

Azure Data Engineer Associate Certification Guide

A hands-on reference guide to developing your data engineering skills and preparing for the DP-203 exam

Newton Alex



Capítulo 11: Gestión de batches y pipelines	3
11.1. Requisitos técnicos.....	3
11.2. Triggering batches.....	4
11.3. Manejo de cargas Batch fallidas.....	7
11.3.1. Errores de pool.....	7
11.3.2. Errores de nodo.....	7
11.3.3. Job errors.....	8
11.3.4. Errores en las tareas.....	9
11.4. Validación de cargas por lotes	10
11.5. Programación de data pipelines en Data Factory/Synapse pipelines.....	13
11.6. Gestión de data pipelines en Data Factory/Synapse pipelines	15
11.6.1. Integration runtimes	15
11.6.2. Monitorización de ADF.....	16
11.7. Gestión de Spark jobs en un pipeline.....	18
11.8. Implementación del control de versiones para los pipeline artifacts.....	22
11.8.1. Configurando el control de fuentes en ADF	22
11.8.2. Integración con Azure DevOps	23
11.8.3. Integración con GitHub	25
Resumen.....	27

Capítulo 11: Gestión de batches y pipelines

Bienvenido al capítulo 11. Este es uno de los capítulos más pequeños y fáciles de leer. En este capítulo, nos centraremos en cuatro grandes categorías: la activación de trabajos batch, la gestión de fallos en trabajos batch, la gestión de pipelines y la configuración del control de versiones para nuestros pipelines. Una vez que hayas completado este capítulo, deberías ser capaz de configurar y gestionar cómodamente los pipelines Batch utilizando Azure Batch, Azure Data Factory (ADF) o Synapse pipelines.

En este capítulo, cubriremos los siguientes temas:

- Triggering Batches
- Manejo de cargas Batch fallidas
- Validación de cargas de Batch
- Gestión de data pipelines en Data Factory/Synapse pipelines
- Programación de data pipelines en pipelines de Data Factory/Synapse
- Gestión de trabajos Spark en un pipeline
- Implementación del control de versiones para los artefactos de los pipelines

11.1. Requisitos técnicos

Para este capítulo, necesitarás lo siguiente

- Una cuenta de Azure (gratuita o de pago)
- Un área de trabajo de Synapse activa
- Un área de trabajo de Azure Data Factory activa

¡Empecemos!

11.2. Triggering batches

Aprendimos sobre Azure Batch en el Capítulo 9, Diseño y desarrollo de una solución de procesamiento por lotes, en la sección Introducción a Azure Batch. En esta sección, aprenderemos a desencadenar esos trabajos de Batch utilizando Azure Functions. Azure Functions es un servicio serverless proporcionado por Azure que ayuda a construir aplicaciones en la nube con un código mínimo, sin que tengas que preocuparte de alojar y mantener la tecnología que ejecuta el código. Azure se encarga de todas las complejidades del alojamiento, como los despliegues, las actualizaciones, los parches de seguridad, el escalado y mucho más. Aunque el nombre diga serverless, tiene servidores que se ejecutan en segundo plano. Sólo significa que no tienes que mantener esos servidores - Azure lo hace por ti.

Para nuestro requisito de desencadenar un trabajo Batch, vamos a utilizar la funcionalidad Trigger de Azure Functions. Un Trigger define cuándo y cómo invocar una función de Azure. Azure Functions admite una gran variedad de triggers, como timer trigger, HTTP trigger, Azure Blob trigger, Azure EventHub trigger, etc.

NOTA

Una función de Azure sólo puede tener un activador asociado. Si necesitas varias formas de activar una función, tendrás que definir diferentes funciones para ella.

Para nuestro ejemplo, definiremos un trigger blob y definiremos una función Azure que pueda llamar a nuestro trabajo Batch:

1. Desde el portal de Azure, busca Function App y crea una nueva aplicación de función. Una vez creada, haz clic en el botón + Crear para crear una nueva función, como se muestra en la siguiente captura de pantalla:

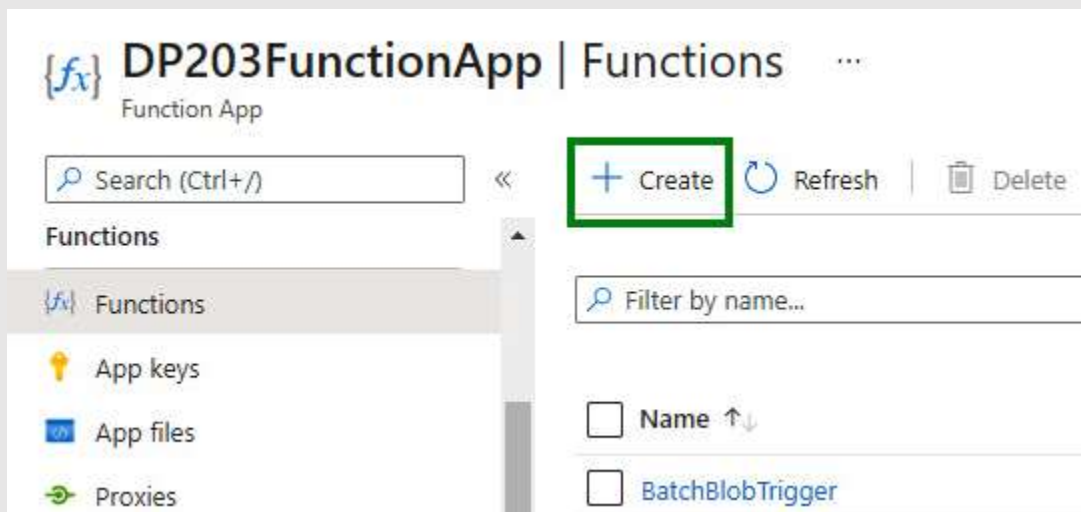


Figura 11.1 - Creación de una nueva Azure Function App

2. La pantalla de creación de la función tendrá el aspecto que se muestra en la siguiente imagen. Seleccione la opción de activación de Azure Blob Storage en la sección Select a template y configure los detalles necesarios, como el nombre de la función (New Function), Path, etc:

The screenshot shows the 'Create function' interface in the Azure portal. It includes sections for 'Select development environment', 'Select a template', and 'Template details'. The 'Azure Blob Storage trigger' is highlighted in the template list. The 'Template details' section contains input fields for 'New Function', 'Path', and 'Storage account connection'.

Select a template	
HTTP trigger	A function that will be run whenever it receives an HTTP request, responding based on data in the body or query string
Timer trigger	A function that will be run on a specified schedule
Azure Queue Storage trigger	A function that will be run whenever a message is added to a specified Azure Storage queue
Azure Service Bus Queue trigger	A function that will be run whenever a message is added to a specified Service Bus queue
Azure Service Bus Topic trigger	A function that will be run whenever a message is added to the specified Service Bus topic
Azure Blob Storage trigger	A function that will be run whenever a blob is added to a specified container
Azure Event Hub trigger	A function that will be run whenever an event hub receives a new event
Azure Cosmos DB trigger	A function that will be run whenever documents change in a document collection

Template details	
We need more information to create the Azure Blob Storage trigger function. Learn more	
New Function *	<input type="text" value="BlobTrigger1"/>
Path * ⓘ	<input type="text" value="samples-workitems/{name}"/>
Storage account connection * ⓘ	<input type="text" value="AzureWebJobsStorage"/> New

Figura 11.2 - Creación de un trigger de Azure Blob Storage

3. Haga clic en el botón Crear para crear la función que utilizará como trigger.
4. Ahora que tenemos el trigger, necesitamos definir qué hacer cuando el trigger se dispare. Vaya a la pestaña Code + Test y añada allí su lógica de negocio. El código de la lógica de negocio puede estar en C#, Java, Python o PowerShell:

