Modulo 9

Actividad **PORTAFOLIO ABP**

Nombre: Marco Neira

**Lección 1 – Computación en la nube**

**1. Clasificación de servicios AWS usados en el proyecto**



**2. Cuadro comparativo de modelos de servicio**



**3. Ventajas del modelo cloud frente al modelo tradicional**

1. Escalabilidad: en la nube se puede crecer o reducir recursos bajo demanda; en un datacenter físico se requieren compras anticipadas de hardware.
2. Pago por uso: no es necesario invertir en infraestructura fija, solo se paga lo que se consume.
3. Alta disponibilidad: servicios distribuidos en varias zonas de disponibilidad, lo que reduce caídas.
4. Agilidad: aprovisionamiento en minutos, en contraste con semanas/meses en modelo on-premise.
5. Menor carga operativa: el equipo de TI se enfoca en tareas de valor y no en mantenimiento de hardware.

**Lección 2 – Tecnologías de almacenamiento en la nube**

**1. Justificación del uso de S3, EBS, EFS o Glacier en distintas capas de la solución**



**2. Simulación de estructura de buckets y clases de almacenamiento**

Bucket principal: fintech-data-lake

* /raw/ → Datos en crudo (logs, CSV de clientes) → Clase S3 Standard.
* /processed/ → Datos transformados por Glue/EMR → Clase S3 Standard-IA (Infrequent Access).
* /reports/ → Resultados analíticos cargados a Redshift o QuickSight → Clase S3 Standard.
* /archive/ → Datos antiguos (más de 1 año) → Clase Glacier / Glacier Deep Archive.

**3. Estimación comparativa de costos según frecuencia de uso**

* **S3 Standard:** más caro que Glacier, pero necesario para datos de acceso frecuente.
* **S3 Standard-IA:** 40% más barato que Standard; ideal para datos procesados con consultas ocasionales.
* **Glacier/Deep Archive:** hasta 90% más barato que Standard; recomendado para almacenamiento a largo plazo con bajas necesidades de recuperación.
* **EBS/EFS:** más caros que S3, pero esenciales para aplicaciones transaccionales y procesamiento intensivo (base de datos, ETL en vivo).

**4. Ventajas generales del enfoque**

* Se optimizan **costos** al asignar cada tipo de almacenamiento según la frecuencia de acceso.
* Se mejora la **eficiencia** del flujo de datos al separar lo “caliente” (frecuente) de lo “frío” (histórico).
* Se garantiza **durabilidad** (11 9’s en S3) y seguridad (cifrado, IAM policies).

**Lección 3 – Servicios de cómputo**

**1. Comparación entre EC2, Contenedores y Lambda**



**2. Propuesta de tipo de cómputo para distintos flujos**

* **ETL (extracción, transformación y carga):**

**AWS Glue + EMR sobre EC2** → para cargas grandes y procesamiento distribuido.

* **Inferencia de modelos (ML/IA):**

**ECS/EKS con contenedores** → despliegue de modelos como microservicios.

* **APIs de consulta o servicios web:**

**Contenedores (ECS/EKS)** para endpoints REST/GraphQL escalables.

* **Procesos automatizados/eventos puntuales:**

**AWS Lambda** para triggers de carga de datos en S3 o notificaciones en SNS.

* **Procesos intensivos batch:**

**Amazon EMR con Spark sobre EC2/EBS** para análisis de grandes volúmenes en paralelo.

**3. Justificación técnica, económica y operativa**

* **Técnica:** cada tipo de cómputo se asigna según la naturaleza de la carga (transaccional, batch, event-driven, servicios web).
* **Económica:**

EC2 → mayor costo fijo (pero necesario para cargas largas).

Contenedores → optimización de recursos, ideal para workloads medianos y repetitivos.

Lambda → ahorro total en servidores inactivos, se paga solo por ejecución.

* **Operativa:**

Se reduce la carga de gestión usando **FaaS** y **PaaS**.

EC2 se reserva para procesos críticos que no pueden ejecutarse bajo límites de Lambda.

**Lección 4 – Servicios de bases de datos en la nube**

**1. Comparación de Amazon RDS, DynamoDB y Redshift**



**2. Diseño de arquitectura que incluya una base relacional gestionada**

**Arquitectura propuesta:**

1. **Amazon RDS (Aurora)** : Base principal de operaciones transaccionales (clientes, cursos, pagos).
2. **Amazon DynamoDB** : Complemento para datos no estructurados (sesiones, logs, interacciones en la app).
3. **Amazon Redshift** : Consolida datos desde RDS y DynamoDB (vía AWS Glue) para análisis histórico y reporting.
4. **Amazon S3** : Data Lake que almacena datos brutos y respaldos antes de entrar a Redshift.

