Programa Profesional en Inteligencia Artificial y Data Science

Módulo 4. Analítica escalable

Índice

AWS EMR (I)

AWS EMR (II)

Documentación oficial de MongoDB

Apache Cassandra

Redis

Neo4j

AWS DynamoDB

AWS DocumentDB

Documentación de AWS SageMaker

AWS Comprenhed

AWS Rekognition

Tema 1. Procesamiento de datos escalable

1.1. Objetivos del tema e introducción al procesamiento de datos escalable

Objetivos

Los objetivos de este tema son los siguientes:

- Comprender la escalabilidad en sistemas de procesamiento de datos.
- Analizar los paradigmas de procesamiento de datos.
- Superar desafíos técnicos y organizativos en el procesamiento de datos escalable.
- Valorar el papel de la computación en la nube.
- Comprender los conceptos fundamentales de Apache Kafka.
- Configurar y operar Apache Kafka.
- Analizar la arquitectura distribuida de Kafka.
- Implementar casos de uso de Kafka en tiempo real.
- Comparar distribuciones de Kafka.
- Comprender los fundamentos de AWS EMR (Elastic MapReduce).
- Configurar y operar clústeres EMR.
- Implementar flujos de trabajo en EMR.
- Aplicar herramientas y *frameworks* de análisis de datos en EMR.
- Gestionar costos, seguridad y monitoreo en EMR.

Tema 1. Procesamiento de datos escalable

Introducción

El procesamiento de datos escalable se refiere a la capacidad de manejar grandes volúmenes de datos de manera eficiente, adaptándose a cambios en la carga de trabajo sin perder rendimiento ni comprometer la disponibilidad.

Este concepto es clave en un mundo donde la cantidad de datos generados crece exponencialmente debido al auge de tecnologías como IoT, redes sociales y aplicaciones empresariales. Para comprender mejor este concepto, tenemos que profundizar en algunos de sus aspectos clave, vamos a realizarlo a continuación.

Concepto de escalabilidad

La escalabilidad es la capacidad de un sistema, red o aplicación para manejar un **incremento en la carga** de trabajo mediante el aumento de sus **recursos.** Este concepto es esencial en la planificación de infraestructuras tecnológicas para asegurar el rendimiento, la disponibilidad y la sostenibilidad del sistema en condiciones de crecimiento.

Existen dos tipos principales de escalabilidad:

- Escalabilidad horizontal *(scaling out).*
- Escalabilidad vertical *(scaling up).*

A continuación, vamos a profundizar sobre ellos.

- **Escalabilidad Horizontal *(scaling out)*:**

La escalabilidad horizontal implica agregar más máquinas o nodos al sistema para distribuir la carga de trabajo. Este enfoque es común en arquitecturas distribuidas como las de aplicaciones en la nube.

- Características:
  - Incrementa el número de instancias o nodos en la infraestructura.
  - Ideal para sistemas distribuidos, clústeres y microservicios.
  - Cada nodo trabaja en paralelo con otros para procesar datos o solicitudes.
- Ventajas:
  - Mayor tolerancia a fallos (si un nodo falla, otros siguen operando).
  - Fácil de implementar en entornos en la nube como AWS, Google Cloud o Azure.
  - Aumenta la capacidad global sin necesidad de modificar los nodos existentes.
- Desventajas:
  - Requiere sistemas distribuidos bien diseñados (por ejemplo, bases de datos distribuidas como Cassandra o MongoDB).
  - Introduce complejidad en la coordinación y sincronización entre nodos.

- Ejemplos:
  - Un sitio web de comercio electrónico distribuye la carga de usuarios entre múltiples servidores.
  - Servicios de *streaming* como Netflix añaden servidores para gestionar picos de usuarios.

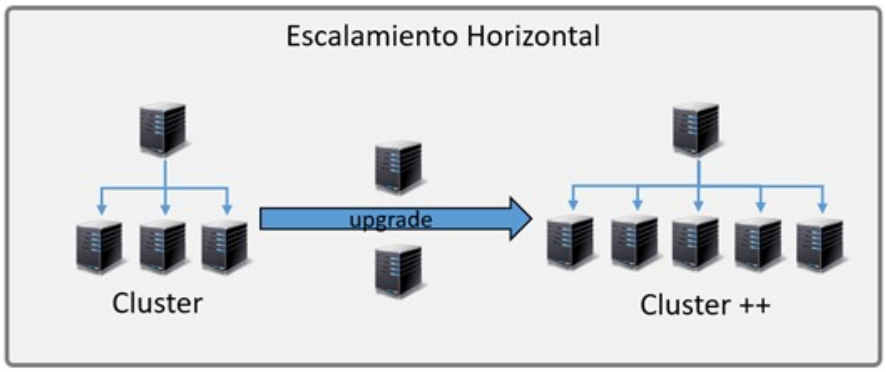

Figura 1. Diagrama comparativo de escalabilidad horizontal y vertical. Fuente: Blancarte, 2017.