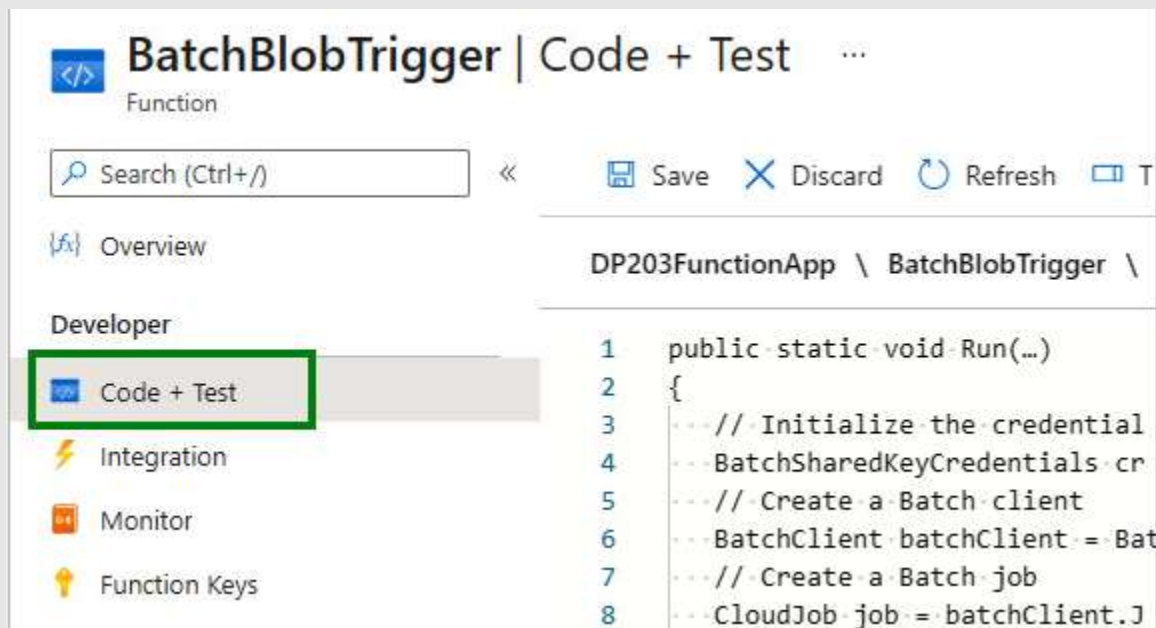


Figura 11.3 - Introducción de código para la función Trigger

En nuestro ejemplo, utilizaremos fragmentos de código C# para demostrar el flujo. Debemos añadir la lógica de negocio dentro de un método Run(). Aquí hay un bloque de referencia de código de ejemplo:

```
public static void Run(...)
{
    // Initialize the credentials
    BatchSharedKeyCredentials cred = new BatchSharedKeyCredentials(...);
    // Create a Batch client
    BatchClient batchClient = BatchClient.Open(cred)
    // Create a Batch job
    CloudJob job = batchClient.JobOperations.GetJob(<JOB_ID>);
    // Add your business logic
    // Add tasks to the job.
    CloudTask task = new CloudTask(<TASK_ID>, <COMMAND>);
    batchClient.JobOperations.AddTask(<JOB_ID>, task);
}
```

Ahora, cada vez que añadamos o actualicemos un archivo en el blob storage en el campo Path, se llamará a la función anterior, que iniciará un trabajo Batch.

Puedes consultar el ejemplo completo de trabajo en C# en el repositorio GitHub de este libro: <https://github.com/Azure-Samples/batch-functions-tutorial>.

A continuación, vamos a aprender a manejar los trabajos fallidos en Azure Batch.

11.3. Manejo de cargas Batch fallidas

Un trabajo de Azure Batch puede fallar debido a cuatro tipos de errores:

- Pool errors
- Node errors
- Job errors
- Task errors

Veamos algunos de los errores más comunes en cada grupo y las formas de manejarlos.

11.3.1. Errores de pool

Los errores de pool ocurren principalmente debido a problemas de infraestructura, problemas de cuota o problemas de tiempo de espera. Estos son algunos ejemplos de errores de pool:

- **Cuota insuficiente:** Si no hay suficiente cuota para su cuenta de Batch, la creación del pool podría fallar. La solución es solicitar un aumento de la cuota. Puede comprobar los límites de cuota aquí: <https://docs.microsoft.com/en-us/azure/batch/batch-quota-limit>.
- **Recursos insuficientes en su VNet:** Si su red virtual (VNet) no tiene suficientes recursos, como direcciones IP disponibles, grupos de seguridad de red (NSG), máquinas virtuales, etc., el proceso de creación del pool puede fallar. La mitigación es buscar estos errores y solicitar una mayor asignación de recursos o moverse a una VNet diferente que tenga suficiente capacidad.
- **Tiempos de espera cortos:** Algunas de las operaciones, como el redimensionamiento del pool, pueden tardar más de 15 minutos, que es el valor por defecto que establece Azure Batch. En estos casos, aumentar el tiempo de espera utilizando la siguiente API podría ayudar:

```
POST {batchUrl}/pools?timeout={timeout}&api-version=2021-06-01.14.0
```

A continuación, veamos algunos de los errores de nodo más comunes.

11.3.2. Errores de nodo

Los errores de nodo pueden deberse a problemas de hardware o a fallos en las actividades de configuración del trabajo, como los scripts de inicio, los scripts de descarga, etc. Estos son algunos ejemplos de errores de nodo:

- **Fallos en la tarea de inicio:** Si ha configurado una tarea de inicio para su trabajo y ésta falla, el nodo queda inutilizado. Puede buscar los errores en los campos TaskExecutionResult y TaskFailureInformation en la respuesta de la siguiente API:

```
GET {batchUrl}/pools/{poolId}/nodes/{nodeId}?api-version=2021-06-01.14.0
```

Una vez identificado el error, tendrá que tomar medidas correctivas, como arreglar los scripts de inicio del usuario.

- **Fallos en la descarga de aplicaciones:** Si Batch no puede descargar su paquete de aplicaciones, entonces lanzará un error de nodo. La propiedad `ComputeNodeError` en la respuesta GET API mostrada anteriormente listará dichos fallos de descarga de aplicaciones. Compruebe los permisos de los archivos, las ubicaciones correctas, etc. para solucionar estos problemas.
- **Nodos que entran en mal estado:** Esto puede ocurrir debido a fallos en la infraestructura, problemas de red, malas actualizaciones del sistema operativo/seguridad, el disco está lleno, etc.

En estos casos, la propiedad `ComputeNodeError` en la respuesta GET API indicará el error. Basándose en el error, tendrá que tomar medidas correctivas, como hacer girar nuevos nodos, arreglar problemas de red, limpiar el disco, etc.

Puedes aprender más sobre el manejo de errores de pool y nodos aquí: <https://docs.microsoft.com/en-us/azure/batch/batch-pool-node-error-checking>.

A continuación, veamos los errores de trabajo.

11.3.3. Job errors

Los jobs pueden fallar debido a múltiples razones. Veamos algunas de ellas:

- Las restricciones de los jobs, como la configuración de `maxWallClockTime`, pueden provocar fallos en los trabajos.
- Errores en la preparación del job o en las tareas de liberación del job. Estas son tareas que se ejecutan antes de la primera tarea y después de la última tarea del job. Cualquier fallo en ellas puede hacer que el job falle.

En todos estos casos, las propiedades `JobExecutionInformation` y `JobSchedulingError` en la respuesta de la siguiente API contendrán los detalles del error:

```
GET {batchUrl}/jobs/{jobId}?api-version=2021-06-01.14.0
```

Basándonos en el error real identificado por la respuesta de la API anterior, deberíamos ser capaces de solucionar el problema. Ahora, veamos los errores de las tareas.

