Apache Flink Streaming Application - Комплетна Документација

1. Преглед на Системот

Ова е комплетен streaming систем кој користи Apache Kafka и Apache Flink за процесирање на податоци од сензори во реално време.

Архитектура

```
Producer → Kafka (sensors) → PyFlink App → Kafka (results1, results2) → Consumer
```

Компоненти:

- **Producer**: Генерира random податоци од сензори
- **Kafka**: Messaging систем (topic: sensors, results1, results2)
- **PyFlink Application**: Процесира податоци со временски прозорци
- **Consumer**: Прикажува резултати

---

2. Producer - Документација

Цел

Producer-от генерира симулирани податоци од сензори и ги испраќа на Kafka topic `sensors` во реално време.

Конфигурација

```
bootstrap_servers = 'localhost:9092'
topic = 'sensors'
keys = ['A', 'B', 'C', 'D']  # ID на сензори
```

Генерирање на Податоци

За секоја итерација:

1. Избира random клуч (A, B, C или D)
2. Генерира random вредност (0-1000)
3. Додава timestamp во милисекунди
4. Го сериализира record-от во JSON
5. Го испраќа на Kafka

Формат на Податоци

```
{
  "key": "A",
  "value": 495,
  "timestamp": 1764347836000
}
```

Полиња:

- `key`: Идентификатор на сензорот (A/B/C/D)
- `value`: Мерена вредност (0-1000)
- `timestamp`: Unix timestamp во ms

Карактеристики

- **Интервал**: Random 500-2000ms помеѓу пораки
- **Serialization**: JSON → UTF-8 bytes
- **Security**: PLAINTEXT (без енкрипција)

Пример Output

```
Sending: {"key": "A", "value": 495, "timestamp": 1764347836000}
Sending: {"key": "B", "value": 326, "timestamp": 1764347838500}
Sending: {"key": "D", "value": 850, "timestamp": 1764347841200}
```

3. PyFlink Application - Документација

Цел

Процесира податоци од сензори користејќи временски прозорци (sliding windows) и испраќа агрегирани резултати на два излезни topics.

Барања (Assignment Requirements)

Барање 1 - results1:

- Групирање по клуч (key)
- Sliding window со големина X ms и лизгање Y ms
- Output: Број на пораки во секој прозорец

Барање 2 - results2:

- Групирање по клуч (key)
- Sliding window со големина X ms и лизгање Y ms
- Output: Агрегации (min, max, average, count)

Конфигурација на Прозорци

```
WINDOW_SIZE_MS = 5000  # X ms - Големина на прозорец (5 сек)
SLIDE_SIZE_MS = 2000   # Y ms - Лизгање (2 сек)
```

Sliding Window Логика:

- Прозорец: 5 секунди
- Се лизга на секои 2 секунди
- Преклопување: Една порака припаѓа на повеќе прозорци

Пример:

```
Прозорец 1: [0s - 5s]
Прозорец 2: [2s - 7s]   ← преклопување
Прозорец 3: [4s - 9s]   ← преклопување
```

Излезен Формат

Topic: results1

```
{
  "key": "A",
  "window_start": 1669748895000,
  "window_end": 1669748900000,
  "count": 10
}
```

Topic: results2

```
{
  "key": "B",
  "window_start": 1669748895000,
  "window_end": 1669748900000,
  "min_value": 10,
  "count": 10,
  "average": 55.55,
  "max_value": 100
}
```

Главни Функции

`get_all_window_starts(timestamp)`

- Наоѓа сите прозорци на кои припаѓа timestamp-от
- Овозможува sliding window behavior

`process_window(key, window_start, values)`

- Пресметува агрегации (min, max, avg, count)
- Испраќа резултати на results1 и results2

`check_and_emit_windows(current_time)`

- Проверува кои прозорци се завршени
- Ги процесира и брише од меморијата

Работен Процес

1. **Читање**: Чита од Kafka topic `sensors`
2. **Доделување**: Ја доделува пораката на сите релевантни прозорци
3. **Складирање**: Ги чува вредностите во memory state
4. **Процесирање**: Кога прозорецот завршува, пресметува агрегации
5. **Испраќање**: Резултатите ги праќа на results1 и results2

Пример Output

```
Received: Key=A, Value=495, Timestamp=1764347836000
[RESULTS1] Key: A, Window: [1764347836000-1764347841000], Count: 2
[RESULTS2] Key: A, Window: [1764347836000-1764347841000], Min: 495, Avg: 741.0,
Max: 987
```

4. Consumer - Документација

Цел

Consumer-от ги чита и прикажува резултатите од PyFlink апликацијата од двата излезни topics.

Конфигурација

```
bootstrap_servers = 'localhost:9092'
topics = ["results1", "results2"]
group_id = 'results_consumer_group'
```

Карактеристики

- **Auto-commit**: Enabled (автоматски commit на offset)
- **Offset reset**: Earliest (чита од почеток ако нема зачуван offset)
- **Deserialization**: JSON автоматски parsing

Форматирање на Output

За results1:

```
[RESULTS1] Key: A, Window: [1764347836000-1764347841000], Count: 2
```

За results2:

```
[RESULTS2]
{"key":"A","window_start":1764347836000,"window_end":1764347841000,"min_value":495
,"count":2,"average":741.0,"max_value":987}
```

Работен Процес

1. Се конектира на Kafka
2. Се претплатува на topics: results1, results2
3. Континуирано poll за нови пораки
4. Десериализира JSON
5. Форматира и прикажува според topic

Docker Compose Конфигурација

```yaml
version: '2.4'
services:
  zookeeper:
    image: pr92918/zookeeper3.8
    expose:
      - 2181

  kafka:
    image: wurstmeister/kafka:2.11-2.0.0
    ports:
      - 9092:9092
    environment:
      KAFKA_ADVERTISED_LISTENERS: INSIDE://kafka:9093,OUTSIDE://localhost:9092
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
```

Сликата за zookeeper ја сменив од 3.4.6 во 3.8 бидејќи ми пишуваше дека е outdated и при секој docker-compose up -d ми прекинуваше процесот по некое време.

Очекуван Работен Процес

1. **Producer** генерира и испраќа податоци на `sensors`
2. **PyFlink** ги процесира со временски прозорци
3. **PyFlink** испраќа резултати на `results1` и `results2`
4. **Consumer** ги прикажува резултатите

Валидација на Излез

Producer треба да испраќа:

```
Sending: {"key": "A", "value": 495, "timestamp": 1764347836000}
```

PyFlink треба да процесира:

```
[RESULTS1] Key: A, Window: [...], Count: X
[RESULTS2] Key: A, Window: [...], Min: X, Avg: X, Max: X
```

Consumer треба да прима:

```
[RESULTS1] Key: A, Window: [...], Count: X
[RESULTS2] {...}
```