

### 8.1.1. Ejemplo practico

**Sin SQS**: Usuario sube imagen → API la redimensiona sincronamente (3s de espera) → responde.
**Con SQS**: Usuario sube imagen → API manda mensaje a SQS "procesar imagen X" → responde inmediatamente (200ms). Una Lambda lee la cola y redimensiona en background.

**Beneficio**: El usuario no espera. Si hay 1000 imagenes, se procesan en paralelo. Si Lambda falla, el mensaje vuelve a la cola y se reintenta.

### 8.1.2. Standard vs FIFO

**Standard Queue**

- Throughput **ilimitado**
- At-least-once delivery (puede entregar 2 veces)
- Best-effort ordering (no garantiza orden)
- **Ideal para**: procesamiento masivo donde duplicados son OK

**FIFO Queue**

- Hasta 300 msg/s (3000 con batching)
- Exactly-once delivery (nunca duplicados)
- Orden estricto garantizado
- **Ideal para**: ordenes de compra, transacciones financieras

## 8.2. SES (Simple Email Service)

**Simple Email Service**
Envio y recepcion de emails a escala. Para transaccional (confirmaciones, reseteo password) y marketing.

**Email transaccional**

**Email marketing**

Confirmacion de cuenta, reseteo de password, recibos, alertas. Integra con tu app via SMTP o API. $0.10 por cada 1000 emails.

Newsletters, campañas, notificaciones masivas. Templates HTML, listas de suscriptores, tracking de opens/clicks. Maneja bounces y complaints.

## 8.3. EventBridge

**Amazon EventBridge**
Bus de eventos serverless. Conecta servicios AWS, apps propias, y SaaS externos via eventos.

EventBridge es el **sistema nervioso** de una arquitectura event-driven. Los servicios publican eventos ("usuario creado", "orden pagada", "imagen procesada") y EventBridge los rutea a los consumidores correctos basandose en reglas.

### 8.3.1. Diferencia con SQS

| Aspecto | SQS | EventBridge |
|---|---|---|
| **Patron** | Punto a punto (1 productor → 1 consumidor) | Pub/Sub (1 evento → N destinos) |
| **Routing** | No hay filtering — todo llega | Rules con pattern matching: filtra por tipo, contenido |
| **Destinos** | El consumidor hace poll | Lambda, SQS, SNS, Step Functions, API, etc |
| **Scheduling** | No | Si — cron expressions para triggers periodicos |
| **Caso de uso** | Trabajo asincrono pesado, decoupling | Orquestacion de microservicios, reaccionar a cambios |

**EVENTBRIDGE SCHEDULER**

EventBridge reemplaza CloudWatch Events (que es el servicio legacy de cron). Para programar tareas periodicas:
`rate(5 minutes)` → cada 5 minutos
`cron(0 3 * * ? *)` → todos los dias a las 3 AM
Trigger: Lambda, ECS task, Step Function, SQS, etc.

# 9. CI/CD en AWS

## 9.1. El pipeline completo

AWS tiene una familia de servicios de CI/CD que se integran entre si:

**CodeCommit**

Repositorio Git managed. Equivalente a GitHub/GitLab pero dentro de AWS. Privado, integrado con IAM.

DEPRECADO AWS anuncio que ya no acepta nuevos repos. Usa GitHub.

**CodeBuild**

Servicio de build serverless. Compila codigo, corre tests, crea artifacts. Pagas por minuto de build. Soporta Docker.

Equivalente a: GitHub Actions runner.

**CodeDeploy**

Despliega a EC2, ECS, Lambda, o on-premise. Rolling, blue/green, canary. Health checks automaticos con rollback.