11.3.4. Errores en las tareas

Los fallos de las tareas pueden ocurrir en los siguientes casos:

- El comando de tareas falla y devuelve un código de salida distinto de cero.
- Problemas de descarga de archivos de recursos, por ejemplo, si la tarea no puede descargar los archivos de recursos de las carpetas de origen.
- Problemas de carga de archivos de salida, por ejemplo, si la tarea no puede cargar el archivo de salida en las carpetas de destino.

En todos estos casos, puede comprobar la propiedad `TaskExecutionInformation` en la respuesta de la siguiente API REST para obtener los detalles del error de la tarea:

```
GET {batchUrl}/jobs/{jobId}/tasks/{taskId}?api-version=2021-06-01.14.0
```

Para los problemas de salida de archivos, las tareas por lotes también escriben los archivos `Fileuploadout.txt` y `fileuploaderr.txt` en el nodo. Estos archivos también pueden proporcionar información valiosa sobre los errores.

Puede aprender más sobre el manejo de errores de trabajos y tareas aquí: <https://docs.microsoft.com/en-us/azure/batch/batch-job-task-error-checking>.

En la siguiente sección, veremos las opciones disponibles para validar las cargas de Azure Batch.

11.4. Validación de cargas por lotes

Los trabajos por lotes suelen ejecutarse como parte de Azure Data Factory (ADF). ADF proporciona funcionalidades para validar el resultado de los jobs. Aprendamos a utilizar la actividad de Validación en ADF para comprobar la corrección de las cargas por lotes:

1. La actividad de Validación de ADF puede ser utilizada para comprobar la existencia de un archivo antes de proceder con el resto de las actividades en el pipeline. El pipeline de validación tendrá un aspecto similar al siguiente:

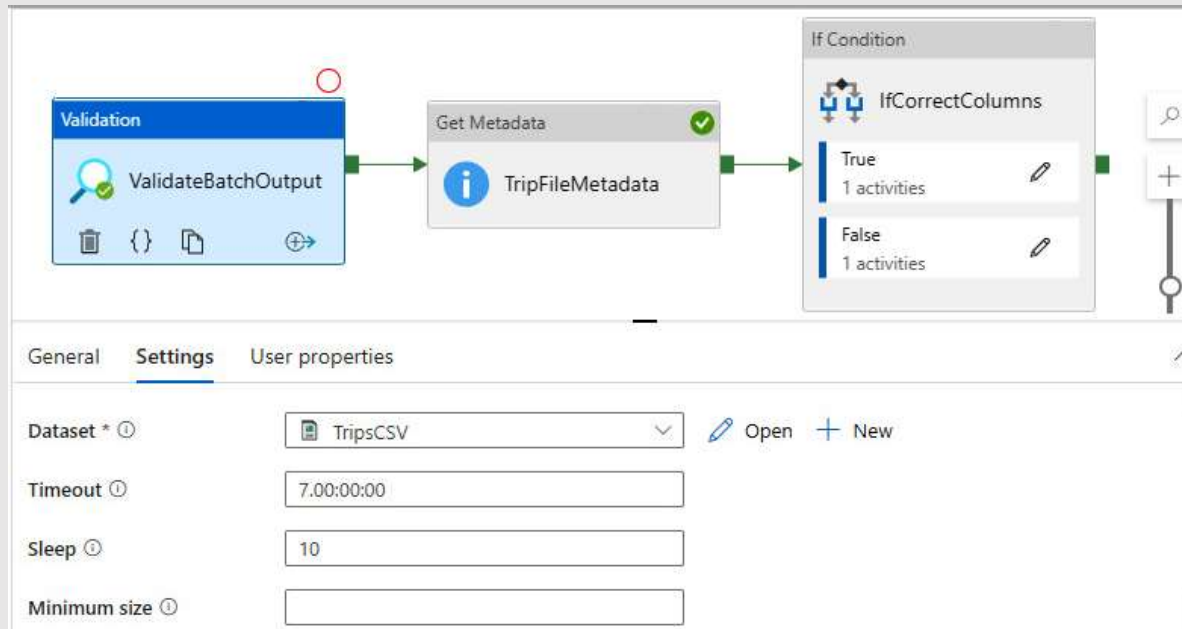


Figura 11.4 - Actividad de validación del ADF

2. Una vez que hayamos validado que los archivos existen, podemos utilizar la actividad Obtener Metadatos para obtener más información sobre los archivos de salida. En la siguiente captura de pantalla, obtenemos el recuento de columnas, que comprobaremos más tarde utilizando una actividad If Condition para decidir si los archivos de salida son buenos:

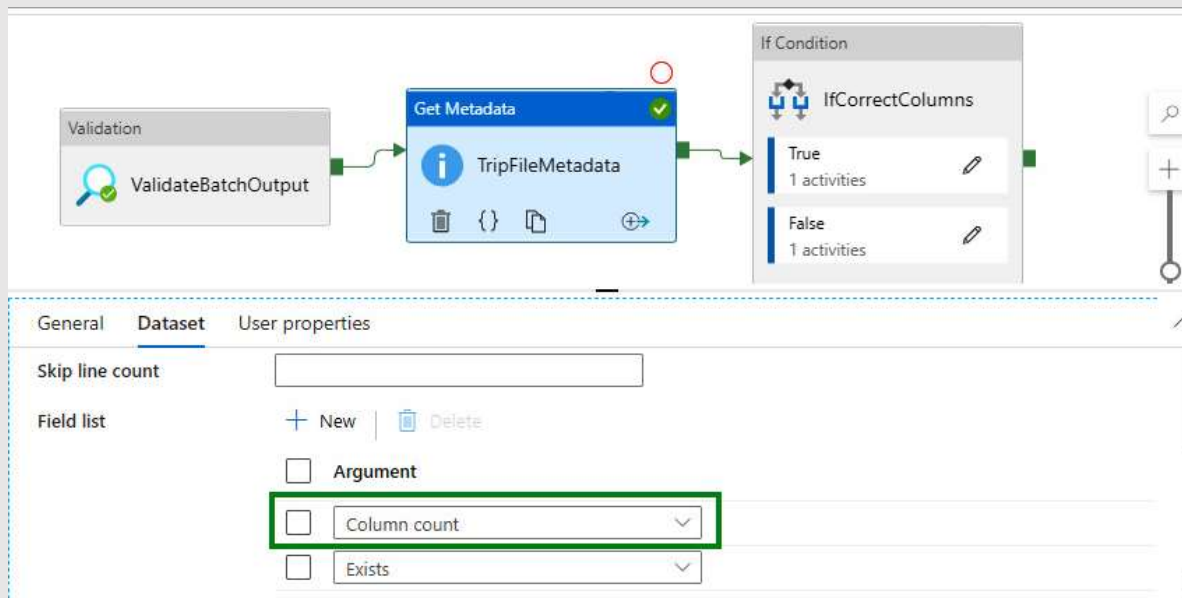


Figura 11.5 - Configuración de la actividad Get Metadata para publicar el recuento de columnas

- Una vez que obtenemos los metadatos, debemos utilizar la actividad If Condition para decidir si queremos seguir procesando el pipeline o no en base a los metadatos de la etapa anterior:

