tenova

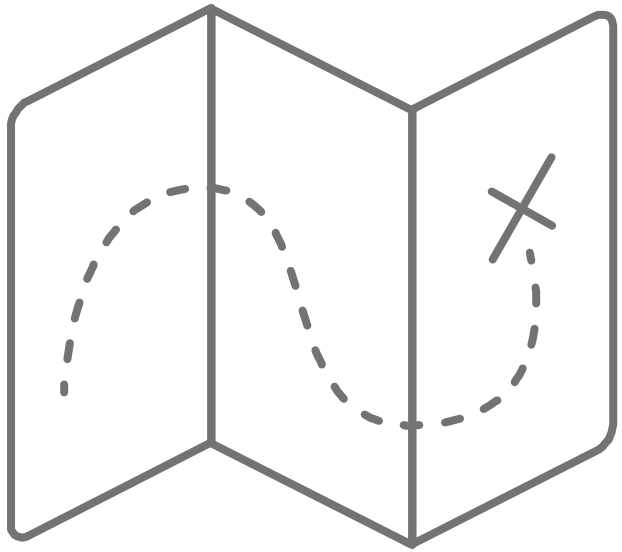
From telemetry data to CSVs with Python, Spark and Azure Databricks

EUROPYTHON 2021

Nicolò Giso

Agenda

- Background
- Data Applications
- Architecture
- Code
  - Some lines of codes
  - Other lines
  - And another bunch of them
- Orchestration
- Conclusion



Background

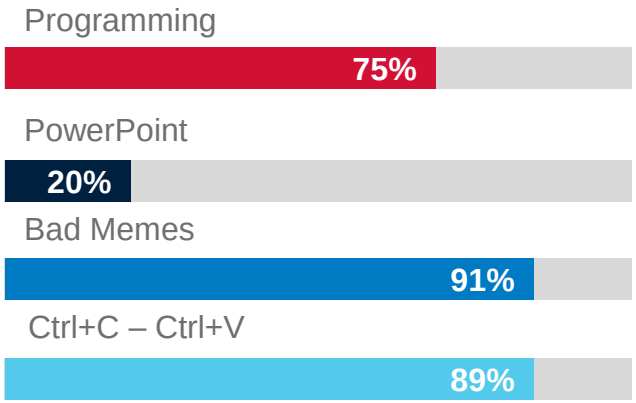
Let me introduce myself

Nicolò Giso

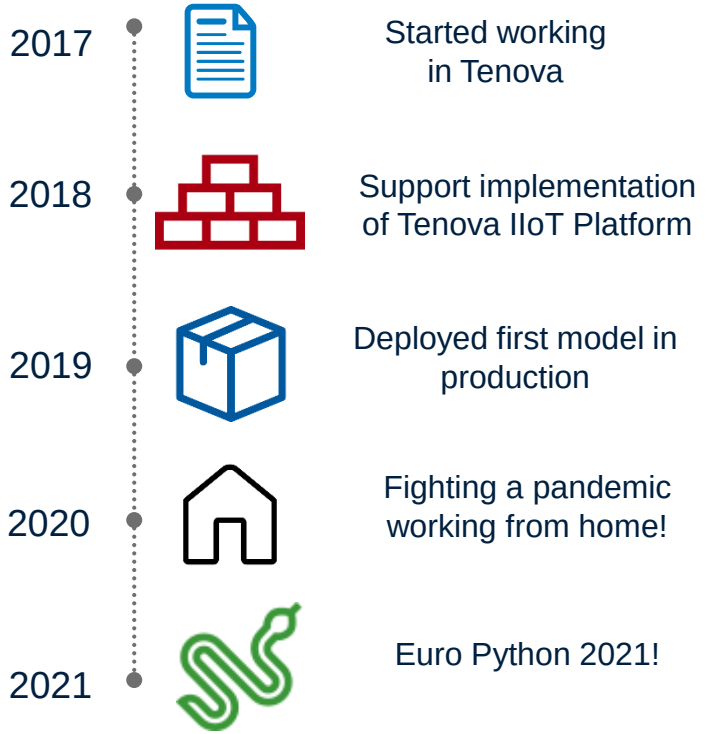
About Me

Role: Data Scientist
Company: Tenova
Interests: books, tv series, data

Professional Skills



My History



TENOVA, A TECHINT GROUP COMPANY, IS YOUR WORLDWIDE PARTNER
FOR **SUSTAINABLE, INNOVATIVE AND RELIABLE SOLUTIONS**
IN THE METALS AND –THROUGH THE WELL-KNOWN TAKRAF AND DELKOR BRANDS
– IN **THE MINING** INDUSTRIES.

WE DESIGN AND DEVELOP SOLUTIONS THAT HELP COMPANIES TO:

REDUCE COSTS

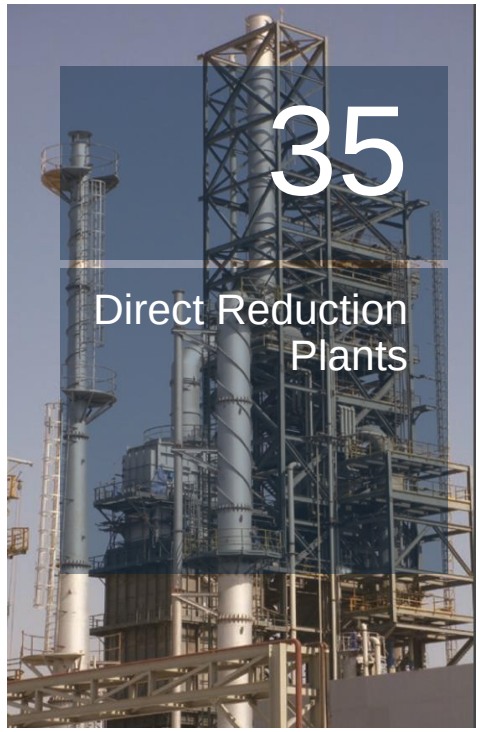
SAVE ENERGY

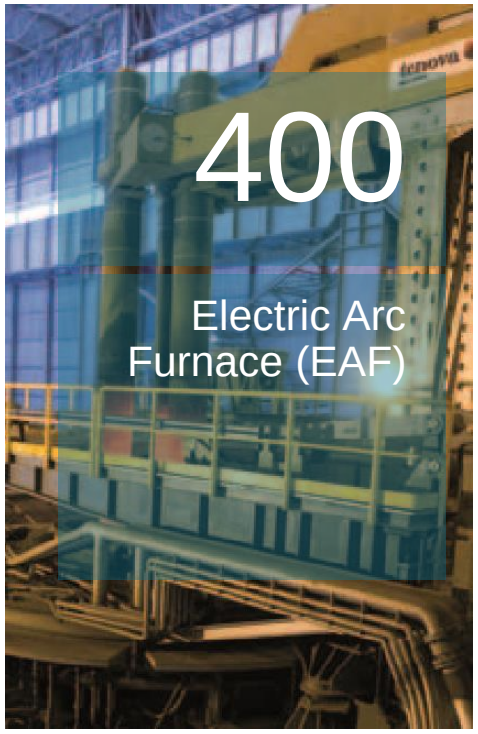
LIMIT ENVIRONMENTAL IMPACT

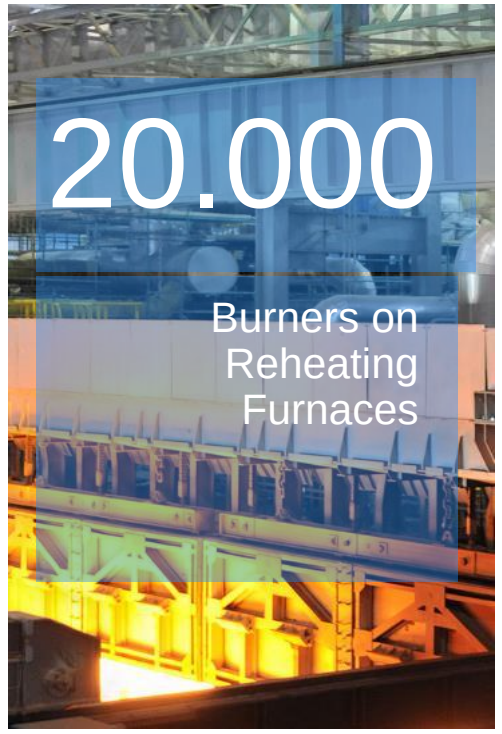
IMPROVE WORKING CONDITIONS

Tenova

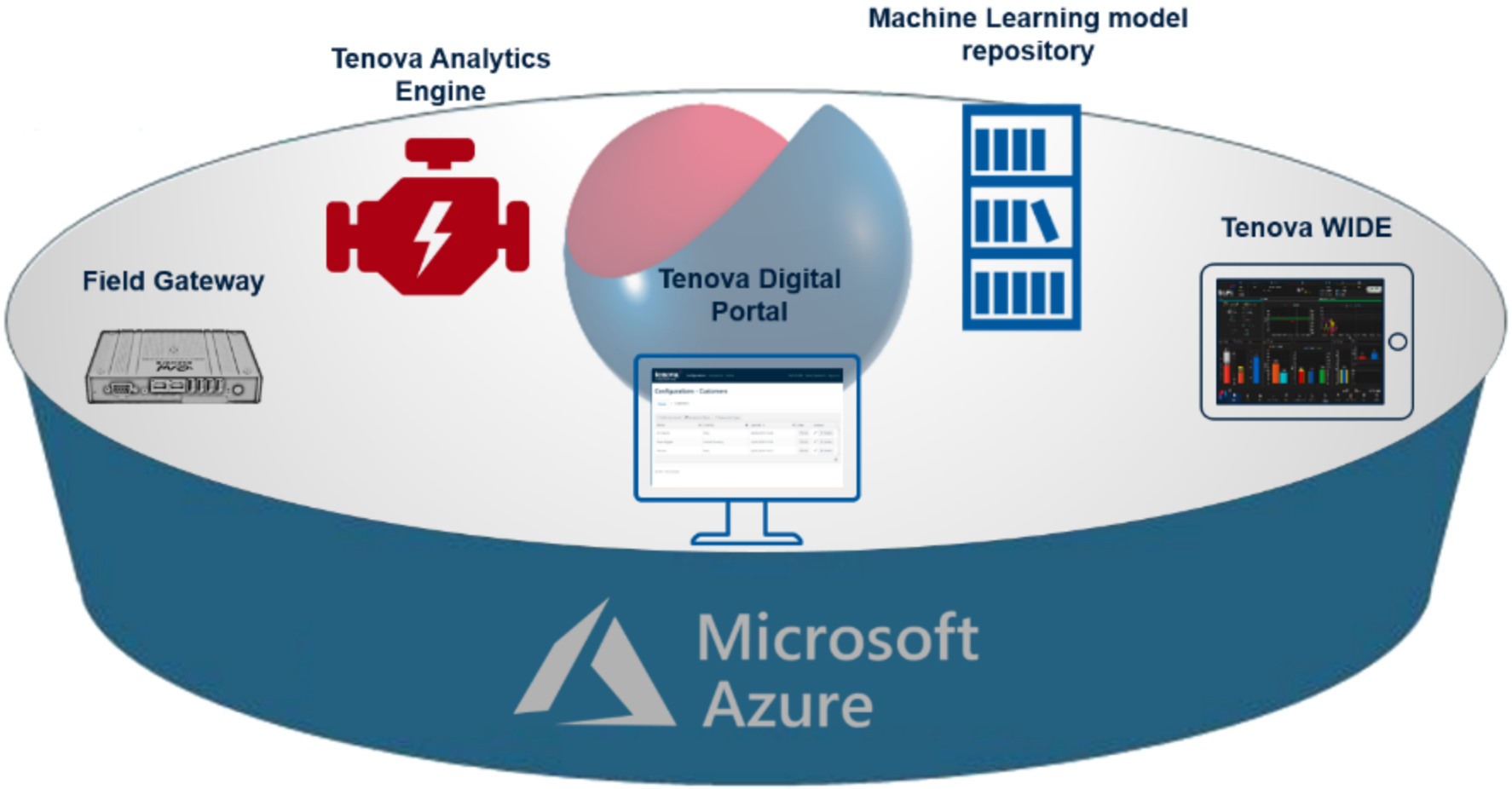
INSTALLED BASE





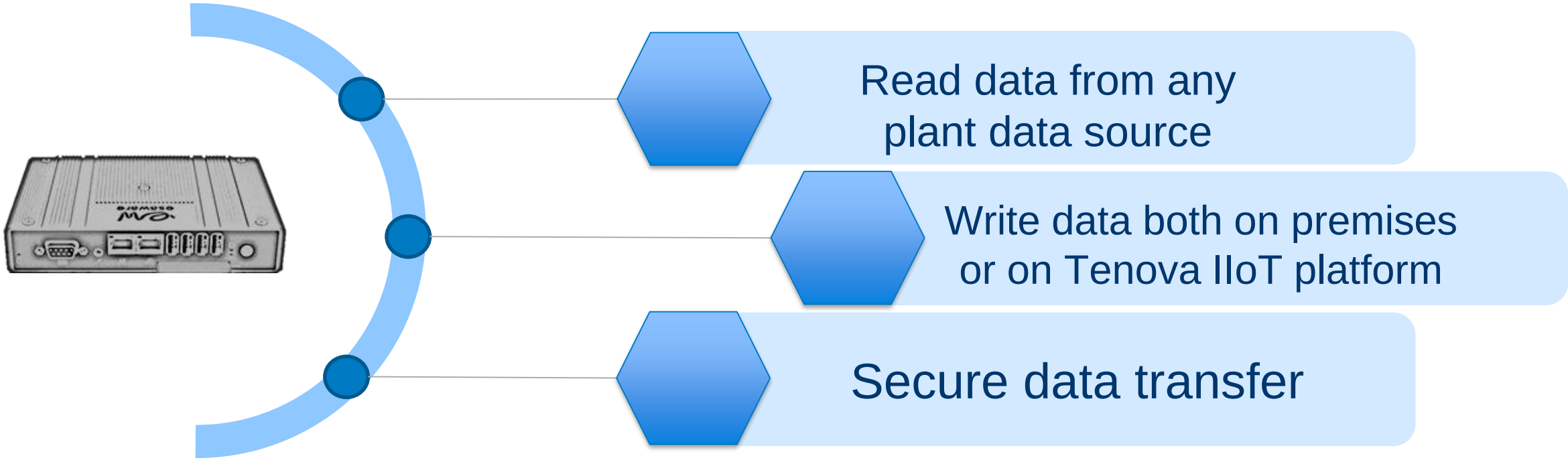


Tenova IIoT platform



Tenova IIoT platform

TENOVA EDGE

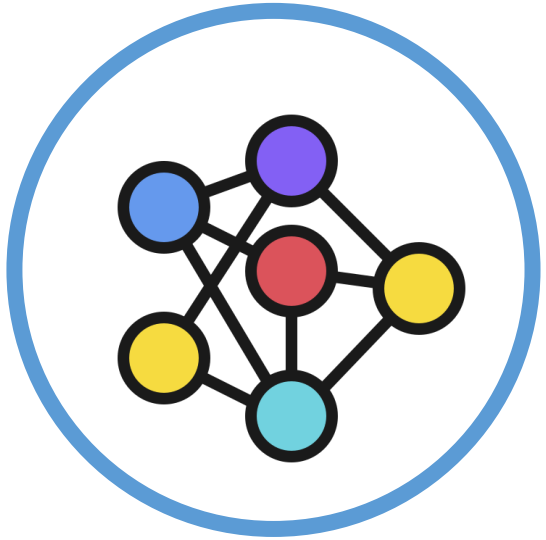


Data Applications

Data Applications

Monitoring

Data Analysis

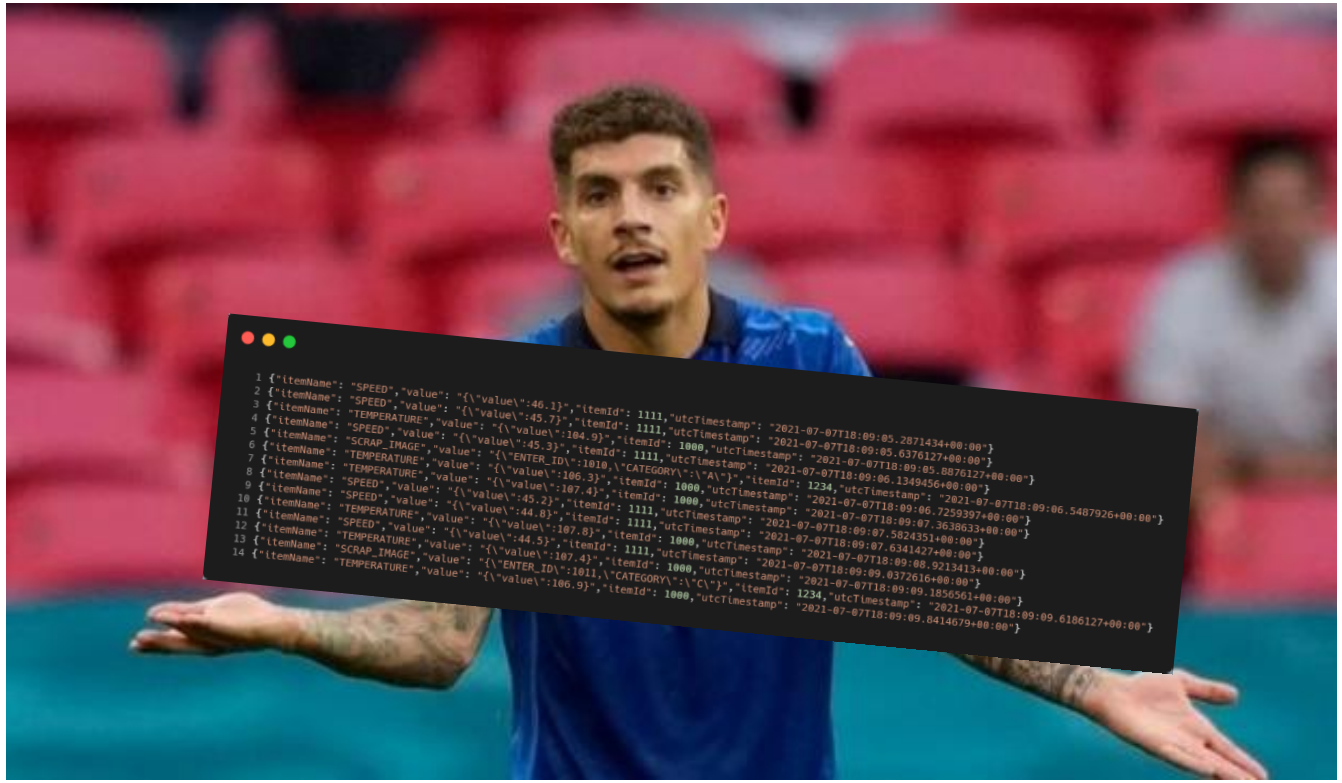


ML Models

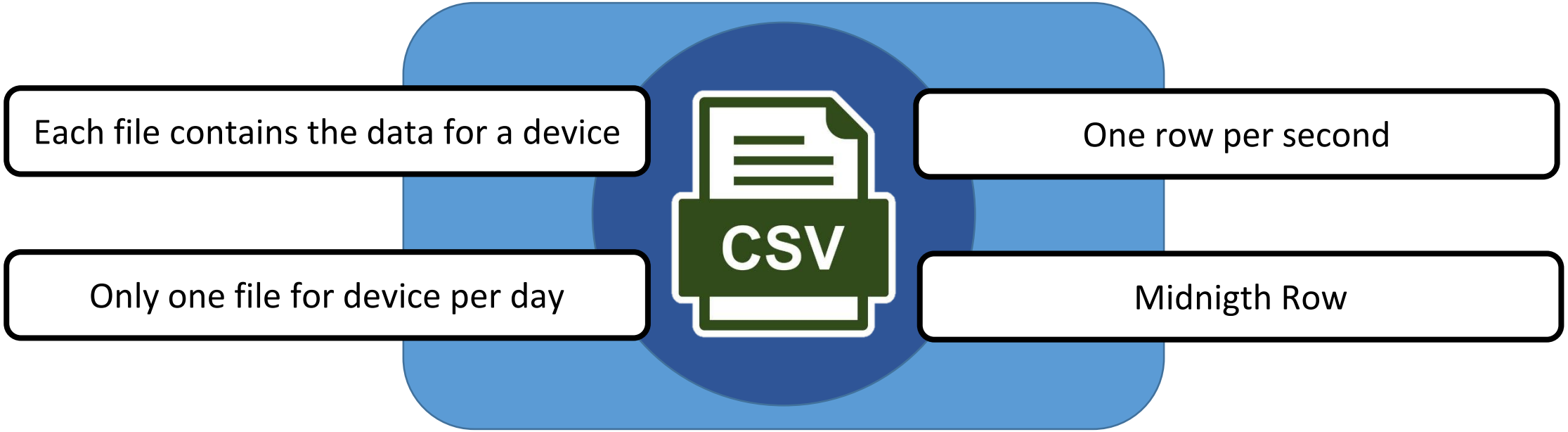
Raw Data


```
{"itemName": "SPEED","value": "{\"value\":46.1}","itemId": 1111,"utcTimestamp": "2021-07-07T18:09:05.2871434+00:00"}
{"itemName": "SPEED","value": "{\"value\":45.7}","itemId": 1111,"utcTimestamp": "2021-07-07T18:09:05.6376127+00:00"}
