DATA+AI SUMMIT BY databricks

UNIT TESTS: HOW TO OVERCOME CHALLENGES IN STRUCTURED STREAMING

Bartosz Konieczny | freelance data engineer | contact@waitingforcode.com

Bartosz Konieczny

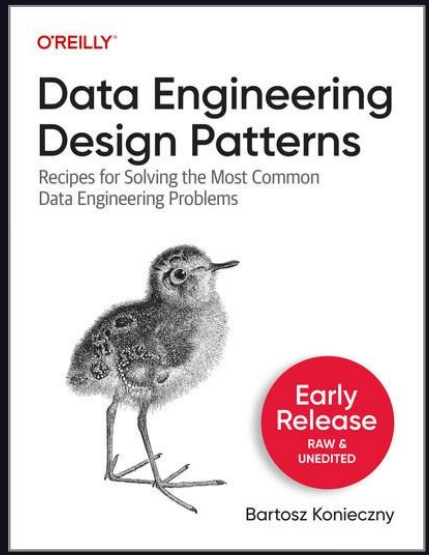
Freelance data engineer and trainer

✉️ contact@waitingforcode.com

Current focus:

🌊 stream processing

✍️ "Data Engineering Design Patterns"



Blogging: waitingforcode.com

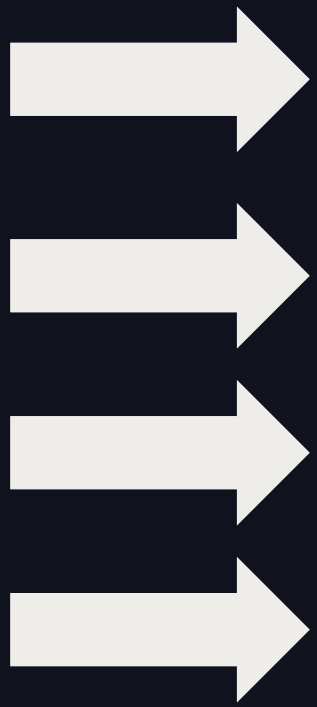
🤝 https://www.linkedin.com/in/bartosz-konieczny-waitingforcode/

2

Challenge 01:

STRUCTURING THE CODE AND THE DATAFLOW MODEL

DATAFLOW MODEL 101



...results your job generates

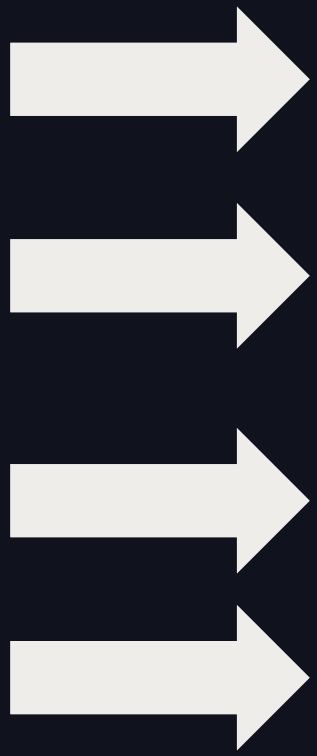
...the job computes the results (event time)

...the job materializes the results (processing time)

...the job handles accumulated data

DATAFLOW MODEL 101

AND APACHE SPARK STRUCTURED STREAMING



WHAT

...results your job generates

...transformation logic

WHERE

...the job computes the results (event time)

...watermark

WHEN

...the job materializes the results (processing time)