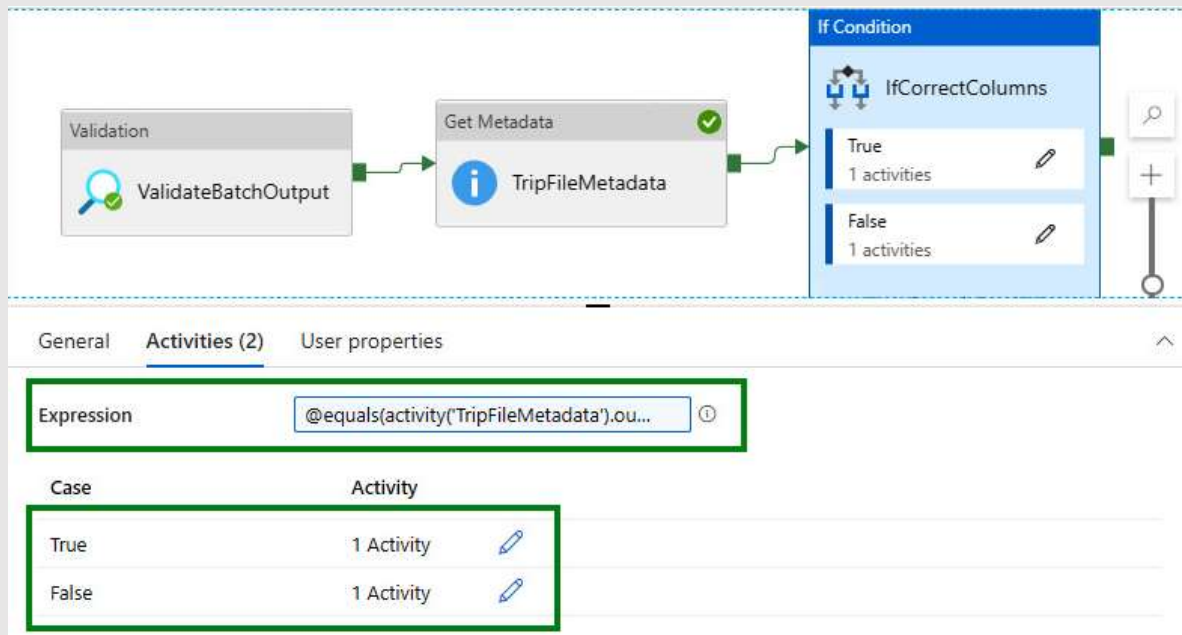


Figura 11.6 - Uso de los metadatos de la actividad Get Metadata para tomar una decisión

La condición podría ser algo como lo siguiente, donde comprobamos si el número de columnas es igual al número de columnas esperado. En este ejemplo, estoy esperando cinco columnas:

```
@equals(activity('TripFileMetadata').output.columnCount, 5)
```

Puede realizar comprobaciones de validación similares utilizando una combinación de actividades de ADF para Batch y otras tecnologías.

Ahora, aprendamos a programar pipelines de datos en ADF y Synapse Pipelines.

11.5. Programación de data pipelines en Data Factory/Synapse pipelines

La programación de pipelines se refiere al proceso de definir cuándo y cómo debe iniciarse un pipeline. El proceso es el mismo entre los pipelines ADF y Synapse. Los pipelines de ADF y Synapse tienen un botón llamado Add Trigger en la pestaña Pipelines que puede ser utilizado para programar los pipelines, como se muestra en la siguiente captura de pantalla:

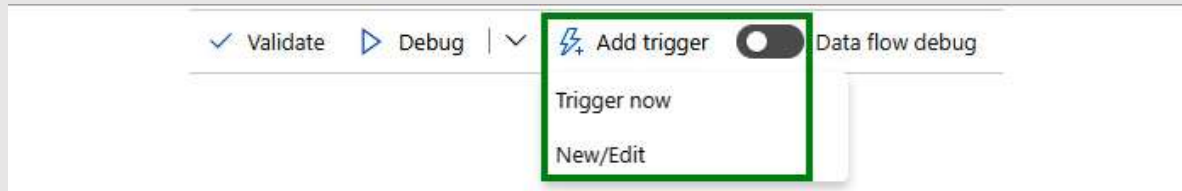
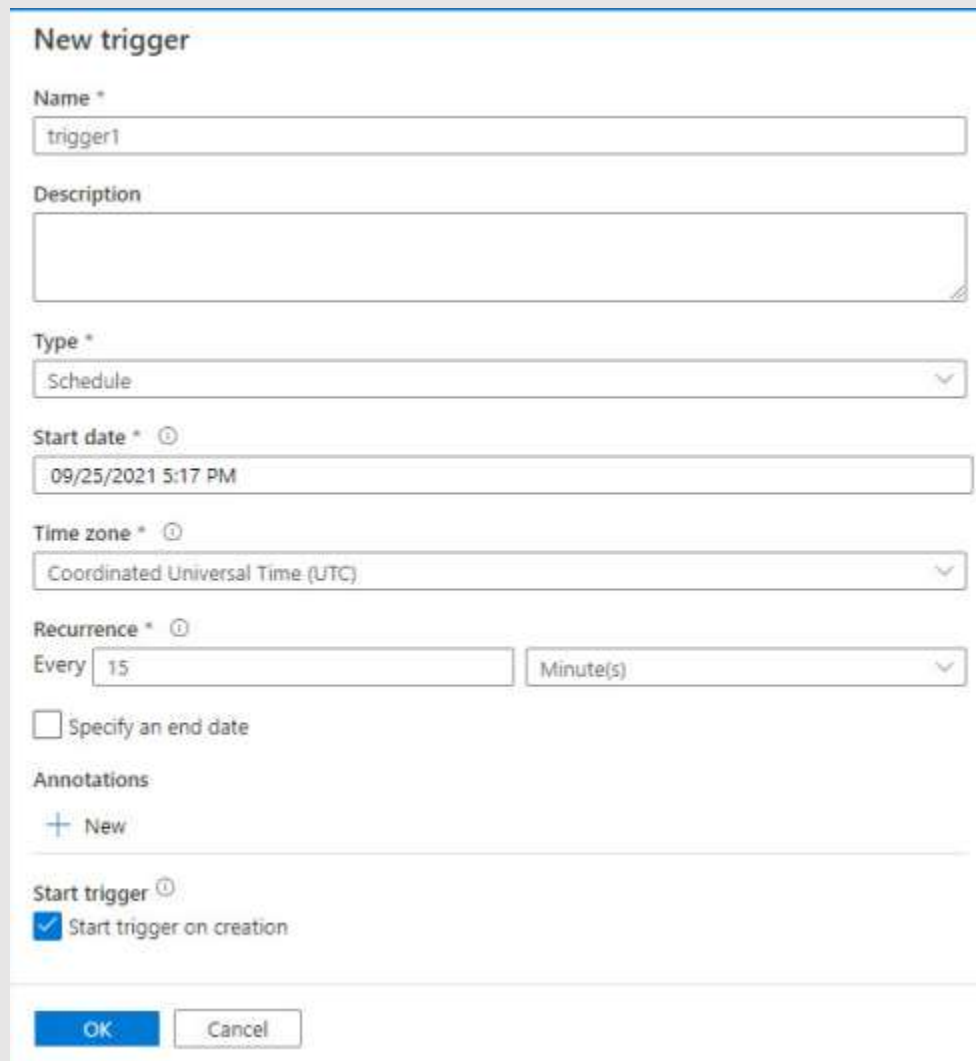


Figura 11.7 - Añadir un trigger desde los pipelines ADF/Synapse

La siguiente captura de pantalla muestra los detalles necesarios para configurar un Schedule trigger:

A screenshot of the 'New trigger' configuration form in Azure Data Factory/Synapse. The form has the following fields and options:

- Name ***: A text input field containing 'trigger1'.
- Description**: A large text area for additional details.
- Type ***: A dropdown menu set to 'Schedule'.
- Start date ***: A date and time picker showing '09/25/2021 5:17 PM'.
- Time zone ***: A dropdown menu set to 'Coordinated Universal Time (UTC)'.
- Recurrence ***: A section with 'Every' followed by a text input '15' and a dropdown menu 'Minute(s)'.
- Specify an end date**: An unchecked checkbox.
- Annotations**: A section with a '+ New' button.
- Start trigger**: A section with a checked checkbox labeled 'Start trigger on creation'.

