BTC Challenge

Deployment Proposal

# Index

[Index 2](#_Toc67659033)

[BTC Challenge 3](#_Toc67659034)

[Objective 3](#_Toc67659035)

[Deployment python and spark With Azure 4](#_Toc67659036)

[Solution 4](#_Toc67659037)

[Deployment 4](#_Toc67659038)

[Alternative deployment 7](#_Toc67659039)

[Conclusion 7](#_Toc67659040)

# BTC Challenge

Deployment Proposal

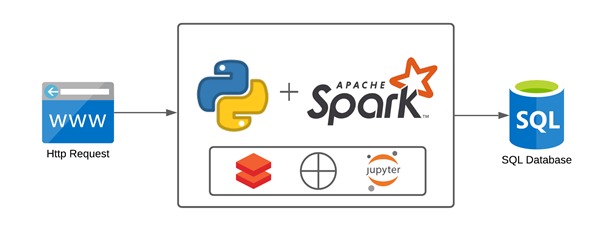
# Objective

This document explains 2 ways in which my solution can be deployed, the first one using Microsoft Azure Resources as this are the ones a have more experience with and understand how they work. Secondly, I also took the liberty to research more about Apache Beam and GCP to try to propose a deployment using your frameworks.

# Deployment python and spark With Azure

## Solution

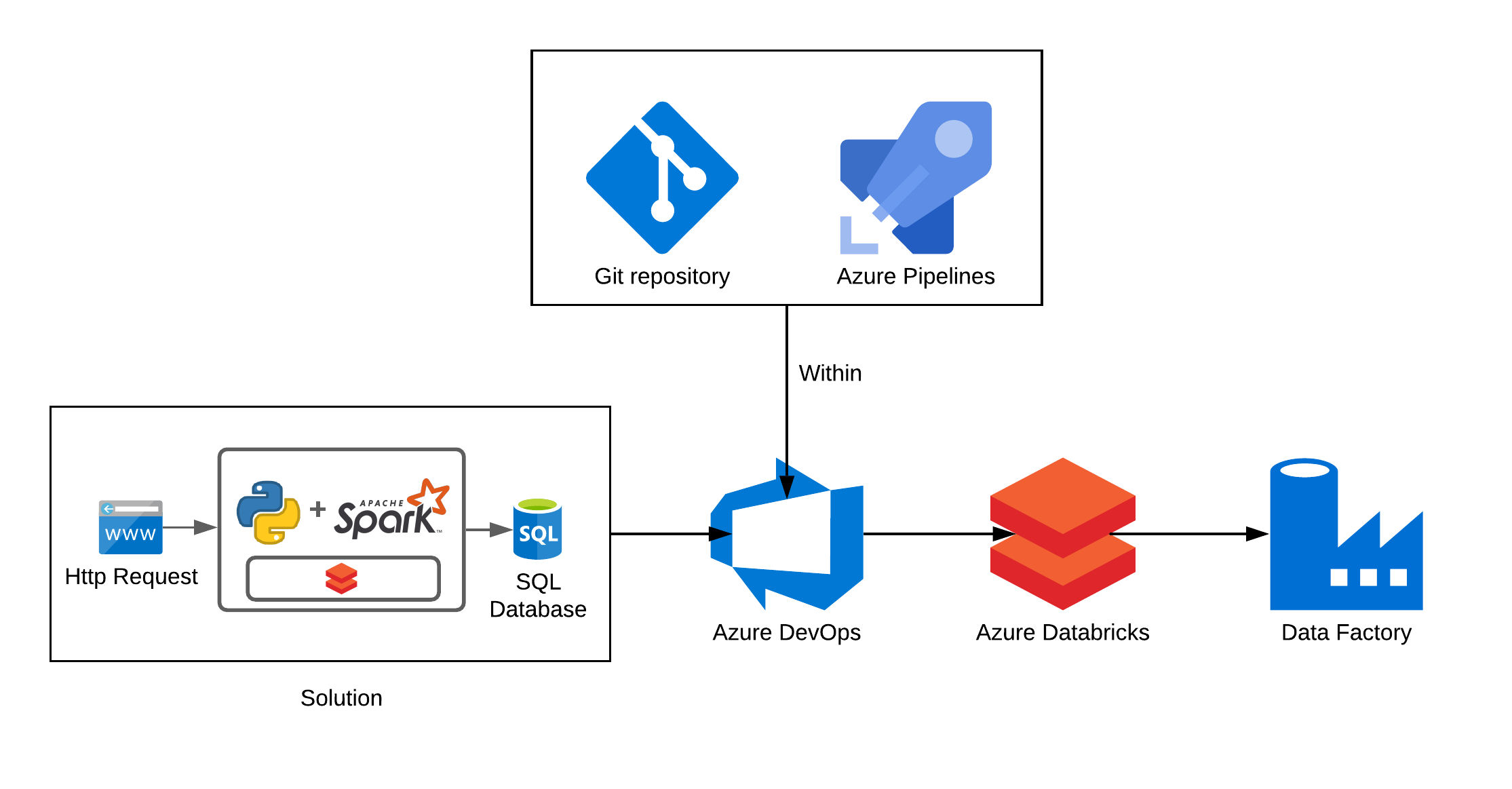
The solution is simple, it is a code in python combined with spark that consumes data from an http request, transforms it and then it stores the raw data and results in a SQL Database. The diagram contains a small box within with the icons of Databricks and Jupyter respectively. This is because this solution can be deployed using any of them as is.