- **Escalabilidad vertical *(scaling up)*:**

La escalabilidad vertical consiste en mejorar la capacidad de una máquina existente, por ejemplo, aumentando la RAM, los núcleos de CPU o el almacenamiento.

- Características:
  - Implica la actualización de *hardware* o la asignación de más recursos a una máquina existente.
  - Común en sistemas monolíticos o aplicaciones centralizadas.

- Ventajas:
  - Menos complejidad, ya que no hay necesidad de coordinar múltiples nodos.
  - Configuración más sencilla comparada con la horizontal.
  - Útil cuando la aplicación no admite arquitecturas distribuidas.
- Desventajas:
  - Límite físico: no puedes escalar indefinidamente (limitaciones del *hardware*).
  - Punto único de fallo: si el servidor falla, todo el sistema queda inoperativo.
  - Costos más altos por unidad al mejorar *hardware* de alto rendimiento.
- Ejemplos:
  - Una base de datos relacional centralizada mejora su rendimiento aumentando el almacenamiento SSD y la memoria RAM.
  - Una aplicación alojada en un servidor único amplía su capacidad aumentando los núcleos de CPU.

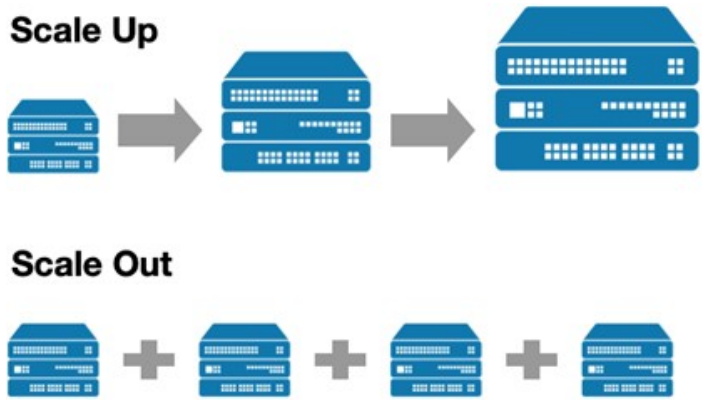

Figura 2. Gráfica demostrativa de escalabilidad horizontal y vertical a través de *hardware*. Fuente: Goller, 2019.

En la siguiente tabla se muestran las principales diferencias entre escalabilidad horizontal y vertical:

Aspecto	Escalabilidad horizontal	Escalabilidad vertical
Enfoque	Añadir más máquinas o nodos.	Mejorar recursos de una máquina existente.
Arquitectura requerida	Sistemas distribuidos o clústeres.	Sistemas monolíticos o centralizados.
Tolerancia a fallos	Alta: más nodos redundantes.	Baja: depende de un único servidor.
Límite físico	Escalabilidad casi ilimitada.	Limitada por el *hardware*.

Tabla 1. Diferencias entre escalabilidad horizontal y vertical. Fuente: elaboración propia.

¿Cuándo debemos usar cada tipo de escalabilidad?

- Escalabilidad horizontal:
  - Aplicaciones web con millones de usuarios simultáneos.
  - Sistemas de procesamiento masivo de datos *(big data)*.
  - Infraestructuras en la nube que buscan elasticidad.
- Escalabilidad vertical:
  - Bases de datos relacionales que no admiten distribución fácil.
  - Aplicaciones *legacy* donde no es posible rediseñar la arquitectura.
  - Sistemas con cargas moderadas y predecibles.

Tema 1. Procesamiento de datos escalable

Procesamiento batch vs. Procesamiento en tiempo real

El procesamiento de datos es una de las tareas clave en la informática moderna y puede realizarse de dos maneras principales: procesamiento *batch* y procesamiento en tiempo real. Ambas estrategias tienen aplicaciones y beneficios específicos, dependiendo de la naturaleza de los datos, la infraestructura disponible y los objetivos del negocio.

