

IND320 part 2 - Elhub, Spark, Cassandra, MongoDB

Goal: fetch 2021 hourly production from Elhub, normalize, store in Cassandra, plot, then upsert to MongoDB for the Streamlit app.

Notes

I had to merge the Oblig2 branch into main to make the Streamlit app available. Whenever i tried to use the other branch i got ann error saying i didn't hve access.

Imports

Load the libraries used in this notebook.

```
In [16]: import time, requests, pandas as pd, numpy as np
from datetime import datetime, timedelta, timezone
import matplotlib.pyplot as plt
```

API setup

Base URL, dataset, and a small helper to fetch one date range.

```
In [17]: BASE = "https://api.elhub.no/energy-data/v0"
ENTITY = "price-areas"
DATASET = "PRODUCTION_PER_GROUP_MBA_HOUR"

YEAR_START = datetime(2021, 1, 1, tzinfo=timezone.utc)
YEAR_END = datetime(2022, 1, 1, tzinfo=timezone.utc)

def iso_offset(dt: datetime) -> str:
    return dt.strftime("%Y-%m-%dT%H:%M:%S+00:00")

def fetch_chunk(start_dt: datetime, end_dt: datetime):
    url = f"{BASE}/{ENTITY}"
    params = {"dataset": DATASET, "startDate": iso_offset(start_dt), "end": iso_offset(end_dt)}
    r = requests.get(url, params=params, timeout=60)
    if r.ok:
        return 200, r.json(), None
    return r.status_code, None, r.text[:400]
```

Probe the API

Call a tiny window to check status and response shape.

```
In [18]: probe_start = datetime(2021, 1, 1, tzinfo=timezone.utc)
probe_end = datetime(2021, 1, 3, tzinfo=timezone.utc)
```

```
code, js, err = fetch_chunk(probe_start, probe_end)
print("HTTP", code, "| keys:", list(js.keys())) if isinstance(js, dict) el
```

HTTP 200 | keys: ['data', 'links', 'meta']

Normalize JSON

Flatten productionPerGroupMbaHour into rows with the four fields we need.

```
In [19]: def normalize_payload_jsonapi_nested(js):
    rows = []
    for item in js.get("data", []):
        attrs = item.get("attributes", {}) or {}
        area = attrs.get("name") or attrs.get("eic")
        for rec in attrs.get("productionPerGroupMbaHour", []) or []:
            rows.append((
                area,
                rec.get("productionGroup"),
                rec.get("startTime") or rec.get("start"),
                rec.get("quantityKwh") or rec.get("kWh") or rec.get("quan
            ))
    df = pd.DataFrame(rows, columns=["priceArea", "productionGroup", "start
df["startTime"] = pd.to_datetime(df["startTime"], utc=True, errors="c
df["quantityKwh"] = pd.to_numeric(df["quantityKwh"], errors="coerce")
    return df

probe_df = normalize_payload_jsonapi_nested(js)
probe_df.head()
```

```
Out [19]:
```

	priceArea	productionGroup	startTime	quantityKwh
0	NO1	hydro	2020-12-31 23:00:00+00:00	2507716.8
1	NO1	hydro	2021-01-01 00:00:00+00:00	2494728.0
2	NO1	hydro	2021-01-01 01:00:00+00:00	2486777.5
3	NO1	hydro	2021-01-01 02:00:00+00:00	2461176.0
4	NO1	hydro	2021-01-01 03:00:00+00:00	2466969.2

