

Dokumentacja Techniczna Projektu: Online shop activity analysis

Bugdol Mateusz • Kowalski Kajetan • Orlikowski Przemysław

1. Cel projektu

Celem projektu "Online shop activity analysis" jest wdrożenie skalowalnego systemu przetwarzania danych w czasie rzeczywistym (Real-Time Data Streaming), który umożliwia monitorowanie zachowań użytkowników sklepu internetowego.

System realizuje architekturę Medallion (Bronze-Silver-Gold) na platformie Azure Databricks, dostarczając kluczowe metryki biznesowe, takie jak konwersja sprzedaży, wykrywanie porzuconych koszyków oraz analiza przychodów w ujęciu sesyjnym.

2. Architektura Rozwiązania

System oparty jest na chmurze Microsoft Azure i wykorzystuje następujące komponenty:

1. **Źródło Danych:** Symulator ruchu (Python) generujący zdarzenia JSON.
2. **Ingestia:** Azure Event Hubs (kolejkowanie strumienia zdarzeń).
3. **Przetwarzanie (Compute):** Apache Spark (Databricks) w trybie Structured Streaming.
4. **Magazyn Danych:** Azure Data Lake Storage Gen2 (format Delta Lake).
5. **Orkiestracja i Infrastruktura:** Terraform (IaC) oraz GitHub Actions (CI/CD).

(Rys. 1. Diagram architektury systemu)

3. Infrastruktura (Infrastructure as a Code)

Całe środowisko jest definiowane i wdrażane automatycznie przy użyciu narzędzia **Terraform**.

3.1. Zasoby Azure

- Resource Group: *rg-clickstream-student-dev* (kontener logiczny zasobów).
- **Storage Account:** Data Lake Gen2 (*stclickstream...*) złączoną hierarchiczną przestrzenią nazw. Kontener data przechowuje tabele Delta.
- **Event Hubs:** Namespace i Hub (*input-stream*) z 2 partycjami do odbioru danych.
- **Databricks Workspace:** Środowisko pracy analitycznej.
- Zmienne konfiguracyjne Terraform:

Nazwa zmiennej	Opis	Wartość domyślna
<code>resource_group_name</code>	Nazwa grupy zasobów	<code>rg-clickstream-student-dev</code>

location	Region Azure	Switzerland North
alert_email	Email do alertów budżetowych	twoj@email.com

3.2. Konfiguracja klastra

Skrypt Terraform automatycznie powołuje klaszter *Clickstream Cluster*:

- **Typ:** Single Node (rozmiar *Standard_DS3_v2*).
- **Runtime:** Databricks Runtime 13.3 LTS (Scala 2.12, Spark 3.x).
- **Biblioteki:** Zainstalowany konektor Maven *com.microsoft.azure:azure-eventhubs-spark*.
- **Bezpieczeństwo:** Klucze dostępowe (Connection Strings) są wstrzykiwane jako zmienne środowiskowe (*EVENT_HUB_CONN_STR*, *STORAGE_ACCOUNT_KEY*), co eliminuje konieczność ich trwałego kodowania.

3.2. Zarządzanie Kosztami i Monitoring (FinOps)

W celu kontroli wydatków chmurowych, w infrastrukturze zdefiniowano zasób *azurerem_consumption_budget_resource_group*, który monitoruje koszty generowane przez grupę zasobów w cyklu miesięcznym.

- **Budżet miesięczny:** Ustalony na sztywno limit **50 jednostek walutowych** (zależnie od waluty subskrypcji Azure).
- **Alerty kosztowe:** System automatycznie wysyła powiadomienia e-mail w momencie przekroczenia określonych progów zużycia budżetu:
 - **10%:** Wczesne ostrzeżenie o rozpoczęciu naliczania kosztów.
 - **50%:** Powiadomienie o wykorzystaniu połowy budżetu.
 - **75%:** Ostrzeżenie krytyczne przed wyczerpaniem środków.
- **Adresat:** Powiadomienia trafiają na adres zdefiniowany w zmiennej *alert_email* (domyślnie konfigurowalny w pliku *variables.tf*).

4. Generator Danych (Symulacja)

Aplikacja Python (*mock_data.py*) symuluje ruch użytkowników na stronie sklepu.

- **Generowane zdarzenia:** *PAGE_VIEW, SEARCH, PRODUCT_VIEW, ADD_TO_CART, PURCHASE*.
- **Struktura danych:** Zdarzenia zawierają ID użytkownika, ID sesji, czas (Timestamp), dane urządzenia oraz szczegóły produktu (cena, kategoria).
- **Konfiguracja:** Skrypt automatycznie pobiera parametry połączenia z pliku *.env*, generowanego przez Terraform podczas wdrożenia infrastruktury.