At the bottom of the form are 'OK' and 'Cancel' buttons.

Figura 11.8 - Definir el trigger en ADF

Los servicios de pipeline de ADF y Synapse soportan cuatro tipos de trigger:

- **Schedule trigger**: Esto desencadena un pipeline una vez o regularmente en base a la hora del reloj de pared.
- **Tumbling window trigger**: Esto desencadena un pipeline basado en intervalos periódicos mientras se mantiene el estado; es decir, el trigger entiende qué ventana de datos fue procesada por última vez y se reinicia desde ahí.
- **Event-based trigger**: Este dispara un pipeline basado en eventos del almacén, como la creación de archivos en Blob o ADLS.
- **Custom trigger**: Esto desencadena pipelines basados en eventos de Azure Event Grid. ADF puede escuchar temas en Azure Event Grid y desencadenar pipelines basados en ciertos mensajes o eventos que ocurren en el tema de Event Grid.

Una vez que se ha configurado el trigger, puede ver cómo se activan automáticamente los pipelines desde la pestaña Monitoring de los pipelines de ADF o Synapse.

Puede aprender más sobre triggers y scheduling pipelines aquí: <https://docs.microsoft.com/en-us/azure/data-factory/concepts-pipeline-execution-triggers>.

Ahora, vamos a aprender a gestionar y supervisar los pipelines de datos en los pipelines de ADF y Synapse.

11.6. Gestión de data pipelines en Data Factory/Synapse pipelines

Los pipelines ADF y Synapse proporcionan dos pestañas llamadas Manage y Monitor, que pueden ayudarnos a gestionar y monitorizar los pipelines, respectivamente.

En la pestaña Manage, podemos añadir, editar y eliminar servicios vinculados, integration runtimes, triggers, configurar Git, etc., como se muestra en la siguiente captura de pantalla:

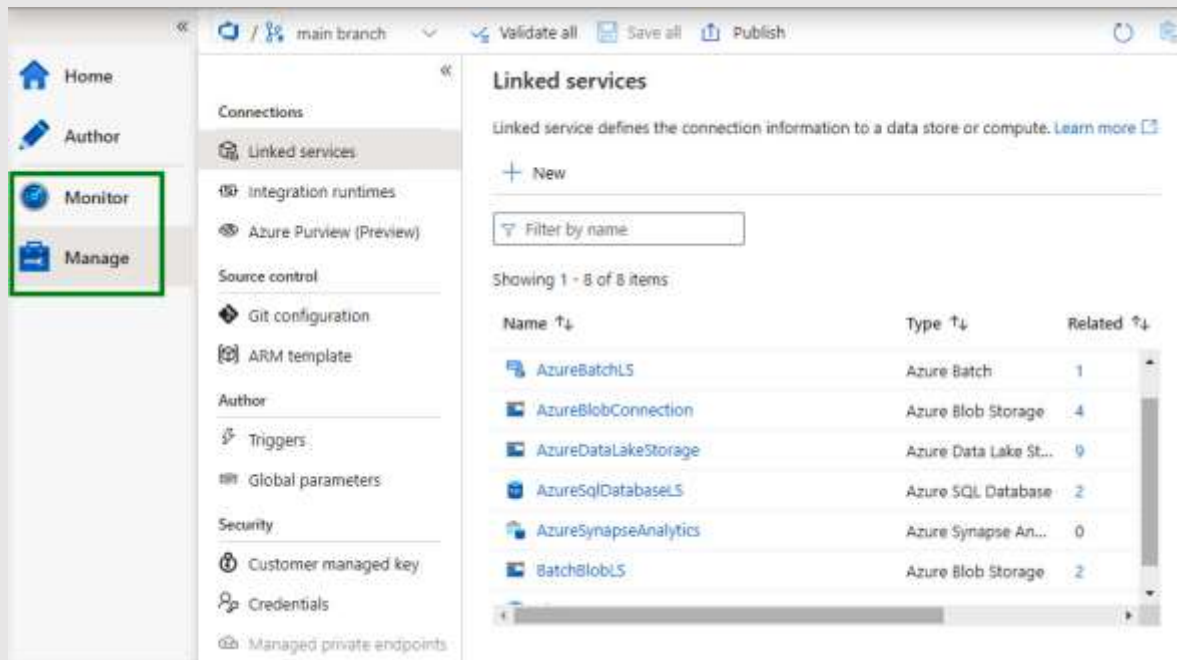


Figura 11.9 - La pantalla Manage de ADF

Ya hemos aprendido sobre los servicios vinculados a lo largo de este libro. Ahora, vamos a explorar el tema de integration runtimes en ADF y los pipelines de Synapse.

11.6.1. Integration runtimes

Un integration runtime (IR) se refiere a la infraestructura informática que es utilizada por ADF y Synapse Pipelines para ejecutar data pipelines y data flows. Se trata de las máquinas o VM reales que ejecutan el trabajo entre bastidores.

La IR se encarga de ejecutar los data flows, copiar los datos a través de redes públicas y privadas, enviar actividades a servicios como Azure HDInsight y Azure Databricks, y ejecutar SQL Server Integration Services (SSIS).

Puedes crear IRs desde la pestaña Manage de los pipelines de ADF y Synapse haciendo clic en el botón + New, como se muestra en la siguiente captura de pantalla:

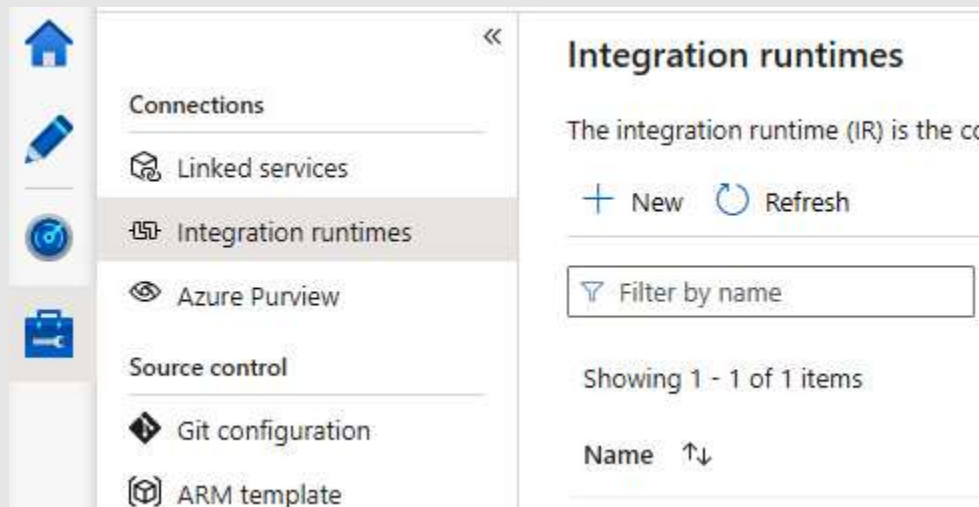


Figura 11.10 - Creación de un nuevo integration runtime

ADF proporciona soporte para tres tipos de IRs:

- **Azure Integration Runtime**: Esta es la opción por defecto y soporta la conexión de almacenes de datos y servicios de cómputo a través de endpoints públicos. Utilice esta opción para copiar datos entre servicios alojados en Azure. Azure IR también admite la conexión de almacenes de datos mediante enlaces privados.
- **Self-Hosted Integration Runtime**: Utilice esta opción cuando necesite copiar datos entre clústeres locales y servicios de Azure. Necesitará máquinas o VM en la red privada local para instalar un self-hosted IR.
- **Azure – SSIS Integration Runtime**: Las IR de SSIS se utilizan para los casos de uso de elevación y desplazamiento de SSIS.