Procesamiento *batch*

El procesamiento *batch,* o por lotes, consiste en recolectar y almacenar datos en bloques o lotes que se procesan juntos en un momento posterior. Este tipo de procesamiento no requiere que los datos se analicen inmediatamente después de ser generados.

Sus **características** principales son:

- Procesa grandes volúmenes de datos en sesiones programadas.
- Generalmente, se realiza en segundo plano.
- Ideal para análisis que no requieren respuestas inmediatas.
- Suele implicar una mayor latencia en los resultados.

Entre sus **ventajas,** podemos destacar que:

- Permite procesar grandes cantidades de datos de manera eficiente.
- Conlleva una menor complejidad en comparación con el procesamiento en tiempo real.
- Es posible realizar análisis más complejos al procesar lotes completos.
- Tiene menores costos si se utilizan recursos en horarios de baja demanda.

Entre sus **desventajas,** enumeramos las siguientes:

- No es apto para tareas que requieren resultados inmediatos.
- Los datos no procesados permanecen inactivos hasta la ejecución del próximo lote.
- Posee un mayor riesgo de pérdida de relevancia si los datos cambian rápidamente.

Algunos **ejemplos** de uso de procesamiento *batch* son los siguientes:

- Procesamiento mensual de nóminas en empresas.
- Generación de informes financieros al final del día.
- Cálculo de recomendaciones *offline* en plataformas de *streaming* (por ejemplo, Netflix).

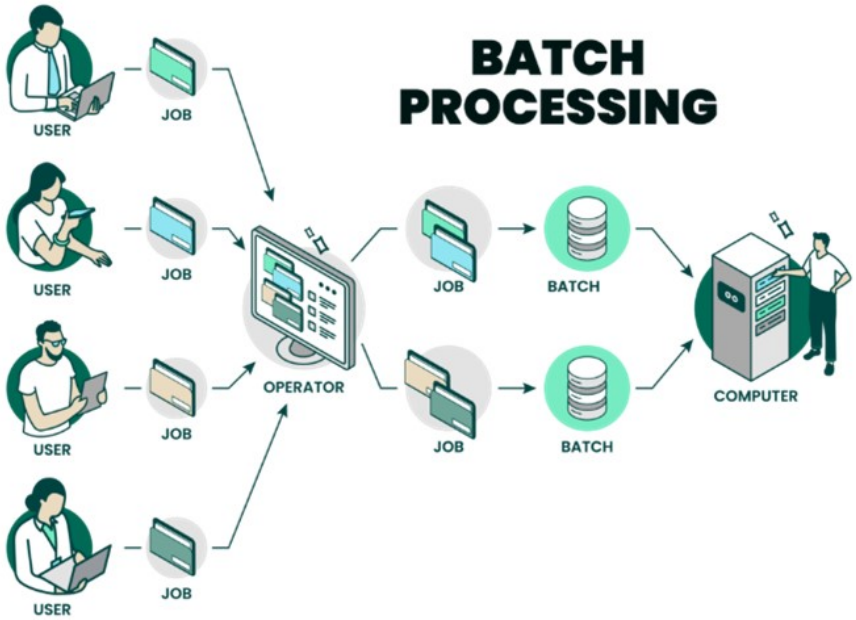

Figura 3. Gráfica de flujo de trabajo con procesamiento *batch.* Fuente: Fortra, s. f.

Procesamiento en tiempo real

El procesamiento en tiempo real implica analizar datos, casi instantáneamente, a medida que se generan. Es esencial en sistemas donde la toma de decisiones debe ser inmediata o en tiempo limitado.

¿Cuáles son sus **características** principales?

- Los datos se procesan de forma continua.
- Las respuestas son rápidas o inmediatas para eventos o transacciones.
- Implica una menor latencia en los resultados.
- Requiere infraestructuras diseñadas para manejar flujos constantes de datos.

Entre sus **ventajas** podemos destacar:

- Resultados inmediatos para una toma de decisiones ágil.
- Capacidad para detectar y responder a eventos críticos (por ejemplo, fraudes).
- Mejora de la experiencia del usuario, al ofrecer respuestas rápidas.
- Ideal para sistemas con datos que pierden valor rápidamente con el tiempo.

Respecto a sus **desventajas** o inconvenientes:

- Mayor complejidad en diseño y operación.
- Requiere una infraestructura más robusta y costosa.
- Dificultad para procesar grandes volúmenes de datos históricos.

