Scenario 2: Real-time Network Monitoring using **Azure Cloud**, we'll design an end-to-end streaming data pipeline leveraging Azure services such as **Azure Event Hubs**, **Azure Databricks**, **Azure Data Lake Storage (ADLS) Gen2**, **Azure Functions**, and **Power BI/Fabric**. This solution will enable you to collect, process, analyze, and visualize network performance metrics in real-time, as well as trigger alerts for anomalies and performance degradation.

Overview of the Solution

1. **Data Ingestion**: Stream network performance data from devices using **Azure Event Hubs**.
2. **Stream Processing and Enrichment**: Use **Azure Databricks** with Structured Streaming to process and enrich the data.
3. **Storage**: Store processed data in **ADLS Gen2**.
4. **Real-time Alerting**: Implement anomaly detection and trigger alerts using **Azure Databricks** and **Azure Functions**.
5. **Visualization**: Create real-time dashboards and reports using **Power BI/Fabric**.
6. **Monitoring and Automation**: Monitor the pipeline and automate processes using **Azure Monitor** and **Azure Data Factory (ADF)**.

Step-by-Step Implementation

1. Data Ingestion with Azure Event Hubs

Objective: Stream network performance data from various devices (e.g., routers) in real-time.

Steps:

1. **Create an Azure Event Hubs Namespace and Event Hub**:
   - **Navigate to Azure Portal**:
     - Search for **Event Hubs** and create a new **Event Hubs Namespace**.
     - Within the namespace, create an **Event Hub** (e.g., network-performance-hub).
   - **Configure Event Hub**:
     - Set the **Partition Count** based on expected throughput.

        - Configure **Capture** settings if you want to automatically capture streaming data to ADLS Gen2.

2. **Configure Devices to Send Data to Event Hub**:
    - **Device Configuration**:
        - Ensure that your network devices (routers, etc.) can send data to Azure Event Hubs. This typically involves setting up an **IoT Edge** or using **SDKs/APIs** to publish events.
    - **Data Format**:
        - Ensure that the data sent follows the defined schema:

json

```
{
  "DeviceID": "device-123",
  "Timestamp": "2024-09-19T12:34:56Z",
  "SignalStrength": 85,
  "CallDropRate": 2.3,
  "DataTransferSpeed": 56
}
```

3. **Secure the Event Hub**:
    - **Access Policies**:
        - Create **Shared Access Policies** with appropriate permissions (e.g., **Send** for devices, **Listen** for consumers).

---

2. Stream Processing and Enrichment with Azure Databricks

Objective: Process, clean, and enrich streaming data in real-time.

Steps:

1. **Set Up Azure Databricks Workspace**:
    - **Create a Databricks Workspace**:
        - In the Azure Portal, search for **Azure Databricks** and create a new workspace.

- **Create a Cluster**:
    - Launch the Databricks workspace and create a **Cluster** with appropriate configurations (e.g., **Standard_DS3_v2** nodes).

2. **Mount ADLS Gen2 in Databricks**:
    - **Mount ADLS Gen2**:

```
# Replace placeholders with your ADLS Gen2 details

configs = {

  "fs.azure.account.auth.type": "OAuth",

  "fs.azure.account.oauth.provider.type":
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",

  "fs.azure.account.oauth2.client.id": "<YOUR_CLIENT_ID>",

  "fs.azure.account.oauth2.client.secret": "<YOUR_CLIENT_SECRET>",

  "fs.azure.account.oauth2.client.endpoint":
"https://login.microsoftonline.com/<YOUR_TENANT_ID>/oauth2/token"

}


dbutils.fs.mount(

  source = "abfss://transformed-
data@<YOUR_STORAGE_ACCOUNT>.dfs.core.windows.net/",

  mount_point = "/mnt/transformed-data",

  extra_configs = configs

)
```

3. **Create a Streaming Job in Databricks**:
    - **Read from Event Hubs**:

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import from_json, col

from pyspark.sql.types import StructType, StructField, StringType, TimestampType,
IntegerType, FloatType
```

