

geowerkstatt

Geodaten-ETL mit Apache Hop

Ein Erfahrungsbericht aus der Praxis

Technischer Austausch KGK, 03.09.2026

Lukas Wiss, geowerkstatt GmbH



Pilotprojekt mit GEOTEST AG



v2.17.0



v1.1.0



Ausgangslage

Data Lake mit räumlich
und zeitlich referenzierten
Rohdaten.
(Sensordaten als CSV-
Textdateien)



Ziel

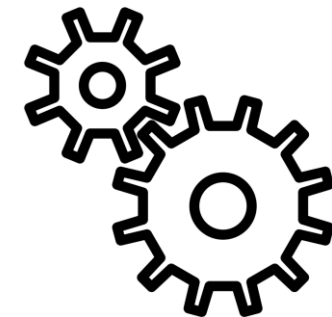
STAC-Katalog als
Datengrundlage für
internen Datennavigator.
Anbindung anderer Tools
möglich (z.B. WebGIS)



Varianten

Python-Script
Umsetzung durch Kundin

Apache Hop Pipeline
(mit Atol CD-GIS-Plugin)
Umsetzung durch
geowerkstatt





Warum STAC?

1. Gemeinsamer Nenner für alle Daten mit Raum- und Zeitbezug

Im Data Lake der Kundin liegen sehr unterschiedliche Assets diversester Projekte über mehrere Jahrzehnte

2. Die Rohdaten liegen als Dateien vor

1 Standort und 1 Zeitintervall pro Datei
→ passt auf den STAC Asset-Begriff

3. Standardisierte Sicht auf der Konsumentenseite

Der Datenbrowser muss nur ein Schema kennen, das immer bekannter wird

4. STAC wird präsenter im Markt

STAC hat in der Geodaten-Community schnell Fuss gefasst



Hypothese

Apache Hop ist für diesen Job passend, weil...

einfache, offene Formate (CSV)

Nur geometrische Basisoperationen
vonnöten (Koordinaten zu Punktdaten,
Aggregationen, Bounding Box Generation).