Entre sus **ejemplos** de uso destacan:

- Detección de fraudes en transacciones bancarias.
- Monitorización de sistemas IoT, como sensores industriales.
- Procesamiento de interacciones en redes sociales en tiempo real.

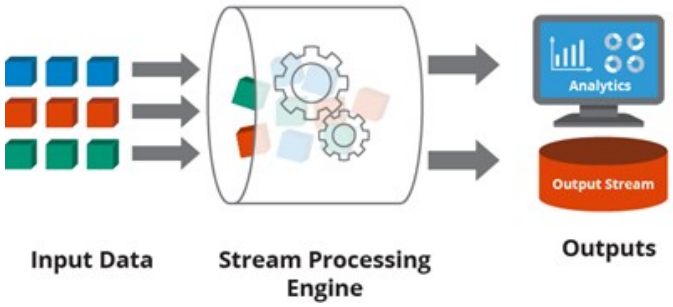

Figura 4. Gráfica de flujo de trabajo procesamiento *streaming.* Fuente: Hazelcast, s. f.

Principales diferencias entre procesamiento batch y en tiempo real

A continuación, se muestra una tabla resumen con las principales diferencias entre el procesamiento *batch* y *streaming* en los principales conceptos del procesamiento de datos.

Aspecto	Procesamiento *batch*	Procesamiento en tiempo real
Modo de operación	Procesa datos en bloques o lotes.	Procesa datos continuamente.
Latencia	Alta: los resultados se obtienen después de cierto tiempo.	Baja: respuestas inmediatas o casi inmediatas.
Complejidad	Baja: sistemas más simples.	Alta: sistemas más complejos.
Volumen de datos	Grandes cantidades de datos acumulados.	Flujos constantes de datos en pequeñas cantidades.

Tabla 2. Diferencias entre procesamiento *batch* y en tiempo real. Fuente: elaboración propia.

¿Cuándo usar cada tipo de procesamiento?

- Procesamiento *batch:*
  - Tareas programadas con grandes volúmenes de datos históricos.
  - Informes empresariales periódicos.
  - Análisis que no son sensibles al tiempo, como tendencias anuales.
- Procesamiento en tiempo real:
  - Detección de anomalías o fraudes.
  - Sistemas críticos que dependen de decisiones inmediatas (por ejemplo, monitorización médica).
  - Aplicaciones que mejoran la experiencia del usuario con respuestas rápidas.

Desafíos del procesamiento de datos escalable

El procesamiento de datos escalable implica manejar grandes volúmenes de información de manera eficiente, ya sea mediante escalabilidad horizontal (agregar más nodos) o vertical (mejorar el *hardware* de una máquina).

Aunque estas estrategias ofrecen flexibilidad y potencia, también presentan desafíos significativos en términos de diseño, implementación y mantenimiento.

Desafíos técnicos

- Volumen de datos creciente:
  - Problema: el crecimiento exponencial de los datos dificulta prever y dimensionar adecuadamente la infraestructura.
  - Impacto: los sistemas pueden volverse ineficientes o incapaces de manejar la carga.
  - Ejemplo: una empresa de *streaming* que maneja *terabytes* diarios de datos de usuarios puede enfrentar tiempos de respuesta lentos si no escala adecuadamente.

- Latencia y rendimiento:
  - Problema: mantener bajos tiempos de respuesta, mientras se procesan grandes volúmenes de datos.
  - Impacto: los usuarios finales pueden experimentar retrasos, afectando la experiencia del cliente.
  - Ejemplo: aplicaciones como las de *trading* financiero requieren latencias en el rango de milisegundos.