... micro-batch, trigger

HOW

...the job handles accumulated data

...output mode

DATAFLOW MODEL 101

AND APACHE SPARK STRUCTURED STREAMING

WHAT

WHERE

WHEN

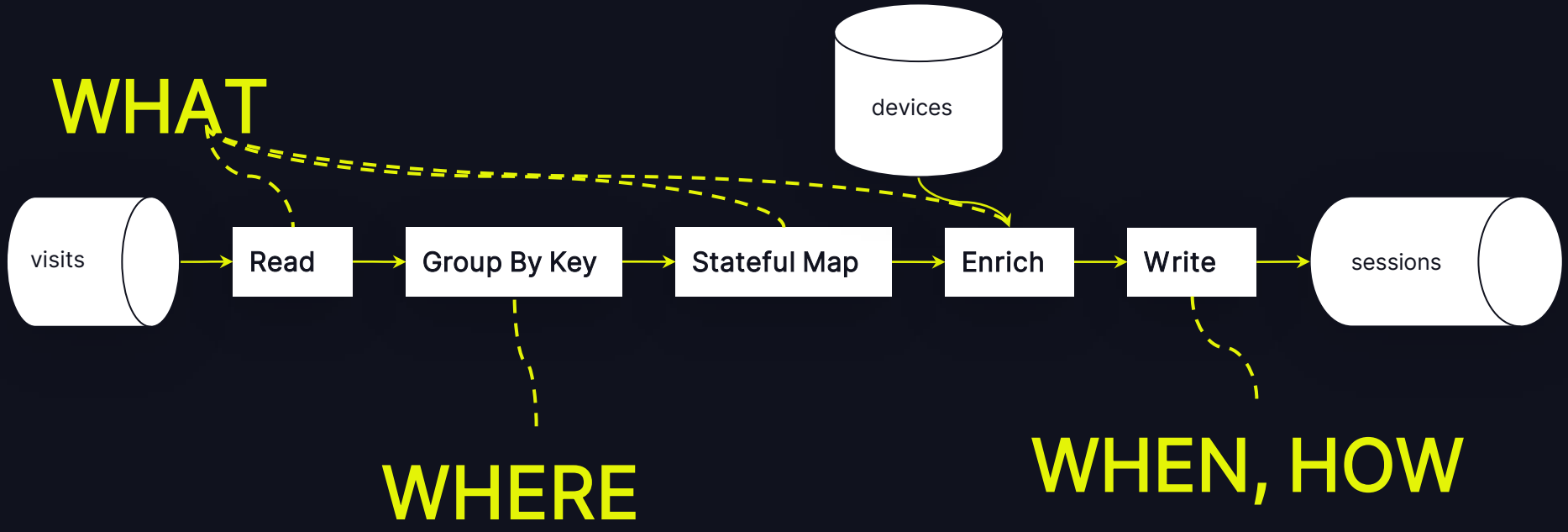
HOW

Project organization → Unit Tests specification

EXAMPLE - THE JOB

💡no I/O if it's provided, it's a matter of configuration only

Stateful processing for https://github.com/bartosz25/data-generator-blogging-platform



UNIT TESTS SPECIFICATION - SNIPPETS

WHAT

reader.py

```
def select_raw_visits(visits_source: DataFrame) -> DataFrame:
   visit_schema = '''
       visit_id STRING, event_time TIMESTAMP, user_id STRING, page STRING,
       ...
   '''
   return (visits_source.select(
       F.from_json(F.col('value').cast('string'), visit_schema).alias('value'))
           .selectExpr('value.*'))
```

UNIT TESTS SPECIFICATION - SNIPPETS

WHERE

sessions_generation_job_logic.py

```
def generate_sessions(visits_source: DataFrame, devices: DataFrame, trigger: Dict[str, str],
                      checkpoint_location: str)->DataStreamWriter:
    raw_visits = select_raw_visits(visits_source)

    grouped_visits = (raw_visits
                      .withWatermark('event_time', '5 minutes')
                      .groupBy(F.col('visit_id')))
    # ...
    return set_up_sessions_writer(sessions_to_write, trigger).option('checkpointLocation', checkpoint_location)
```

UNIT TESTS SPECIFICATION - SNIPPETS

WHEN, HOW

writer.py

```
def set_up_visits_writer(visits_to_write: DataFrame, trigger: Dict[str, str]) -> DataStreamWriter:
   return (visits_to_write.writeStream
           .outputMode("append")
           .trigger(**trigger))
```

EXAMPLE - THE JOB CODE

sessions_generation_job.py

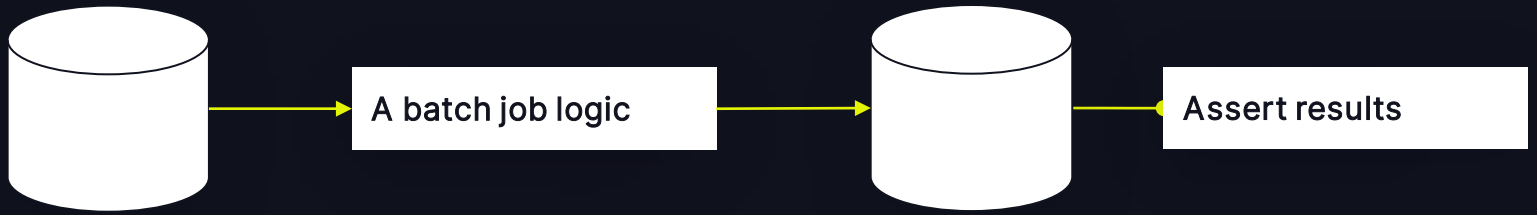
```
input_data_stream = (spark_session.readStream
  .option('kafka.bootstrap.servers', job_args.kafka_servers)
  .option('subscribe', job_args.input_topic).format('kafka')
  .load())

devices_table: DataFrame = spark_session.read.format('delta').load(job_args.devices_table_location)

visits_writer = generate_sessions(input_data_stream, devices_table, {'processingTime': '30 seconds'},
                job_args.checkpointLocation)

write_query = (visits_writer.format("kafka")
  .option('kafka.bootstrap.servers', job_args.kafka_servers).option('topic',job_args.output_topic)
  .start())

write_query.awaitTermination()
```

Challenge 02:

ORCHESTRATING ASSERTIONS IN STREAMING

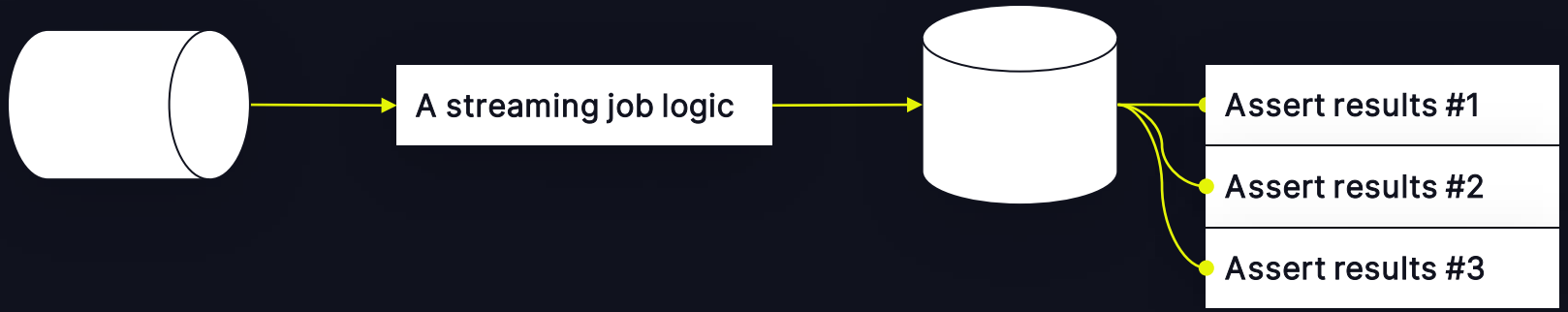
BATCH JOB - ASSERTIONS

Write once, check once



STREAMING JOB - ASSERTIONS

Write once, check once ... in a loop



🤔 Simulate the loop in a controlled manner

StreamingQuery#processAllAvailable

awaitTermination → processAllAvailable

PYTHON

```python
visit_writer = generate_visits(visits_reader, devices_to_test, {'processingTime': '0 seconds'})
started_query = visit_writer.foreachBatch(write_results_to_batch_id_partitioned_storage()).start()
started_query.processAllAvailable()
# ... assert on sessions after the first iteration

add_new_data_for_test(new_data)
visit_writer.processAllAvailable()
# ... assert on sessions after the second iteration

add_new_data_for_test(new_data)
visit_writer.processAllAvailable()
# ... assert on sessions after the third iteration
```

BUT . . .

Processing time trigger gotcha

processAllAvailable() waits the next trigger to confirm there is no new available!

```
assertAwaitThread()
// ...
if (!isActive) return
awaitProgressLock.lock()
try {
  noNewData = false
  while (true) {
    awaitProgressLockCondition.await(10000,
MILLISECONDS)
   // ...
   if (noNewData || !isActive) return
  }

 } finally {
  awaitProgressLock.unlock()
 }
```

⚠ Your production trigger =

Tests can be really slow!

What about availableNow?

📕 Scaladoc: *A Trigger that processes all available data in multiple batches then terminates the query.*

PYTHON

```
visit_writer = generate_visits(visits_reader, devices_to_test, {'availableNow': True})
query = (visit_writer.option('checkpointLocation', '/tmp/test2')
.foreachBatch(write_results_to_batch_id_partitioned_storage()))

started_query = query.start()
started_query.awaitTermination()
# ... assert on sessions after the first iteration

add_new_data_for_test(new_data)
started_query = query.start()
started_query.awaitTermination()

# ... *try to* assert on sessions after the second iteration
```

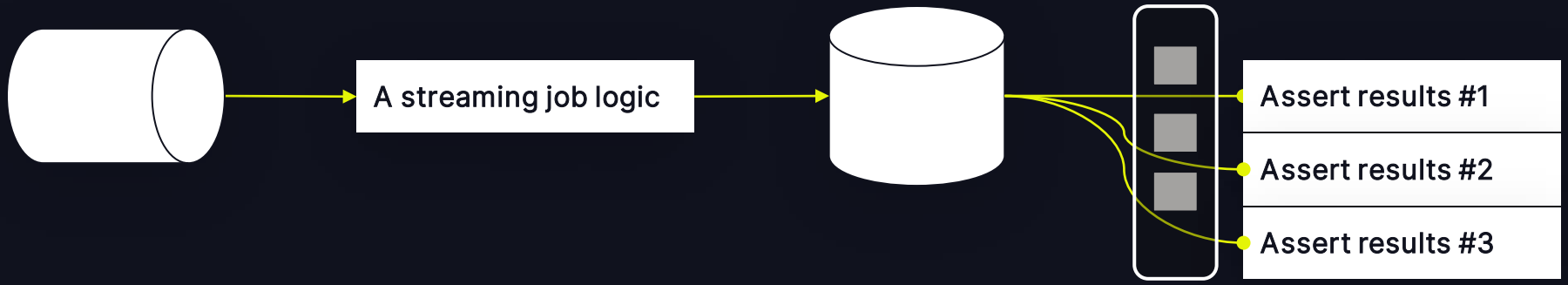
Works too, but semantically different from the processAllAvailable (not continuous processing)

Challenge 03:

SCOPING THE MICRO-BATCH, THE WHERE

STREAMING JOB - ASSERTIONS SCOPE

Knowing what to expect for a micro-batch



DataStreamWriter AND BATCH ID

Leveraging Apache Spark API

DataStreamWriter...

- Scaladoc: *Interface used to write a streaming Dataset to external storage systems (e.g. file systems, key-value stores, etc)*

PYTHON

```python
def generate_sessions(visits_source: DataFrame, devices: DataFrame, trigger: Dict[str, str],
                      checkpoint_location: str) -> DataStreamWriter:
    raw_visits = select_raw_visits(visits_source)

    grouped_visits = raw_visits.withWatermark('event_time', '5 minutes').groupBy(F.col('visit_id'))

    sessions = grouped_visits.applyInPandasWithState(func=map_visits_to_session,
      outputStructType=get_session_output_schema(), stateStructType=get_session_state_schema(),
      outputMode="append", timeoutConf="EventTimeTimeout")

    enriched_sessions = enrich_sessions_with_devices(sessions, devices)

    sessions_to_write = (enriched_sessions.withColumn('value', F.to_json(F.struct('*')))
      .selectExpr('visit_id AS key', 'value'))

    return set_up_sessions_writer(sessions_to_write, trigger).option('checkpointLocation', checkpoint_location)
```