Puede obtener más información sobre los IRs aquí: <https://docs.microsoft.com/en-us/azure/data-factory/create-azure-integration-runtime?tabs=data-factory>.

A continuación, veamos las opciones de monitorización en ADF.

11.6.2. Monitorización de ADF

Los pipelines pueden ser activados manualmente o automáticamente (usando triggers). Una vez activados, podemos monitorizarlos desde la pestaña Monitor, como se muestra en la siguiente captura de pantalla:

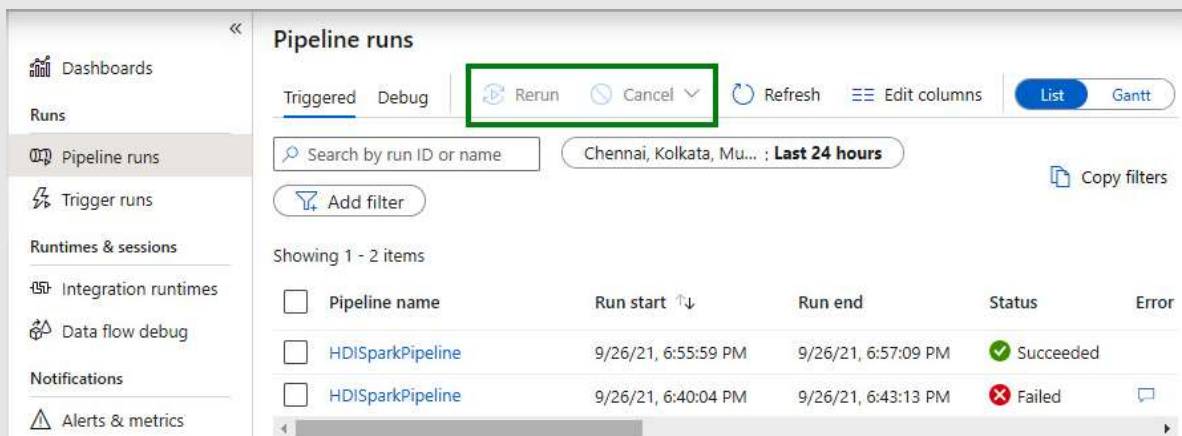


Figura 11.11 - Monitorización de pipelines de ADF

Desde la pestaña Monitor se pueden cancelar las ejecuciones en curso y volver a ejecutar los pipelines que hayan fallado. Puede hacer clic en cada una de esas ejecuciones para ver los detalles de cada flujo, como se muestra en la siguiente captura de pantalla. Haga clic en el icono de las gafas para ver las estadísticas sobre la ejecución, como cuántas etapas había, cuántas líneas se procesaron, cuánto tiempo duró cada etapa, etc:

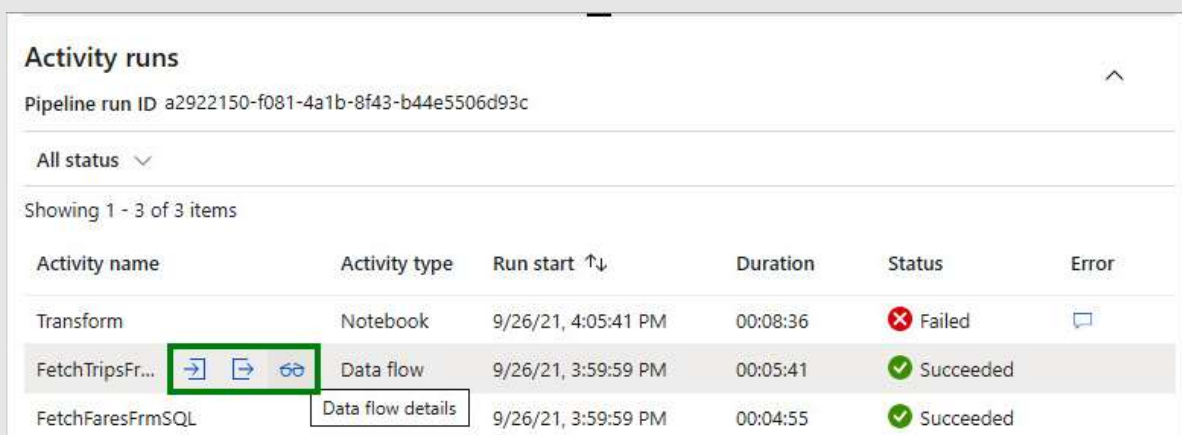


Figura 11.12 - Detalles del data flow en la pestaña de ejecuciones de actividad

Los pipelines de Synapse también proporcionan pantallas similares de Gestión y Monitorización, por lo que no las repetiremos aquí.

A continuación, vamos a aprender a gestionar los Spark jobs en un pipeline de ADF.

11.7. Gestión de Spark jobs en un pipeline

La gestión de los Spark jobs en un pipeline implica dos aspectos:

- **La gestión de los atributos del pipeline's runtime que lanza la actividad Spark:** La gestión de los atributos de la actividad Spark en el pipeline no difiere de la gestión de cualquier otra actividad en un pipeline. Las páginas de Gestión y Monitorización que vimos en la Figura 11.9, Figura 11.11 y Figura 11.12 son las mismas para cualquier actividad Spark también. Puedes utilizar las opciones proporcionadas en estas pantallas para gestionar tu actividad Spark.
- **Gestión de jobs y configuraciones de Spark:** Esto implica entender cómo funciona Spark, ser capaz de afinar los jobs, etc. Tenemos un capítulo completo dedicado a la optimización de los trabajos de Synapse SQL y Spark hacia el final de este libro. Puedes consultar el Capítulo 14, Optimización y resolución de problemas de almacenamiento y procesamiento de datos, para aprender más sobre la gestión y el ajuste de los trabajos de Spark.

En esta sección, aprenderemos a añadir un job de Apache Spark (a través de HDInsight) a nuestro pipeline para que te hagas una idea de los parámetros que se pueden configurar al configurar un job de Spark usando ADF y HDInsight. Esto es ligeramente diferente a añadir Databricks Notebooks al pipeline de ADF. Estos son los pasos:

1. Debemos crear un Linked service para un cluster de HDInsight desde la pestaña Manage de ADF. La siguiente captura de pantalla muestra cómo crear este servicio vinculado. Puedes utilizar AutoResolveIntegrationRuntime como tu clúster de HDInsight ya que también están alojados en Azure. Especifique los demás detalles, como el nombre de un clúster de Hdi existente o un clúster de HDInsight on demand, así como el nombre del servicio vinculado de Azure Storage asociado a la cuenta de almacenamiento de ese clúster de HDI para crear el servicio vinculado:

New linked service (Azure HDInsight)

Type *

☒ Bring your own HDInsight ☐ On-demand HDInsight

Connect via integration runtime * ⓘ

AutoResolveIntegrationRuntime

Account selection method

☒ From Azure subscription ☐ Enter manually

Azure subscription

Azure subscription 1 (ed6b68963-fcd6-400d-87e3-b27629c5da40)

Hdi Cluster *

DP203HDISpark

▲ Storage accounts associated with cluster ⓘ

Storage	Type
dp203hdisparkhdistorage	Blob Storage

Azure Storage linked service

☒ Blob Storage ☐ ADLS Gen 2

Azure Storage linked service *

HDIBlobStorageLS

✓ Connection successful

Create Back Test connection Cancel

Figura 11.13 - Creación de un servicio vinculado de HDInsight

2. A continuación, seleccionamos la actividad Spark en la pestaña de actividades del ADF y seleccionamos el servicio vinculado que hemos creado en el paso anterior para el campo de servicio vinculado de HDInsight:

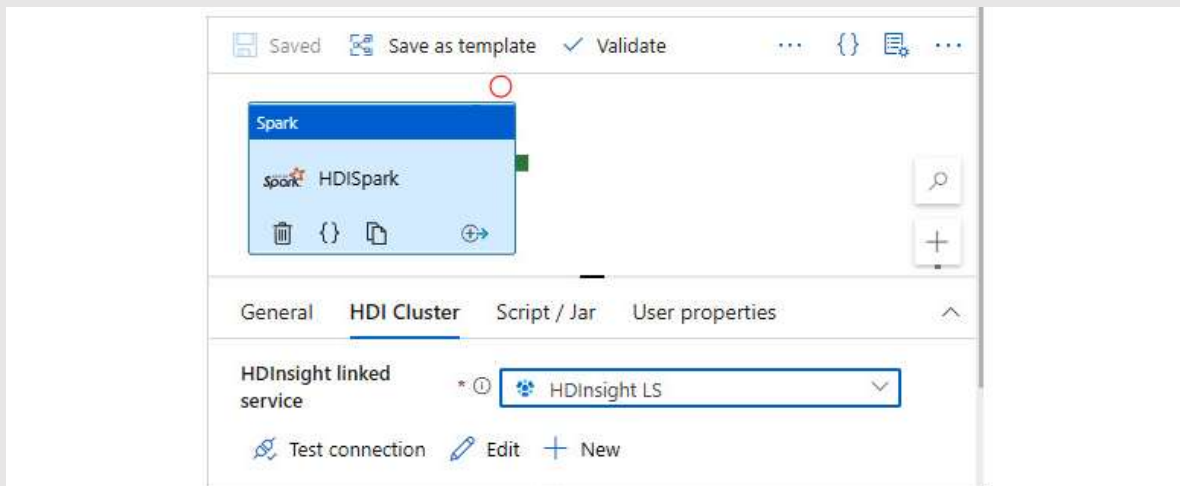


Figura 11.14 - Configuración del cluster de HDInsight Spark dentro del pipeline de ADF

3. En la pestaña Script/Jar, proporciona el enlace al archivo de trabajo del Driver de Spark real. En este caso, apuntaremos a un script de recuento de palabras, que fue cargado en el almacenamiento Blob:

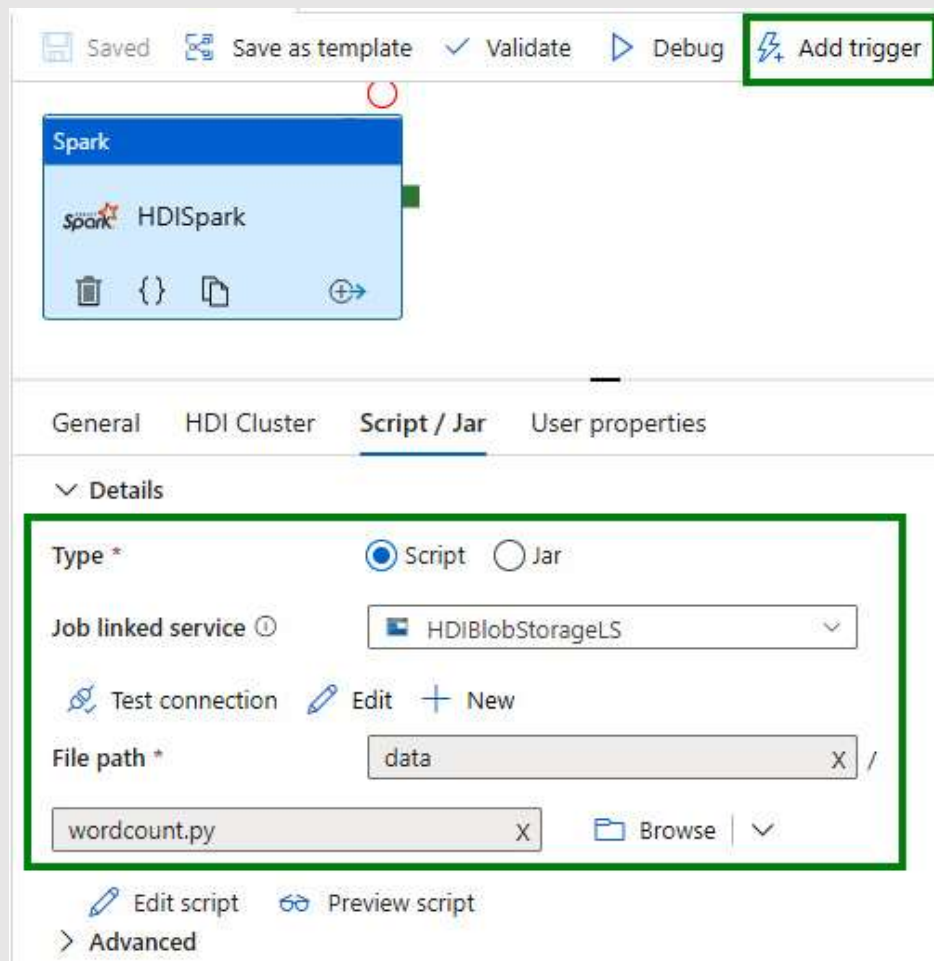


Figura 11.15 - Creación de un pipeline con Spark

4. Ahora, si activamos este pipeline usando el botón Add Trigger, todo el pipeline de Spark se ejecutará.

Esto debería haberte dado una buena comprensión de la gestión de Spark en pipelines. A continuación, vamos a aprender cómo implementar el control de versiones en ADF.

11.8. Implementación del control de versiones para los pipeline artifacts

Por defecto, los pipelines de ADF y Synapse guardan los detalles del pipeline en sus almacenes internos. Estos almacenes internos no ofrecen opciones de colaboración, control de versiones o cualquier otro beneficio proporcionado por los sistemas de control de fuentes. Cada vez que se hace clic en el botón Publicar todo, los últimos cambios se guardan dentro del servicio. Para superar esta carencia, tanto los pipelines de ADF como los de Synapse ofrecen opciones para integrarse con sistemas de control de fuentes como Git. Exploremos cómo configurar el control de versiones para nuestros artefactos de pipeline.

11.8.1. Configurando el control de fuentes en ADF

ADF proporciona un botón para configurar el repositorio de código en la parte superior de la pantalla de inicio, como se muestra en la siguiente captura de pantalla. Puede utilizar este botón para iniciar el proceso de configuración de Git:

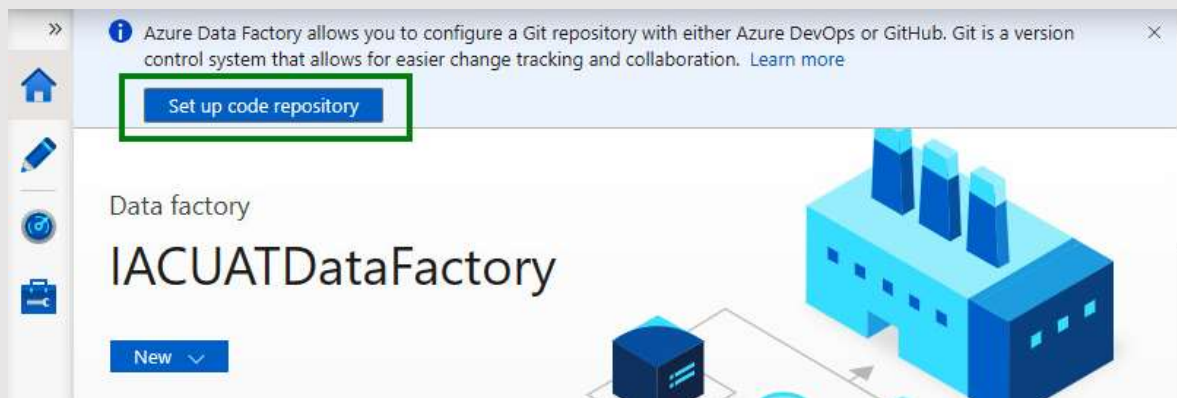


Figura 11.16 - El botón Configurar repositorio de código en la pantalla de inicio de ADF