**Image 1:** Solution diagram.

## Deployment

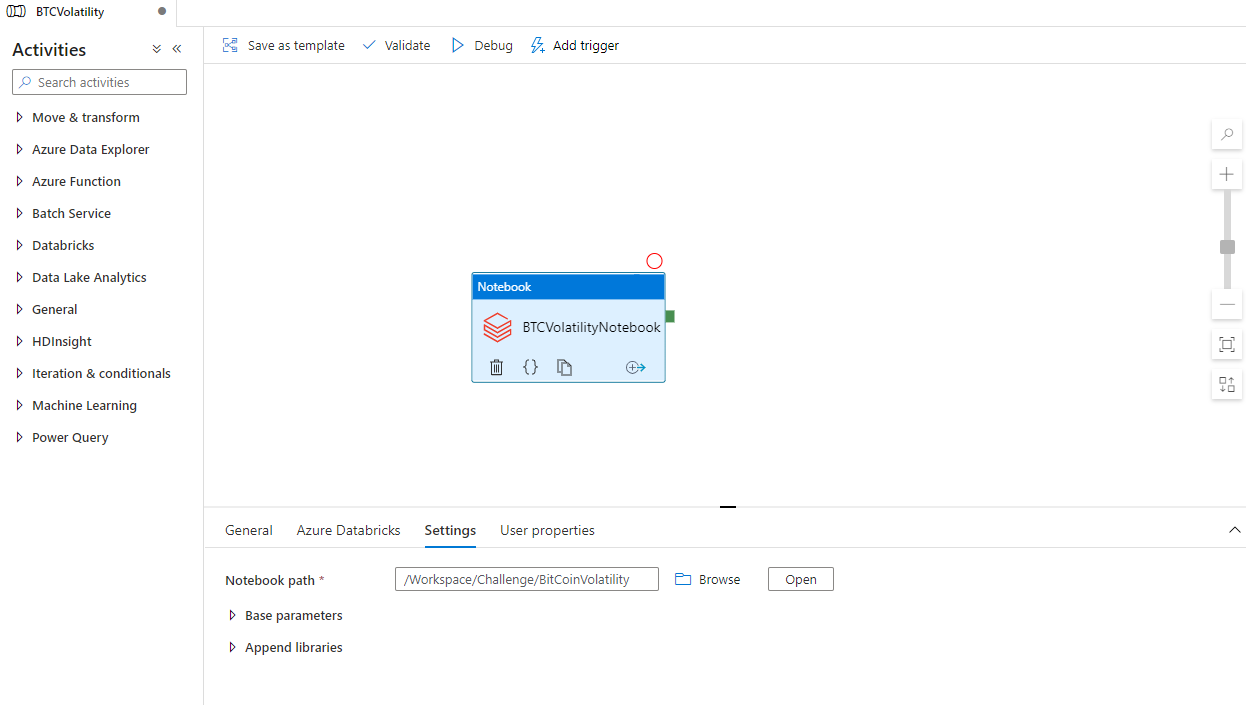
Because of the nature of this challenge, considering it would be a 1-time execution, I propose the use of Databricks, Azure Data Factory v2, and Azure DevOps as shown in **Image 2**. The code would be stored in a git repository using DevOps, from there you can also build pipelines and releases to move the code to the production environment into Databricks. After configuring the connections that are needed for the Data Factory to link to the Databricks environment, where the code is now contained, a Databricks component can be used within the pipeline in the Data Factory to execute the notebook.



