🔥 Encendiendo la chispa 🔥
Desarrollo con Data Engineering en Microsoft Fabric

ADN Fabric – 21/01/2026

Hello! My name is l2aFa

- Amando y odiando los datos desde 2010
- Me encanta aprender cosas nuevas, llevar la tecnología más allá y colaborar en proyectos que ilusionen y aporten valor a todos los niveles
- Apasionado de la tecnología, videojuegos, cómics y memes por igual
- Microsoft Fabric MVP

📖 Agenda

- 👨‍🏫 Data Engineering 101
- 🔥 Encendiendo la chispa
- 🥕 ¿Qué hay de nuevo viejo?
- 🛠️ Tips and tricks

Data Engineering 101

¿Qué es la ingeniería de datos?

Consiste en construir los sistemas que transforman datos desordenados en información limpia y lista para ser analizada y consumida

¿Y en Fabric?

Data Engineering

Cree un lago de datos y operacionalice el flujo de trabajo para crear, transformar y compartir el patrimonio de datos.

Acerca de

Tipos de elemento

- Lakehouse
- Bloc de notas
- Entorno
- Definición de trabajo d...
- Funciones de datos de...
- API para GraphQL

Áreas de trabajo disponibles

Esta carga de trabajo se puede usar en todas las áreas de trabajo

Soporte del anunciante

Documentación

Ayuda

Información general

Descripción

Publicador: Microsoft

La ingeniería de datos permite diseñar, compilar y mantener infraestructuras y sistemas que su organización puede usar para recopilar, almacenar, procesar y analizar grandes volúmenes de datos.

Comenzar

Explorar un ejemplo
Usar un ejemplo para aprender

Seleccionar

¿Qué es un lakehouse?
Introducción a la ingeniería de datos

Abrir

Obtención de la experiencia de datos en ...
Introducción a la ingeniería de datos