{"itemName": "TEMPERATURE","value": "{\"value\":104.9}","itemId": 1000,"utcTimestamp": "2021-07-07T18:09:05.8876127+00:00"}
{"itemName": "SPEED","value": "{\"value\":45.3}","itemId": 1111,"utcTimestamp": "2021-07-07T18:09:06.1349456+00:00"}
{"itemName": "SCRAP_IMAGE","value": "{\"ENTER_ID\":1010,\"CATEGORY\":\"A\"}","itemId": 1234,"utcTimestamp": "2021-07-07T18:09:06.5487926+00:00"}
{"itemName": "TEMPERATURE","value": "{\"value\":106.3}","itemId": 1000,"utcTimestamp": "2021-07-07T18:09:06.7259397+00:00"}
{"itemName": "TEMPERATURE","value": "{\"value\":107.4}","itemId": 1000,"utcTimestamp": "2021-07-07T18:09:07.3638633+00:00"}
{"itemName": "SPEED","value": "{\"value\":45.2}","itemId": 1111,"utcTimestamp": "2021-07-07T18:09:07.5824351+00:00"}
{"itemName": "SPEED","value": "{\"value\":44.8}","itemId": 1111,"utcTimestamp": "2021-07-07T18:09:07.6341427+00:00"}
{"itemName": "TEMPERATURE","value": "{\"value\":107.8}","itemId": 1000,"utcTimestamp": "2021-07-07T18:09:08.9213413+00:00"}
{"itemName": "SPEED","value": "{\"value\":44.5}","itemId": 1111,"utcTimestamp": "2021-07-07T18:09:09.0372616+00:00"}
{"itemName": "TEMPERATURE","value": "{\"value\":107.4}","itemId": 1000,"utcTimestamp": "2021-07-07T18:09:09.1856561+00:00"}
{"itemName": "SCRAP_IMAGE","value": "{\"ENTER_ID\":1011,\"CATEGORY\":\"C\"}","itemId": 1234,"utcTimestamp": "2021-07-07T18:09:09.6186127+00:00"}
{"itemName": "TEMPERATURE","value": "{\"value\":106.9}","itemId": 1000,"utcTimestamp": "2021-07-07T18:09:09.8414679+00:00"}
```