Perspektive

Gleichartige, reproduzierbare ETL-
Prozesse aufbauen und automatisieren

Für automatisierungswürdige Prozesse:
Orchestrierung direkt im selben Tool

No-Code

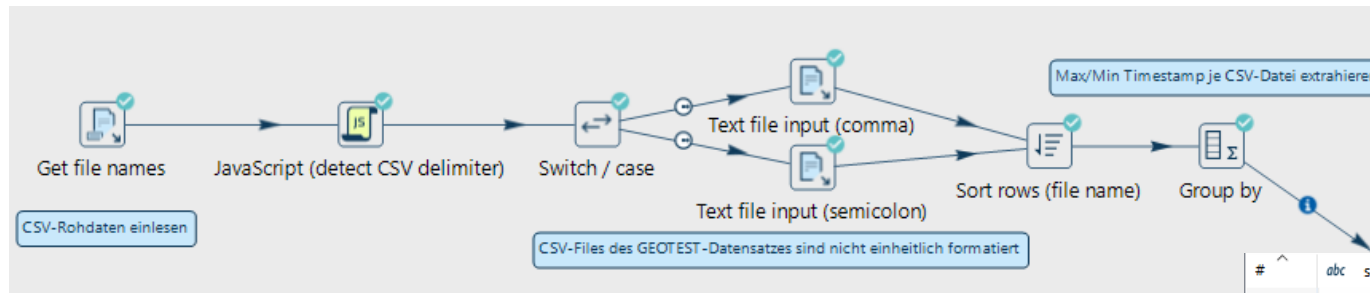
Die vier Prozessierungsschritte



No-Code trägt bis Schritt 3 — Schritt 4 ist Handarbeit in JavaScript



Schritt 1: Rohdaten auslesen und Datensätze gruppieren



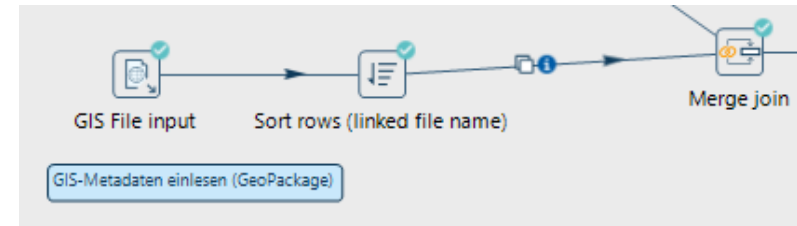
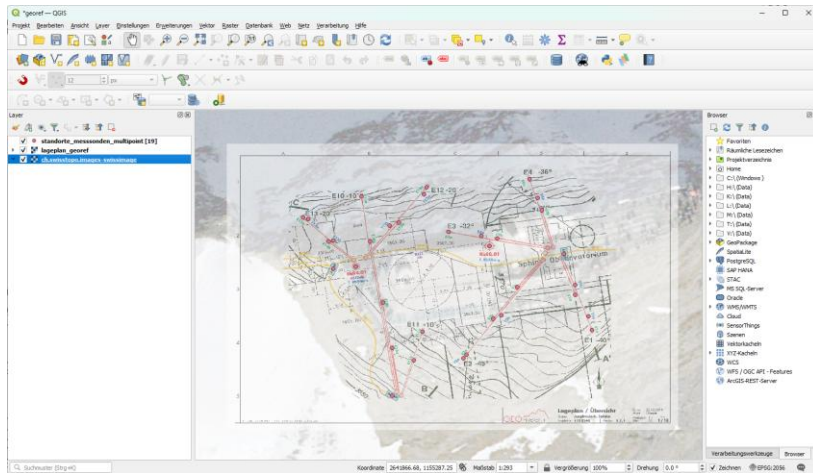
Name	Änderungsdatum	Typ	Größe
A4000_extensometer-e12.csv	02.04.2026 13:13	Microsoft Excel-CSV-Datei	18'242 KB
A4000_extensometer-e13.csv	02.04.2026 13:14	Microsoft Excel-CSV-Datei	18'242 KB
A4000_loggerdaten.csv	02.04.2026 13:14	Microsoft Excel-CSV-Datei	15'512 KB
A4000_thermistoren-12.csv	02.04.2026 13:14	Microsoft Excel-CSV-Datei	18'926 KB

```
UTC Time,T13b (4m),T13a (7m)
2023-03-08 06:05:00,-9.190584,0.000000
2023-03-08 06:00:00,-9.189060,0.000000
2023-03-08 05:55:00,-9.188840,0.000000
2023-03-08 05:50:00,-9.188382,0.000000
2023-03-08 05:45:00,-9.187380,0.000000
2023-03-08 05:40:00,-9.187168,0.000000
2023-03-08 05:35:00,-9.186595,0.000000
```

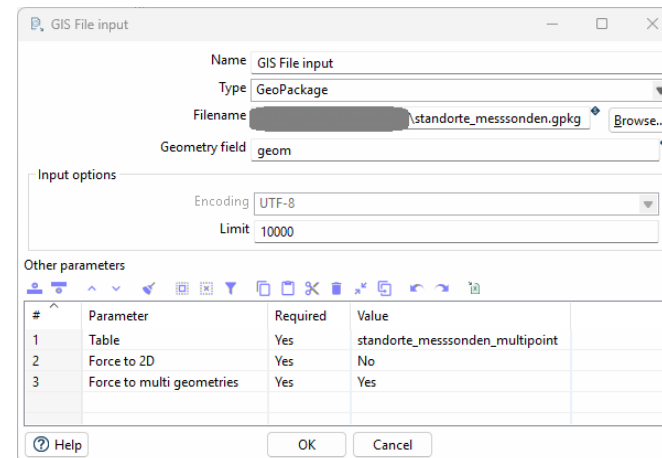
Die Dateien müssen geparsed,
die Max- und Min-Timestamp je
Datei gefunden werden
(ca. 8 Mio Zeilen)