```
# Define the schema
schema = StructType([
   StructField("DeviceID", StringType(), True),
   StructField("Timestamp", TimestampType(), True),
   StructField("SignalStrength", IntegerType(), True),
   StructField("CallDropRate", FloatType(), True),
   StructField("DataTransferSpeed", IntegerType(), True)
])

# Read from Event Hubs
event_hub_connection_string = "<YOUR_EVENT_HUB_CONNECTION_STRING>"

df = (
   spark.readStream
   .format("eventhubs")
   .option("eventhubs.connectionString", event_hub_connection_string)
   .load()
)

# Convert binary data to string
df = df.selectExpr("CAST(body AS STRING) as json_str")

# Parse JSON
df = df.select(from_json(col("json_str"), schema).alias("data")).select("data.*")

# Data Cleansing: Filter out invalid records
df_clean = df.filter((col("SignalStrength").isNotNull()) &
                (col("CallDropRate").isNotNull()) &
```

(col("DataTransferSpeed").isNotNull()))

- **Enrich the Data (Optional)**:
  - Join with reference data, add calculated fields, etc.

```
# Example: Add a calculated field
df_enriched = df_clean.withColumn("PerformanceScore",
                  (col("SignalStrength") * 0.5) +
                  ((1 - col("CallDropRate")) * 0.3) +
                  (col("DataTransferSpeed") * 0.2))
```

- **Write to ADLS Gen2 in Delta Format**:

```
query = (
  df_enriched.writeStream
  .format("delta")
  .option("checkpointLocation", "/mnt/transformed-data/checkpoints/network_performance")
  .option("path", "/mnt/transformed-data/network_performance")
  .outputMode("append")
  .start()
)

query.awaitTermination()
```

---

3. Storage in Azure Data Lake Storage (ADLS) Gen2

Objective: Persist processed and enriched data for further analysis and reporting.

Steps:

1. **Data Storage**:
   - The streaming job writes data to ADLS Gen2 in **Delta Lake** format, enabling ACID transactions and scalable storage.
   - Data is stored under the path: /mnt/transformed-data/network_performance.
2. **Access Control**:
   - Ensure that appropriate **Access Control Lists (ACLs)** are set on the ADLS Gen2 containers to secure data access.

---

4. Real-time Alerting for Network Anomalies

Objective: Detect anomalies in network performance metrics and trigger alerts.

Steps:

1. **Define Anomaly Detection Logic in Databricks**:
   - Implement logic to identify anomalies based on predefined thresholds or statistical methods.

```
from pyspark.sql.functions import when

# Define thresholds
SIGNAL_STRENGTH_THRESHOLD = 50
CALL_DROP_RATE_THRESHOLD = 3.0
DATA_TRANSFER_SPEED_THRESHOLD = 40

# Identify anomalies
df_anomalies = df_enriched.filter(
  (col("SignalStrength") < SIGNAL_STRENGTH_THRESHOLD) |
  (col("CallDropRate") > CALL_DROP_RATE_THRESHOLD) |
  (col("DataTransferSpeed") < DATA_TRANSFER_SPEED_THRESHOLD)
```

```
)
```

2. **Write Anomalies to a Separate Delta Table**:

```
anomaly_query = (
    df_anomalies.writeStream
    .format("delta")
    .option("checkpointLocation", "/mnt/transformed-data/checkpoints/network_anomalies")
    .option("path", "/mnt/transformed-data/network_anomalies")
    .outputMode("append")
    .start()
)

anomaly_query.awaitTermination()
```

3. **Trigger Alerts Using Azure Functions**:
    - **Create an Azure Function**:
        - Develop an Azure Function that listens to the network_anomalies Delta table or is triggered by events from Event Hubs.
    - **Example: Azure Function to Send Alerts via Email or Teams**:

```
import requests
import json
import os
from azure.storage.blob import BlobServiceClient

def main(req: func.HttpRequest) -> func.HttpResponse:
```

```
    device_id = req.params.get('DeviceID')
    timestamp = req.params.get('Timestamp')
    signal_strength = req.params.get('SignalStrength')
    call_drop_rate = req.params.get('CallDropRate')
    data_transfer_speed = req.params.get('DataTransferSpeed')

    alert_message = f"Anomaly detected for Device {device_id} at {timestamp}.\n" \
            f"Signal Strength: {signal_strength}\n" \
            f"Call Drop Rate: {call_drop_rate}\n" \
            f"Data Transfer Speed: {data_transfer_speed}"