Our engineers looking at raw data

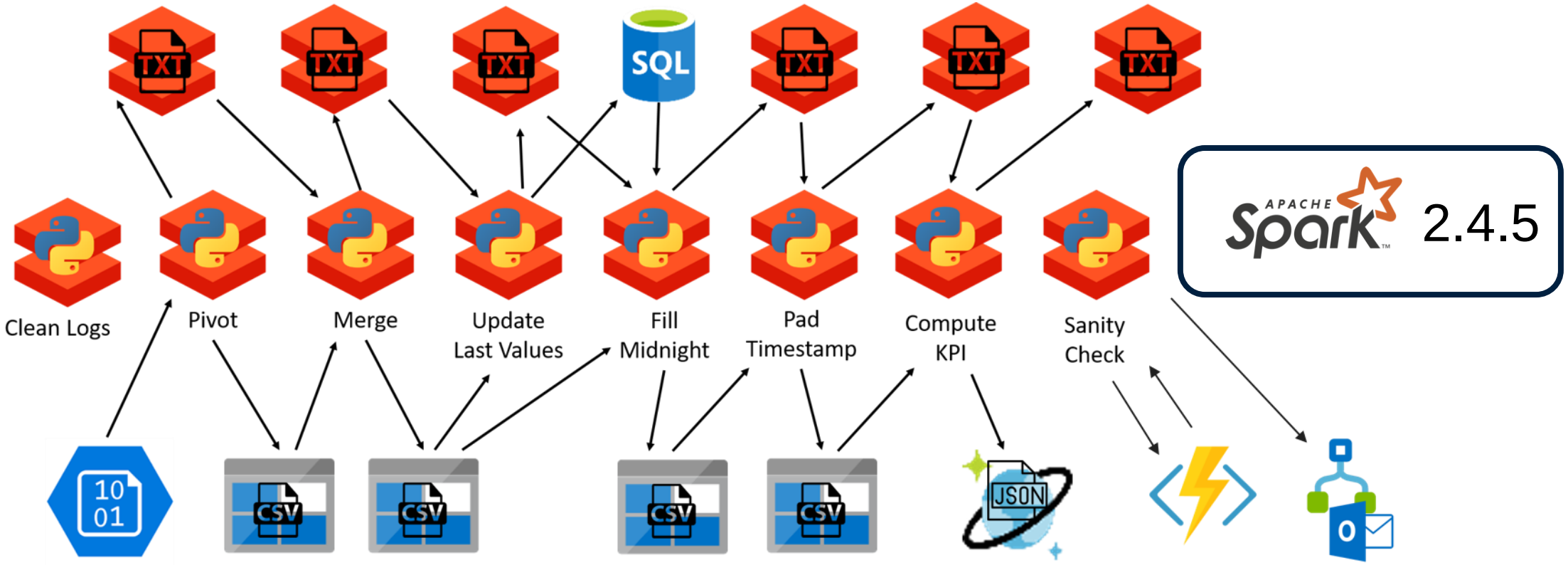


Requirements



Architecture

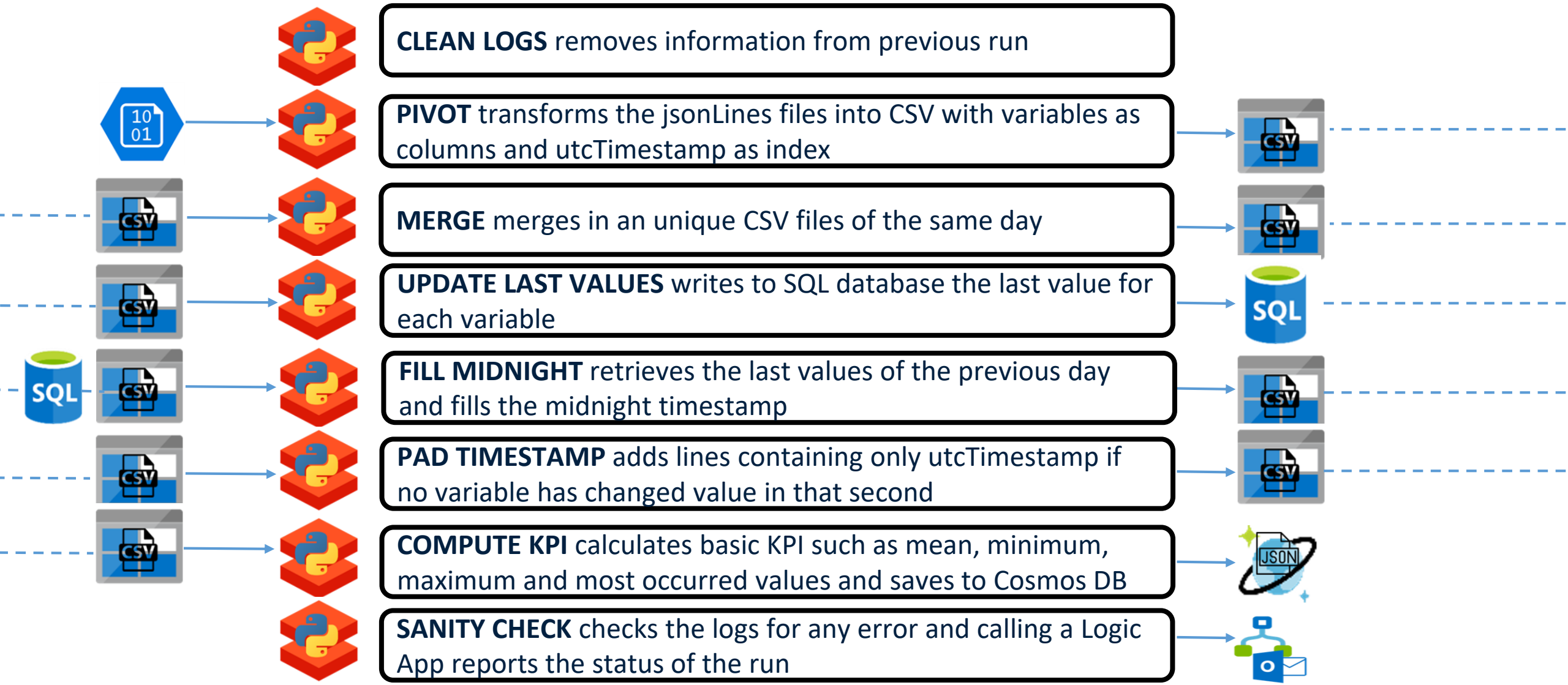
Architecture



Our intern looking at the architecture

Architecture

I HOPE IT'S A LITTLE CLEARER



Code

Raw data

```
{"itemName": "SPEED","value": "{\"value\":46.1}","itemId": 1111,"utcTimestamp": "2021-07-07T18:09:05.2871434+00:00"}
{"itemName": "SPEED","value": "{\"value\":45.7}","itemId": 1111,"utcTimestamp": "2021-07-07T18:09:05.6376127+00:00"}
{"itemName": "TEMPERATURE","value": "{\"value\":104.9}","itemId": 1000,"utcTimestamp": "2021-07-07T18:09:05.8876127+00:00"}
{"itemName": "SPEED","value": "{\"value\":45.3}","itemId": 1111,"utcTimestamp": "2021-07-07T18:09:06.1349456+00:00"}
{"itemName": "SCRAP_IMAGE","value": "{\"ENTER_ID\":1010,\"CATEGORY\":\"A\"}","itemId": 1234,"utcTimestamp": "2021-07-07T18:09:06.5487926+00:00"}
{"itemName": "TEMPERATURE","value": "{\"value\":106.3}","itemId": 1000,"utcTimestamp": "2021-07-07T18:09:06.7259397+00:00"}