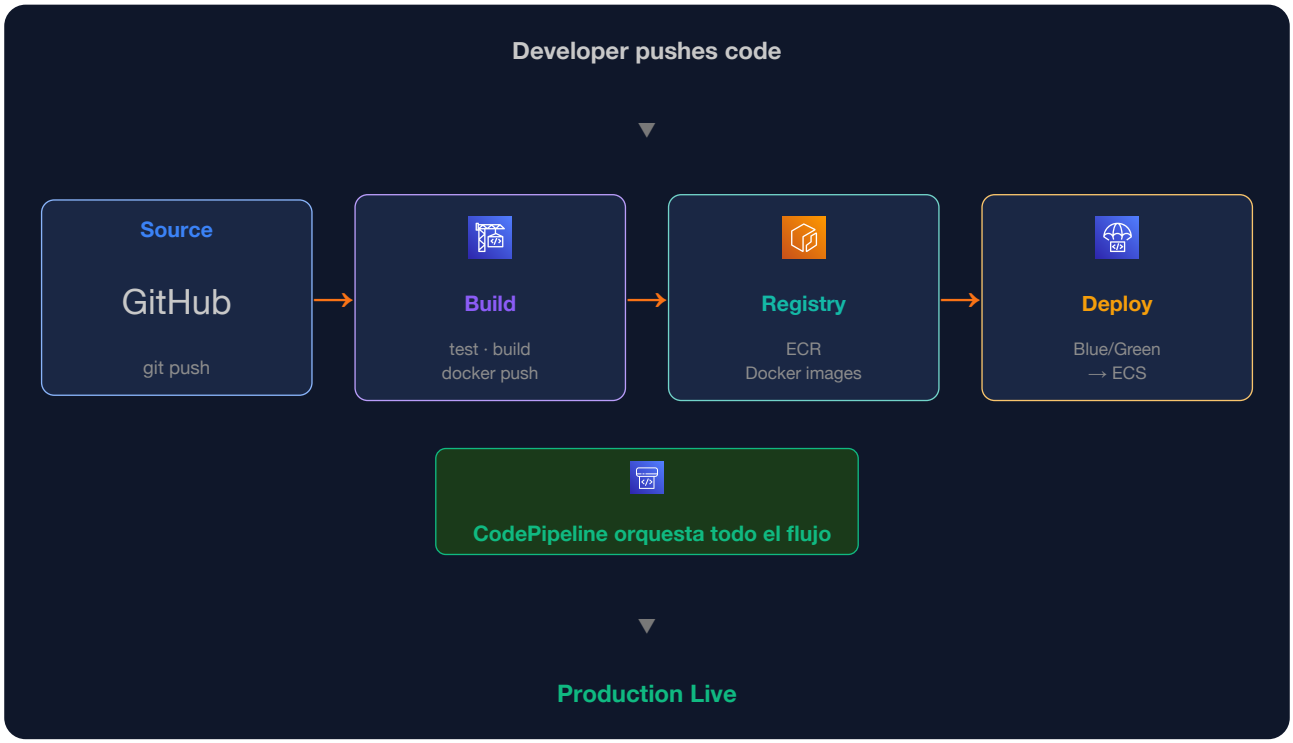
Equivalente a: SSH deploy + rollback.

**CodePipeline**

Orquestador del pipeline completo. Source → Build → Test → Deploy. Aprobaciones manuales. Integra los 3 anteriores.

Equivalente a: GitHub Actions workflow.

### 9.1.1. Flujo tipico AWS CI/CD



## 9.2. ECR (Elastic Container Registry)

**Elastic Container Registry**
Registro de imagenes Docker managed. Integrado con ECS, Fargate, CodeBuild, y IAM.

| Feature | Detalle |
|---|---|
| **Vulnerability scanning** | Inspector escanea imagenes automaticamente al pushear. Reporta CVEs. |
| **Lifecycle policies** | Borrar imagenes sin tag despues de 7 dias. Mantener solo las ultimas 10 taggeadas. |
| **Cross-region replication** | Replicar imagenes a otra region automaticamente (DR). |
| **Immutable tags** | Evitar que alguien sobreescriba `latest`. Un tag = una imagen, siempre. |
| **Auth** | `aws ecr get-login-password \| docker login`. Token temporal via IAM. |

# 10. DNS, CDN y Distribucion

## 10.1. Route 53

**Amazon Route 53**
DNS managed. Registro de dominios, routing inteligente (geolocation, failover, weighted), health checks.