    # Example: Send alert to Microsoft Teams via Incoming Webhook
    teams_webhook_url = os.getenv("TEAMS_WEBHOOK_URL")
    headers = {'Content-Type': 'application/json'}
    payload = {"text": alert_message}
    response = requests.post(teams_webhook_url, headers=headers,
data=json.dumps(payload))

    if response.status_code == 200:
        return func.HttpResponse("Alert sent successfully.", status_code=200)
    else:
        return func.HttpResponse(f"Failed to send alert: {response.text}", status_code=500)
```

- **Integrate with Databricks**:
    - Modify the Databricks streaming job to call the Azure Function when an anomaly is detected.

```
import requests
```

```
def send_alert(device_id, timestamp, signal_strength, call_drop_rate,
data_transfer_speed):
  url = "https://<YOUR_FUNCTION_APP>.azurewebsites.net/api/SendAlert"
  params = {
    "DeviceID": device_id,
    "Timestamp": timestamp,
    "SignalStrength": signal_strength,
    "CallDropRate": call_drop_rate,
    "DataTransferSpeed": data_transfer_speed
  }
  response = requests.get(url, params=params)
  if response.status_code != 200:
    print(f"Failed to send alert: {response.text}")

# Apply the function to each anomaly record
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType


def trigger_alert(device_id, timestamp, signal_strength, call_drop_rate,
data_transfer_speed):
  send_alert(device_id, timestamp, signal_strength, call_drop_rate,
data_transfer_speed)


trigger_alert_udf = udf(trigger_alert, StringType())