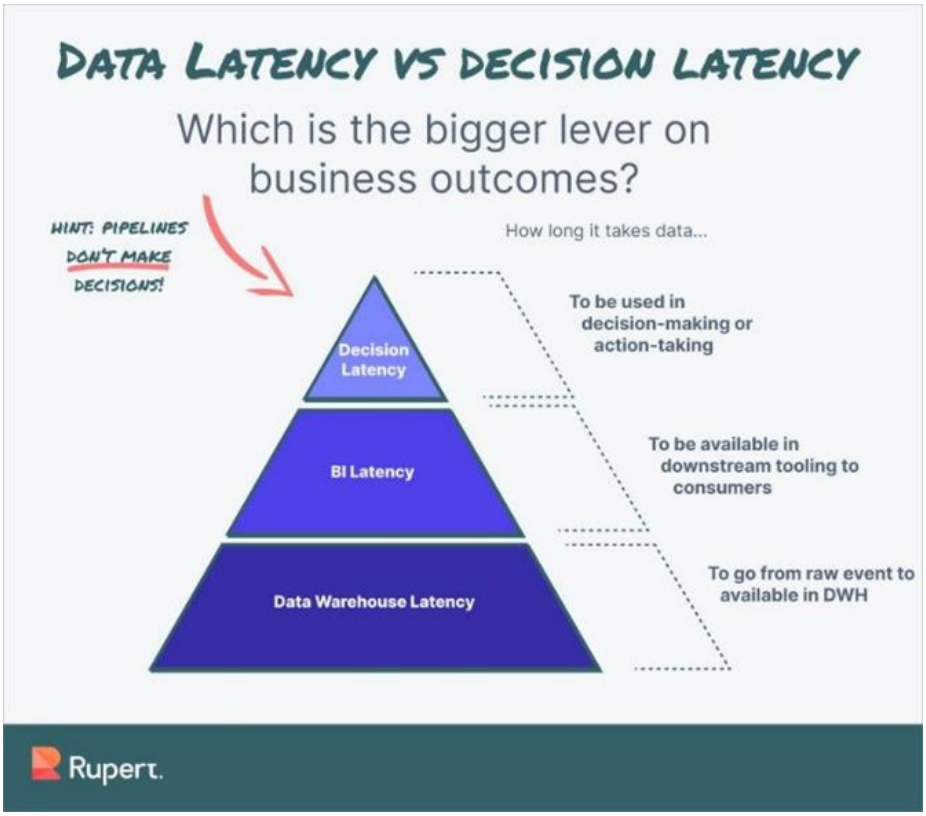

Figura 5. Gráfica que demuestra la importancia de la latencia en diferentes sistemas de datos. Fuente: Rupert, 2024.

- Consistencia de los datos:
  - Problema: garantizar la consistencia de los datos en sistemas distribuidos.
  - Impacto: las inconsistencias pueden llevar a decisiones empresariales incorrectas.
  - Ejemplo: una plataforma de comercio electrónico que no sincroniza correctamente su inventario en tiempo real puede vender productos agotados.

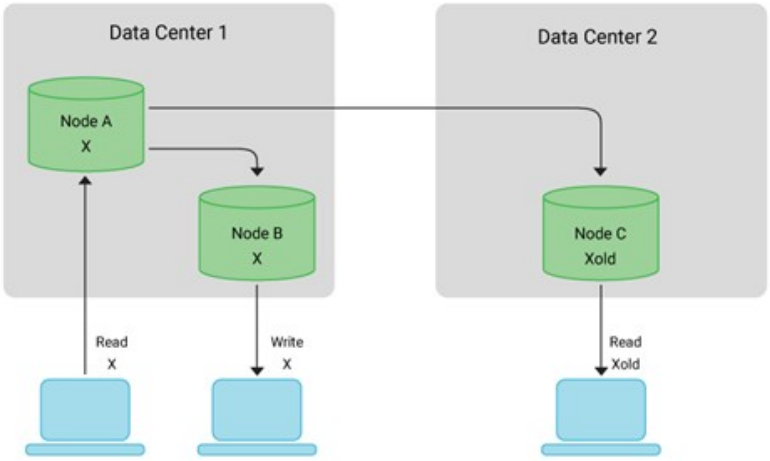

Figura 6. Diagrama de alta disponibilidad para consistencia datos. Scylla DB, s. f.

- Gestión de fallos:
  - Problema: garantizar que el sistema sea tolerante a fallos en arquitecturas distribuidas.
  - Impacto: un fallo en un nodo puede propagarse y afectar el sistema completo.
  - Ejemplo: un clúster de bases de datos debe manejar automáticamente el redireccionamiento de solicitudes en caso de fallo de un nodo.

- Escalabilidad de la infraestructura:
  - Problema: diseñar sistemas que puedan escalar fácilmente sin necesidad de reestructuraciones importantes.
  - Impacto: soluciones mal diseñadas pueden ser costosas y poco efectivas al escalar.
  - Ejemplo: migrar de un sistema monolítico a uno basado en microservicios para manejar cargas dinámicas.