5. Przetwarzanie Danych (ETL Pipeline)

Logika ETL zaimplementowana jest w notebooku *medallion_pipeline* z wykorzystaniem **Spark Structured Streaming**.

5.1. Warstwa Bronze (Raw)

- **Zadanie:** Pobranie strumienia z Event Hubs.
- **Format:** Dane zapisywane są w formacie Delta w trybie *append*.
- **Charakterystyka:** Dane są surowe, nieprzetworzone, służą jako historyczne źródło prawdy.

5.2. Warstwa Silver (Cleansed)

- **Parsowanie:** Rozpakowanie pola *body* z formatu binarnego do struktury JSON.
- **Typowanie:** Rzutowanie pól na odpowiednie typy (Timestamp, Double).
- **Watermarking:** Ustawienie 10-minutowego progu tolerancji dla opóźnionych danych.
- **Deduplikacja:** Usuwanie powtórzeń na podstawie unikalnego *eventId*.

5.3. Warstwa Gold (Aggregated Business Logic)

Warstwa ta tworzy gotowe do analizy statystyki sesji (*gold_session_stats*).

- **Session Windows:** Grupowanie zdarzeń w **30-minutowe okna sesji** (brak aktywności przez 30 min zamyka sesję).
- **Logika Biznesowa:**
 - *is_purchased*: Flaga oznaczająca sfinalizowanie transakcji.
 - *is_abandoned_cart*: Wykrywanie sytuacji, gdy użytkownik dodał produkt do koszyka, ale nie kupił.
 - *session_revenue*: Suma wartości zakupów w sesji.
- **Tryb Zapisu:** *OutputMode.Complete* (aktualizacja stanu dla całego okna).

6. Raportowanie i Wizualizacja

Notebook *dashboard_visualization* pełni rolę warstwy prezentacyjnej, generując interaktywne wykresy:

- 6.1. **Główne KPI:** Liczniki całkowitej sprzedaży, liczby sesji, porzuconych koszyków i przychodu.
- 6.2. **Lejek Konwersji (Funnel):** Wizualizacja etapów: *Wszystkie Sesje -> Aktywność w Koszyku -> Zakup*.
- 6.3. **Analiza Czasowa:** Wykres liniowy pokazujący natężenie ruchu i sprzedaży w godzinowych oknach czasowych.
- 6.4. **Top Klienci:** Tabela rankingowa użytkowników z najwyższym LTV (Lifetime Value).

7. DevOps i CI/CD (Github Actions)

Projekt wykorzystuje dwa automatyczne przepływy pracy (Workflows) do zapewnienia jakości kodu.

7.1. Auto format and Lint

- **Wyzwalacz:** Pull Request do main.
- **Działanie:** Automatycznie formatuje kod Python (Black, Isort) i Terraform (terraform fmt). Usuwa nieużywane importy (Autoflake).
- **Cel:** Utrzymanie spójnego stylu kodu bez integracji programisty.

8. Instrukcja Uruchomienia (Deployment)

1. Infrastruktura:

```
cd iac
terraform init
terraform apply
```

Po zakończeniu zostanie utworzony plik *.env* z kluczami dostępu.

2. Generator Danych:

```
cd ../data_generator  
python mock_data.py
```

3. Uruchomienie Pipeline'u

- Zaloguj się do Azure Databricks.
- Uruchom klaster Clickstream Cluster.
- Otwórz notebook medallion_pipeline i kliknij "Run All".
- **Analiza:**
 - Otwórz notebook dashboard_visualization i odświeżaj wykresy, aby obserwować napływanie danych.

9. Estymowane koszty systemu:

Usługa	Warstwa/Typ	Szacowany miesięczny koszt [USD]
Azure Event Hubs	Standard	~\$5
Azure Databricks	Standard_DS3_v2	~\$120
Storage Account	Standard LRS	~\$1-2
SUMA:		~\$127

10.