#	abc	short_filename	UTC Time MAX	UTC Time MIN
1		A4000_extensometer-e12.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
2		A4000_extensometer-e13.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
3		A4000_loggerdaten.csv	08.03.2023 06:40:00	19.06.2018 11:49:44
4		A4000_thermistoren-12.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
5		A4000_thermistoren-13.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
6		A4000_thermistoren-Rb94.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
7		A4001_extensometer-E1.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
8		A4001_extensometer-E2.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
9		A4001_extensometer-E3.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
10		A4001_extensometer-E4.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
11		A4001_loggerdaten.csv	08.03.2023 06:40:00	12.07.2018 10:20:00
12		A4001_thermistoren-Rb88-abc.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
13		A4001_thermistoren-Rb88-defg.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
14		A4001_thermistoren-T-12.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
15		A4001_thermistoren-T-34.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
16		A4001_thermistoren-T.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
17		A4010_extensometer-E10.csv	08.03.2023 06:05:00	01.01.2011 12:00:00
18		A4010_extensometer-E11.csv	08.03.2023 06:05:00	01.01.2011 12:00:00

Schritt 2: Geometadaten einlesen, Join mit Rohdaten



Lageinformation der Messsonden aus PDF georeferenziert und in einem GeoPackage gespeichert (QGIS)



Einlesen (Hop Atol CD-GIS-Plugin) und Join mit den extrahierten Timestamps der Datenfiles



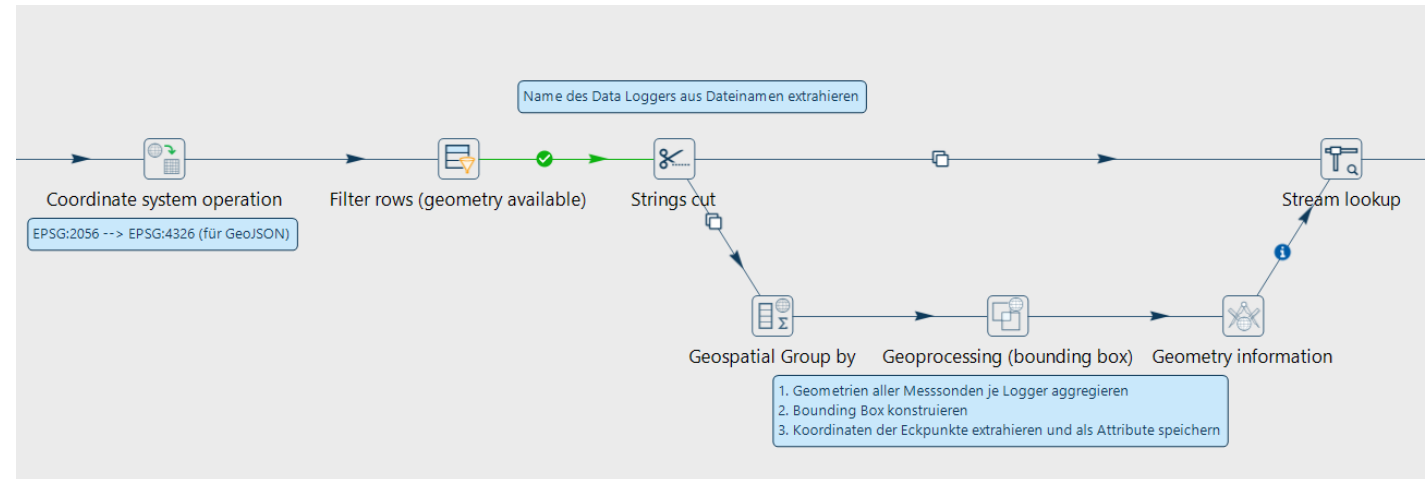
Schritt 3: Einfache Geodatenoperationen durchführen

Für die STAC Collection benötigen wir eine Bounding Box.