Desafíos de implementación

- Complejidad del diseño:
  - Problema: diseñar arquitecturas escalables implica comprender en profundidad los patrones de uso y los requerimientos del sistema.
  - Impacto: implementaciones subóptimas pueden aumentar la latencia o los costos operativos.
  - Ejemplo: decidir entre escalabilidad horizontal o vertical según el caso de uso.
- Costos de infraestructura:
  - Problema: a medida que el sistema escala, los costos pueden crecer exponencialmente.
  - Impacto: presiones financieras significativas para *startups* o empresas en crecimiento.
  - Ejemplo: los costos de servicios en la nube, como AWS o Google Cloud, aumentan drásticamente con el procesamiento de datos en tiempo real.

- Integración de herramientas:
  - Problema: seleccionar y coordinar herramientas adecuadas para la infraestructura (por ejemplo: Kafka, Hadoop o Spark).
  - Impacto: herramientas mal integradas pueden reducir la eficiencia del sistema.
  - Ejemplo: configurar un *pipeline* de datos en tiempo real que integre Kafka y Spark Streaming.

Desafíos organizativos

- Falta de talento especializado:
  - Problema: la demanda de profesionales en *big data,* DevOps e ingenieros de datos supera la oferta.
  - Impacto: dificultades para diseñar, implementar y mantener sistemas escalables.
  - Ejemplo: la falta de expertos en Kubernetes puede retrasar la implementación de clústeres escalables.
- Gestión de cambios:
  - Problema: cambiar de sistemas *legacy* a arquitecturas modernas puede ser complejo y costoso.
  - Impacto: resistencias internas y problemas de compatibilidad tecnológica.
  - Ejemplo: migrar de una base de datos SQL centralizada a una base de datos NoSQL distribuida, como MongoDB.

- Seguridad y privacidad:
  - Problema: escalar un sistema incrementa las superficies de ataque y los riesgos de brechas de seguridad.
  - Impacto: pérdida de datos sensibles y confianza del cliente.
  - Ejemplo: proteger flujos de datos en tiempo real con cifrado y autenticación sólida.

Desafíos de operación y mantenimiento

- Monitoreo y observabilidad:
  - Problema: supervisar sistemas distribuidos y escalables puede ser complejo.
  - Impacto: problemas de rendimiento no detectados pueden crecer exponencialmente.
  - Ejemplo: implementar herramientas como Prometheus o Datadog para monitorear clústeres de datos.
- Actualizaciones y escalabilidad sin interrupciones:
  - Problema: realizar actualizaciones o escalar recursos sin interrumpir los servicios.
  - Impacto: experiencia negativa para el cliente durante tiempos de inactividad.
  - Ejemplo: escalar nodos en un clúster de Kubernetes, mientras se manejan solicitudes activas.

- Balance de carga:
  - Problema: distribuir uniformemente el tráfico entre nodos para evitar cuellos de botella.
  - Impacto: desempeño desigual y posibles sobrecargas en nodos específicos.
  - Ejemplo: configurar balanceadores de carga en una arquitectura basada en microservicios.

Estrategias para superar los desafíos

- Diseño eficiente de arquitectura:
  - Adoptar arquitecturas basadas en microservicios y APIs bien definidas.
  - Diseñar *pipelines* de datos resilientes.
- Automatización:
  - Usar herramientas de infraestructura como código (por ejemplo: Terraform, Ansible), para implementar recursos escalables.
- Optimización de costos:
  - Monitorizar el uso de recursos en la nube y adoptar estrategias de optimización (por ejemplo: instancias *spot*).

- Capacitación y desarrollo de talento:
  - Invertir en la formación continua de equipos en tecnologías clave como Spark, Kafka y Kubernetes.
- Implementación de monitoreo avanzado:
  - Integrar herramientas como Grafana, Prometheus o Elastic Stack para visibilidad en tiempo real.

Importancia de la nube en el procesamiento escalable

El avance de la tecnología ha hecho que los datos sean el centro de muchas actividades humanas, desde las compras en línea hasta las investigaciones científicas. Con este crecimiento explosivo en la generación de datos, surge una pregunta clave:

¿Cómo podemos procesar y analizar grandes volúmenes de datos de manera rápida, eficiente y accesible?

Aquí es donde entra en juego la computación en la nube. Este modelo revolucionario permite que cualquier persona, organización o empresa pueda acceder a potentes recursos tecnológicos sin necesidad de grandes inversiones iniciales. Vamos a explorar por qué la nube es tan importante para el procesamiento escalable.

¿Qué es la computación en la nube?