DataStreamWriter...AND BATCH ID

PYTHON

```python
class DataToAssertWriterReader:
 def __init__(self, base_dir: str, spark_session: SparkSession):
   self.spark_session = spark_session
   self._results_per_micro_batch: Dict[int, List[Dict[str, Any]]] = {}

 def write_results_to_batch_id_partitioned_storage(self):
  def write_to_test_output_dir(dataframe: DataFrame, batch_number: int):
    rows = dataframe.collect()
    results = []
    for row in rows:
     results.append(json.loads(row.value))
    self._results_per_micro_batch[batch_number] = results
  return write_to_test_output_dir

 def get_dataframe_to_assert(self, batch_number: int) -> List[Dict[str, Any]]:
  return self._results_per_micro_batch[batch_number]
```

DataStreamWriter...AND BATCH ID

💡 Avoid assertions in the foreachBatch function; you'll need to switch context

PYTHON

```
started_query = visit_writer.foreachBatch(assertions_io.write_results_to_batch_id_partitioned_storage()).start()
started_query.processAllAvailable()

# ...
emitted_visits_4 = assertions_io.get_dataframe_to_assert(4)
assert_that(emitted_visits_4).is_empty()

emitted_visits_5 = assertions_io.get_results_to_assert_for_the_last_micro_batch()
assert_that(emitted_visits_5).is_not_empty()
assert_that(emitted_visits_5).is_length(2)
sorted_emitted_visits_5 = sorted(emitted_visits_5, key=lambda row: row['visit_id'])
assert_that(sorted_emitted_visits_5[0]['visit_id']).is_equal_to('v1')
assert_that(sorted_emitted_visits_5[1]['visit_id']).is_equal_to('v2')
# ...
```

⚠ *Flaky test, sometimes 3 final micro-batches instead of 2!*

In-memory table

⛔ No micro-batch id

PYTHON

```
visits_reader: DataFrame = spark_session.readStream.schema(visits_to_test.schema).json(visits_output_dir)
visit_writer = generate_visits(visits_reader, devices_to_test, {'processingTime': '0 seconds'})
started_query = visit_writer.format('memory').queryName('visits_once').start()
started_query.processAllAvailable()

add_new_data_to_test()
started_query.processAllAvailable()

add_new_data_to_test()
started_query.processAllAvailable()

spark_session.sql('SELECT * FROM visits_once').collect()

# assert...
```

Challenge 04:

EXECUTION TIME

they said, unit tests were fast...

LATENCY FACTORS

Why your unit tests may be slow?

Apache Spark configuration

- **spark.ui.enabled**
- **spark.sql.shuffle.partitions**
- maintenance jobs
  - spark.sql.streaming.stateStore.maintenanceInterval
  - spark.sql.streaming.stateStore.minDeltasForSnapshot
  - spark.sql.streaming.fileSource.log.compactInterval
- compaction
  - spark.sql.streaming.minBatchesToRetain
- If Delta Lake:
  - spark.databricks.delta.snapshotPartitions
  - spark.databricks.delta.log.cacheSize
  - spark.sql.sources.parallelPartitionDiscovery.parallelism

Your code / Dependencies

- `Thread.sleep(...) / time.sleep(...)`
- Processing time trigger
- Big test dataset
- Network and external dependencies

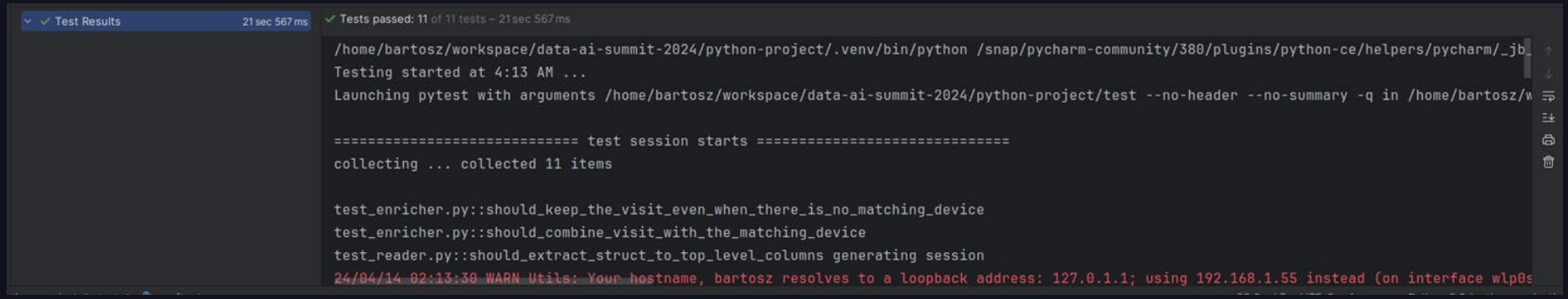
LATENCY FACTORS

Quick test...

