WELCOME TO THE TECHNICAL COMMUNITY BLOG OF PAUL ANDREW

Data Platform Principal Consultant & Solution Architect

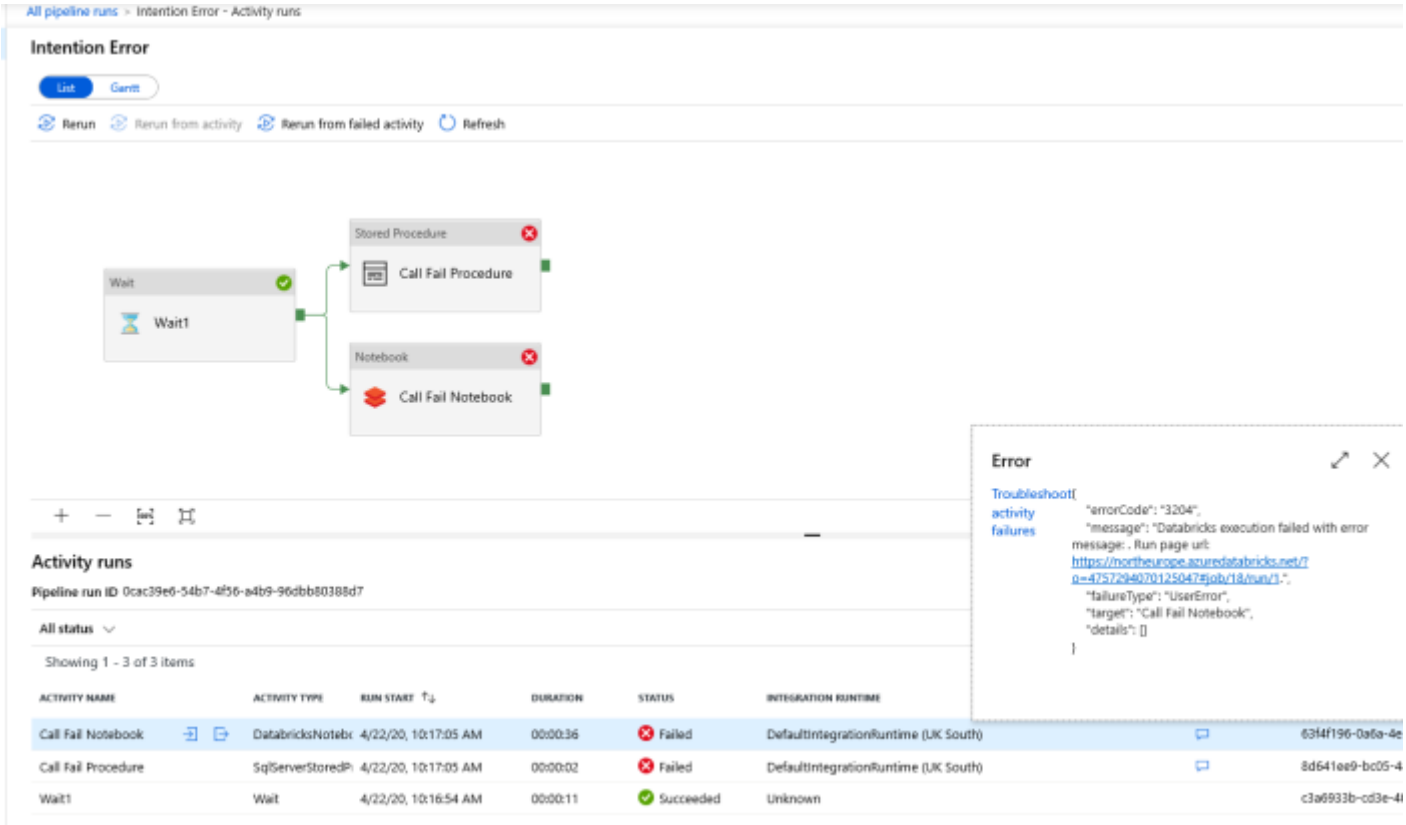
Get Any Azure Data Factory Pipeline Activity Error Details with Azure Functions

Posted on April 22, 2020July 15, 2020
A quick blog friends… I've done a few different thing now with Azure Functions and Azure Data Factory (ADF). They are definitely two of my favourite Azure Resources. In previous post I've:

- Executed Any Azure Data Factory Pipeline with an Azure Function (https://mrpaulandrew.com/2020/02/18/execute-any-azure-data-factory-pipeline-with-an-azure-function/)
- Get Any Azure Data Factory Pipeline Run Status with Azure Functions (https://mrpaulandrew.com/2019/11/21/get-any-azure-data-factory-pipeline-run-status-with-azure-functions/)

Building on this theme I've put together a similar Function to now return the error details for our failed Pipeline Activities.