La computación en la nube es el acceso a recursos como servidores, almacenamiento, bases de datos, *software* y procesamiento de datos a través de Internet, en lugar de hacerlo desde un servidor físico local. Estos servicios son ofrecidos por proveedores como Amazon Web Services **(AWS)**, Google Cloud Platform **(GCP)** o **Microsoft Azure.**

En lugar de comprar y mantener un *hardware* costoso, la nube permite a los usuarios utilizar lo que necesitan y pagar únicamente por lo que consumen.

> Imagina que necesitas procesar datos para un proyecto escolar. En lugar de usar tu computadora personal, puedes acceder a una máquina virtual en la nube, con el poder de cien computadoras juntas.

Escalabilidad: la clave de la nube

La escalabilidad es la capacidad de aumentar o disminuir recursos según la demanda. La nube hace que esta tarea sea sencilla y eficiente, convirtiéndose en un aliado crucial para proyectos grandes o en constante crecimiento.

- Escalabilidad horizontal: puedes agregar más servidores en la nube para procesar datos.
- Escalabilidad vertical: puedes aumentar la capacidad (memoria, CPU) de un servidor existente con solo un clic.

Una tienda en línea experimenta un aumento de visitas durante el Black Friday. Con la nube, pueden aumentar rápidamente sus recursos para manejar el tráfico y, luego, reducirlos cuando la demanda regrese a niveles normales.

Figura 7. Infografía de ejemplos de beneficios del *cloud computing*. Fuente: nOps, 2024.

Beneficios de la nube para el procesamiento escalable

- Acceso a potentes recursos computacionales. La nube permite usar recursos de alto rendimiento que serían inaccesibles para muchas organizaciones si tuvieran que comprarlos.
- Ejemplo práctico: usando la nube, un científico puede analizar *terabytes* de datos genómicos en horas en lugar de en meses utilizando una computadora estándar.
- Costo eficiente. Solo pagas por lo que utilizas. Esto evita grandes gastos iniciales en infraestructura y reduce costos de mantenimiento.
- Ejemplo práctico: una *startup* puede iniciar sus operaciones con bajos costos, usando instancias pequeñas en la nube, y expandirse a medida que crecen sus necesidades.

- Flexibilidad y velocidad. La nube permite escalar recursos rápidamente, lo cual es esencial para aplicaciones en tiempo real o necesidades repentinas de mayor capacidad.

- Ejemplo práctico: un sistema de videollamadas puede aumentar recursos durante una conferencia global sin interrupciones.

- Acceso global. Los datos y aplicaciones alojados en la nube pueden ser accesibles desde cualquier lugar del mundo, lo que permite colaboración en tiempo real.

  - Ejemplo práctico: un equipo de desarrollo en distintos países puede trabajar en un mismo proyecto, usando herramientas en la nube como GitHub o Google Drive.

- Resiliencia y seguridad. Los proveedores de la nube cuentan con mecanismos avanzados para proteger los datos y garantizar que los servicios estén siempre disponibles, incluso ante fallos de *hardware.*

  - Ejemplo práctico: un banco puede confiar en la nube para garantizar que sus operaciones nunca se detengan, incluso si ocurre un fallo en un servidor.

Casos de uso reales

- Netflix: Netflix usa la nube para procesar enormes cantidades de datos sobre las preferencias de sus usuarios. Esto permite ofrecer recomendaciones personalizadas y transmitir millones de horas de contenido a nivel mundial sin interrupciones.
- NASA: la NASA utiliza la nube para analizar datos espaciales. Gracias a la escalabilidad, pueden procesar rápidamente imágenes satelitales o datos de misiones en otros planetas.
- *Startups* de inteligencia artificial: pequeñas empresas pueden entrenar modelos complejos de IA usando la nube, sin necesidad de invertir en servidores propios.

Desafíos superados gracias a la nube

- **Procesar datos masivos:** la nube permite manejar desde *gigabytes* hasta *petabytes* de información con facilidad.
- **Adaptarse a la demanda:** los recursos se ajustan automáticamente, evitando tiempos de inactividad.
- **Reducir barreras tecnológicas:** acceso inmediato a tecnología de punta, incluso para usuarios principiantes.

1.2 Procesamiento de datos escalable con Apache Kafka

Introducción

En un mundo cada vez más conectado y digitalizado, la capacidad de procesar grandes volúmenes de datos en tiempo real se ha convertido en una necesidad imperiosa para las empresas. Apache Kafka emerge como una solución potente y escalable para abordar este desafío. En este módulo, exploraremos los fundamentos de Kafka y su papel en la construcción de sistemas de datos modernos.