Route 53 no es solo un DNS — tiene **routing inteligente**:

| Routing Policy | Como funciona |
|---|---|
| **Simple** | Un record, un destino. Como cualquier DNS normal. |
| **Weighted** | 70% del trafico a la version A, 30% a la version B. Para canary deployments. |
| **Latency** | Enviar al usuario a la region con menor latencia. Si estas en Mexico → us-east-1, si estas en Europa → eu-west-1. |
| **Failover** | Primary en us-east-1 con health check. Si falla, automaticamente rutea a secondary en us-west-2. |
| **Geolocation** | Usuarios de Mexico → servidor en Mexico. Usuarios de España → servidor en Europa. |
| **Multi-value** | Retorna multiples IPs healthy. DNS-level load balancing. |

## 10.2. CloudFront

**Amazon CloudFront**
CDN global con 400+ edge locations. Cachea contenido cerca del usuario. Reduce latencia dramaticamente.

**Sin CDN**: Usuario en Mexico → request viaja a Virginia (us-east-1) → 200ms latencia.
**Con CloudFront**: Primer request viaja a Virginia, CloudFront cachea la respuesta en edge de CDMX. Segundo request → 10ms latencia desde el edge node mas cercano.

**Origins soportados**: S3 bucket, ALB, EC2, API Gateway, cualquier HTTP endpoint custom.

### 10.2.1. CloudFront + S3 = Static Website Hosting

**PATRON COMUN**

**Frontend SPA (Angular/React)**: Subes el build (`dist/`) a S3. Creas distribucion CloudFront apuntando a ese bucket. Agregas certificado ACM. Configuras Route 53 alias. Resultado: tu frontend se sirve desde 400+ edge locations globalmente, con HTTPS, por centavos al mes.

# 11. Observabilidad

## 11.1. CloudWatch

**Amazon CloudWatch**
Metricas, logs, alarmas y dashboards. El sistema nervioso de monitoreo de toda tu infra AWS.

### 11.1.1. Componentes

**Metrics**

Datos numericos: CPU de EC2, latencia de ALB, invocaciones de Lambda, IOPS de RDS. AWS envia metricas automaticamente. Puedes crear metricas custom.

**Logs**

Centraliza logs de todos los servicios. Lambda, ECS, API Gateway envian logs automaticamente. Logs Insights para queries tipo SQL sobre tus logs.

**Alarms**

Si CPU > 80% por 5 min → alarma → notificacion SNS → email/Slack. Si errores 5xx > 10/min → alarma → trigger auto-scaling o rollback.

**Dashboards**

Dashboards visuales custom. Mezcla metricas de EC2, RDS, Lambda, ALB en un solo panel. Sharing y auto-refresh.

## 11.2. CloudTrail

**AWS CloudTrail**
Registra CADA llamada API en tu cuenta. Quien hizo que, cuando, desde donde. Auditoria completa.

**PARA QUE SIRVE**

- ¿Quien borro la instancia EC2 ayer a las 3 AM? → CloudTrail lo tiene.
- ¿Quien modifico el Security Group? → CloudTrail lo tiene.
- ¿Alguien accedio al secret de produccion? → CloudTrail lo tiene.
- ¿El root user se logueo? → CloudTrail + alarma inmediata.

Es el equivalente a un **audit log de toda tu infraestructura**. Mandatorio para compliance (SOC2, HIPAA, PCI).

# 12. Seguridad avanzada

## 12.1. WAF (Web Application Firewall)

**AWS WAF**

Firewall de aplicacion web. Protege contra SQL injection, XSS, bots, rate limiting a nivel de HTTP.