df_anomalies.foreach(lambda row: trigger_alert(row.DeviceID, row.Timestamp,
row.SignalStrength, row.CallDropRate, row.DataTransferSpeed))
```

---

5. Visualization with Power BI/Fabric

Objective: Create real-time dashboards to monitor network performance metrics.

Steps:

1. **Connect Power BI to ADLS Gen2 or Azure Databricks**:
   - **Option 1: Connect Directly to Databricks**:
     - In Power BI Desktop, select **Azure Databricks** as a data source.
     - Provide the Databricks workspace URL and authentication token.
     - Import the Delta tables (e.g., network_performance).
   - **Option 2: Connect via ADLS Gen2**:
     - In Power BI Desktop, use the **Azure Data Lake Storage Gen2** connector.
     - Provide the storage account details and access credentials.
     - Import the Delta tables.
2. **Create Real-time Dashboards**:
   - **Design Visualizations**:
     - **Signal Strength Trend**: Line chart showing signal strength over time.
     - **Call Drop Rate Analysis**: Bar chart displaying call drop rates per device.
     - **Data Transfer Speed Comparison**: Grouped bar chart comparing data transfer speeds across devices or time periods.
     - **Geographical Heatmap**: Heatmap showing signal strength variations across different regions.
     - **Real-time Alerts**: KPI cards or alert indicators showing current anomalies.
   - **Set Up Streaming Datasets (if using Direct Streaming)**:
     - Use **Push Datasets** or **Streaming Tiles** in Power BI to enable real-time updates.
     - Configure **Power BI** to refresh data at short intervals (e.g., every minute).
3. **Publish and Share Dashboards**:

    - Publish the Power BI reports to the **Power BI Service**.
    - Share dashboards with stakeholders and set up **Row-Level Security (RLS)** if needed.

---

6. Monitoring and Automation

Objective: Ensure the streaming pipeline runs smoothly and handle any failures or performance issues.

Steps:

1. **Monitor Streaming Jobs in Databricks**:
    - **Databricks Workspace**:
        - Navigate to **Jobs** and monitor the status of your streaming jobs.
        - Check **Streaming Queries** under **Clusters** for real-time monitoring.
2. **Set Up Alerts with Azure Monitor**:
    - **Create Metrics and Alerts**:
        - Use **Azure Monitor** to track metrics such as Event Hub throughput, Databricks job status, and storage utilization.
        - Configure alerts to notify you of any issues (e.g., job failures, high latency).
3. **Automate Recovery and Scaling**:
    - **Auto-scaling in Databricks**:
        - Configure **Auto-scaling** for your Databricks clusters to handle varying workloads.
    - **Retry Logic**:
        - Implement retry mechanisms in your streaming jobs and alerting functions to handle transient failures.
4. **Use Azure Data Factory for Orchestration (Optional)**:
    - While **Azure Databricks** handles the stream processing, you can use **Azure Data Factory (ADF)** to orchestrate additional workflows or manage dependencies between batch and streaming pipelines.

Schema Definition

Network Performance Schema:

Field	**Data Type**	**Description**
DeviceID	String	Unique identifier for the network device
Timestamp	Timestamp	Time when the data was recorded
SignalStrength	Integer	Signal strength measured in dBm or a similar unit
CallDropRate	Float	Percentage rate at which calls are dropped
DataTransferSpeed	Integer	Data transfer speed measured in Mbps or similar
PerformanceScore	Float (Optional)	Calculated performance score for anomaly detection

---

Code Examples

1. Streaming ETL Job in Databricks

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import from_json, col, to_timestamp

from pyspark.sql.types import StructType, StructField, StringType, TimestampType,
IntegerType, FloatType


# Initialize Spark Session

spark =
SparkSession.builder.appName("NetworkPerformanceStreaming").getOrCreate()


# Define schema

schema = StructType([

  StructField("DeviceID", StringType(), True),

  StructField("Timestamp", TimestampType(), True),
```

```
    StructField("SignalStrength", IntegerType(), True),
    StructField("CallDropRate", FloatType(), True),
    StructField("DataTransferSpeed", IntegerType(), True)
])

# Read from Event Hubs
event_hub_connection_string =
"Endpoint=sb://<YOUR_EVENT_HUB_NAMESPACE>.servicebus.windows.net/;SharedAccessKeyName=<KEY_NAME>;SharedAccessKey=<KEY_VALUE>;EntityPath=network-performance-hub"

df = (
    spark.readStream
    .format("eventhubs")
    .option("eventhubs.connectionString", event_hub_connection_string)
    .load()
)

# Convert binary data to string
df = df.selectExpr("CAST(body AS STRING) as json_str")

# Parse JSON
df = df.select(from_json(col("json_str"), schema).alias("data")).select("data.*")

# Data Cleansing
df_clean = df.filter((col("SignalStrength").isNotNull()) &
                (col("CallDropRate").isNotNull()) &
                (col("DataTransferSpeed").isNotNull()))
```

```
# Enrich Data
df_enriched = df_clean.withColumn("PerformanceScore",
                          (col("SignalStrength") * 0.5) +
                          ((1 - col("CallDropRate")) * 0.3) +
                          (col("DataTransferSpeed") * 0.2))

# Write to Delta Lake
query = (
   df_enriched.writeStream
   .format("delta")
   .option("checkpointLocation", "/mnt/transformed-data/checkpoints/network_performance")
   .option("path", "/mnt/transformed-data/network_performance")
   .outputMode("append")
   .start()
)

query.awaitTermination()
```