{"itemName": "TEMPERATURE","value": "{\"value\":107.4}","itemId": 1000,"utcTimestamp": "2021-07-07T18:09:07.3638633+00:00"}
{"itemName": "SPEED","value": "{\"value\":45.2}","itemId": 1111,"utcTimestamp": "2021-07-07T18:09:07.5824351+00:00"}
{"itemName": "SPEED","value": "{\"value\":44.8}","itemId": 1111,"utcTimestamp": "2021-07-07T18:09:07.6341427+00:00"}
{"itemName": "TEMPERATURE","value": "{\"value\":107.8}","itemId": 1000,"utcTimestamp": "2021-07-07T18:09:08.9213413+00:00"}
{"itemName": "SPEED","value": "{\"value\":44.5}","itemId": 1111,"utcTimestamp": "2021-07-07T18:09:09.0372616+00:00"}
{"itemName": "TEMPERATURE","value": "{\"value\":107.4}","itemId": 1000,"utcTimestamp": "2021-07-07T18:09:09.1856561+00:00"}
{"itemName": "SCRAP_IMAGE","value": "{\"ENTER_ID\":1011,\"CATEGORY\":\"C\"}","itemId": 1234,"utcTimestamp": "2021-07-07T18:09:09.6186127+00:00"}
{"itemName": "TEMPERATURE","value": "{\"value\":106.9}","itemId": 1000,"utcTimestamp": "2021-07-07T18:09:09.8414679+00:00"}
```