▶ ✓ 3 minutes ago (3s)

```
status_df = df.withColumn(
    "session_status",
    when(col("is_purchased") == 1, "Purchased")
    .when(col("is_abandoned_cart") == 1, "Abandoned Cart")
    .otherwise("Browsing Only"),
)

display(status_df.groupBy("session_status").count())
```

▶ (2) Spark Jobs

> status_df: pyspark.sql.dataframe.DataFrame = [userid: string, window_start: timestamp ... 6 more fields]

Table +

	session_status	count
1	Purchased	714
2	Abandoned Cart	1096
3	Browsing Only	2536

↓ 3 rows | 3.32s runtime

Refreshed 2 minutes ago

▶ ✓ 2 minutes ago (4s)

```
kpi_df = df.select(
    count("*").alias("total_sessions"),
    sum("is_purchased").alias("total_orders"),
    sum("is abandoned cart").alias("total_abandoned"),
    round(sum("session_revenue"), 2).alias("total_revenue"),
    round(avg("session_revenue"), 2).alias("avg_order_value"),
)
```

display(kpi_df)

▶ (2) Spark Jobs

> kpi_df: pyspark.sql.dataframe.DataFrame = [total_sessions: long, total_orders: long ... 3 more fields]

Table +

	total_sessions	total_orders	total_abandoned	total_revenue	avg_order_value
1	4346	714	1096	1072904.5	1502.67

↓ 1 row | 4.00s runtime

Refreshed 2 minutes ago

▶ ✓ 2 minutes ago (4s)

```
display(df)
```

▶ (3) Spark Jobs

Table +

userid	window_start	window_end	events_count	is_purchased	session_revenue	is_abandoned_cart
user_2053	2026-01-29T18:55:12.184+00:00	2026-01-29T19:25:35.256+00:00	3	0	[0.01]	0
user_7536	2026-01-29T19:56:22.796+00:00	2026-01-29T20:26:37.577+00:00	2	0	[0.01]	0
user_9688	2026-01-29T19:35:02.447+00:00	2026-01-29T20:09:11.819+00:00	3	0	[0.01]	0
user_8713	2026-01-29T18:25:22.264+00:00	2026-01-29T18:55:39.694+00:00	3	0	[0.01]	0
user_2053	2026-01-29T18:19:50.036+00:00	2026-01-29T18:54:05.313+00:00	3	0	[0.01]	0
user_2968	2026-01-29T18:47:53.009+00:00	2026-01-29T19:18:08.771+00:00	4	0	[0.01]	0
user_1988	2026-01-29T19:36:24.140+00:00	2026-01-29T20:06:44.856+00:00	2	0	[0.01]	0
user_3001	2026-01-29T19:45:05.011+00:00	2026-01-29T20:16:09.842+00:00	7	0	[0.01]	1
user_5069	2026-01-29T18:16:23.541+00:00	2026-01-29T18:46:46.706+00:00	5	0	[0.01]	1
user_6754	2026-01-29T18:23:03.910+00:00	2026-01-29T18:53:38.315+00:00	4	1	350	0
user_3685	2026-01-29T19:01:24.349+00:00	2026-01-29T19:42:48.324+00:00	4	0	[0.01]	1
user_7342	2026-01-29T19:06:17.060+00:00	2026-01-29T19:36:17.060+00:00	1	0	[0.01]	0
user_9634	2026-01-29T19:05:27.819+00:00	2026-01-29T19:35:37.323+00:00	4	0	[0.01]	0
user_4338	2026-01-29T19:01:35.991+00:00	2026-01-29T19:38:59.912+00:00	10	1	5300	0
user_8960	2026-01-29T19:36:00.239+00:00	2026-01-29T20:06:18.123+00:00	3	0	[0.01]	1

↓ 4346 rows | 3.69s runtime

Refreshed 2 minutes ago

▶ ✓ 2 minutes ago (1s)

```
from pyspark.sql.functions import *
from pyspark.sql.types import *

df = spark.read.table("gold_session_stats")
```

▶ df: pyspark.sql.dataframe.DataFrame = [userid: string, window_start: timestamp ... 5 more fields]

```

    interrupt
silver_stream = spark.readStream.table("silver_clickstream")

gold_df = (
    silver_stream.groupBy(
        col("userId"),
        sessionWindow(col("event_time"), "30 minutes").alias("session_window"),
    )
    .agg(
        count("*").alias("events_count"),
        max(when(col("eventType") == "PURCHASED", 1).otherwise(0)).alias("is_purchased"),
        max(when(col("eventType") == "ADD_TO_CART", 1).otherwise(0)).alias(
            "has_cart_activity"
        ),
        sum(col("total_amount")).alias("session_revenue"),
        min("event_time").alias("session_start"),
        max("event_time").alias("session_end"),
    )
    .select(
        col("userId"),
        col("session_window.start").alias("window_start"),
        col("session_window.end").alias("window_end"),
        col("events_count"),
        col("is_purchased"),
        col("session_revenue"),
        when((col("has_cart_activity") == 1) & (col("is_purchased") == 0), 1)
        .otherwise(0)
        .alias("is_abandoned_cart"),
    )
)

gold_query = (
    gold_df.writeStream.format("delta")
    .outputMode("complete")
    .option("checkpointLocation", f"{base_path}/checkpoints/gold")
    .table("gold_session_state")
)