Definición

Apache Kafka es una plataforma distribuida de transmisión de eventos de código abierto, diseñada para publicar, suscribirse, almacenar y procesar flujos de registros en tiempo real. Imagina Kafka como una tubería gigante por la que fluyen continuamente datos, permitiendo que múltiples aplicaciones se conecten y consuman esta información de manera eficiente.

Puedes encontrar la documentación oficial en el siguiente enlace Apache Kafka:

https://kafka.apache.org/

Historia y evolución

Kafka nació en LinkedIn como una solución interna para gestionar el gran volumen de mensajes generados por su plataforma. Dada su eficacia, se liberó como *software* de código abierto en 2011 y rápidamente ganó popularidad en la comunidad de desarrolladores. Desde entonces, Kafka ha evolucionado significativamente, incorporando nuevas características y optimizaciones para satisfacer las demandas de las empresas más exigentes.

Características clave

- Escalabilidad: Kafka puede manejar flujos de datos masivos y crecer horizontalmente para adaptarse a las necesidades cambiantes.
- Durabilidad: los mensajes se almacenan de forma persistente en discos, lo que garantiza que no se pierdan datos en caso de fallos.
- Alta disponibilidad: Kafka está diseñado para ser altamente disponible, con mecanismos de replicación y tolerancia a fallos.
- Baja latencia: permite el procesamiento de eventos en tiempo real con baja latencia.
- Orden de los mensajes: garantiza que los mensajes se procesen en el orden en el que fueron producidos.
- Flexibilidad: se integra fácilmente con otras tecnologías y herramientas del ecosistema de *big data.*

Figura 8. Diez principales características de Apache Kafka. Fuente: Way to easy learn, 2024.

Casos de uso iniciales

Los primeros adoptadores de Kafka encontraron aplicaciones en diversos sectores:

- Sistemas de recomendación: Netflix utilizó Kafka para construir un sistema de recomendación en tiempo real, personalizando las sugerencias de películas y series para cada usuario.
- Análisis de *logs:* muchas empresas utilizan Kafka para recopilar y analizar *logs* de aplicaciones y servidores, facilitando la detección de problemas y la optimización del rendimiento.
- Mensajería entre microservicios: Kafka se convirtió en una herramienta popular para desacoplar microservicios y permitir una comunicación asíncrona entre ellos.
- IoT: Kafka es ideal para procesar los datos generados por dispositivos IoT en tiempo real, permitiendo la toma de decisiones basadas en datos en tiempo real.

¿Por qué aprender Kafka?

- Demanda en el mercado laboral: la demanda de profesionales con conocimientos en Kafka está en constante crecimiento.
- Solución escalable y robusta: Kafka es una tecnología probada y confiable para construir sistemas de datos en tiempo real.
- Versatilidad: se puede aplicar a una amplia gama de casos de uso.
- Comunidad activa: cuenta con una gran comunidad de desarrolladores que contribuyen a su evolución y ofrecen soporte.

Arquitectura de Apache Kafka: el corazón de los sistemas de streaming

Apache Kafka es una plataforma distribuida de transmisión de datos que permite gestionar grandes volúmenes de datos en tiempo real. Para comprender cómo Kafka puede manejar este procesamiento, es esencial entender su arquitectura y cómo están organizados sus componentes. A continuación, conoceremos los conceptos clave y la arquitectura de Kafka de manera clara y directa.

Componentes principales de Apache Kafka

Apache Kafka se basa en varios componentes fundamentales que trabajan juntos para permitir el procesamiento y la transmisión de datos en tiempo real. Estos componentes son:

- ***Brokers*** **(servidores Kafka):**
- Función: los *brokers* son los servidores que gestionan el almacenamiento de los datos y las solicitudes de lectura y escritura. Un clúster de Kafka se compone de varios *brokers* que trabajan juntos para manejar los datos de manera distribuida.
- Características:
  - Cada *broker* gestiona particiones de uno o más *topics.* Un *broker* puede tener múltiples particiones de diferentes *topics.*
  - Los *brokers* se comunican entre sí para asegurarse de que los datos estén replicados y disponibles, incluso si algunos *brokers* fallan.
  - Un *broker* puede tener información sobre el estado de las particiones que maneja, como los mensajes más recientes.
- Ejemplo: un clúster de Kafka podría tener varios brokers, cada uno encargado de almacenar y gestionar las particiones de los topics, garantizando la disponibilidad y redundancia de los datos.