Zudem werden in STAC Geometadaten grundsätzlich in GeoJSON formatiert:

→ KBS-Transformation von CH1903+/LV95 (EPSG:2056) nach WGS84 (EPSG:4326)<

→ Koordinatenformatierung in GeoJSON





Schritt 4: STAC-Elemente aufbauen

```
Script1 X
var clean = String(geom_union);
var sridIdx = clean.indexOf(";");
if (sridIdx > -1) clean = clean.substring(sridIdx + 1).trim();

var nums = [];
var current = "";
var inNum = false;
for (var i = 0; i < clean.length; i++) {
    var c = clean.charAt(i);
    if ((c >= "0" && c <= "9" || c === "." || c === "-") {
        current += c; inNum = true;
    } else if (inNum) {
        nums.push(parseFloat(current));
        current = ""; inNum = false;
    }
}
if (inNum) nums.push(parseFloat(current));

var coords = [];
for (var i = 0; i < nums.length; i += 2) {
    coords.push("[\" + nums[i] + \",\" + nums[i+1] + \"]");
}

// STAC-konforme Geometrien bilden
var geomType = clean.indexOf("MULTIPOINT") === 0 ? "MultiPoint" : "Point";
var geometry_json = "{\\\"type\\\":\\\"\" + geomType + \"\\\",\\\"coordinates\\\":[" + coords.join(",") + "]}";

// Asset-Einträge (1 oder 2 Files je Station)
var asset_entry_1 = '{\"type\":\"text/csv\",\"title\":\"' + String(logger) + ' - ' + String(art) + ' ' + String(bezeichnung) + '\",\"roles\":[\"data\"]}';

var asset_entry_2 = "";
if (datafile_2 != null && String(datafile_2).trim().length > 0) {
    var _df2 = String(datafile_2).trim();
    asset_entry_2 = '{\"type\":\"text/csv\",\"title\":\"' + String(logger) + ' - ' + String(art) + ' ' + String(bezeichnung) + ' (2)\"' + '\",\"roles\":[\"data\"]}';
}

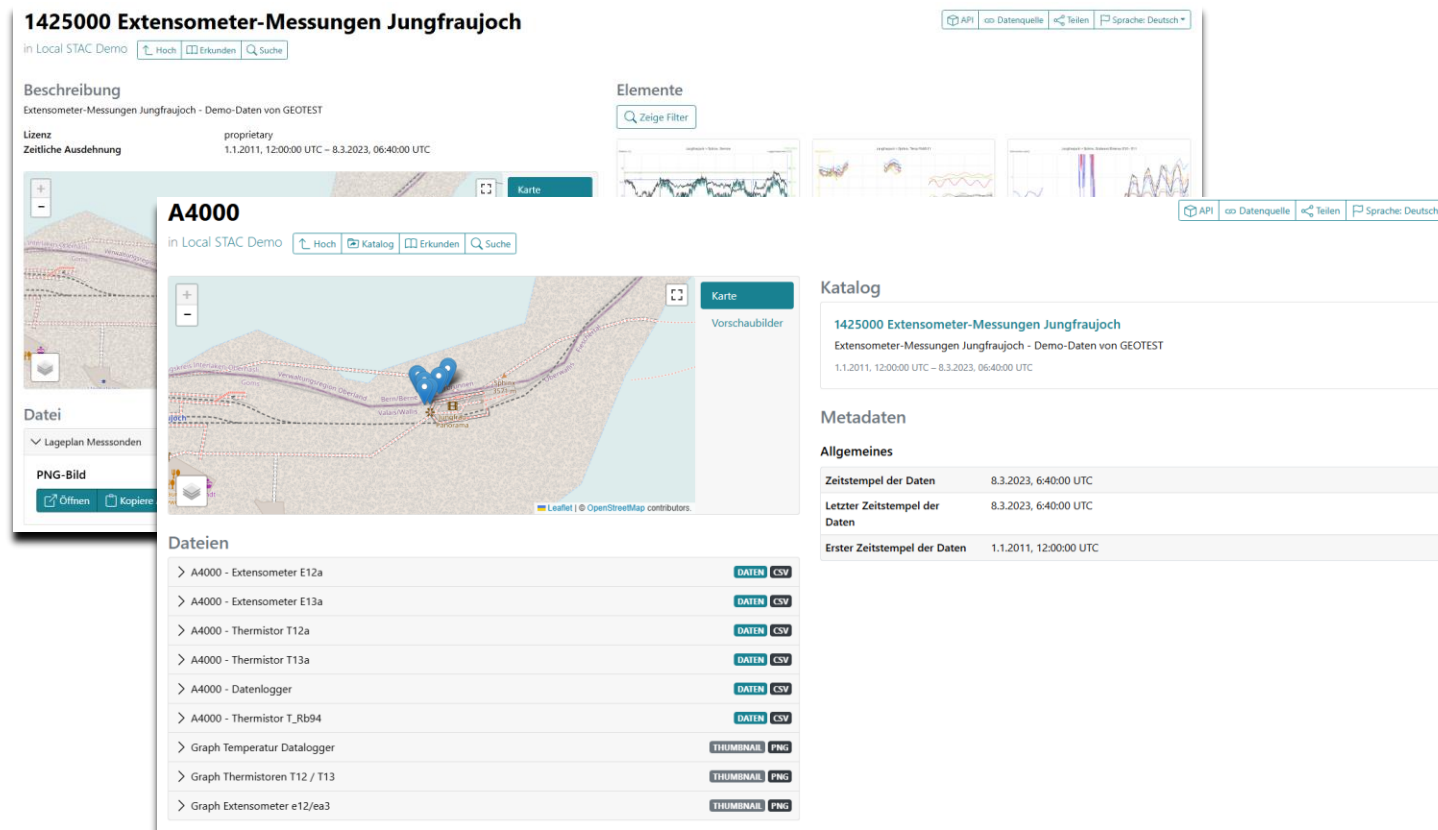
var asset_entry = asset_entry_1 + asset_entry_2;
```