Optimized

```
spark.ui.enabled=false

spark.sql.shuffle.partitions=2
```

21 sec 567 ms

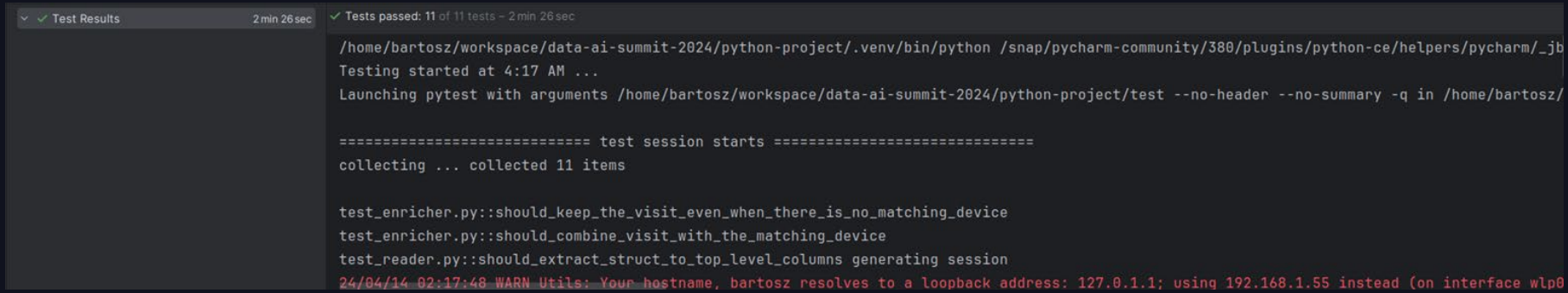


LATENCY FACTORS

Quick test...

Default

2 min 26 sec



Challenge 05:

"THINKLESS" ASSERTIONS

POLL#1

Have you already spent > 1 minute on understanding failures?

```
        assert item1 == item2
AssertionError


AssertionError: Expected
<test_abc.<locals>.InventoryItem(name='abc', unit_price=9.99,
delivery_countries=['us', 'uk', 'fr'], quantity_on_hand=3)> to
be equal to <test_abc.<locals>.InventoryItem(name='abc',
unit_price=9.99, delivery_countries=['de', 'us', 'uk', 'ie',
'pl'], quantity_on_hand=2)>, but was not.
```

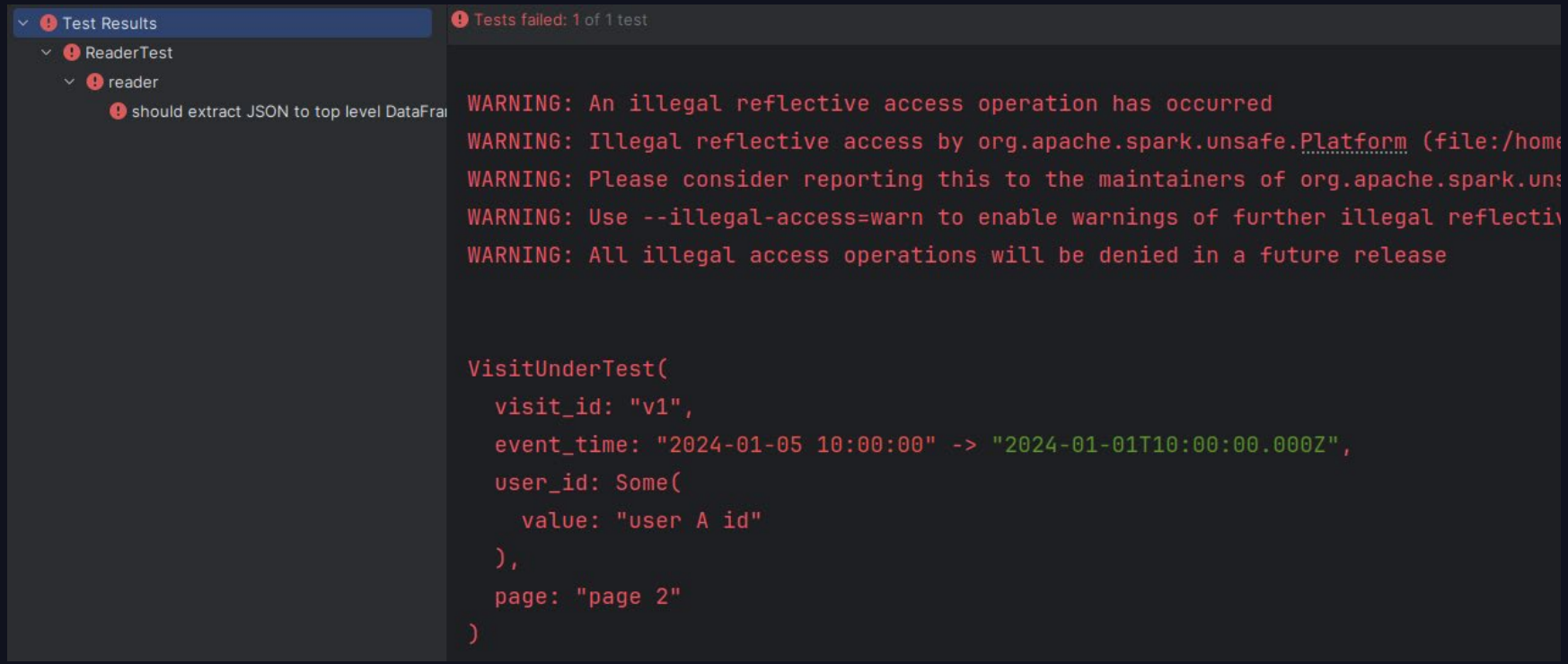
POLL#2

Can you spot the issue in less than 1 minute?

```
>       assert item1 == item2
E       AssertionError: assert == failed. [pytest-clarity diff shown]
E
E         LHS vs RHS shown below
E
E         should_test_abc.<locals>.InventoryItem(name='abc', unit_price=9.99,
E         delivery_countries=['us', 'uk', 'ie'], quantity_on_hand=0)
E         should_test_abc.<locals>.InventoryItem(name='abc', unit_price=9.99,
E         delivery_countries=['de', 'us', 'uk', 'ie', 'pl'], quantity_on_hand=2)
E
```