WAF se coloca **delante del ALB, API Gateway, o CloudFront**. Inspecciona cada request HTTP y puede bloquear basandose en reglas:

**Managed Rules**

Reglas pre-hechas por AWS o vendors: bloquea OWASP Top 10 (SQLi, XSS), bots conocidos, IPs de tor.

**Rate Limiting**

Bloquear IPs que hagan >1000 requests en 5 min. Por IP, por path, por header. Previene DDoS capa 7.

**Custom Rules**

Bloquear paises, user-agents, patterns en body/headers. Geo-blocking. IP whitelist/blacklist.

## 12.2. AWS Shield

| Tier | Que incluye | Costo |
|---|---|---|
| **Shield Standard** | Proteccion automatica contra DDoS capa 3/4 (SYN floods, UDP reflection). Activado por defecto en todos los recursos AWS. | Gratis |
| **Shield Advanced** | Proteccion DDoS capa 7, equipo de respuesta 24/7 (DRT), proteccion de costos (AWS absorbe cargos extra por DDoS), metricas detalladas. | $3,000/mes |

## 12.3. Secrets Manager vs Parameter Store

| Aspecto | Secrets Manager | Parameter Store |
|---|---|---|
| **Proposito** | Secrets (passwords, API keys, DB creds) | Configuracion general + secrets |
| **Rotacion** | Automatica (Lambda integrada) | Manual |
| **Costo** | $0.40/secret/mes + $0.05/10K requests | Gratis (standard) / $0.05 (advanced) |

| | | |
|---|---|---|
| **Encryption** | KMS siempre | KMS opcional (SecureString) |
| **Versioning** | Automatico | Basico |
| **Recomendacion** | Para DB passwords y API keys criticas | Para feature flags, config, URLs |

# 13. Infrastructure as Code (IaC)

## 13.1. CloudFormation

**AWS CloudFormation**
Define TODA tu infraestructura en templates YAML/JSON. Un comando crea, actualiza o destruye todo.

CloudFormation es el servicio nativo de AWS para IaC. Escribes un template que describe tus recursos (VPC, subnets, EC2, RDS, ALB…) y CloudFormation los crea en el orden correcto, maneja dependencias, y puede hacer rollback si algo falla.

### 13.1.1. Ejemplo basico

```yaml
# cloudformation-template.yml
AWSTemplateFormatVersion: '2010-09-09'
Resources:
  MyVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16

  MySubnet:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref MyVPC
      CidrBlock: 10.0.1.0/24
      AvailabilityZone: us-east-1a

  MyDatabase:
    Type: AWS::RDS::DBInstance
    Properties:
      DBInstanceClass: db.t3.micro
      Engine: postgres
      MasterUsername: admin
      MasterUserPassword: !Ref DBPassword
```

### 13.1.2. CloudFormation vs Terraform

| Aspecto | CloudFormation | Terraform |
|---|---|---|
| **Proveedor** | Solo AWS | Multi-cloud (AWS, GCP, Azure, etc) |
| **Lenguaje** | YAML/JSON | HCL (Hashicorp Config Language) |
| **State** | AWS lo maneja (en su backend) | Tu lo manejas (S3 + DynamoDB) |
| **Drift detection** | Nativo | Con `terraform plan` |

| | | |
|---|---|---|
| **Ecosistema** | AWS nativo, soporta todo | Mas modulos comunitarios |
| **Curva aprendizaje** | Media (YAML verboso) | Media (HCL intuitivo) |
| **Recomendacion** | Si solo usas AWS | Si usas multi-cloud o quieres portabilidad |