Kein STAC-Writer oder Plugin
In Hop muss der STAC-Katalog
händisch per eingebettetem
JavaScript-Code zusammengesetzt
werden

Aufwändiger und manueller
als mit einer einschlägigen Python-
Library wie z.B. [PySTAC](#)



Ziel erreicht. Mit No-Code?



No Low Coding: STAC-Katalog muss manuell in JavaScript Transform zusammengestellt werden

Python-Integration in Script-Transform ist zwar vorhanden, basiert aber auf dem veralteten Jython (auf Python 2 beschränkt!). Daher kam das eingebettete JavaScript zum Einsatz.



Kleiner Exkurs: Infrastructure as Code mit Hop

```
<transform>
  <name>Add constants (ogr2ogr command)</name>
  <type>Constant</type>
  <description/>
  <distributed>Y</distributed>
  <custom_distribution/>
  <copies>1</copies>
  <partitioning>
    <method>none</method>
    <schema_name/>
  </partitioning>
  <fields>
    <field>
      <currency/>
      <decimal/>
      <format/>
      <group/>
      <length>-1</length>
      <name>ogr2ogr</name>
      <nullif>C:\OSGeo4W\OSGeo4W.bat ogr2ogr -f Parquet "C:\Users\lwiss\Downloads\test.parquet" "${output_filename}"</nullif>
      <precision>-1</precision>
      <set_empty_string>N</set_empty_string>
      <type>String</type>
    </field>
  </fields>
  <attributes/>
  <GUI>
    <xloc>832</xloc>
    <yloc>336</yloc>
  </GUI>
</transform>
```

Hop Pipelines und Workflows sind sehr einfach gehaltene XML-Files

→ Change-Tracking sehr einfach

→ Verwaltung in einem Git-Repository möglich

→ Wenig Overhead und Speicherbedarf



Das O in Hop steht für Orchestrierung.

Im Projektbeispiel zwar nicht benötigt, aber dennoch erwähnenswert weil...