TRANSFORMED ASSERTIONS

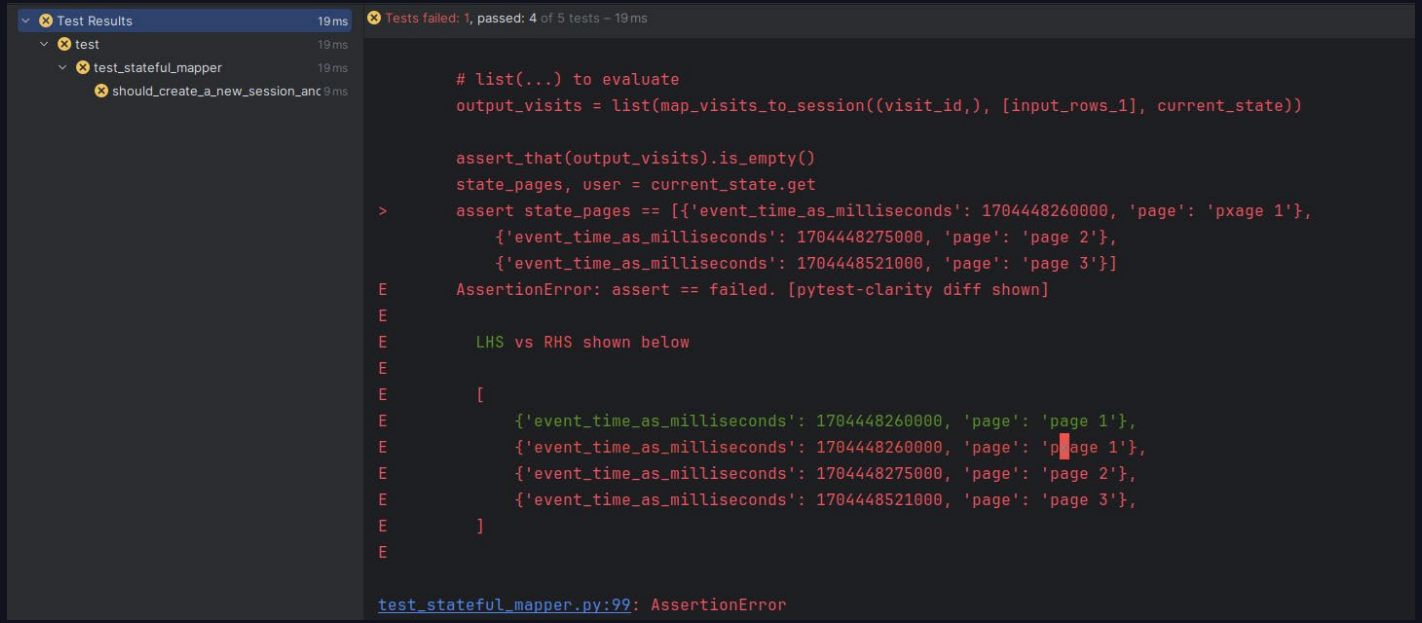
Scala Spark with difflicious



TRANSFORMED ASSERTIONS

PySpark with pytest-clarity

- ⛔ Library maintenance



TRANSFORMED ASSERTIONS

PySpark with pyassert

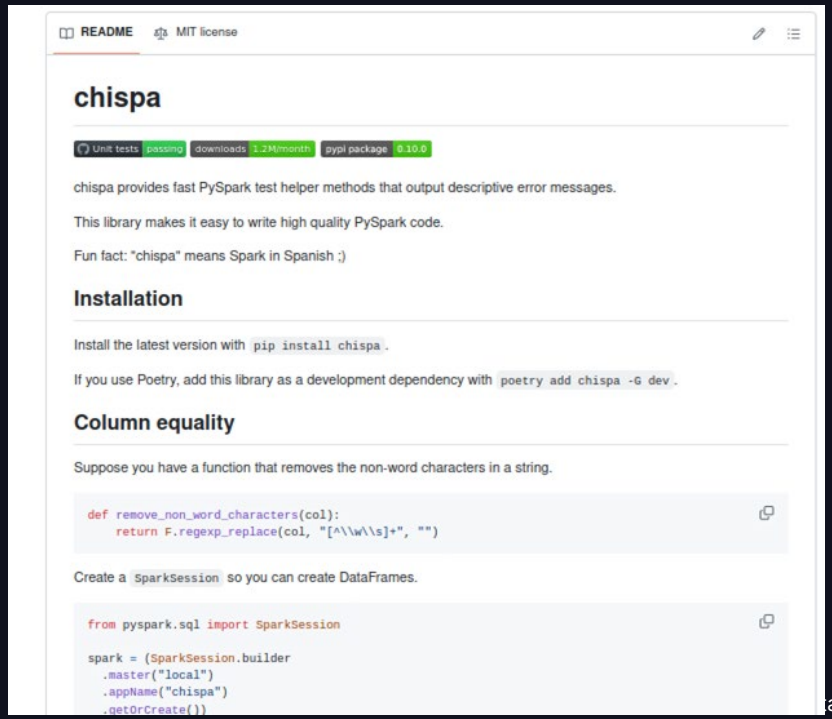
⛔ Requires changing the habits (no more 'assert x == y')

```
>       assert_that(state_pages).contains(
            {'event_time_as_milliseconds': 1704448260000, 'page': 'page 1'},
            {'event_time_as_milliseconds': 1704448275000, 'page': 'page 22'},
            {'event_time_as_milliseconds': 1704448521000, 'page': 'page 3'}
        )
E       AssertionError: Expected <[{'event_time_as_milliseconds': 1704448260000, 'page': 'page 1'}, {'event_time_as_milliseconds': 1704448275000, 'page': 'page 2'},
  {'event_time_as_milliseconds': 1704448521000, 'page': 'page 3'}]> to contain items <{'event_time_as_milliseconds': 1704448260000, 'page': 'page 1'},
 {'event_time_as_milliseconds': 1704448275000, 'page': 'page 22'}, {'event_time_as_milliseconds': 1704448521000, 'page': 'page 3'}>, but did not contain
 <{'event_time_as_milliseconds': 1704448275000, 'page': 'page 22'}>.
```