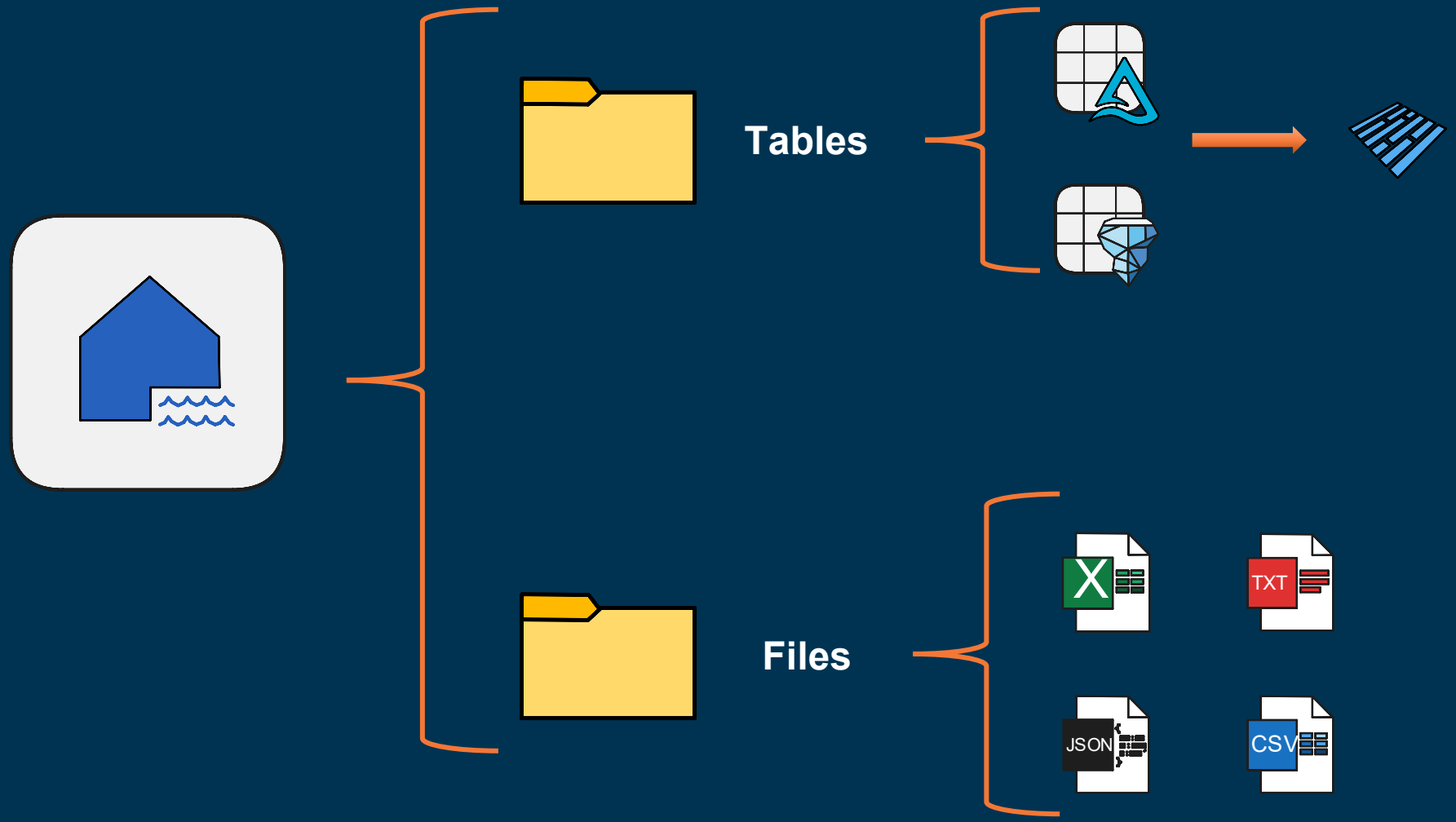
Abrir

Introducción a las definiciones de trabajo...
Introducción a la ingeniería de datos

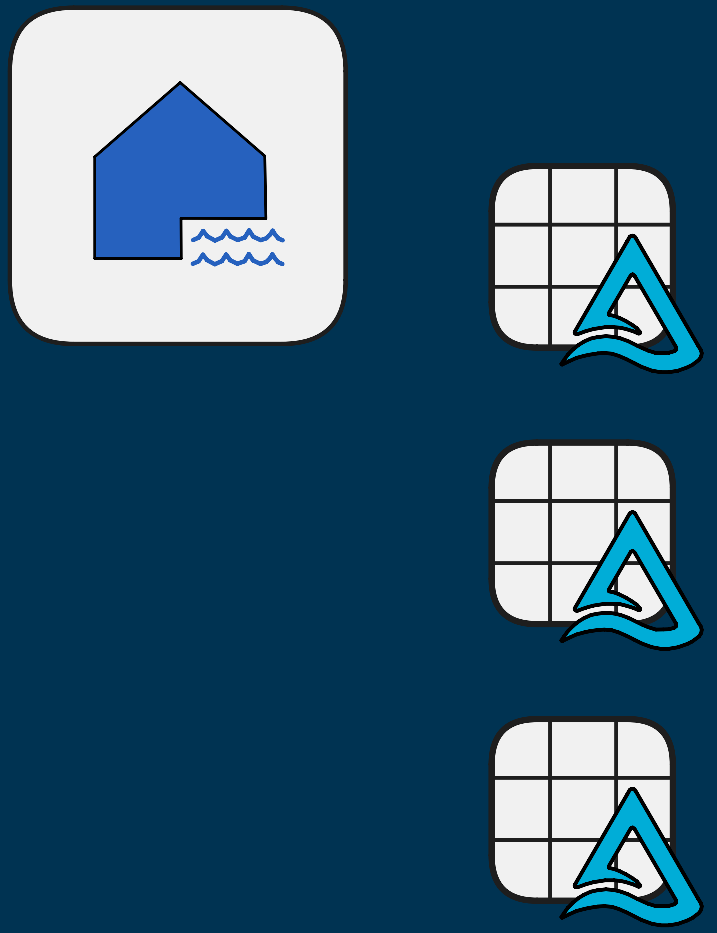
Desarrollar y ejecutar cuadernos
Introducción a la ingeniería de datos

Cómo usar NotebookUtils
Introducción a la ingeniería de datos

En algún sitio hay que guardar las cosas



Dos por el precio de uno



¡Ojo con los timings de sincronización!

Cada escenario tiene su herramienta

- Enfoque mayormente pro code
- Posibilidad de personalización
- Contamos con librerías y útiles propios
  - Semantic link
  - NotebookUtils
- Dos vertientes

Tigre y dragón

- Arranque inmediato
- Datos pequeños/medianos
- Acceso limitado a características
- Soporte parcial sobre Delta Lake
- Idóneo para tareas cortas y prototipos/pruebas de características
- Menor coste de recursos

- Arranque más lento*
- Datos enormes
- Acceso a todas las características
- Soporte total sobre Delta Lake
- Más adecuado para cargas de trabajo pesadas
- Mayor coste de recursos

Como todo en Fabric, ¡la unión hace la fuerza!