2. Azure Function for Alerting

Function Code ():

```
import logging
import requests
import os
import azure.functions as func

def main(req: func.HttpRequest) -> func.HttpResponse:
```

```
    logging.info('Azure Function triggered for network anomaly alert.')

    try:
        # Extract query parameters
        device_id = req.params.get('DeviceID')
        timestamp = req.params.get('Timestamp')
        signal_strength = req.params.get('SignalStrength')
        call_drop_rate = req.params.get('CallDropRate')
        data_transfer_speed = req.params.get('DataTransferSpeed')

        if not all([device_id, timestamp, signal_strength, call_drop_rate,
data_transfer_speed]):
            return func.HttpResponse("Missing parameters", status_code=400)

        # Create alert message
        alert_message = f"""
        **Network Anomaly Detected**

        **Device ID**: {device_id}
        **Timestamp**: {timestamp}
        **Signal Strength**: {signal_strength}
        **Call Drop Rate**: {call_drop_rate}%
        **Data Transfer Speed**: {data_transfer_speed} Mbps

        Please investigate the issue immediately.
        """

        # Send alert to Microsoft Teams via Incoming Webhook
```

```
        teams_webhook_url = os.getenv("TEAMS_WEBHOOK_URL")
        headers = {'Content-Type': 'application/json'}
        payload = {
            "text": alert_message
        }
        response = requests.post(teams_webhook_url, headers=headers, json=payload)

        if response.status_code == 200:
            return func.HttpResponse("Alert sent successfully.", status_code=200)
        else:
            logging.error(f"Failed to send alert: {response.text}")
            return func.HttpResponse(f"Failed to send alert: {response.text}",
status_code=500)

    except Exception as e:
        logging.error(f"Error in alert function: {str(e)}")
        return func.HttpResponse(f"Error: {str(e)}", status_code=500)
```

Function Configuration:

- **Environment Variables**:
  - TEAMS_WEBHOOK_URL: URL of the Microsoft Teams Incoming Webhook connector.
- **HTTP Trigger**:
  - Configure the function to be triggered via HTTP requests.

Triggering the Function from Databricks:

```
import requests
```

```
def send_alert(device_id, timestamp, signal_strength, call_drop_rate, data_transfer_speed):
  url = "https://<YOUR_FUNCTION_APP>.azurewebsites.net/api/SendAlert"
  params = {
    "DeviceID": device_id,
    "Timestamp": timestamp,
    "SignalStrength": signal_strength,
    "CallDropRate": call_drop_rate,
    "DataTransferSpeed": data_transfer_speed
  }
  response = requests.get(url, params=params)
  if response.status_code != 200:
    print(f"Failed to send alert: {response.text}")

# Apply the function to each anomaly record
df_anomalies.foreach(lambda row: send_alert(row.DeviceID, row.Timestamp, row.SignalStrength, row.CallDropRate, row.DataTransferSpeed))
```

Diagram of the Streaming Pipeline

plaintext

```
+----------------------+     +----------------------+     +-------------------------+
|  Network Devices    | ----> |   Azure Event Hubs  | ----> |    Azure Databricks    |
| (Routers, Switches)  |      |  (Ingestion Service) |      |  (Stream Processing)    |
+----------------------+     +----------------------+     +-----------+-------------+
                                                                      |
                                                                      v
                                                          +-------------------------+
                                                          |   ADLS Gen2 (Delta)     |
```

```
| (Storage & Query)     |
+-----------+-------------+
            |
            v
+-------------------------+
|    Azure Functions      |
| (Real-time Alerting)    |
+-----------+-------------+
            |
            v
+-------------------------+
|    Power BI/Fabric      |
| (Real-time Dashboards)  |
+-------------------------+
```

Detailed Configuration Steps

1. Setting Up Azure Event Hubs

1. **Create an Event Hubs Namespace**:
   - **Navigate to Azure Portal** > **Create a Resource** > **Event Hubs**.
   - **Fill in the Details**:
     - **Name**: telecom-event-hubs
     - **Pricing Tier**: Choose based on throughput needs (e.g., **Standard**).
     - **Resource Group**: Select or create a new one.
   - **Review and Create**.
2. **Create an Event Hub**:
   - **Within the Namespace**, select **Event Hubs** > **+ Event Hub**.
   - **Name**: network-performance-hub.
   - **Configure Partitions**: Set based on expected load (e.g., 4 partitions).

    - **Capture**: Optionally enable to automatically capture streaming data to ADLS Gen2.

3. **Obtain Connection String**:
    - **Shared Access Policies**:
        - Under **Settings** > **Shared Access Policies**, select **RootManageSharedAccessKey** or create a new policy with **Send** permissions.
    - **Connection String**: Copy the **Primary Connection String** for use in Databricks.