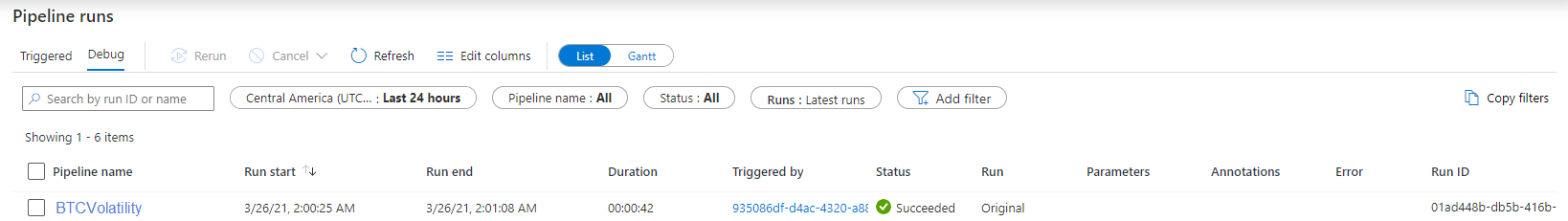
**Image 2:** Deployment with Azure Resources.

As you can see in the **Image 3** below there are settings within the Databricks component, the **Base parameter** can be useful in case the notebook could receive parameters. Some parameters can be the URL of the http request and the database table names and/or credentials. After the setup is finished the pipeline can be executed manually and you can monitor the activity in real-time or afterwards to check if it was successful or if it had errors as shown in **Image 4**.

There is also the possibility to schedule the execution of the pipeline using the Data Factory **triggers**. These can be configured to run in a specific date and time. Also if you want to have recurrence they are very useful as well.



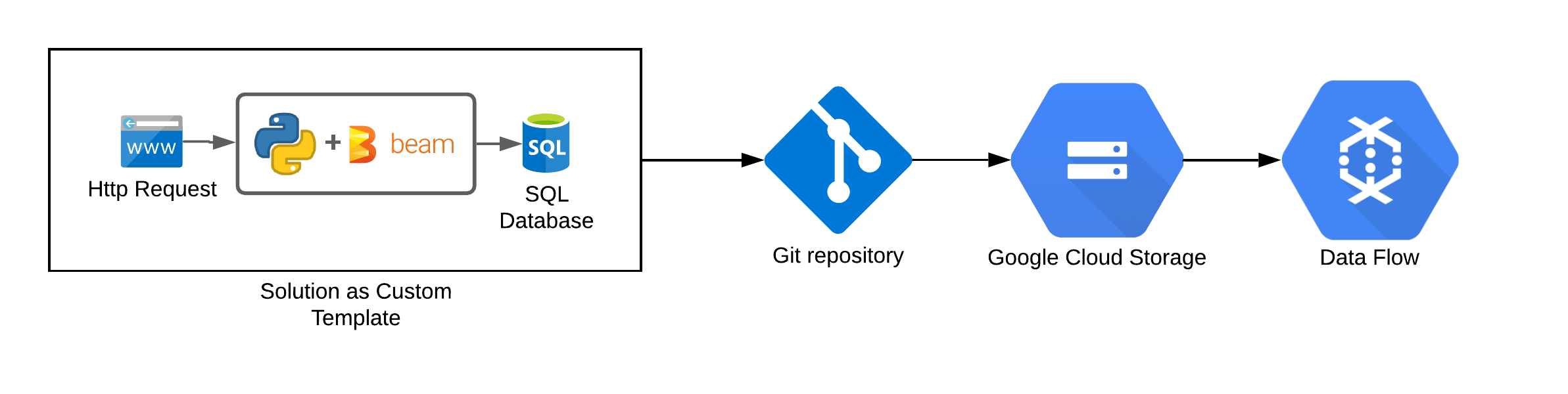
**Image 3:** Azure Data Factory v2 pipeline.



**Image 4:** Azure Data Factory v2 monitor activity.

# Alternative deployment

I would also like to propose the alternative to use Apache Beam, git repository and Google Cloud’s Data Flow as this are tools that you use. I’ve done some research and it seems that the same code I made with PySpark can be translated to Apache Beam using its libraries and python. After constructing the pipeline in Apache Beam a template must be created, this template can be stored in a git repository then moved to the CGS to have a better control on the code versions. Finally deploy the template using Data Flow.



**Image 5:** Deployment with GCP Resources.

# Conclusion

I would like to thank you for this opportunity, I’ve had a lot of fun with this challenge and hope you like what I tried to accomplish. I would like to hear your comments on the GCP deployment as it is based upon my research, feel free to teach me more about GCP and how you would implement a solution or if there are other ways that I am sure there are.

Here I include a link to the repository: <https://github.com/Areyes-Trimarchi/BitCoinVolatility> containing the solution.