**Case Study: Real-time Streaming with Azure Databricks**

**Business Problem**

MediHealth Hospitals wants to monitor **real-time patient vital signs** (e.g., heart rate, oxygen level, temperature) across multiple branches.

* IoT devices continuously send patient vitals.
* Hospital IT wants to detect **abnormal readings in real-time** (e.g., heart rate > 120).
* They want to store raw events for audit and also maintain a curated table for alerts and dashboards.

**Architecture**

1. **Data Source (Producer):** IoT devices send JSON messages into **Azure Event Hubs** (Kafka-compatible).
2. **Ingestion:** Databricks Structured Streaming reads from Event Hubs.
3. **Processing:** Filter abnormal vitals, enrich with patient metadata.
4. **Storage:**
   * Raw stream → **ADLS Gen2 (Bronze Layer)**.
   * Curated alerts → **Delta Lake (Silver Layer)**.
5. **Consumption:** Power BI dashboard for hospital staff.

**Step 1 – Setup Prerequisites**

* Azure Event Hubs namespace + event hub (patient-vitals).
* Azure Data Lake Storage Gen2 with containers: bronze, silver.
* Azure Databricks workspace + cluster.
* Patient metadata file (patients.csv) uploaded to ADLS.

**patients.csv**

PatientID,Name,Age,Department

P001,John Smith,45,Cardiology

P002,Sarah Lee,38,Emergency

P003,David Tan,50,ICU

**Step 2 – Sample Streaming Data**

Event Hubs will receive JSON messages like:

{

"PatientID": "P001",

"HeartRate": 128,

"OxygenLevel": 96,

"Temperature": 99.5,

"Timestamp": "2025-09-12T10:30:00Z"

}

**Step 1 – Create Azure Resources**

1. **Azure Event Hubs**
   * Go to Azure Portal → Create **Event Hubs Namespace** (Standard tier).
   * Inside it, create an **Event Hub** named: patient-vitals.
   * Copy the **Event Hub connection string** with EntityPath (we’ll use in producer.py).
2. **Azure Data Lake Storage Gen2**
   * Create a **Storage Account** with Hierarchical Namespace enabled.
   * Add 3 containers:
     + bronze (raw data)
     + silver (curated/cleaned)
     + gold (aggregates for BI)
3. **Azure Databricks**
   * Create **Databricks Workspace** (Premium tier).
   * Inside Databricks, create a **Cluster** (choose runtime: 11.x or later, with Spark 3.3+).

**Step 2 – Create VM for Producer**

1. **Provision VM**
   * Azure Portal → Create Resource → Virtual Machine.
   * Choose **Ubuntu 20.04 LTS**, size B1s or B2s.
   * Allow SSH (port 22).
   * Attach to same virtual network (not required but helps if you expand later).
2. **SSH into VM**
3. ssh azureuser@<vm-public-ip>
4. **Install Python & Git**
5. sudo apt update
6. sudo apt install -y python3 python3-pip git
7. **Install Event Hubs SDK**
8. pip3 install azure-eventhub

**Step 3 – Create producer.py on VM**

nano producer.py

Paste:

import json, time, random

from datetime import datetime

from azure.eventhub import EventHubProducerClient, EventData

# Replace with your Event Hub connection string

CONNECTION\_STR = "Endpoint=sb://<namespace>.servicebus.windows.net/;SharedAccessKeyName=<keyname>;SharedAccessKey=<key>;EntityPath=patient-vitals"

EVENTHUB\_NAME = "patient-vitals"

producer = EventHubProducerClient.from\_connection\_string(conn\_str=CONNECTION\_STR, eventhub\_name=EVENTHUB\_NAME)

patients = ["P001", "P002", "P003", "P004", "P005"]

def generate\_vitals():

return {

"PatientID": random.choice(patients),

"HeartRate": random.randint(60, 140),

"OxygenLevel": random.randint(85, 100),

"Temperature": round(random.uniform(97.0, 103.0), 1),

"Timestamp": datetime.utcnow().isoformat()

}

print("Sending events...")

try:

while True:

batch = producer.create\_batch()

vitals = generate\_vitals()

batch.add(EventData(json.dumps(vitals)))

producer.send\_batch(batch)

print(f"Sent: {vitals}")

time.sleep(2)

except KeyboardInterrupt:

print("Stopped producer.")

finally:

producer.close()

Run in background:

nohup python3 producer.py > producer.log 2>&1 &

Now your VM will continuously send patient vitals into Event Hubs.