Via the ADF monitoring portal something like the below might be a common sight (no judgement!). However, in my case, as the Pipeline name suggests this is an 'Intentional Error' 🙂



(https://mrpaulandrew.files.wordpress.com/2020/04/pipeline-intentional-error.png)

My pipeline contains three Activities:

1. A simple Wait, left with all default values.
2. An Azure SQLDB Stored Procedure call.
3. An Azure Databricks Notebook call.

In my Stored Procedure and Notebook I've done nothing more than raise/throw exceptions using a Pipeline parameter to establish if I actually want an error status to be created. T-SQL and Scala code snippets below.

```
CREATE PROCEDURE [dbo].[FailProcedure]
    (
    @RaiseError VARCHAR(50)
    )
AS

BEGIN

    IF(@RaiseError = 'true')
    BEGIN
        RAISERROR('The Stored Procedure intentionally failed.',16,1);
        RETURN;
    END

END
```

```
RaiseError : true

import scala.util.Try

dbutils.widgets.text("RaiseError", "","")
```

```
val raiseError = Try(dbutils.widgets.get("RaiseError").toBoolean).getOrElse(false)

if(raiseError)
{
  throw new Exception("The Notebook intentionally failed.")
}
```

(https://mrpaulandrew.files.wordpress.com/2020/04/fail-procedure-and-notebook.png)

Anyway, these bits aren't really the point of the blog.

The main thing to consider is how these error details are reported programmatically, via C# in my case, from the ADF Pipeline run. The main class used is called 'Query By Pipeline Run (https://docs.microsoft.com/en-us/rest/api/datafactory/activityruns/querybypipelinerun)' which in the .NET SDK is available via the **DataFactoryManagementClient**. This query response contains details of everything about the pipeline run and all executed Activities; success or fail. Therefore exacting the error message information can be a little tricky when presented with a huge response body for every Activity. This is my motivation for wanting to simplify things into a targeted Azure Function call. Furthermore, given my Pipeline structure above an array is required if we need to deal with multiple Activities responses.

Get Activity Error Details

In my Function, after creating the ADF client, I firstly query my Pipeline using the Run ID as the primary filter and use these to get the Activity Run details.

```
//Get pipeline details
int daysOfRuns = 7; //max duration for mandatory RunFilterParameters
DateTime today = DateTime.Now;
DateTime lastWeek = DateTime.Now.AddDays(-daysOfRuns);

PipelineRun pipelineRun;
pipelineRun = client.PipelineRuns.Get(resourceGroup, factoryName, runId);

RunFilterParameters filterParams = new RunFilterParameters(lastWeek, today);
ActivityRunsQueryResponse queryResponse = client.ActivityRuns.QueryByPipelineRun(
    resourceGroup, factoryName, runId, filterParams);
```

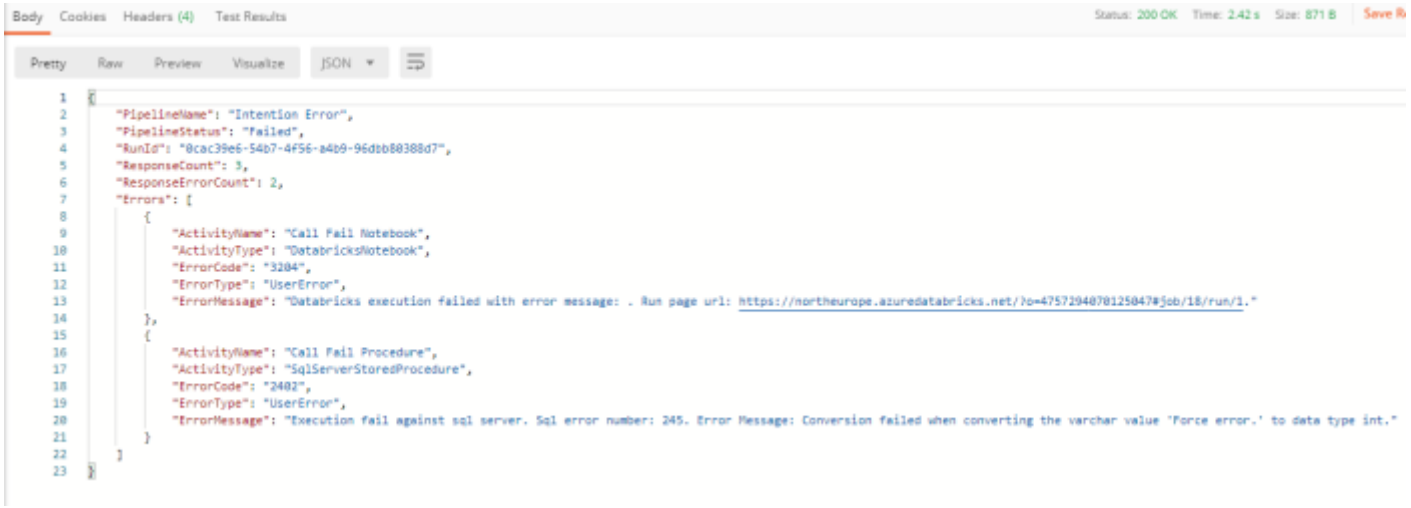
(https://mrpaulandrew.files.wordpress.com/2020/04/get-activity-details.png)