Fetch full 2021

Loop over months in 2021, fetch, normalize, and concatenate to one DataFrame.

```
In [20]: parts, cursor = [], YEAR_START
while cursor < YEAR_END:
    month_end = (cursor + timedelta(days=32)).replace(day=1, hour=0, minu
    end_dt = min(month_end, YEAR_END)
    code, jsm, err = fetch_chunk(cursor, end_dt)
    if code == 200 and jsm:
        dfp = normalize_payload_jsonapi_nested(jsm)
        parts.append(dfp)
        print(f"OK {cursor.date()}..{end_dt.date()} rows={len(dfp)}")
        cursor = end_dt
        time.sleep(0.2)
    continue
```

```

print(f"Monthly failed {cursor.date()}..{end_dt.date()} code={code}."
wk = cursor
while wk < end_dt:
    wk_end = min(wk + timedelta(days=7), end_dt)
    c2, jsw, err2 = fetch_chunk(wk, wk_end)
    if c2 == 200 and jsw:
        parts.append(normalize_payload_jsonapi_nested(jsw))
        print(f" OK {wk.date()}..{wk_end.date()}")
        time.sleep(0.2)
    else:
        raise RuntimeError(f"Weekly failed {wk}..{wk_end} code={c2} e
    wk = wk_end
cursor = end_dt

df = pd.concat(parts, ignore_index=True) if parts else pd.DataFrame(colum
df = df[(df["startTime"] >= "2021-01-01") & (df["startTime"] < "2022-01-0
len(df), df.head())

```

```

OK 2021-01-01..2021-02-01 rows=17856
OK 2021-02-01..2021-03-01 rows=16128
OK 2021-03-01..2021-04-01 rows=17832
OK 2021-04-01..2021-05-01 rows=17280
OK 2021-05-01..2021-06-01 rows=17856
OK 2021-06-01..2021-07-01 rows=17976
OK 2021-07-01..2021-08-01 rows=18600
OK 2021-08-01..2021-09-01 rows=18600
OK 2021-09-01..2021-10-01 rows=18000
OK 2021-10-01..2021-11-01 rows=18625
OK 2021-11-01..2021-12-01 rows=18000
OK 2021-12-01..2022-01-01 rows=18600

```

```

Out[20]: (215329,
          priceArea productionGroup      startTime  quantityKwh
1          N01          hydro 2021-01-01 00:00:00+00:00    2494728.0
2          N01          hydro 2021-01-01 01:00:00+00:00    2486777.5
3          N01          hydro 2021-01-01 02:00:00+00:00    2461176.0
4          N01          hydro 2021-01-01 03:00:00+00:00    2466969.2
5          N01          hydro 2021-01-01 04:00:00+00:00    2467460.0)

```

Start Spark

Local Spark with the Cassandra connector and localhost settings.

```

In [21]: import os
os.environ["SPARK_LOCAL_IP"] = "127.0.0.1"
from pyspark.sql import SparkSession

spark = (
    SparkSession.builder
        .appName("ElhubToCassandra")
        .config("spark.master", "local[*]")
        .config("spark.jars.packages", "com.datastax.spark:spark-cassandra-con
        .config("spark.cassandra.connection.host", "127.0.0.1")
        .config("spark.cassandra.connection.port", "9042")
        .config("spark.cassandra.connection.localDC", "datacenter1")
        .getOrCreate()
)

```

Write to Cassandra

Map columns, cast types, and write to power.production_2021. Primary key is (pricearea, productiongroup, starttime).

```
In [22]: sdf = spark.createDataFrame(df).selectExpr(
    "priceArea as pricearea",
    "productionGroup as productiongroup",
    "startTime as starttime",
    "CAST(quantityKwh AS double) as quantitykwh"
)

sdf.write.format("org.apache.spark.sql.cassandra") \
    .options(keyspace="power", table="production_2021") \
    .mode("append").save()
```

Read back for plots

Load only pricearea, productiongroup, starttime, quantitykwh. Show schema and a quick count.

```
In [23]: from pyspark.sql.functions import col, sum as _sum

sdf = (spark.read.format("org.apache.spark.sql.cassandra")
    .options(keyspace="power", table="production_2021").load()
    .select("pricearea", "productiongroup", "starttime", "quantitykwh")
    .cache())

sdf.count()
```

Out[23]: 215304

Choose price area

Pick one area to plot. Change the code value if you want another area.

```
In [24]: areas = [r.pricearea for r in sdf.select("pricearea").distinct().orderBy(
    areas
```

Out[24]: ['N01', 'N02', 'N03', 'N04', 'N05']

```
In [25]: price_area = "N01"
```