Flujo:  
Aplicaciones → RDS/DynamoDB → Glue ETL → Redshift → QuickSight (dashboards).

**3. Configuraciones para alta disponibilidad y recuperación**

* **RDS:**

Multi-AZ (réplica en otra zona de disponibilidad).

Backups automáticos + snapshots manuales.

Read Replicas para cargas de lectura intensivas.

* **DynamoDB:**

Replicación global con **Global Tables** para tolerancia a fallos.

On-Demand Backup y Point-in-Time Recovery (PITR).

* **Redshift:**

RA3 nodes con almacenamiento gestionado.

Snapshots automáticos en S3.

Clústeres en múltiples AZs (Redshift Serverless si se requiere elasticidad).

**4. Beneficios de esta arquitectura**

* **Escalabilidad:** cada servicio escala de forma independiente según carga.
* **Resiliencia:** tolerancia a fallos mediante réplicas y backups automáticos.
* **Seguridad:** cifrado en reposo (KMS) y en tránsito (TLS).
* **Optimización de costos:** se paga según uso y capacidad provisionada.
* **Eficiencia analítica:** separación entre operaciones transaccionales (OLTP) y análisis masivo (OLAP).

**Lección 5 – Movimiento de datos en la nube mediante ETL**

**1. Flujo ETL completo con Glue Studio**

El flujo propuesto incluye las siguientes etapas:

1. **Ingesta de datos**: Datos brutos llegan a **Amazon S3** (bucket fintech-data-lake/raw/).
2. **Crawlers**: **AWS Glue Crawlers** detectan esquemas automáticamente y actualizan el **Data Catalog**.
3. **Transformación**: Un **Glue Job** en PySpark limpia y transforma los datos (normalización de columnas, formato Parquet).
4. **Carga final**: Los datos procesados se guardan en **Amazon Redshift** (para análisis OLAP) o en **RDS** (para operaciones OLTP).

Flujo:  
**S3 (raw) → Glue Crawler → Data Catalog → Glue Job (PySpark) → S3 (processed) → Redshift**

**2. Simulación de ingesta de datos**

* **Fuente:** CSV con transacciones de clientes en s3://fintech-data-lake/raw/transactions/.
* **Transformación:**

Conversión a formato **Parquet** (más eficiente en almacenamiento y consultas).

Limpieza de valores nulos y estandarización de fechas.

* **Destino:**

s3://fintech-data-lake/processed/transactions/

Carga incremental en **Redshift table** transactions\_history.

**3. Documentación de componentes y dependencias**

* **AWS S3:** almacenamiento de datos brutos y transformados.
* **AWS Glue Data Catalog:** metadatos centralizados (tablas y esquemas).
* **AWS Glue Crawlers:** descubrimiento automático de esquemas.
* **AWS Glue Jobs:** ejecución de procesos ETL (PySpark).
* **AWS Glue Triggers:** automatizan la ejecución del job según horario o evento en S3.
* **Amazon Redshift:** destino analítico para reporting.
* **Amazon CloudWatch:** monitoreo de ejecución y logs de jobs.

**4. Calendario de ejecución (ejemplo)**

* **Crawler:** cada 6 horas para detectar nuevos esquemas.
* **ETL Job:** ejecución diaria (00:00 hrs) para consolidar información.
* **Carga a Redshift:** incremental, con actualización de tabla histórica.

**5. Beneficios de esta solución**

* **Automatización:** flujo ETL orquestado sin intervención manual.
* **Escalabilidad:** Glue ajusta recursos automáticamente.
* **Costo eficiente:** pago por uso (serverless).
* **Flexibilidad:** soporta múltiples formatos (CSV, JSON, Parquet).
* **Integración nativa:** con Redshift, RDS, DynamoDB y QuickSigh

**Lección 6 – Servicio de procesamiento distribuido (Amazon EMR)**

**1. Configuración de un clúster EMR simulado**

* **Versión:** Amazon EMR con **Apache Spark** y **Hadoop** preinstalados.
* **Nodos del clúster:**

**Master Node** → coordina el clúster y asigna tareas.

**Core Nodes** → ejecutan procesos y almacenan datos en HDFS.

**Task Nodes** → se agregan dinámicamente para procesamiento temporal.

* **Almacenamiento:**

**Amazon S3** como Data Lake (entrada/salida).

EBS para almacenamiento de los nodos.