2. Configuring Azure Databricks for Streaming

1. **Create and Configure the Cluster**:
    - **Databricks Workspace** > **Clusters** > **Create Cluster**.
    - **Cluster Name**: network-performance-cluster.
    - **Spark Version**: Choose the latest stable version (e.g., 7.3.x).
    - **Node Type**: Standard_DS3_v2.
    - **Workers**: 2 (adjust based on data volume).
    - **Libraries**: Install necessary libraries (e.g., azure-eventhubs-spark).
2. **Mount ADLS Gen2**:
    - **Notebook Cell**:

```
configs = {
  "fs.azure.account.auth.type": "OAuth",
  "fs.azure.account.oauth.provider.type":
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
  "fs.azure.account.oauth2.client.id": "<YOUR_CLIENT_ID>",
  "fs.azure.account.oauth2.client.secret": "<YOUR_CLIENT_SECRET>",
  "fs.azure.account.oauth2.client.endpoint":
"https://login.microsoftonline.com/<YOUR_TENANT_ID>/oauth2/token"
}
```

```
dbutils.fs.mount(
  source = "abfss://transformed-data@<YOUR_STORAGE_ACCOUNT>.dfs.core.windows.net/",
  mount_point = "/mnt/transformed-data",
  extra_configs = configs
)
```

3. **Develop the Streaming Notebook**:
    - **Streaming Logic**:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col, to_timestamp
from pyspark.sql.types import StructType, StructField, StringType, TimestampType, IntegerType, FloatType

# Define the schema
schema = StructType([
  StructField("DeviceID", StringType(), True),
  StructField("Timestamp", TimestampType(), True),
  StructField("SignalStrength", IntegerType(), True),
  StructField("CallDropRate", FloatType(), True),
  StructField("DataTransferSpeed", IntegerType(), True)
])

# Read from Event Hubs
event_hub_connection_string = "Endpoint=sb://telecom-event-hubs.servicebus.windows.net/;SharedAccessKeyName=<KEY_NAME>;SharedAccessKey=<KEY_VALUE>;EntityPath=network-performance-hub"
```

```
df = (
   spark.readStream
   .format("eventhubs")
   .option("eventhubs.connectionString", event_hub_connection_string)
   .load()
)

# Convert binary data to string
df = df.selectExpr("CAST(body AS STRING) as json_str")

# Parse JSON
df = df.select(from_json(col("json_str"), schema).alias("data")).select("data.*")

# Data Cleansing
df_clean = df.filter((col("SignalStrength").isNotNull()) &
                (col("CallDropRate").isNotNull()) &
                (col("DataTransferSpeed").isNotNull()))

# Enrich Data
df_enriched = df_clean.withColumn("PerformanceScore",
                          (col("SignalStrength") * 0.5) +
                          ((1 - col("CallDropRate")) * 0.3) +
                          (col("DataTransferSpeed") * 0.2))

# Write to Delta Lake
query = (
   df_enriched.writeStream
```

```
    .format("delta")
    .option("checkpointLocation", "/mnt/transformed-data/checkpoints/network_performance")
    .option("path", "/mnt/transformed-data/network_performance")
    .outputMode("append")
    .start()
)

query.awaitTermination()
```

4. **Run the Streaming Job**:
   - Execute the notebook to start the streaming job. Ensure that the cluster remains active to continue processing incoming data.

---

3. Real-time Alerting with Azure Functions

Objective: Automatically trigger alerts when anomalies are detected in network performance metrics.

Steps:

1. **Create an Azure Function App**:
   - **Navigate to Azure Portal** > **Create a Resource** > **Compute** > **Function App**.
   - **Fill in the Details**:
     - **Name**: NetworkAnomalyAlertFunction.
     - **Runtime Stack**: .
     - **Hosting**: Choose an appropriate plan (e.g., **Consumption Plan** for serverless).
     - **Storage Account**: Create or use an existing one.
     - **Region**: Choose the same region as other resources for lower latency.
   - **Review and Create**.
2. **Develop the Function**:

- **Function Code ()**:

```
import logging
import requests
import os
import azure.functions as func

def main(req: func.HttpRequest) -> func.HttpResponse:
  logging.info('Azure Function triggered for network anomaly alert.')