```

(1) Spark Jobs # 1

```

> 1f13c35-c75-4a72-89e6-07ffab2c7d7 Last updated: 1 minute ago
> └── gold_df: pyspark.sql.DataFrame = [userId string, window_start timestamp ... 5 more fields]
> └── silver_stream: pyspark.sql.DataFrame = [eventId string, eventType string ... 8 more fields]

```



```

    interrupt
json_schema = StructType([
    StructField("eventId", StringType(), True),
    StructField("eventType", StringType(), True),
    StructField("timeStamp", TimestampType(), True),
    StructField("userId", StringType(), True),
    StructField("sessionId", StringType(), True),
    StructField("pageUrl", StringType(), True),
    StructField("device",
        StructType([
            StructField("type", StringType(), True),
            StructField("os", StringType(), True),
            StructField("ip", StringType(), True),
        ]),
        True,
    ),
    StructField("data",
        StructType([
            StructField("productId", StringType(), True),
            StructField("productName", StringType(), True),
            StructField("price", DoubleType(), True),
            StructField("currency", StringType(), True),
            StructField("searchQuery", StringType(), True),
            StructField("totalAmount", DoubleType(), True),
        ]),
        True,
    ),
),
])

```

```

bronze_df = spark.readStream.table("bronze_clickstream")

silver_df = (
    bronze_df.select(from_json(col("body").cast("string"), json_schema).alias("parsed"))
    .select("parsed.*")
    .withColumn("id", col("eventId").cast("string"))
    .withColumn("total_amount", col("data.totalAmount"))
    .withWatermark("event_time", "10 minutes")
    .dropDuplicates(["eventId", "event_time"])
)

```

```

silver_query = (
    silver_df.writeStream.format("delta")
    .outputMode("append")
    .option("checkpointLocation", f"{base_path}/checkpoints/silver")
    .table("silver_clickstream")
)

```

(1) Spark Jobs # 2

```

> 48bc0eff-6aae-4319-a755-f1bb5b19e8ff Last updated: 25 seconds ago
> └── bronze_df: pyspark.sql.DataFrame = [body binary, partition string ... 7 more fields]
> └── silver_df: pyspark.sql.DataFrame = [eventId string, eventType string ... 8 more fields]

```

```
8
top_users_df = (
    df.groupBy("userId")
    .agg(
        sum("session_revenue").alias("lifetime_value"),
        count("*").alias("total_sessions"),
        sum("is_purchased").alias("total_purchases"),
    )
    .orderBy(col("lifetime_value").desc())
    .limit(10)
)

display(top_users_df)
▶ (4) Spark Jobs
> top_users_df: pyspark.sql.dataframe.DataFrame = [userId: string, lifetime_value: double ... 2 more fields]

Table + 
+---+
| A0 userId | 12 lifetime_value | 13 total_sessions | 13 total_purchases |
+---+
| 1 user_7761 | 11000 | 2 | 2 |
| 2 user_8588 | 9000 | 1 | 1 |
| 3 user_1987 | 9000 | 1 | 1 |
| 4 user_9724 | 9000 | 1 | 1 |
| 5 user_7165 | 9000 | 2 | 1 |
| 6 user_2049 | 6500 | 2 | 2 |
| 7 user_5885 | 5700 | 2 | 2 |
| 8 user_4733 | 5700 | 1 | 1 |
| 9 user_5476 | 5700 | 1 | 1 |
| 10 user_6176 | 5700 | 2 | 1 |
+---+
↓ 10 rows | 6.10s runtime
Refreshed 3 minutes ago
```

```
7
conversion_df = (
    df.select(lit("All Sessions").alias("step"), count("*").alias("count"))
    .union(
        df.filter(col("is_abandoned_cart") == 1).select(
            lit("Cart Activity").alias("step"), count("*").alias("count")
        )
    )
    .union(
        df.filter(col("is_purchased") == 1).select(
            lit("Purchased").alias("step"), count("*").alias("count")
        )
    )
)

display(conversion_df)
▶ (7) Spark Jobs
> conversion_df: pyspark.sql.dataframe.DataFrame = [step: string, count: long]

Table + 
+---+
| A0 step | 13 count |
+---+
| 1 Cart Activity | 1096 |
| 2 Purchased | 714 |
| 3 All Sessions | 4346 |
+---+
↓ 3 rows | 6.78s runtime
Refreshed 3 minutes ago
```