Pie chart, full 2021

Sum by production group for the chosen area. Show a clean pie with a legend.

```
In [26]: totals_pdf = (
    sdf.filter((col("pricearea")==price_area) & col("quantitykwh").isNotN
```

```

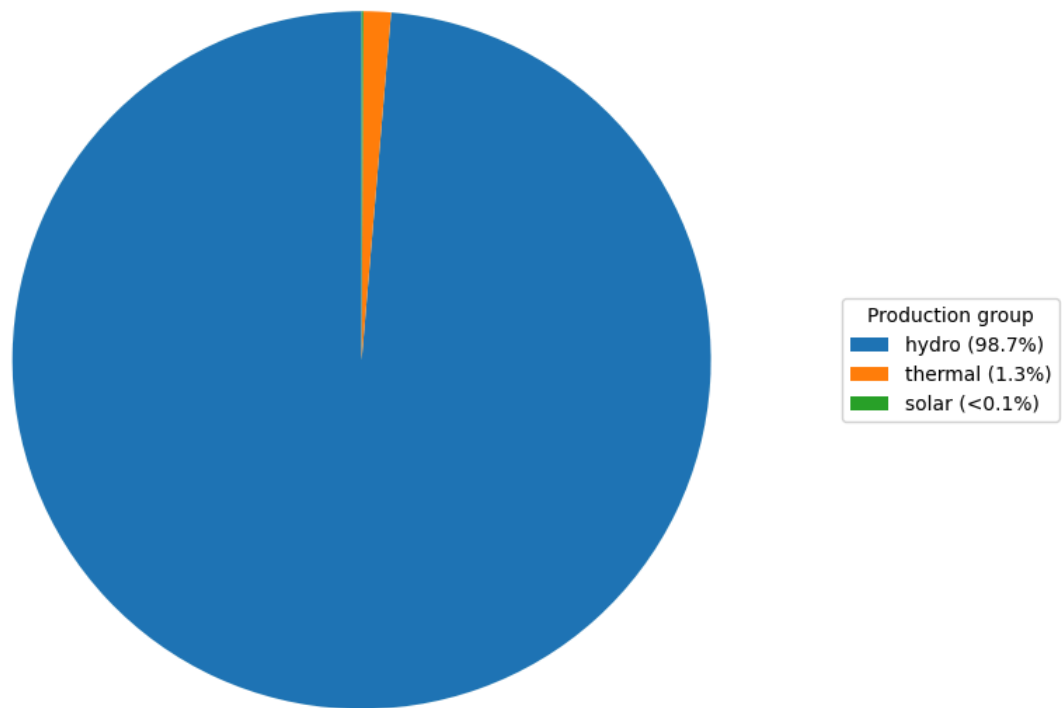
        .groupBy("productiongroup")
        .agg(_sum("quantitykwh").alias("totalKwh"))
        .toPandas()
    )
    totals_pdf = totals_pdf.dropna(subset=["productiongroup"])
    totals_pdf["totalKwh"] = pd.to_numeric(totals_pdf["totalKwh"], errors="co
    totals_pdf = totals_pdf[totals_pdf["totalKwh"] > 0].sort_values("totalKwh

    grand = totals_pdf["totalKwh"].sum()
    totals_pdf["pct"] = (100 * totals_pdf["totalKwh"] / grand) if grand > 0 e
    legend_labels = [f"{g} ({p:.1f}%)" if p >= 0.1 else f"{g} (<0.1%)" for g,

    sizes = totals_pdf["totalKwh"].to_numpy(dtype=float)
    fig, ax = plt.subplots(figsize=(8.5, 6))
    wedges = ax.pie(sizes, labels=None, autopct=None, startangle=90, pctdista
    ax.axis("equal")
    ax.set_title(f"Total production 2021 - {price_area}")
    ax.legend(wedges, legend_labels, title="Production group", loc="center le
    plt.tight_layout(); plt.show()

```

Total production 2021 - NO1



Line chart, January

Filter 2021-01-01 to 2021-02-01 UTC, sum per day, one line per group.

```

In [27]: from pyspark.sql.functions import col, sum as _sum, to_date
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.dates import DayLocator
from matplotlib.ticker import FuncFormatter

daily_sdf = (
    sdf.filter(