  try:
    # Extract query parameters
    device_id = req.params.get('DeviceID')
    timestamp = req.params.get('Timestamp')
    signal_strength = req.params.get('SignalStrength')
    call_drop_rate = req.params.get('CallDropRate')
    data_transfer_speed = req.params.get('DataTransferSpeed')

    if not all([device_id, timestamp, signal_strength, call_drop_rate,
data_transfer_speed]):
      return func.HttpResponse("Missing parameters", status_code=400)

    # Create alert message
    alert_message = f"""
    **Network Anomaly Detected**

    **Device ID**: {device_id}
```

```
    **Timestamp**: {timestamp}

    **Signal Strength**: {signal_strength}

    **Call Drop Rate**: {call_drop_rate}%

    **Data Transfer Speed**: {data_transfer_speed} Mbps


    Please investigate the issue immediately.

    """


    # Send alert to Microsoft Teams via Incoming Webhook

    teams_webhook_url = os.getenv("TEAMS_WEBHOOK_URL")

    headers = {'Content-Type': 'application/json'}

    payload = {

      "text": alert_message

    }

    response = requests.post(teams_webhook_url, headers=headers, json=payload)


    if response.status_code == 200:

      return func.HttpResponse("Alert sent successfully.", status_code=200)

    else:

      logging.error(f"Failed to send alert: {response.text}")

      return func.HttpResponse(f"Failed to send alert: {response.text}",
status_code=500)


  except Exception as e:

    logging.error(f"Error in alert function: {str(e)}")

    return func.HttpResponse(f"Error: {str(e)}", status_code=500)
```

- **Configure Environment Variables**:
    - In the Function App settings, under **Configuration**, add a new application setting:

        - **Name**: TEAMS_WEBHOOK_URL
        - **Value**: Your Microsoft Teams Incoming Webhook URL.

3. **Integrate Azure Databricks with Azure Functions**:
    - **Modify the Streaming Job to Call the Azure Function**:

```
import requests

def send_alert(device_id, timestamp, signal_strength, call_drop_rate, data_transfer_speed):
  url = "https://<YOUR_FUNCTION_APP>.azurewebsites.net/api/NetworkAnomalyAlertFunction"
  params = {
    "DeviceID": device_id,
    "Timestamp": timestamp,
    "SignalStrength": signal_strength,
    "CallDropRate": call_drop_rate,
    "DataTransferSpeed": data_transfer_speed
  }
  response = requests.get(url, params=params)
  if response.status_code != 200:
    print(f"Failed to send alert: {response.text}")

# Apply the function to each anomaly record
df_anomalies = df_enriched.filter(
  (col("SignalStrength") < SIGNAL_STRENGTH_THRESHOLD) |
  (col("CallDropRate") > CALL_DROP_RATE_THRESHOLD) |
  (col("DataTransferSpeed") < DATA_TRANSFER_SPEED_THRESHOLD)
```

```
)

query_anomalies = (
    df_anomalies.writeStream
    .foreach(lambda row: send_alert(row.DeviceID, row.Timestamp, row.SignalStrength, row.CallDropRate, row.DataTransferSpeed))
    .outputMode("append")
    .start()
)