TRANSFORMED ASSERTIONS

DataFrame alternatives

https://github.com/MrPowers/chispa



README MIT license

chispa

Unit tests passing | downloads 1.2M/month | pypi package 0.10.0

chispa provides fast PySpark test helper methods that output descriptive error messages.

This library makes it easy to write high quality PySpark code.

Fun fact: "chispa" means Spark in Spanish ;)

Installation

Install the latest version with `pip install chispa`.

If you use Poetry, add this library as a development dependency with `poetry add chispa -G dev`.

Column equality

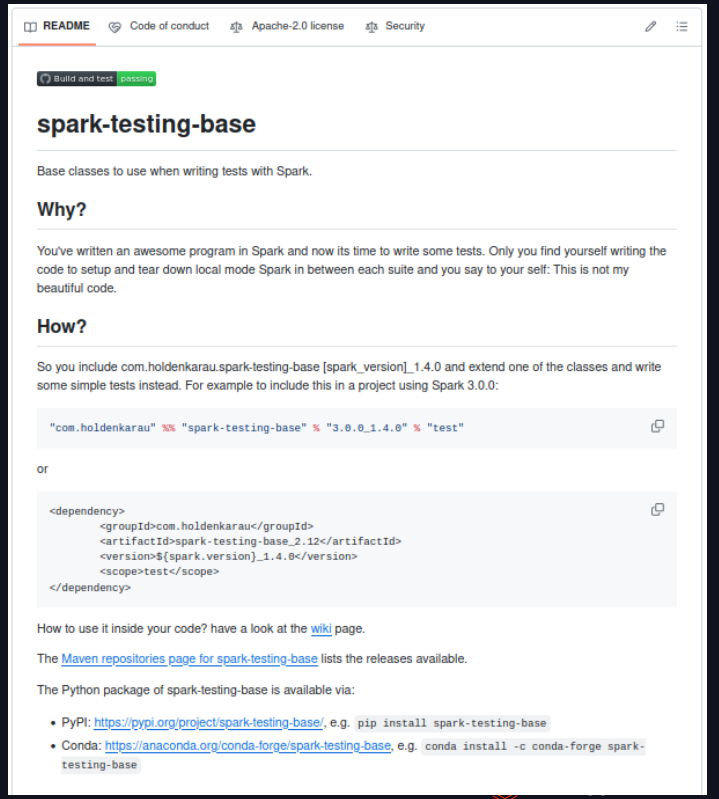
Suppose you have a function that removes the non-word characters in a string.

```
def remove_non_word_characters(col):
    return F.regexp_replace(col, "[^\\w\\s]+", "")
```

Create a `SparkSession` so you can create DataFrames.

```
from pyspark.sql import SparkSession

spark = (SparkSession.builder
  .master("local")
  .appName("chispa")
  .getOrCreate())
```

https://github.com/holdenk/spark-testing-base/



README Code of conduct Apache-2.0 license Security

Build and test passing

spark-testing-base

Base classes to use when writing tests with Spark.

Why?

You've written an awesome program in Spark and now its time to write some tests. Only you find yourself writing the code to setup and tear down local mode Spark in between each suite and you say to your self: This is not my beautiful code.

How?

So you include com.holdenkarau.spark-testing-base [spark_version]_1.4.0 and extend one of the classes and write some simple tests instead. For example to include this in a project using Spark 3.0.0:

```
"com.holdenkarau" %% "spark-testing-base" % "3.0.0_1.4.0" % "test"
```

or

```
<dependency>
        <groupId>com.holdenkarau</groupId>
        <artifactId>spark-testing-base_2.12</artifactId>
        <version>${spark.version}_1.4.0</version>
        <scope>test</scope>
</dependency>
```

How to use it inside your code? have a look at the wiki page.

The Maven repositories page for spark-testing-base lists the releases available.