```

```

        (col("pricearea") == price_area) &
        (col("starttime") >= "2021-01-01") &
        (col("starttime") < "2021-02-01") &
        col("quantitykwh").isNotNull()
    )
    .withColumn("day", to_date(col("starttime")))
    .groupBy("productiongroup", "day")
    .agg(_sum("quantitykwh").alias("kWh_day"))
    .orderBy("day")
)

daily_pdf = daily_sdf.toPandas()
daily_pdf["day"] = pd.to_datetime(daily_pdf["day"])
pivot = daily_pdf.pivot_table(
    index="day",
    columns="productiongroup",
    values="kWh_day",
    fill_value=0
).sort_index()

fig, ax = plt.subplots(figsize=(10.5, 4.5))
pivot.plot(ax=ax, linewidth=2)

ax.set_title(f"Daily production, Jan 2021, {price_area}")
ax.set_xlabel("day of month")
ax.set_ylabel("kWh, daily total")
ax.grid(True, alpha=0.3)

JAN_START = pd.Timestamp("2021-01-01")
FEB_START = pd.Timestamp("2021-02-01")
ax.set_xlim(JAN_START, FEB_START)

ax.xaxis.set_major_locator(DayLocator(interval=1))

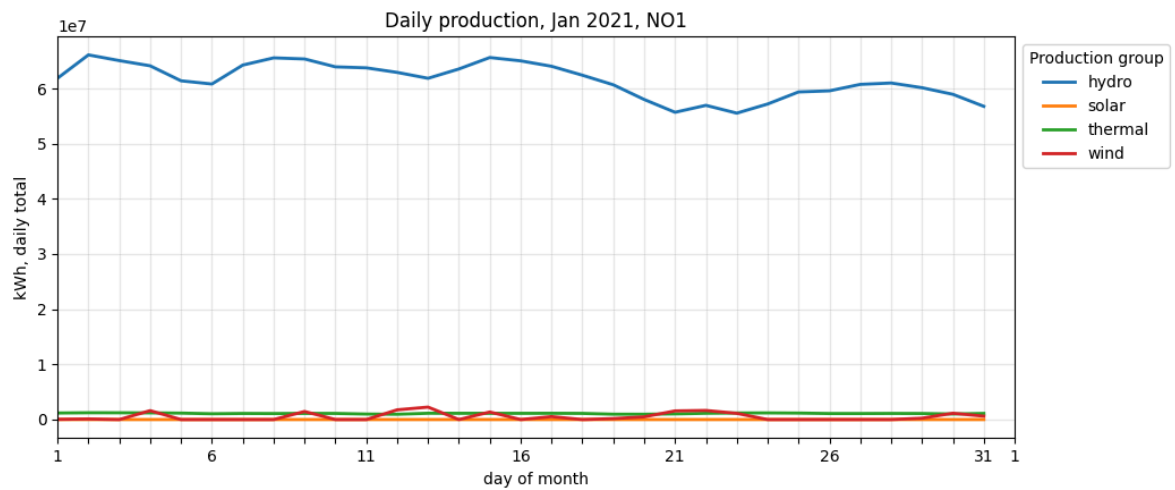
def label_every_5th(x, pos):
    d = mdates.num2date(x)
    return str(d.day) if d.day in {1, 6, 11, 16, 21, 26, 31} else ""

ax.xaxis.set_major_formatter(FuncFormatter(label_every_5th))

ax.xaxis.set_minor_locator(DayLocator(interval=1))
ax.tick_params(axis="x", which="major", labelsize=10, pad=4)

ax.legend(title="Production group", loc="upper left", bbox_to_anchor=(1.0
plt.tight_layout()
plt.show()

```



MongoDB settings

Read Mongo URI from env or secrets. Collection holds the same four fields.

```
In [28]: MONGODB_URI = "mongodb+srv://younes_user:Younes2002@oblig.okycarf.mongodb
DB_NAME = "power"
COLL_NAME = "production_2021"
```