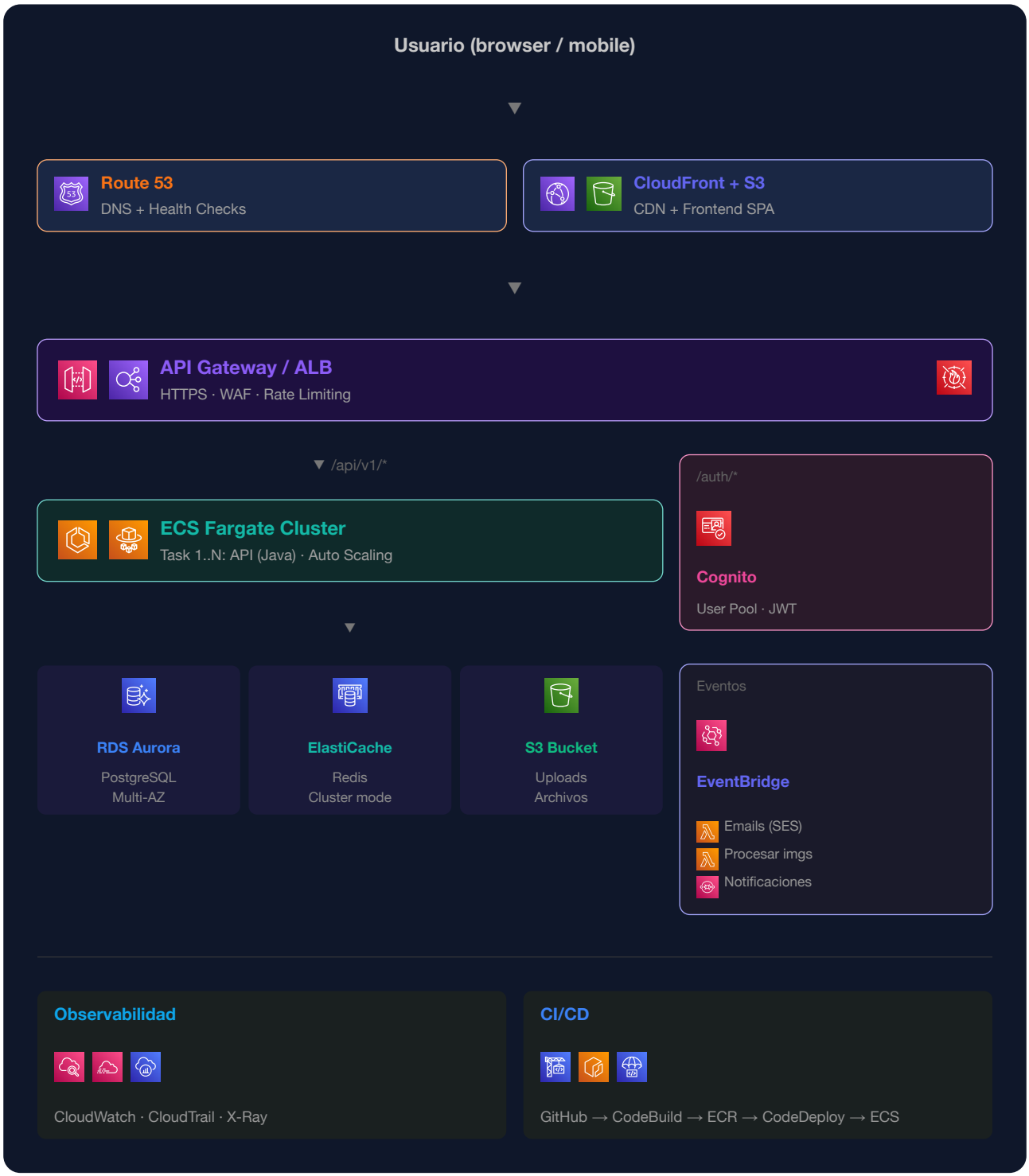
# 14. Cheat Sheet: Todos los servicios

| Sigla | Nombre | En una frase | Categoria |
|---|---|---|---|
| **EC2** | Elastic Compute Cloud | Servidores virtuales (VMs) que tu administras | Computo |
| **Lambda** | AWS Lambda | Funciones serverless que corren por evento | Computo |
| **ECS** | Elastic Container Service | Orquestador de containers Docker | Computo |
| **Fargate** | AWS Fargate | Containers serverless (sin administrar EC2) | Computo |
| **App Runner** | AWS App Runner | Deploy de containers con zero config | Computo |
| **VPC** | Virtual Private Cloud | Tu red privada aislada en la nube | Redes |
| **ALB** | Application Load Balancer | Distribuye trafico HTTP entre targets | Redes |
| **APIGW** | API Gateway | Puerta de entrada managed para APIs | Redes |
| **R53** | Route 53 | DNS managed con routing inteligente | Redes |
| **CF** | CloudFront | CDN global, 400+ edge locations | Redes |
| **S3** | Simple Storage Service | Object storage infinito y barato | Storage |
| **EBS** | Elastic Block Store | Discos duros virtuales para EC2 | Storage |
| **RDS** | Relational Database Service | PostgreSQL/MySQL managed con backups y Multi-AZ | DB |
| **Aurora** | Amazon Aurora | RDS premium: 3-5x mas rapido, serverless option | DB |
| **ElastiCache** | Amazon ElastiCache | Redis/Memcached managed | DB |
| **SQS** | Simple Queue Service | Cola de mensajes para trabajo asincrono | Mensajeria |
| **SES** | Simple Email Service | Envio de emails transaccional y marketing | Mensajeria |
| **EB** | EventBridge | Bus de eventos serverless + cron scheduler | Mensajeria |
| **IAM** | Identity and Access Mgmt | Quien puede hacer que en tu cuenta | Seguridad |
| **WAF** | Web Application Firewall | Proteccion contra SQLi, XSS, DDoS capa 7 | Seguridad |