The Python package of spark-testing-base is available via:

- PyPI: https://pypi.org/project/spark-testing-base/, e.g. `pip install spark-testing-base`
- Conda: https://anaconda.org/conda-forge/spark-testing-base, e.g. `conda install -c conda-forge spark-testing-base`

Challenge 06:

FLUENT AND PAIN-FREE DATASET GENERATION

is it hard to refresh a dataset?


```
{"visit_id": "140177386105728_1", "event_time": "2024-01-01T02:56:00+00:00", "user_id":
"140177386105728_cfeeaf2a-9a09-4989-9049-7d1982f2ec16", "keep_private": false, "page": "main", "context":
{"referral": "Twitter", "ad_id": null, "user": {"ip": "133.184.20.112", "login": "lauraburke",
"connected_since": "2023-12-28T00:00:00+00:00"}, "technical": {"browser": "Firefox", "browser_version":
"24.11", "network_type": "4G", "device_type": "Smartphone", "device_version": "3.0"}}}
{"visit_id": "140177386105728_10", "event_time": "2024-01-01T00:00:00+00:00", "user_id":
"140177386105728_ad4d7b1f-358a-42ea-9388-baec2fca06c7", "keep_private": false, "page": "category_19",
"context": {"referral": null, "ad_id": "ad 1", "user": {"ip": "18.106.61.69", "login": "jaredstewart",
"connected_since": "2023-12-05T00:00:00+00:00"}, "technical": {"browser": "Firefox", "browser_version":
"23.0", "network_type": "Wi-Fi", "device_type": "PC", "device_version": "1.0"}}}
{"visit_id": "140177386105728_2", "event_time": "2024-01-01T03:08:00+00:00", "user_id":
"140177386105728_ef9f7a75-277e-4dbb-8446-076a43237d66", "keep_private": false, "page": "category_8",
"context": {"referral": "StackOverflow", "ad_id": null, "user": {"ip": "60.135.200.110", "login": "sandra17",
"connected_since": null}, "technical": {"browser": "Chrome", "browser_version": "20.1", "network_type": "Wi-
Fi", "device_type": "iPad", "device_version": "1.0"}}}
{"visit_id": "140177386105728_3", "event_time": "2024-01-01T02:57:22+00:00", "user_id":
"140177386105728_0319cbab-73ff-4963-836b-072014e69ee0", "keep_private": false, "page": "categories",
"context": {"referral": "LinkedIn", "ad_id": "ad 5", "user": {"ip": "86.131.220.213", "login": "jasmin85",
"connected_since": null}, "technical": {"browser": "Safari", "browser_version": "20.0", "network_type": "4G",
"device_type": "iPad", "device_version": "1.0"}}}
```

BUILDER DESIGN PATTERN

Set up insignificant properties for the test, focus on the significant ones

PYTHON

```
def visit(visit_id='visit_1', event_time='2024-01-05T10:00:00.000Z', user_id='user A id',
    page='page1.html', referral='search', ad_id='ad 1',
    user_cxt=user_context(), technical_cxt=technical_context()) -> Visit:
  generated_visit = Visit(
    visit_id=visit_id, event_time=event_time, user_id=user_id, page=page,
    context=VisitContext(referral=referral, ad_id=ad_id, user=user_cxt,
     technical=technical_cxt
    )
  )
  DataGeneratorWrapper.add_visit(generated_visit)
  return generated_visit
```

BUILDER DESIGN PATTERN

💡 Keep the definition readable

PYTHON

```python
def should_extract_struct_to_top_level_columns(generate_spark_session):
 spark_session = generate_spark_session[0]
 visit_1, visit_2, visit_3, visit_4 = (
                  visit(visit_id='v1', event_time='2024-01-05T10:00:00.000Z', page='page 2'),
                  visit(visit_id='v1', event_time='2024-01-05T10:01:00.000Z', page='page 3'),
                  visit(visit_id='v1', event_time='2024-01-05T10:03:00.000Z', page='page 4'),
                  visit(visit_id='v2', event_time='2024-01-05T10:00:00.000Z', user_id=None, page='home_page'))
```

UNCOVERED CHALLENGES...

...because I have only few minutes left

- Challenge 07: Data representation skew (aka Who is guarding the guards, part 1)
- Challenge 08: Schema skew
- Challenge 09: Who is guarding the guards, part 2?
- Challenge 10: Who is guarding the guards, part 3?
- Challenge 11: Exhaustiveness
- Challenge 12: Slower and faster tests classification
- Challenge 13: External dependencies (IO)
- Challenge 14: Beyond unit tests, validation testing
- ...

TAKEAWAYS

- Dataflow model - test specification
- Project structure - enabler for testing
- I/O API - already in Apache Spark and tested by the community
- Explicitiveness - don't make me think
- User engagement - ease of definition
- Performance - disable the unnecessary components and decrease the numbers

RESOURCES

- Github repo: https://github.com/bartosz25/data-ai-summit-2024
- Books, blogs:
  - Streaming: The world beyond batch, part 1 and part 2, by Tyler Akidau
  - "Streaming systems" by Tyler Akidau, Slava Chernyak, Reuven Lax
- Talks:
  - Learn to Efficiently Test ETL Pipelines by Jacqueline Bilston
  - Productizing Structured Streaming Jobs by Burak Yavuz
  - My 25 Laws of Test Driven Development by Dennis Doomen
- Follow-up on waitingfocode.com/tags/dais2024

With full links 👇

Thank you!

Bartosz Konieczny | contact@waitingforcode.com