🔥 Encendiendo la chispa

¿Qué es Apache Spark?

- Motor de análisis unificado diseñado para el procesamiento de datos a gran escala
  - Realiza el procesamiento en memoria
  - Divide tareas de datos gigantescas en trozos y los procesa en paralelo
  - Dispone de APIs para diferentes lenguajes
  - Open-source*
  - Completamente todo terreno

El secreto está en la masa

- Los procesos en Spark:
  - Se dividen físicamente en trozos, denominados particiones
    - 1 partición = 1 tarea
  - Siguen un patrón de orquesta, contamos con:
    - 1 Driver
    - N Executors
- Spark es perezoso por naturaleza

El secreto está en la masa

- Los pro
  - Se di
    - 1
  - Sigue
    - 1
    - N
- Spark e

```python
# 1. TRANSFORMACIÓN (Lazy): Spark solo "toma nota"
# Aquí NO se leen datos, solo se verifican metadatos (nombres de columnas).
df = spark.read.parquet("ventas_gigantes.parquet")

# 2. TRANSFORMACIÓN (Lazy): Spark añade un paso al plan
# Todavía NO se ha procesado ni una sola fila.
df_filtrado = df.filter(df["pais"] == "España")

# 3. TRANSFORMACIÓN (Lazy): Otro paso al plan
# Seguimos sin usar CPU para mover datos.
df_final = df_filtrado.select("producto", "importe")

# ------------------------------------------------------
# 4. ACCIÓN (Eager): ¡El detonante!
# Aquí Spark mira todo el plan anterior, lo optimiza y EJECUTA.
df_final.show()
```

Pero cuidado no nos quememos

```python
# Definimos el plan (Lazy) -> Coste: 0 segundos
df = spark.read.parquet("datos_masivos.parquet").filter(...)

# 1. ACCIÓN: Count
# Spark lee el archivo -> filtra -> cuenta.
print(df.count())

# 2. ACCIÓN: Show
# ¡Spark NO recuerda el resultado anterior!
# Vuelve a leer el archivo -> vuelve a filtrar -> muestra.
df.show()

# 3. ACCIÓN: Write
# ¡Otra vez! Lee -> filtra -> escribe.
df.write.mode("overwrite").save("destino")
```

Pero cuidado no nos quememos

```python
# --- STAGE 1 (Todo esto ocurre en paralelo y memoria local) ---
df = spark.read.parquet("datos.parquet")
df_limpio = df.filter(df["edad"] > 18)   # Narrow (Local)
df_final = df_limpio.select("nombre", "ciudad") # Narrow (Local)

# --- SHUFFLE (La barrera de red) ---
# Aquí Spark detiene el proceso local, intercambia datos y crea la STAGE 2
df_agrupado = df_final.groupBy("ciudad").count() # Wide (Requiere mover datos)

df_agrupado.show() # Acción que dispara todo
```

 ¿Qué hay de nuevo, viejo?

