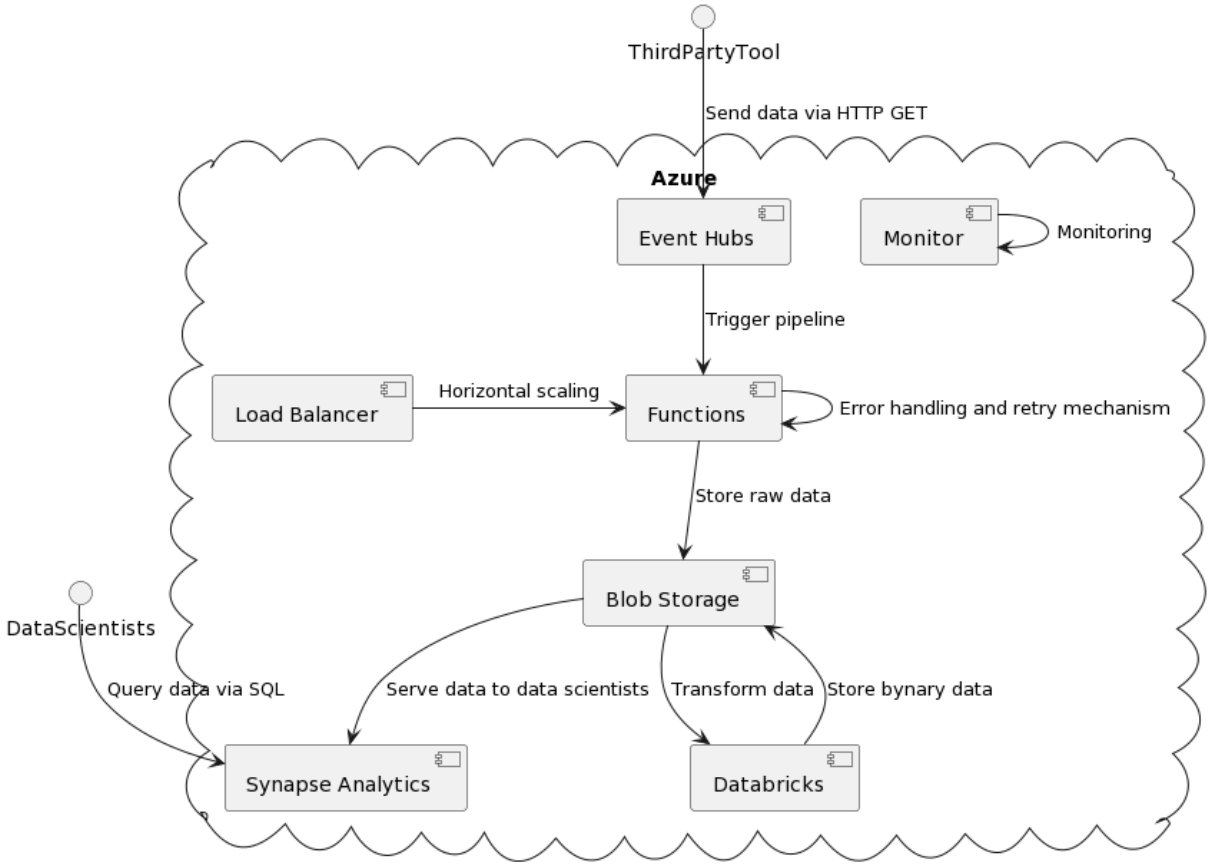
First exercise – Architecture



Third-Party Tool: This component represents the source of data that needs to be ingested into the pipeline. The tool exposes an HTTP endpoint that can only be accessed using the GET method.

Azure Event Hub: The Event Hub serves as a data ingestion point for the pipeline. It receives data from the Third-Party Tool and triggers the Event Hub Trigger in Azure Functions.

Azure Functions: Azure Functions provide the error-handling and retry functionality for the pipeline. They also handle the data ingestion from the Event Hub and store the raw data in Azure Blob Storage.

Azure Blob Storage: Azure Blob Storage is used to store both the raw data and the transformed data in compressed binary format. The raw data is stored in a designated container, and the transformed data is stored in a separate container.

Azure Databricks: Databricks is used to perform the data transformation and binary conversion of the raw data stored in Azure Blob Storage.

In the data pipeline architecture proposed for this exercise, the trigger for the Databricks job is an event triggered by the ingestion of data into the Blob Storage by the Azure Function. Specifically, the Databricks job is triggered by an Event Hub trigger that is connected to the Blob Storage. Whenever new data is ingested into the Blob Storage, the Event Hub trigger detects the event and initiates the Databricks job to transform and compress the data. This trigger mechanism ensures that the Databricks job is only executed when there is new data to process, optimizing the pipeline's performance and reducing the use of resources.

Azure Synapse Analytics: Synapse Analytics provides a SQL interface to the transformed data stored in Azure Blob Storage. This allows data scientists to interact with the data using standard SQL queries.

Azure Monitor: Monitor collects metrics, logs, and alerts from all components of the pipeline. This allows for the detection of issues and the monitoring of the health and performance of the pipeline.

In summary, this proposed Azure data pipeline architecture ingests data from a Third-Party Tool using Azure Event Hub and stores the raw data in Azure Blob Storage. The raw data is then transformed and converted into a compressed binary format using Azure Databricks and stored back in Azure Blob Storage. Data scientists can then access the transformed data through a SQL interface provided by Azure Synapse Analytics. Fealth and performance of the pipeline are monitored by Azure Monitor.

Second exercise – Devops

To simplify and automate the day-to-day management of the data pipeline, we can adopt the following strategies, technologies, and tools:

1. Metrics and monitoring: Azure provides several services for monitoring the performance and health of the data pipeline, such as Azure Monitor, Azure Log Analytics, and Azure Application Insights. We can use these services to collect metrics, logs, and alerts from all components of the pipeline and gain visibility into the dataflow's behavior. We can set up alerts and notifications to notify us when specific conditions are met, such as when a dataflow fails or when the system exceeds a threshold for resource utilization.
2. Recovering from a failed dataflow: To recover from a failed dataflow, we can leverage Azure's built-in resiliency and fault-tolerance features. For example, we can use Azure Functions to implement retry logic and error handling in case of failure. We can also use Azure Databricks to perform batch recovery of failed jobs or use Azure Synapse Analytics to perform point-in-time recovery of the transformed data. Moreover, we can implement a backup and recovery strategy for the data stored in Azure Blob Storage, such as replicating the data across different regions or using Azure Backup to take regular backups of the data.
3. Testing a dataflow in a development environment and then deploying it to a production environment: To test a dataflow in a development environment, we can use Azure DevOps to automate the build, test, and deployment of the pipeline. We can use Azure DevOps to create a CI/CD pipeline that automatically builds and tests the pipeline code and deploys it to a staging environment for further testing. Once the testing is complete, we can use Azure DevOps to promote the code to the production environment. We can also use Azure Virtual Machines or Azure Kubernetes Service to create isolated development and testing environments that replicate the production environment's configuration.
4. Modifying code on the dataflow without stopping it: To modify code on the dataflow without stopping it, we can use Azure DevOps to implement a rolling deployment strategy. This strategy involves deploying the new code to a subset of the pipeline's components while keeping the other components running on the old code. We can gradually increase the number of components running on the new code until all components are updated. This approach ensures that the pipeline remains operational during the deployment and minimizes the risk of downtime or data loss. Additionally, we can use Azure Container Instances or Azure Kubernetes Service to deploy the pipeline as containerized applications, which enable easy updates and rollbacks of the pipeline's code.