* **Red:** Configuración en **VPC** privada con acceso controlado vía IAM.

**2. Diseño de un job batch con Spark**

**Ejemplo de flujo:**

1. **Input:** Archivos de transacciones grandes en formato CSV ubicados en s3://fintech-data-lake/raw/transactions/.
2. **Transformación con Spark:**

Limpieza de duplicados.

Conversión a formato **Parquet**.

Enriquecimiento con datos de clientes desde otra tabla en S3.

1. **Output:** Archivos procesados en s3://fintech-data-lake/processed/transactions/.
2. **Carga final:** Glue o Redshift para análisis y reporting.

**3. Beneficios del enfoque distribuido**

* **Escalabilidad horizontal:** Spark divide los datos y los procesa en paralelo en múltiples nodos.
* **Velocidad:** reducción drástica de tiempos de procesamiento frente a un servidor único.
* **Elasticidad:** posibilidad de agregar o quitar **task nodes** según la carga.
* **Integración con AWS:** trabaja de forma nativa con **S3, Glue, Redshift**.
* **Optimización de costos:** con nodos **spot** se abaratan cargas batch no críticas.

**4. Buenas prácticas de escalabilidad**

* Usar **Auto Scaling** para nodos de tarea según la carga de trabajo.
* Procesar datos en formatos optimizados como **Parquet/ORC** en lugar de CSV.
* Dividir jobs en etapas más pequeñas (pipelines).
* Monitorear con **CloudWatch + Ganglia** para detectar cuellos de botella.
* Configurar **IAM Roles** con privilegios mínimos para los nodos.

**Lección 7 – Procesamiento sin servidor (AWS Lambda)**

**1. Funciones Lambda disparadas por eventos**

Ejemplos de configuración en el proyecto:

* **Evento en S3 (ObjectCreated):**  
  Cada vez que se sube un archivo a s3://fintech-data-lake/raw/, se dispara una **función Lambda** que:

Valida el archivo.

Envía notificación a SNS.

Opcional: invoca un **Glue Job** para procesar el archivo.

* **Evento programado (cron job con CloudWatch Events / EventBridge):**  
  Función Lambda que ejecuta validaciones diarias, como limpieza de logs o generación de reportes.

**2. Tareas posibles de Lambda en la solución**

* **Notificación:** al detectar carga de nuevos datos, envía correo mediante **Amazon SNS**.
* **Transformación rápida:** validación de columnas o formateo de datos antes de enviarlos a Glue.
* **Orquestación:** invocación de **Glue Jobs** o inicio de pipelines ETL.
* **Monitoreo:** integración con CloudWatch para registrar errores y métricas

**3. Documentación de eventos, triggers y políticas IAM**

* **Eventos soportados:**

s3:ObjectCreated:\* para cargas de archivos.

cron(0 0 \* \* ? \*) para ejecución diaria programada.

* **Triggers:**

S3 (bucket de datos).

EventBridge (cron jobs).

* **IAM Policies:**

Rol con permisos mínimos:

* + - s3:GetObject, s3:PutObject → acceso a buckets.
    - glue:StartJobRun → invocar jobs de Glue.
    - sns:Publish → enviar notificaciones.
    - logs:CreateLogGroup, logs:CreateLogStream, logs:PutLogEvents → CloudWatch Logs.

**4. Buenas prácticas**

* **Seguridad:**

Principio de **menor privilegio** en roles IAM.

Cifrado en tránsito y en reposo para los datos procesados.

* **Rendimiento:**

Ajustar **memory y timeout** según carga del archivo.

Minimizar dependencias externas en el código Lambda.

Usar capas (Layers) para librerías comunes.

* **Escalabilidad:**

Configurar concurrencia reservada si hay picos de carga.

Integración con SQS como buffer si se esperan miles de archivos simultáneos.

**5. Reflexión final**

Los principales desafíos de implementar Lambda en esta arquitectura son:

* **Límites de ejecución (15 min):** algunos procesos deben pasar a Glue o EMR.
* **Control de costos:** si hay demasiados eventos simultáneos, puede escalar rápidamente la factura.
* **Complejidad de orquestación:** cuando hay múltiples Lambdas, es recomendable integrar **Step Functions** para controlar el flujo.