Pivot

CODE

PIVOT transforms the jsonLines files into CSV with variables as columns and utcTimestamp as index


```python
from pyspark.sql.functions import to_timestamp, explode, col, desc, last, date_trunc, row_number
from pyspark.sql.window import Window


def transform_dataframe(raw_df):
    df = raw_df.select(
        to_timestamp(raw_df.utcTimestamp).alias("utcTimestamp"),
        raw_df.itemName,
        explode(parse(raw_df.value)).alias("key", "value"),
        raw_df.itemId,
    )
    df = df.withColumn("variable", merge_columns("key", "itemName", "itemId"))
    df = df.select(df.utcTimestamp, df.variable, df.value)

    # truncatedUtc is introduced to select just the last value (desc) for each pair of [utcTimestamp, variable]
    df = df.withColumn("truncatedUtc", date_trunc("second", "utcTimestamp"))
    w = Window.partitionBy("variable", "truncatedUtc").orderBy(desc("utcTimestamp"))
    # for each variable at a given truncatedUtc the first row is selected.
    # Since the window is in descending order the last value is selected.
    df = df.withColumn("row_number", row_number().over(w)).where(col("row_number") == 1).drop("rownumber")

    # there is just one value per timestamp, per variable. last is used just to take the unique value.
    df = df.groupby("truncatedUtc").pivot("variable").agg(last("value"))
    df = df.drop("utcTimestamp").withColumnRenamed("truncatedUtc", "utcTimestamp").orderBy("utcTimestamp")

    return df
```

Pivot

OUTPUT

utcTimestamp	1000|TEMPERATURE	1111|SPEED	1234|SCRAP_IMAGE_ENTER_ID	1234|SCRAP_IMAGE_CATEGORY
2021-07-07T18:09:05	104.9	45.7		
2021-07-07T18:09:06	106.3	45.3	1010	A
2021-07-07T18:09:07	107.4	44.8		
2021-07-07T18:09:08	107.8			
2021-07-07T18:09:09	106.9	44.5	1011	C

Merge

CODE


MERGE merges in an unique CSV files of the same day

```python
def merge(day_files, conf):
    dfs = [spark.read.csv(_, header=True, multiLine=True) for _ in day_files]
    df_raw = reduce(
        lambda df1, df2: df1.join(
            df2, list(set(df1.columns).intersection(df2.columns)), "outer"
        ),
        dfs,
    )
    df_raw = sort_columns(df_raw)
    schema = df_raw.schema
    df_no_duplicates = df_raw.dropDuplicates(subset=["utcTimestamp"])
    if df_no_duplicates.count() == df_raw.count():
        return df_raw.sort(asc("utcTimestamp"))
    else:
        df_raw = df_raw.toPandas()
        df_duplicated = df_raw[df_raw.duplicated(subset="utcTimestamp", keep=False)]
        df_no_duplicates = df_raw[~df_raw.duplicated(subset="utcTimestamp", keep=False)]

        resolved_duplicates = df_duplicated.groupby("utcTimestamp").agg(unique_udf)
        resolved_duplicates["utcTimestamp"] = resolved_duplicates.index
        resolved_duplicates.index = range(len(resolved_duplicates))

        result = pd.concat([df_no_duplicates, resolved_duplicates]).sort_values(
            "utcTimestamp"
        )
        result = result[schema.fieldNames()]
        result = spark.createDataFrame(result, schema)
        return result
```

Update Last Values

CODE

UPDATE LAST VALUES writes to SQL database the last value for each variable


```python
def find_last_value_column(obj):
    last_non_null = obj.dropna().tail(1)
    try:
        last_non_null_value = last_non_null[0]
        last_non_null_timestamp = last_non_null.index[0]
        return {"value": last_non_null_value, "timestamp": last_non_null_timestamp}
    except:
        return None


def generate_last_values_dict(df):
    last_values_dict = {
        column: find_last_value_column(df[column])
        for column in df.columns
        if column != "utcTimestamp"
    }
    last_values_dict = {
        key: value for key, value in last_values_dict.items() if value is not None
    }
    return last_values_dict
```

Update Last Values

tenova

QUERY

Timestamp	ItemId	Value	DeviceId	ItemName
2021-07-26 23:59:20	1000	103.5	1	TEMPERATURE
2021-07-26 23:59:56	1111	46.8	1	SPEED
2021-07-26 23:55:15	1234	1982	1	SCRAP_IMAGE_ENTER_ID
2021-07-26 23:55:15	1234	A	1	SCRAP_IMAGE_CATEGORY

```
1 /*get timestamp*/
2 SELECT Timestamp FROM lastValues  WHERE ItemId={itemId} AND CONVERT(DATE, Timestamp)=CONVERT(DATE, '{last_timestamp}') AND ItemName='{itemName}'
3
4 /*update last value*/
5 UPDATE lastValues SET ItemId = {itemId}, Timestamp = '{last_timestamp}', Value = '{value}', ItemName='{itemName}', DeviceId= {device} WHERE ItemId={itemId}
  AND ItemName='{itemName}' AND CONVERT(DATE, Timestamp)=CONVERT(DATE, '{last_timestamp}')
6
7 /*insert last value*/
8 INSERT INTO lastValues(ItemId, Timestamp, Value, DeviceId, ItemName) VALUES ('{itemId}', '{last_timestamp}', '{value}', {device}, '{itemName}')
```

Fill Midnight

RETRIEVE LAST VALUES


```python
from pandas import read_sql


def retrieve_last_values(device, day):
    data = read_sql(
        f"""SELECT itemid, itemname, value FROM LastValues
          WHERE DATEADD(DAY, +1, CONVERT(DATE, Timestamp))='{day}'
          and deviceid={device}""",
        cnxn,
    )
    yesterday_column_names = [
        f'{value}|{data["itemname"][key]}' for key, value in enumerate(data["itemid"])
    ]
    yesterday_last_values = dict(zip(yesterday_column_names, data["value"]))
    return yesterday_last_values
```