6

```

revenue_df = (
    df.filter(col("session_revenue") > 0)
    .select("userId", "session_revenue", "window_start")
    .orderBy(col("window_start").desc())
)

display(revenue_df)

```

(1) Spark Jobs

revenue_df: pyspark.sql.dataframe.DataFrame = [userId: string, session_revenue: double ... 1 more field]

Table +

#	userId	session_revenue	window_start
1	user_2049	1200	2026-01-29T20:00:13.746+00:00
2	user_5799	350	2026-01-29T20:00:07.469+00:00
3	user_3126	120.5	2026-01-29T19:59:45.152+00:00
4	user_1837	350	2026-01-29T19:59:38.628+00:00
5	user_7629	800	2026-01-29T19:59:29.627+00:00
6	user_6881	120.5	2026-01-29T19:59:29.627+00:00
7	user_4054	120.5	2026-01-29T19:59:24.491+00:00
8	user_7360	1200	2026-01-29T19:59:20.035+00:00
9	user_5885	4500	2026-01-29T19:59:12.769+00:00
10	user_9110	120.5	2026-01-29T19:58:54.813+00:00
11	user_5419	120.5	2026-01-29T19:58:53.016+00:00
12	user_2461	350	2026-01-29T19:58:53.016+00:00
13	user_3857	120.5	2026-01-29T19:58:48.048+00:00
14	user_7902	120.5	2026-01-29T19:58:26.827+00:00
15	user_7349	1200	2026-01-29T19:58:06.586+00:00

↓ 714 rows | 3.26s runtime

Refreshed 2 minutes ago

5

```

timeline_df = (
    df.groupBy(window(col("window_start"), "1 hour").alias("time_window"))
    .agg(
        count("*").alias("sessions"),
        sum("is_abandoned_cart").alias("abandoned_count"),
        sum("is_purchased").alias("purchases_count"),
    )
    .orderBy("time_window")
)

display(timeline_df)

```

(2) Spark Jobs

timeline_df: pyspark.sql.dataframe.DataFrame = [time_window: struct, sessions: long ... 2 more fields]

Table +

#	time_window	sessions	abandoned_count	purchases_count
1	> ["start": "2026-01-29T17:00:00Z", "end": "2026-01-29T18:00:00Z"]	110	27	14
2	> ["start": "2026-01-29T18:00:00Z", "end": "2026-01-29T19:00:00Z"]	2168	545	363
3	> ["start": "2026-01-29T19:00:00Z", "end": "2026-01-29T20:00:00Z"]	2053	520	335
4	> ["start": "2026-01-29T20:00:00Z", "end": "2026-01-29T21:00:00Z"]	15	4	2

↓ 4 rows | 3.30s runtime

Refreshed 2 minutes ago

2

```

connectionString = f"{eh_conn_str};EntityPath={eh_name}"
ehConf = {
    "eventhubs.connectionString": sc._jvm.org.apache.spark.eventhubs.EventHubsUtils.encrypt(
        connectionString
    )
}

raw_stream_df = spark.readStream.format("eventhubs").options(**ehConf).load()

bronze_query = (
    raw_stream_df.writeStream.format("delta")
    .outputMode("append")
    .option("checkpointLocation", f"{base_path}/checkpoints/bronze")
    .table("bronze_clickstream")
)

```

(1) Spark Jobs

78440e2e-2359-4a8f-a3f2-3a6c61c935d Last updated: 35 seconds ago

raw_stream_df: pyspark.sql.dataframe.DataFrame = [body: binary, partition: string ... 7 more fields]

06:56 PM (2s)

```

import os

from pyspark.sql.functions import *
from pyspark.sql.types import *

eh_conn_str = os.environ.get("EVENT_HUB_CONN_STR")
eh_name = os.environ.get("EVENT_HUB_NAME")
storage_name = os.environ.get("STORAGE_ACCOUNT_NAME")
storage_key = os.environ.get("STORAGE_ACCOUNT_KEY")

spark.conf.set(f"fs.azure.account.key.{storage_name}.dfs.core.windows.net", storage_key)
base_path = f"abfss://data@{storage_name}.dfs.core.windows.net"

```

Compute > Single from: ON >

Clickstream Cluster

Metrics

CPU utilization and active nodes

Memory utilization and swap

Network received and transmitted

Free memory space