Upsert to MongoDB

Bulk upsert using a stable `_id` built from `priceArea`, `productionGroup`, `startTime`.
Create a helpful compound index.

```
In [29]: from pymongo import MongoClient, ReplaceOne

MONGO_URI = MONGODB_URI
DB = "power"
COLL = "production_2021"
BATCH = 3000

def insert_partition(rows):
    client = MongoClient(
        MONGO_URI,
        serverSelectionTimeoutMS=8000,
        connectTimeoutMS=8000,
        socketTimeoutMS=20000,
        retryWrites=True
    )
    coll = client[DB][COLL]
    ops = []
    try:
        client.admin.command("ping")
        for r in rows:
            start_iso = r.starttime.isoformat() if hasattr(r.starttime, "isoformat") else ""
            _id = f"{r.pricearea}|{r.productiongroup}|{start_iso}"
            doc = {
                "_id": _id,
                "priceArea": r.pricearea,
                "productionGroup": r.productiongroup,
                "startTime": start_iso,
            }
```

```

        "quantityKwh": float(r.quantitykwh) if r.quantitykwh is not None
    }
    ops.append(ReplaceOne({"_id": _id}, doc, upsert=True))
    if len(ops) >= BATCH:
        coll.bulk_write(ops, ordered=False)
        ops.clear()
    if ops:
        coll.bulk_write(ops, ordered=False)
    finally:
        client.close()

target = sdf
target.repartition(6).foreachPartition(insert_partition)
print("Bulk upsert complete.")

```

```

[Stage 53:=====> (5 + 1) / 6]
Bulk upsert complete.

```

Verify Mongo

Print total docs, distinct areas, and one example document.

```

In [30]: from pymongo import MongoClient
cli = MongoClient(MONGODB_URI)
col = cli["power"]["production_2021"]
print("Total docs:", col.count_documents({}))
print("Distinct price areas:", sorted(col.distinct("priceArea")))
print("One example:", col.find_one({}, {"_id": 0}))
cli.close()

```

```

Total docs: 215304
Distinct price areas: ['N01', 'N02', 'N03', 'N04', 'N05']
One example: {'priceArea': 'N02', 'productionGroup': 'solar', 'startTime':
'2021-09-14T22:00:00', 'quantityKwh': 1.602}

```

AI usage

I am new to this stack, so I used an chatgpt5 like a teacher throughout the work. It helped me plan the order of tasks, set up the development environment, and keep tool versions consistent. I am not particullary good at coding, so i used the ai by explaining what i wanted to achieve and commenting on the changes i wanted to be made. When things failed to start or connect, it explained common problems and suggested simple fixes and how to implement them. For the API part, it guided me to fetch the data in the right time windows and reshape the payload into a table with the fields I needed. For Spark and the databases, it guided my setup and the code I needed to read and write data. For the plots it helped me get the right visualizations. MongoDB and the app, it helped me with the setup, the upsert approach, and the connection to the UI.

Log

I started by setting up a clean Python environment in conda and making sure I used Java 17 with Spark 3.5.1. I ran Cassandra in Docker and verified it, this was the easiest part since i ws already familiar with Docker. Getting the tools to work together was harder. I launched Jupyter from the same conda env. When the Spark gateway failed once, I fixed it by restarting the kernel and setting SPARK_LOCAL_IP to 127.0.0.1.

Next I looked at the Elhub Energy Data API. I tested a tiny date window to confirm the JSON shape and keys. Then I fetched the full year 2021 in monthly chunks to respect date range limits. The payload nests productionPerGroupMbaHour, so I included a normalizer that flattened it to four fields that were used later: priceArea, productionGroup, startTime, quantityKwh. I kept all timestamps in UTC and used a half open window for January to have boundaries.

For storage I used Spark with the Cassandra connector and created a table power.production_2021. The important part was choosing a primary key that fit the query I wanted to do. I used pricearea, productiongroup, starttime. My first write failed but then I aligned DataFrame column names and types with the table. After writing, I read the same four columns back with Spark to verify the schema and counts.

For visualization I made two plots from the Spark data. A pie chart that sums the full year by production group for the chosen price area. A line chart for January where i used daily totals and drew one line per production group.

Finally I set up the remote database. I connected to MongoDB Atlas from Python and upserted the same four fields. I used a stable id built from priceArea, productionGroup, and startTime. In the Streamlit app I added a page that reads from Mongo that shows a radio to pick price area for a pie chart. And a linechart that lets me pick group and month. And at the bottom i added a short data source note.

Links

Github repo link: <https://github.com/youneshansen/ind320-yohan3351> Streamlit link: <https://ind320-yohan3351.streamlit.app/>