**Step 4 – Configure Databricks to Read Event Hubs**

1. Go to Databricks cluster → Libraries → Install:
2. com.microsoft.azure:azure-eventhubs-spark\_2.12:2.3.22
3. Create Notebook: **PatientVitalsStream**

from pyspark.sql.types import StructType, StringType, IntegerType, DoubleType

from pyspark.sql.functions import from\_json, col, window, avg

# Event Hub config

event\_hub\_connection = "Endpoint=sb://<namespace>.servicebus.windows.net/;SharedAccessKeyName=<keyname>;SharedAccessKey=<key>;EntityPath=patient-vitals"

ehConf = {"eventhubs.connectionString": event\_hub\_connection}

# Schema

schema = StructType() \

.add("PatientID", StringType()) \

.add("HeartRate", IntegerType()) \

.add("OxygenLevel", IntegerType()) \

.add("Temperature", DoubleType()) \

.add("Timestamp", StringType())

# Read stream

raw\_df = (spark.readStream

.format("eventhubs")

.options(\*\*ehConf)

.load())

df = raw\_df.selectExpr("cast(body as string) as json") \

.select(from\_json(col("json"), schema).alias("data")) \

.select("data.\*")

# --------------------

# Bronze: Raw Storage

# --------------------

bronze\_query = df.writeStream \

.format("delta") \

.option("checkpointLocation", "abfss://bronze@<storageaccount>.dfs.core.windows.net/checkpoints/patient\_vitals") \

.outputMode("append") \

.start("abfss://bronze@<storageaccount>.dfs.core.windows.net/patient\_vitals")

# --------------------

# Silver: Filtered/Curated

# --------------------

silver\_df = df.filter(col("HeartRate") > 120)

silver\_query = silver\_df.writeStream \

.format("delta") \

.option("checkpointLocation", "abfss://silver@<storageaccount>.dfs.core.windows.net/checkpoints/patient\_alerts") \

.outputMode("append") \

.start("abfss://silver@<storageaccount>.dfs.core.windows.net/patient\_alerts")

# --------------------

# Gold: Aggregates (5-min window avg per patient)

# --------------------

gold\_df = (df.withColumn("ts", col("Timestamp").cast("timestamp"))

.groupBy(window(col("ts"), "5 minutes"), col("PatientID"))

.agg(avg("HeartRate").alias("AvgHeartRate"),

avg("OxygenLevel").alias("AvgOxygen"),

avg("Temperature").alias("AvgTemp")))

gold\_query = gold\_df.writeStream \

.format("delta") \

.option("checkpointLocation", "abfss://gold@<storageaccount>.dfs.core.windows.net/checkpoints/patient\_agg") \

.outputMode("append") \

.start("abfss://gold@<storageaccount>.dfs.core.windows.net/patient\_agg")

**Step 5 – Validate**

1. Start producer.py on VM.  
   Check with:
2. tail -f producer.log

You should see messages being sent.

1. In Databricks:
   * Query Bronze (all messages):
   * display(spark.read.format("delta").load("abfss://bronze@<storageaccount>.dfs.core.windows.net/patient\_vitals"))
   * Query Silver (alerts only):
   * display(spark.read.format("delta").load("abfss://silver@<storageaccount>.dfs.core.windows.net/patient\_alerts"))
   * Query Gold (aggregates):
   * display(spark.read.format("delta").load("abfss://gold@<storageaccount>.dfs.core.windows.net/patient\_agg"))

**Step 6 – Optional: Power BI Dashboard**

1. Create **Databricks SQL Endpoint**.
2. Connect Power BI → Databricks.
3. Create visuals:
   * **Line chart**: Avg Heart Rate over time per patient.
   * **Bar chart**: Number of alerts by patient.
   * **Gauge**: Oxygen level vs threshold.