Fill Midnight

UPDATE DATAFRAME

```python
def midnight_exists(df):
    first_timestamp_string = df.select("utcTimestamp").first()["utcTimestamp"]
    # very bad trick
    hour_minute_seconds_string = first_timestamp_string.split("T")[1][:8]
    if hour_minute_seconds_string == "00:00:00":
        return True
    else:
        return False


def update_dataframe(df, info):
    variables_df = [_ for _ in df.columns if _ != "utcTimestamp"]
    if info == {}:
        return df
    for _ in variables_df:
        if _ not in info.keys():
            info[_] = ""
    if midnight_exists(df):
        df = fill_midnight_timestamp(df, info)
        print("fill midnight")
    else:
        print("insert midnight")
        df = insert_midnight_timestamp(df, info)
    return df
```

Pad Timestamp


PAD TIMESTAMP adds lines containing only utcTimestamp if no variable has changed value in that second

```python
from pyspark.sql.functions import col, min as min_, max as max_


def pad_timestamp(df, step=1):
    minp, maxp = df.select(
        min_("utcTimestamp").cast("timestamp").cast("int"),
        max_("utcTimestamp").cast("timestamp").cast("int"),
    ).first()
    reference = spark.range(
        (minp / step) * step, ((maxp / step) + 1) * step, step
    ).select(col("id").cast("timestamp").alias("utcTimestamp"))
    result = reference.join(df, ["utcTimestamp"], "leftouter")
    return result
```

Compute KPI

COMPUTE KPI calculates basic KPI such as mean, minimum, maximum and most occurred values and saves to Cosmos DB

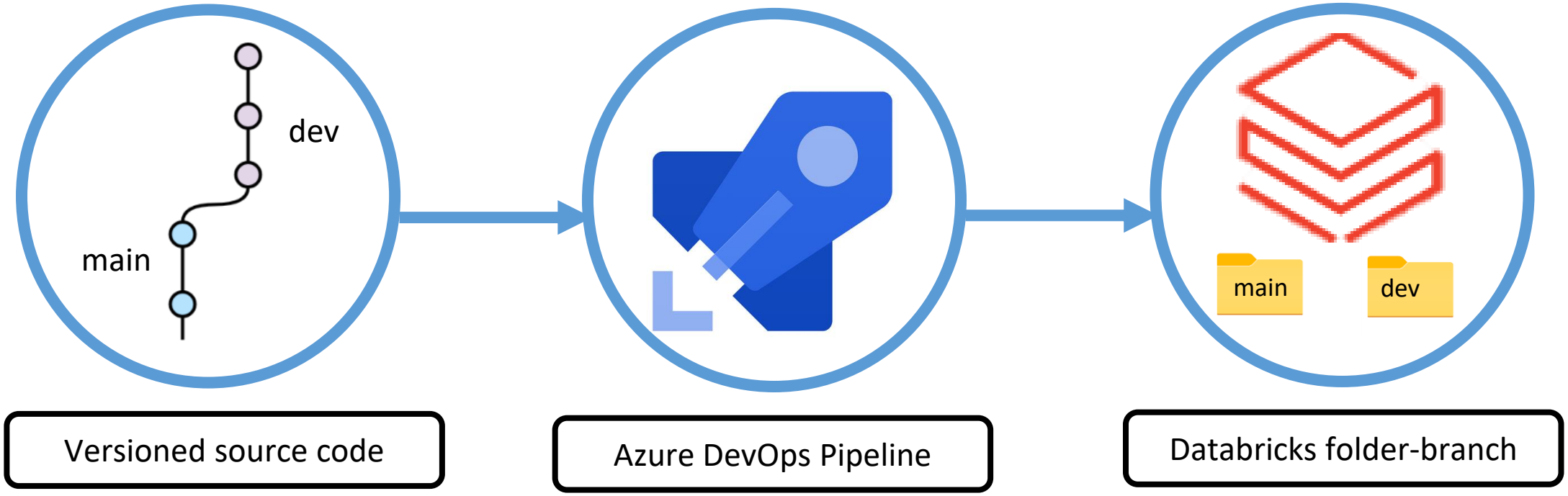

```python
import pandas as pd


def compute_basic_KPIs(df):
    df = df.apply(pd.to_numeric, errors="ignore")
    summary = df.describe(include="all")
    summary_pd = summary.drop(["utcTimestamp"], axis=1)
    df_filled = df.ffill()
    summary_filled = df_filled.describe(include="all")
    summary_filled_pd = summary_filled.drop(["utcTimestamp"], axis=1)

    result_pd = []
    for column in summary_pd.columns:
        item = {}
        tag_id, tag_name = column.split("|")
        item["TagId"] = tag_id
        item["KPI"] = build_KPI_dict_for_column(
            summary_not_filled=summary_pd,
            summary_filled=summary_filled_pd,
            column=column,
        )
        time_occ = df_filled[column].value_counts()
        item["TimeOccurencies"] = dict(time_occ[time_occ > 1])
        item["TagName"] = tag_name
        serialized_item = pd.Series(item).to_json()
        result_pd.append(item)
    return result_pd
```

Orchestration

Continuous Deployment



Orchestration

Azure Databricks

- Same environment as the notebooks
- Little flexibility for future developments

Azure Data Factory

- Easy to customize
- Flexibility in replacing some notebooks with other services

Conclusion

Next Steps

UPGRADE TO SPARK 3

COMPUTE AD HOC KPI

IMPROVE PERFORMANCE COMPUTE KPI

The End!