Con sabor propio

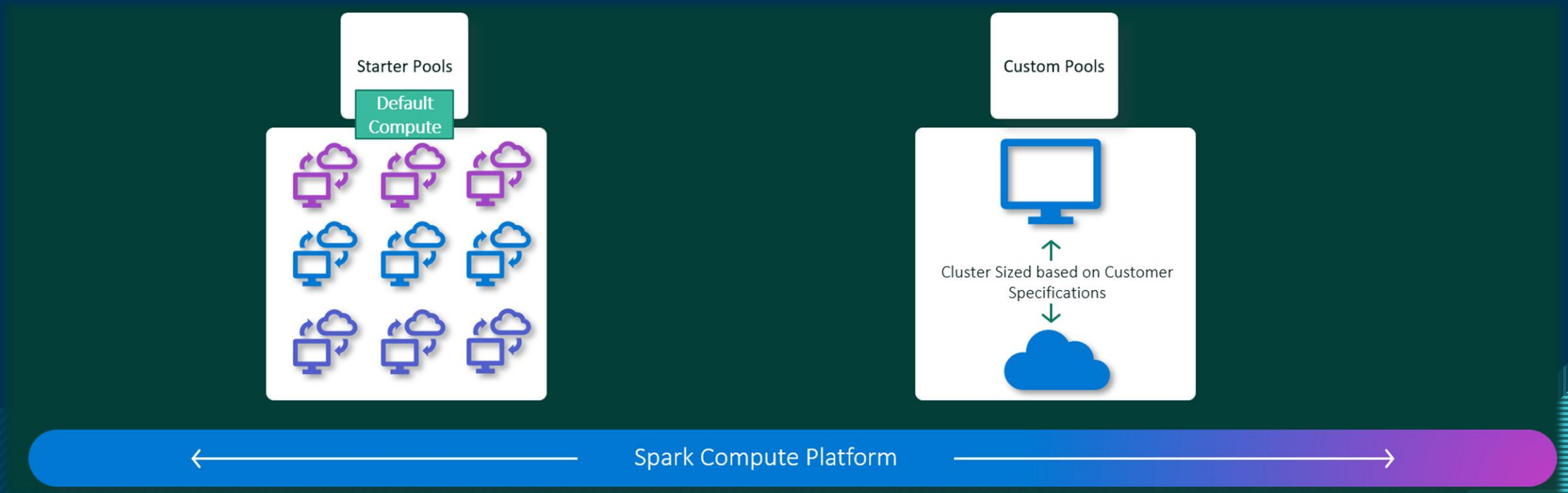
- Spark en Fabric se basa en la versión open-source:
  - Ya instalado y configurado, permite dimensionar y ajustar
    - Acceso a diferentes runtimes
  - Totalmente integrado con OneLake
  - Incluye personalizaciones y mejoras únicas
    - V-ORDER
    - Adaptative target file size
    - Native Execution Engine

Única en su especie

- Admite trabajo en varios lenguajes y formatos
- Aislamiento de costes y modo de facturación único
- Dimensionamiento y configuración del “músculo”
  - Límites a nivel de capacidad
  - Diferentes versiones
- Personalización y ajustes sin parangón

¿Qué hay bajo el capo?

- Grupos de recursos definidos para la ejecución de tareas
  - Tamaños predefinidos (de S a XXL)
  - Ajustes de escalado y asignación dinámicos
  - Configuración de duración de sesión



Hasta el infinito, pero no más allá

- Tenemos un número máximo de vCores por capacidad
  - Los pools definidos utilizan esos núcleos para la ejecución de tareas
  - Se evalúan los núcleos disponibles previamente a ejecutar nada
  - Disponemos de un sistema de cola para ejecución
- Es posible usar más cores de los disponibles gracias al bursting
  - Controlable a voluntad
- Resulta indispensable
  - Programar bien los procesos y cargas de trabajo
  - Dimensionar correctamente los clústers

Hagamos porque todo salga bien

Job Admission (Optimistic Approach)

Jobs are admitted with 1 node and scale up requests go through Spark Job Admission to approve or reject based on cores available

- By enabling autoscale, F32 capacity could support running 24 concurrent jobs (192 Total Cores/8 Core per job)
- Job scale up is approved/ rejected based on cores available in a fair manner
- Scale up or new job admission exceeding the available cores are queued or throttled

De todas las tallas y colores

SKU Fabric	vCores Spark	Bursting (3-5x)	Límite de cola
F2	4	20	4
F4	8	24	4
F8	16	48	8
F16	32	96	16
F32	64	192	32
F64	128	384	64
F128	256	768	128
F256	512	1536	256
F512	1024	3072	512
F1024	2048	6144	1024
F2048	4096	12288	2048
T1	128	128	N/A

🛠️ Tips and tricks

Recetas para el éxito

- Filtrar, filtrar, filtrar y también filtrar
- Utiliza la Delta, Luke
- Indicar es mejor que adivinar
- Uso eficiente de recursos
  - Modo de alta concurrencia
  - Limite de sesión & dimensionamiento
- Ajustar las propiedades según cada capa/propósito

Pezqueñines no gracias

- No olvidemos que SELECT * sigue matando gatitos
- El tamaño sí importa
- Más no es necesariamente mejor
- Las serpientes son traicioneras
- Cuidado con atragantarnos
- Ojo con nuestros pasos y la huella que dejamos

¡Esto no es todo amigos!

- Probar, testear y volver a empezar
  - Dimensionamientos de clúster
  - Nº de particiones
- Mecanismos de tunning adicional
  - Z-ORDER
  - Liquid Clustering
- Aprovechemos “el jugar en casa”

¡Muchas gracias!