| **SM** | Secrets Manager | Almacena y rota secrets automaticamente | Seguridad |
| **KMS** | Key Management Service | Llaves de encriptacion managed | Seguridad |
| **CW** | CloudWatch | Metricas, logs, alarmas, dashboards | Observabilidad |
| **CT** | CloudTrail | Audit log de cada accion en tu cuenta | Observabilidad |
| **ECR** | Elastic Container Registry | Registro de imagenes Docker | DevOps |

| | | | |
|---|---|---|---|
| **CB** | CodeBuild | Build y test serverless | DevOps |
| **CD** | CodeDeploy | Deploy automatizado con rollback | DevOps |
| **CP** | CodePipeline | Orquestador de CI/CD pipeline | DevOps |
| **CFN** | CloudFormation | Infrastructure as Code nativo de AWS | DevOps |
| **ACM** | Certificate Manager | Certificados TLS gratuitos para servicios AWS | Seguridad |

# 15. Arquitectura de referencia: App web moderna en AWS

Esta es la arquitectura tipica de una app web en produccion usando AWS.




### 15.0.1. Costo estimado de esta arquitectura

| Servicio | Costo/mes | Config |
|---|---|---|

| | | |
|---|---|---|
| ECS Fargate (2 tasks) | ~$35 | 0.5 vCPU, 1GB RAM cada uno |
| ALB | ~$22 | Fijo + LCU usage |
| RDS Aurora | ~$30 | db.t3.medium, Multi-AZ |
| ElastiCache | ~$13 | cache.t3.micro |
| S3 + CloudFront | ~$5 | Static hosting + CDN |
| Route 53 | ~$1 | 1 hosted zone + queries |
| CloudWatch | ~$5 | Metricas + logs basicos |
| ECR | ~$2 | Unas pocas imagenes |
| Secrets Manager | ~$2 | 5-10 secrets |
| NAT Gateway | ~$35 | 1 en una AZ (ahorro) |
| **TOTAL** | **~$150-200/mes** | Para una app con trafico moderado |

### Consejo final

No intentes aprender los 200+ servicios de AWS. Domina estos 15-20 servicios core y podras construir el 95% de las aplicaciones. El resto lo aprendes cuando lo necesites. **Construye proyectos reales** — es la unica forma de entender como se conectan entre si.