1. keine zusätzlichen Tools vonnöten

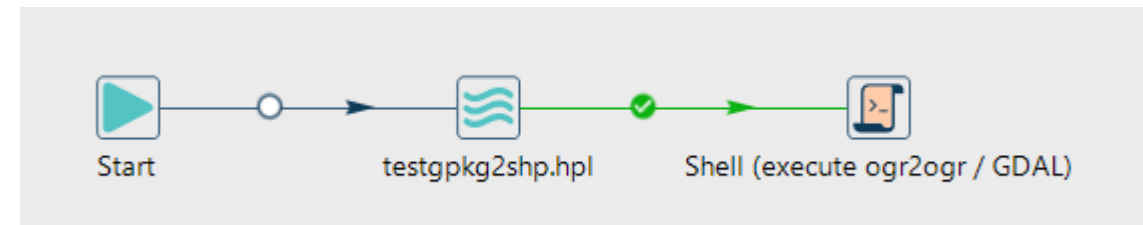
hop und hop-gui decken ETL und Orchestrierung ab

2. Gleiche Logik

kein UI-Bruch zwischen Pipelines und Workflows

3. Leicht und plattformunabhängig

gleiche Applikation läuft auch headless auf Linux, Unix, Windows-Servern.
Hardwareanforderungen dürften überschaubar sein.





Das Hop-GUI: Rustikal, aber funktional

The screenshot displays the Hop-GUI interface. On the left, a data pipeline is visible with several components: 'Fields Converter LFP3', 'Select values LFP3', 'Fields Converter HFP3', 'Select values HFP3', 'Fields Converter GP', 'Select values GP', and 'JSON input LFP1'. The 'Select values' configuration window is open on the right, showing the 'Select & Alter' tab. The window title is 'Select values', and the 'Transform name' is 'Select values HFP3'. The 'Fields' section shows a table with the following data:

#	Fieldname	Rename to	Length	Precision
1	nummer	PktNr		
2	X			3
3	Y			3
4	hoehengeometrie	Z		3
5	lagegenauigkeit	LageGen		
6	hoehengenaueigkeit	HoeheGen		



Das Hop-GUI: Rustikal, aber funktional

1. Für techniknahe User:innen

Für Anwendungsfälle und Zielpublikum i.O..

Beim Erstellen von ETL-Pipelines ist aber auch der Zeitaufwand ein wichtiger Faktor. Komfortvorteile gegenüber codebasierten Alternativen wie DuckDB oder Python sind für GUI ETL-Tools ein Muss, im AI-Zeitalter erst recht!

2. Wenig UI-Automatismen

bei manchen Transforms muss jeder Datentyp explizit von Hand konfiguriert werden.

3. Pipeline erst lauffähig, wenn sie 100% valide ist

nervig beim Prototyping: Ist ein unfertig konfigurierter Transformer (nicht angeschlossen) in meiner Pipeline, kann ich diese nicht starten.



Hop-Achillesferse? Geodaten (Stand heute)

Kein Data Viewing bei Geodaten möglich

Vorschau nur in Tabellenform, Geometrie bestenfalls als WKT

Hop-Engine handhabt Geometrien mit JTS

keine Bogengeometrien unterstützt!

geometry
SRID=2056;POINT (2585908.803 1224231.906)
SRID=2056;POINT (2589720.95 1225176.815)
SRID=2056;POINT (2586439.705 1224039.769)
SRID=2056;POINT (2586836.054 1224061.247)

Native Unterstützung von Geodatenformaten?

