# Day‑1 Ingestion: Polygon daily OHLCV + schemas

This document contains ready‑to‑paste files for your repo:

* **SQL** for Postgres & ClickHouse tables
* **Python** ingestion script for daily OHLCV (Polygon) + corporate actions (splits & dividends)
* **Universe CSV** stub for the S&P 100 tickers
* **Run instructions**

Notes

* Uses **Bearer** header for Polygon authentication.
* Writes to **Postgres** and **ClickHouse**.
* Creates tables if they don’t exist; safe to run repeatedly (idempotent upserts/merges).

## 1) SQL — Postgres tables (reference; the Python script also creates these if missing)

\*\*File: \*\*``

CREATE SCHEMA IF NOT EXISTS core;  
  
CREATE TABLE IF NOT EXISTS core.prices\_daily (  
 ticker TEXT NOT NULL,  
 trade\_date DATE NOT NULL,  
 open DOUBLE PRECISION,  
 high DOUBLE PRECISION,  
 low DOUBLE PRECISION,  
 close DOUBLE PRECISION,  
 volume BIGINT,  
 vwap DOUBLE PRECISION,  
 adjusted\_close DOUBLE PRECISION,  
 PRIMARY KEY (ticker, trade\_date)  
);

\*\*File: \*\*``

CREATE SCHEMA IF NOT EXISTS core;  
  
CREATE TABLE IF NOT EXISTS core.corporate\_actions (  
 ticker TEXT NOT NULL,  
 ca\_type TEXT NOT NULL, -- 'split' or 'dividend'  
 ex\_date DATE, -- execution\_date for splits; ex\_dividend\_date for dividends  
 record\_date DATE,  
 payable\_date DATE,  
 declared\_date DATE,  
 split\_from NUMERIC(18,8), -- for splits  
 split\_to NUMERIC(18,8), -- for splits  
 amount NUMERIC(18,8), -- dividend cash amount  
 frequency TEXT, -- e.g., 0,1,2,4,12 per Polygon docs  
 source TEXT DEFAULT 'polygon',  
 PRIMARY KEY (ticker, ca\_type, ex\_date)  
);

## 2) SQL — ClickHouse tables (reference; the Python script also creates these if missing)

\*\*File: \*\*``

CREATE DATABASE IF NOT EXISTS core;  
  
CREATE TABLE IF NOT EXISTS core.prices\_daily  
(  
 ticker String,  
 trade\_date Date,  
 open Float64,  
 high Float64,  
 low Float64,  
 close Float64,  
 volume UInt64,  
 vwap Float64,  
 adjusted\_close Float64  
)  
ENGINE = MergeTree  
ORDER BY (ticker, trade\_date);

## 3) Python — Ingestion script

\*\*File: \*\*``

#!/usr/bin/env python3  
"""  
Fetch daily OHLCV from Polygon for a universe of tickers and write to Postgres & ClickHouse.  
Also fetches corporate actions (splits & dividends) for the same universe.  
  
Usage examples:  
 # In project root where .env lives  
 python services/ingestion/polygon\_daily.py \  
 --start 2020-01-01 --end today \  
 --universe-file data/universe/sp100.csv \  
 --fetch-events  
  
Env vars expected (see .env):  
 POLYGON\_API\_KEY  
 POSTGRES\_URI  
 CLICKHOUSE\_URL  
  
Notes:  
 - Share class tickers use dot notation on Polygon (e.g., BRK.B).  
 - The script is idempotent: it upserts into Postgres; ClickHouse inserts are de-duped by full reload or by pre-delete window.  
"""  
from \_\_future\_\_ import annotations  
  
import os  
import sys  
import time  
import math  
import argparse  
import datetime as dt  
from typing import List, Dict, Any, Iterable  
  
import requests  
import pandas as pd  
from loguru import logger  
from dotenv import load\_dotenv  
from sqlalchemy import create\_engine, text  
from sqlalchemy.engine import Engine  
from clickhouse\_connect import get\_client as get\_ch\_client  
  