También puede llegar a la página de configuración de Git desde la pestaña Manage (el icono del kit de herramientas), como se muestra en la siguiente captura de pantalla. Esta captura de pantalla muestra los pipelines de Synapse, pero ADF tiene una página muy similar:

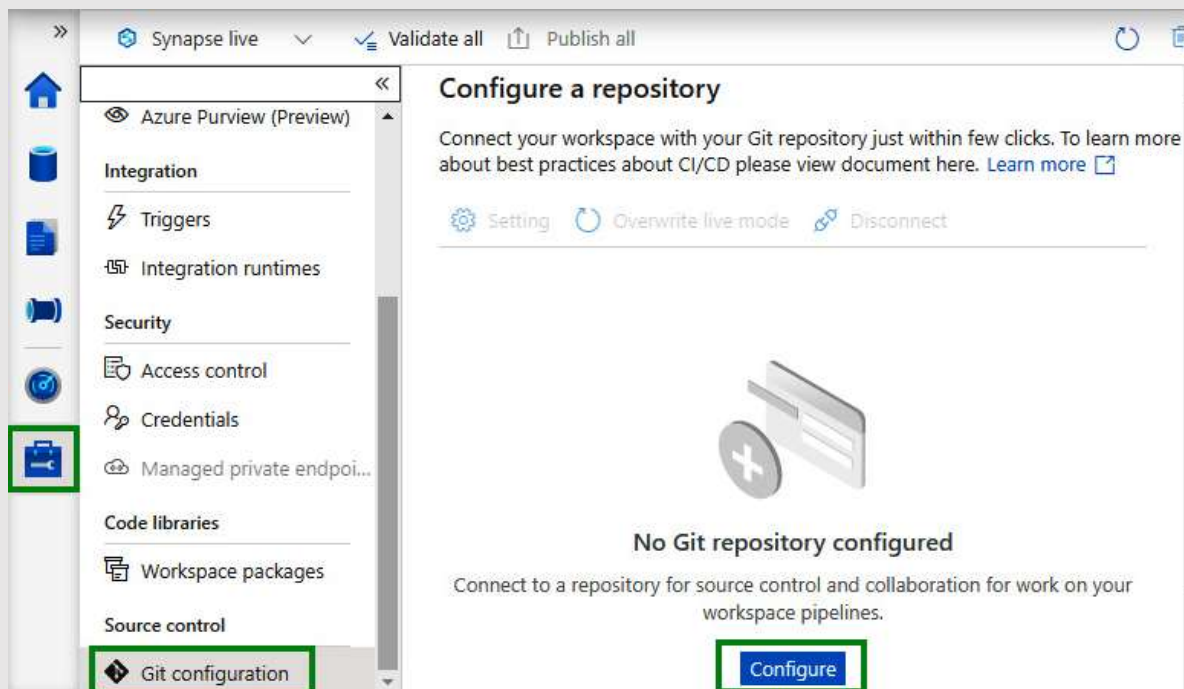


Figura 11.17 - Lanzamiento de la configuración de Git desde la pestaña Manage de Synapse

Tanto los pipelines de ADF como los de Synapse soportan la versión Azure DevOps de Git y las cuentas externas de GitHub. Aprendamos a integrar estos dos sistemas de control de fuentes en ADF y Synapse.

11.8.2. Integración con Azure DevOps

Veamos la configuración para Azure DevOps. Necesitarás tener una cuenta Azure DevOps existente. Si no tienes una, puedes crear una fácilmente:

1. Solo tienes que buscar Azure DevOps Organizations desde el portal de Azure. Haga clic en el botón My Azure DevOps Organizations en la pantalla de Azure DevOps, como se muestra en la siguiente captura de pantalla, y configure una. Como alternativa, también puedes visitar directamente <https://azure.microsoft.com/en-us/services/devops/> para empezar con Azure DevOps:



Figura 11.18 - Creación de una nueva organización DevOps

Una vez que hayas creado una nueva organización Azure DevOps, puedes crear nuevos repositorios Git bajo ella.

2. Ahora, vuelve a la pantalla de configuración de Git en Azure Data Factory y haz clic en el botón Configure (Figura 11.17). aquí, podrás elegir entre Azure DevOps y GitHub. Si eliges Azure DevOps, verás la pantalla Configurar un repositorio, como se muestra en la siguiente captura de pantalla. Aquí, debes proporcionar un nombre de organización de Azure DevOps, un nombre de proyecto, un nombre de repositorio, una rama de colaboración, una rama de publicación, una carpeta raíz, etc:

Configure a repository

Specify the settings that you want to use when connecting to your repository.

☒ Select repository ☐ Use repository link

Azure DevOps organization name * ⓘ
newton-dp203

Project name * ⓘ
adf-dev

Repository name * ⓘ
adf-dev

Collaboration branch * ⓘ

Publish branch * ⓘ
adf_publish

Root folder * ⓘ
/

Import existing resource
☒ Import existing resources to repository

Import resource into this branch ⓘ

Apply Back Cancel

Figura 11.19 - Configuración de Azure DevOps Git como control de origen para ADF

3. Rellene los detalles y haga clic en Aplicar para configurar el repositorio Azure DevOps Git. A partir de ahora, cada vez que hagas clic en Publicar se guardarán los cambios en el repositorio git especificado por ti.

Ahora, veamos los detalles de configuración de GitHub.

11.8.3. Integración con GitHub

La pantalla de configuración para GitHub es también muy similar. Tendrás que especificar atributos como el nombre del repositorio, la rama de colaboración, la rama de publicación, la carpeta raíz, etc., como se muestra en la siguiente captura de pantalla:

Configure a repository

newtonalex

Specify the settings that you want to use when connecting to your repository.

☒ Select repository ☐ Use repository link

Repository name * ⓘ
adf

Collaboration branch * ⓘ

Publish branch * ⓘ
adf_publish

Root folder * ⓘ
/

Import existing resource
☒ Import existing resources to repository

Import resource into this branch ⓘ

Apply Back Cancel

Figura 11.20 - Configuración de GitHub como control de origen para el ADF

NOTA

En ambos casos, ADF crea una nueva rama llamada `adf_publish`, que utilizará como fuente para publicar en el servicio ADF. No podrás hacer cambios en esta rama directamente, pero puedes fusionar tus cambios a través de pull requests.

Una vez que hayas configurado la versión de Azure DevOps o GitHub de Git, cada vez que hagas clic en Publicar, los artefactos del pipeline se almacenarán en el repositorio Git.

Ahora que sabes cómo configurar el control de versiones para los artefactos del pipeline, vamos a resumir este capítulo.

Resumen

Con esto, hemos llegado al final de este pequeño capítulo. Comenzamos aprendiendo cómo desencadenar cargas por lotes, cómo manejar errores y validar trabajos por lotes, y luego pasamos a los pipelines de ADF y Synapse. Aprendimos sobre la configuración de triggers, la gestión y monitorización de pipelines, la ejecución de Spark pipelines y la configuración del control de versiones en ADF y Synapse Pipelines. Con todos estos conocimientos, ahora deberías estar seguro de crear y gestionar pipelines utilizando ADF, Synapse Pipelines y Azure Batch.

Este capítulo marca el final de la sección de Diseño y Desarrollo de Procesamiento de Datos, que representa alrededor del 25-30% de los objetivos de la certificación. A partir del próximo capítulo, pasaremos a la sección de Diseño e Implementación de Seguridad de Datos, donde nos centraremos en los aspectos de seguridad del procesamiento de datos.