Nur wenn man CSVs und Datenbanken mit Koordinatenfeldern als Geodaten betrachtet

Die vorhandene Funktionalität wird bislang nur über Community Plugins abgedeckt

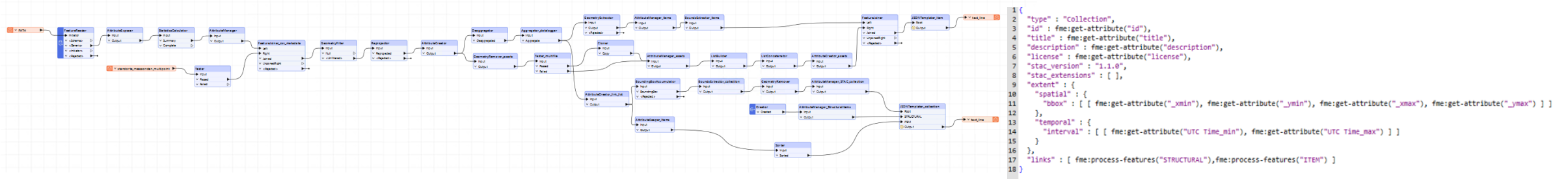
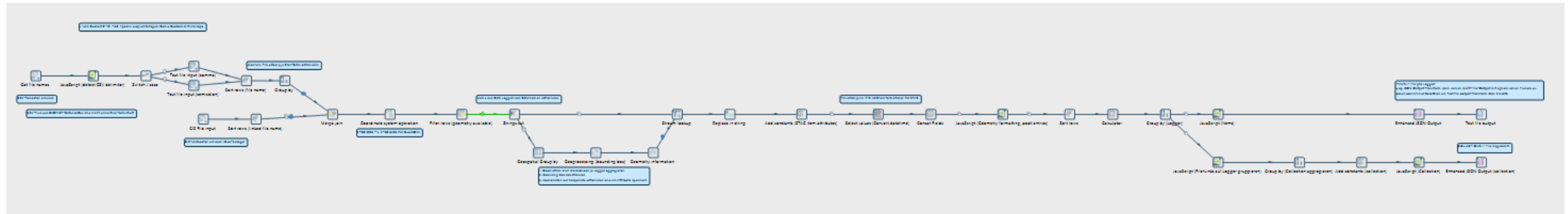
Atol CD, FH Ost, Stefan Ziegler...



Bei Geodaten-ETL steht immer ein Elefant im Raum...



**FME = No-Code? Ja, auch in diesem Fallbeispiel, aber...
Stichwort: manuelles JSON-Formatting**





Performance: Hop war im Beispiel schnell(er) als FME

	Apache Hop 2.17	FME 2025.1
Laufzeit mit aktivem Cache Mittelwert aus 3 Durchläufen	-	3m 29 s
Laufzeit ohne aktiven Cache Mittelwert aus 3 Durchläufen	50 s	1m 56 s

Fazit: Hop als graphischer Geo-ETL-Contender?



1. Ja, weil ...

- Für Daten ohne Geometrien bewährt und robust
- Performant, prozessierte im Beispiel schneller als FME
- ETL und Orchestrierung im selben Tool, ohne Zusatzapplikation.
- OSS, leicht, plattformunabhängig, headless betreibbar, IaC-fähig

2. Nein, weil ...

- Noch zu wenig native Geodatenunterstützung, keine Bogengeometrien, kein Data Viewing, native Formatunterstützung quasi nicht vorhanden
- Die Geo-Funktionalität hängt an Community-Plugins
- Über die Basisoperationen hinaus endet No-Code. Im vorliegenden Pilotprojekt leistete PySTAC dasselbe mit weniger Codezeilen

3. Künftig ja, falls ...

- Bogengeometrien unterstützt werden und ein Geometrie-Viewer in die Pipeline kommt
- Geodatenformate nativ in die Engine wandern
- Das GUI moderner wird. Komfort => Zeitersparnis => Wirtschaftlichkeit
- Jython einem zeitgemässen eingebetteten Python-Environment weicht

Die Messlatte hat sich in den letzten Jahren verschoben...

...denn ein graphisches ETL-Tool konkurriert nicht mehr mit Handarbeit, sondern mit KI-generiertem Code.

«Gewinner» des Pilotprojekts?



In diesem Praxisbeispiel hat sich die Kundin dennoch für Python als Tool entschieden

→ No Code war wünschenswert, aber untergeordnet

→ PySTAC Library ermöglicht Bildung von STAC-Elementen mit wenigen Codezeilen

→ Inhouse-Kompetenzen in Python vorhanden (wie bei den meisten Ingenieurbüros...)

**Merci für eure
Aufmerksamkeit!**