query_anomalies.awaitTermination()
```

---

4. Visualization with Power BI/Fabric

Objective: Create real-time dashboards to monitor network performance and visualize anomalies.

Steps:

1. **Connect Power BI to ADLS Gen2 or Databricks**:
   - **Option 1: Connect Directly to Databricks**:
     - **In Power BI Desktop**:
       - Click on **Get Data** > **Azure** > **Azure Databricks**.
       - Enter the **Databricks workspace URL** and **Personal Access Token**.
       - Select the Delta tables (e.g., network_performance).
   - **Option 2: Connect via ADLS Gen2**:
     - **In Power BI Desktop**:
       - Click on **Get Data** > **Azure** > **Azure Data Lake Storage Gen2**.
       - Enter the **ADLS Gen2 URL** and authenticate using OAuth or Access Key.

            - Navigate to the path /transformed-data/network_performance and load the Delta tables.

2. **Create Real-time Dashboards**:
    - **Design Visualizations**:
        - **Signal Strength Trend**:
            - **Visualization**: Line Chart.
            - **X-Axis**: Timestamp.
            - **Y-Axis**: SignalStrength.
            - **Filter**: By DeviceID or aggregate across devices.
        - **Call Drop Rate Analysis**:
            - **Visualization**: Bar Chart.
            - **X-Axis**: DeviceID.
            - **Y-Axis**: CallDropRate.
        - **Data Transfer Speed Comparison**:
            - **Visualization**: Grouped Bar Chart.
            - **X-Axis**: DeviceID.
            - **Y-Axis**: DataTransferSpeed.
        - **Geographical Heatmap**:
            - **Visualization**: Filled Map or Shape Map.
            - **Location Data**: If available (requires device location data).
        - **Real-time Alerts**:
            - **Visualization**: KPI Cards or Conditional Formatting Tables.
            - **Indicators**: Display latest anomalies or counts of anomalies.
        - **Performance Score Overview**:
            - **Visualization**: Scatter Plot.
            - **X-Axis**: CallDropRate.
            - **Y-Axis**: PerformanceScore.
            - **Color**: Indicate severity or device category.

3. **Enable Real-time Data Streaming**:
   - **Using Power BI Streaming Datasets**:
     - Set up a **Streaming Dataset** in Power BI for real-time data.
     - Use **Power BI REST API** or **Azure Stream Analytics** to push real-time data to Power BI.
   - **Configure Automatic Refresh**:
     - Schedule **DirectQuery** or **Live Connection** for near real-time updates.
     - Ensure that data sources are optimized for frequent querying.
4. **Publish and Share Dashboards**:
   - **Publish**:
     - Publish the Power BI report to the **Power BI Service**.
   - **Share**:
     - Share dashboards with stakeholders and set up **Row-Level Security (RLS)** if needed.
   - **Mobile Access**:
     - Enable mobile access to dashboards for on-the-go monitoring.

---

5. Monitoring and Automation

Objective: Ensure the streaming pipeline operates smoothly and efficiently.

Steps:

1. **Monitor Streaming Jobs in Databricks**:
   - **Databricks Workspace**:
     - Navigate to **Jobs** and monitor the status of your streaming jobs.
     - Use **Streaming Query** monitoring to check for any backlogs or issues.
2. **Set Up Alerts with Azure Monitor**:
   - **Create Metrics and Alerts**:
     - Monitor **Event Hubs** metrics (e.g., incoming requests, errors).

        - Monitor **Databricks** job metrics (e.g., processing rates, errors).
        - Configure alerts for any anomalies or failures in the pipeline.

3. **Use Azure Data Factory (Optional)**:
    - **Orchestrate Additional Workflows**:
        - Use **ADF** to manage dependencies between batch and streaming pipelines.
        - Schedule data ingestion tasks or integrate with other Azure services as needed.
4. **Implement Auto-scaling**:
    - **Databricks Auto-scaling**:
        - Configure your Databricks clusters to auto-scale based on workload to handle variable data volumes.
5. **Optimize Performance**:
    - **Efficient Data Partitioning**:
        - Ensure Delta tables are properly partitioned for optimized query performance.
    - **Caching**:
        - Use caching strategies in Databricks for frequently accessed data.

Conclusion

By following this end-to-end solution, you can effectively implement **Scenario 2: Real-time Network Monitoring**within your telecom data engineering project using Azure Cloud services. This setup ensures that network performance metrics are ingested, processed, stored, and visualized in real-time, providing immediate insights and enabling proactive responses to network anomalies.

Key Benefits:

- **Real-time Insights**: Immediate visibility into network performance allows for swift action on issues.
- **Scalability**: Azure services scale seamlessly to handle varying data volumes and processing needs.

- **Automation**: Automated data pipelines reduce manual intervention and enhance reliability.
- **Integration**: Seamless integration with Azure's ecosystem ensures robust security, monitoring, and management.
- **Visualization**: Power BI/Fabric provides intuitive dashboards for stakeholders to monitor network health continuously.