# ----------------------------  
# Config & helpers  
# ----------------------------  
load\_dotenv()  
POLYGON\_API\_KEY = os.getenv("POLYGON\_API\_KEY")  
POSTGRES\_URI = os.getenv("POSTGRES\_URI")  
CLICKHOUSE\_URL = os.getenv("CLICKHOUSE\_URL", "http://localhost:8123")  
  
if not POLYGON\_API\_KEY:  
 logger.error("POLYGON\_API\_KEY is not set in your .env; please add it.")  
 sys.exit(1)  
  
SESSION = requests.Session()  
SESSION.headers.update({"Authorization": f"Bearer {POLYGON\_API\_KEY}"}) # Polygon auth  
  
ISO = "%Y-%m-%d"  
AGGS\_URL\_TMPL = "https://api.polygon.io/v2/aggs/ticker/{ticker}/range/1/day/{start}/{end}"  
SPLITS\_URL = "https://api.polygon.io/v3/reference/splits"  
DIVS\_URL = "https://api.polygon.io/v3/reference/dividends"  
  
  
def parse\_args() -> argparse.Namespace:  
 p = argparse.ArgumentParser(description="Ingest daily OHLCV + corp actions from Polygon")  
 p.add\_argument("--start", required=True, help="YYYY-MM-DD or 'yesterday' or 'today'")  
 p.add\_argument("--end", default="today", help="YYYY-MM-DD or 'today'")  
 p.add\_argument("--universe-file", required=True, help="CSV with header 'ticker'")  
 p.add\_argument("--sleep", type=float, default=0.25, help="Seconds between API calls")  
 p.add\_argument("--fetch-events", action="store\_true", help="Also fetch splits & dividends")  
 p.add\_argument("--delete-existing-window", type=int, default=0,  
 help="If >0, delete existing rows in Postgres for last N days before insert")  
 return p.parse\_args()  
  
  
def norm\_date(s: str) -> str:  
 s = s.strip().lower()  
 today = dt.date.today()  
 if s == "today":  
 return today.strftime(ISO)  
 if s == "yesterday":  
 return (today - dt.timedelta(days=1)).strftime(ISO)  
 # Assume already in ISO  
 return s  
  
  
def load\_universe(csv\_path: str) -> List[str]:  
 df = pd.read\_csv(csv\_path)  
 if "ticker" not in df.columns:  
 raise ValueError("Universe CSV must have a 'ticker' column")  
 tickers = (  
 df["ticker"].astype(str).str.strip().str.upper().tolist()  
 )  
 # Drop obvious blanks  
 return [t for t in tickers if t]  
  
  
# ----------------------------  
# Polygon fetchers  
# ----------------------------  
  
def http\_get(url: str, params: Dict[str, Any] | None = None, retries: int = 3) -> Dict[str, Any]:  
 for attempt in range(1, retries + 1):  
 try:  
 r = SESSION.get(url, params=params, timeout=30)  
 if r.status\_code == 429:  
 # Rate limited — back off  
 wait = attempt \* 2  
 logger.warning(f"429 rate limit; backing off {wait}s ...")  
 time.sleep(wait)  
 continue  
 r.raise\_for\_status()  
 return r.json()  
 except Exception as e:  
 logger.warning(f"GET failed (attempt {attempt}/{retries}) {url} : {e}")  
 time.sleep(1.5 \* attempt)  
 raise RuntimeError(f"Failed GET after {retries} attempts: {url}")  
  