Next, I parse the response to extract the error information required for each failed Activity and construct my own cleaner Function response content.

```
//Create initial output content
dynamic outputValues = new JObject();

outputValues.PipelineName = pipelineName;
outputValues.PipelineStatus = pipelineRun.Status;
outputValues.RunId = runId;
outputValues.ResponseCount = queryResponse.Value.Count;
outputValues.ResponseErrorCount = 0;
outputValues.Errors = new JArray();
JObject errorDetails;

log.LogInformation("Pipeline status: " + pipelineRun.Status);
log.LogInformation("Activities found in pipeline response: "
                                        + queryResponse.Value.Count.ToString());
//Loop over activities in pipeline run
foreach (var activity in queryResponse.Value)
{
    if (String.IsNullOrEmpty(activity.Error.ToString()))
    {
        continue; //just incase
    }

    //Parse error output to customise output
    dynamic outputData = JsonConvert.DeserializeObject(activity.Error.ToString());

    string errorCode = outputData?.errorCode;
    string errorType = outputData?.failureType;
    string errorMessage = outputData?.message;

    //Get output details
    if (!String.IsNullOrEmpty(errorCode))
    {
      log.LogInformation("Activity name: " + activity.ActivityName);
      log.LogInformation("Activity type: " + activity.ActivityType);
      log.LogInformation("Error message: " + errorMessage);

      outputValues.ResponseErrorCount += 1;

      //Construct custom error information block
      errorDetails = JObject.Parse("{ \"ActivityName\": \"" + activity.ActivityName +
                     "\", \"ActivityType\": \"" + activity.ActivityType +
                     "\", \"ErrorCode\": \"" + errorCode +
                     "\", \"ErrorType\": \"" + errorType +
                     "\", \"ErrorMessage\": \"" + errorMessage +
                     "\" }");

      outputValues.Errors.Add(errorDetails);
    }
}
return new OkObjectResult(outputValues);
```

(https://mrpaulandrew.files.wordpress.com/2020/04/activity-error-output-code-1.png)

The output for the ADF Pipeline shown above looks like this, via Postman:



(https://mrpaulandrew.files.wordpress.com/2020/04/activity-error-postman-output.png)I hope you agree this refined response to the Activity run(s) is a lot nicer when error details is really all you want to know.

As with my previous blogs the Function body should contain the following details:

```
{
"tenantId": "1234-1234-1234-1234-1234",
"applicationId": "1234-1234-1234-1234-1234",
"authenticationKey": "Passw0rd123!",
"subscriptionId": "1234-1234-1234-1234-1234",
"resourceGroup": "CommunityDemos",
"factoryName": "PaulsFunFactoryV2",
"pipelineName": "Intentional Error",
"runId": "1234-1234-1234-1234-1234"
}
```

Feel free to grab the code from my usual GitHub repo…

Blog Supporting Content in my **GitHub** repository:
(https://mrpaulandrew.files.wordpress.com/2018/11/github-icon.png)
https://github.com/mrpaulandrew
(https://github.com/mrpaulandrew)/BlobSupportingContent/{Blog_Title}

Also, for those following my ADF.procfwk (https://mrpaulandrew.com/category/azure/data-factory/adf-procfwk/) blogs you can probably guess where I'm heading with this and the reason for wanting to develop a discrete Function to get Activity error details 😉

Many thanks for reading.

Tagged Azure Data Factory, Azure Functions, C#

Published by mrpaulandrew

Principal consultant and architect specialising in big data solutions on the Microsoft Azure cloud platform. Data engineering competencies include Azure Data Factory, Data Lake, Databricks, Stream Analytics, Event Hub, IoT Hub, Functions, Automation, Logic Apps and of course the complete SQL Server business intelligence stack. Many years' experience working within healthcare, retail and gaming verticals delivering analytics using industry leading methods and technical design patterns. STEM ambassador and very active member of the data platform community delivering training and technical sessions at conferences both nationally and internationally. Father, husband, swimmer, cyclist, runner, blood donor, geek, Lego and Star Wars fan!
View all posts by mrpaulandrew

2 thoughts on "Get Any Azure Data Factory Pipeline Activity Error Details with Azure Functions"

1. Pingback: ADF.procfwk v1.6 – Error Details for Failed Activities Captured – Welcome to the Technical Community Blog of Paul Andrew
2. Pingback: Best Practices for Implementing Azure Data Factory – Welcome to the Technical Community Blog of Paul Andrew