  
def fetch\_daily\_aggregates(ticker: str, start: str, end: str) -> pd.DataFrame:  
 url = AGGS\_URL\_TMPL.format(ticker=ticker, start=start, end=end)  
 params = {  
 "adjusted": "true", # explicit; Polygon defaults to adjusted  
 "sort": "asc",  
 "limit": 50000,  
 }  
 data = http\_get(url, params)  
 results = data.get("results", []) or []  
 if not results:  
 return pd.DataFrame(columns=[  
 "ticker", "trade\_date", "open", "high", "low", "close", "volume", "vwap", "adjusted\_close"  
 ])  
 rows = []  
 for row in results:  
 # Polygon fields: o,h,l,c,v,vw,t  
 ts\_ms = int(row.get("t"))  
 trade\_date = dt.datetime.utcfromtimestamp(ts\_ms / 1000).date()  
 rows.append({  
 "ticker": ticker,  
 "trade\_date": trade\_date,  
 "open": row.get("o"),  
 "high": row.get("h"),  
 "low": row.get("l"),  
 "close": row.get("c"),  
 "volume": int(row.get("v")) if row.get("v") is not None else None,  
 "vwap": row.get("vw"),  
 "adjusted\_close": row.get("c"), # adjusted=true so c is adjusted  
 })  
 return pd.DataFrame.from\_records(rows)  
  
  
def fetch\_splits(ticker: str, start: str, end: str) -> pd.DataFrame:  
 params = {  
 "ticker": ticker,  
 "execution\_date.gte": start,  
 "execution\_date.lte": end,  
 "order": "asc",  
 "limit": 1000,  
 }  
 data = http\_get(SPLITS\_URL, params)  
 results = data.get("results", []) or []  
 if not results:  
 return pd.DataFrame(columns=[  
 "ticker", "ca\_type", "ex\_date", "record\_date", "payable\_date", "declared\_date",  
 "split\_from", "split\_to", "amount", "frequency"  
 ])  
 rows = []  
 for r in results:  
 rows.append({  
 "ticker": r.get("ticker"),  
 "ca\_type": "split",  
 "ex\_date": r.get("execution\_date"),  
 "record\_date": None,  
 "payable\_date": None,  
 "declared\_date": None,  
 "split\_from": r.get("split\_from"),  
 "split\_to": r.get("split\_to"),  
 "amount": None,  
 "frequency": None,  
 })  
 return pd.DataFrame.from\_records(rows)  
  
  
def fetch\_dividends(ticker: str, start: str, end: str) -> pd.DataFrame:  
 params = {  
 "ticker": ticker,  
 "ex\_dividend\_date.gte": start,  
 "ex\_dividend\_date.lte": end,  
 "order": "asc",  
 "limit": 1000,  
 }  
 data = http\_get(DIVS\_URL, params)  
 results = data.get("results", []) or []  
 if not results:  
 return pd.DataFrame(columns=[  
 "ticker", "ca\_type", "ex\_date", "record\_date", "payable\_date", "declared\_date",  
 "split\_from", "split\_to", "amount", "frequency"  
 ])  
 rows = []  
 for r in results:  
 rows.append({  
 "ticker": r.get("ticker"),  
 "ca\_type": "dividend",  
 "ex\_date": r.get("ex\_dividend\_date"),  
 "record\_date": r.get("record\_date"),  
 "payable\_date": r.get("pay\_date"),  
 "declared\_date": r.get("declaration\_date"),  
 "split\_from": None,  
 "split\_to": None,  
 "amount": r.get("cash\_amount"),  
 "frequency": str(r.get("frequency")) if r.get("frequency") is not None else None,  
 })  
 return pd.DataFrame.from\_records(rows)  
  
  
# ----------------------------  
# Storage: Postgres & ClickHouse  
# ----------------------------  
  
def init\_postgres(engine: Engine):  
 with engine.begin() as conn:  
 conn.execute(text("""  
 CREATE SCHEMA IF NOT EXISTS core;  
 CREATE TABLE IF NOT EXISTS core.prices\_daily (  
 ticker TEXT NOT NULL,  
 trade\_date DATE NOT NULL,  
 open DOUBLE PRECISION,  
 high DOUBLE PRECISION,  
 low DOUBLE PRECISION,  
 close DOUBLE PRECISION,  
 volume BIGINT,  
 vwap DOUBLE PRECISION,  
 adjusted\_close DOUBLE PRECISION,  
 PRIMARY KEY (ticker, trade\_date)  
 );  
 CREATE TABLE IF NOT EXISTS core.co