DHBW Loerrach
Baden-Wuerttemberg
Cooperative State University

Data Lakes und Data Warehouses

Prof Dr Veit U.B. Schenk, schenkv@dhbw-loerrach.de

Von DB zu DW zu DL

- Es waren einmal viele Daten (in der DB)…
- … und wir wollten aber BI machen …
- … und luden deswegen unsere „Tagesgeschäftsdaten“ ins Archiv (das DW) wo wir für immer und ewig slicen und dicen können
- der Ladevorgang heißt übrigens ETL: https://de.wikipedia.org/wiki/ETL-Prozess
- Tja, und nun sagen mir plötzlich alle, dass ich einen Data Lake brauche.

Warum sollen wir uns überhaupt mit Data Lakes beschäftigen?

- https://www.theregister.com/2021/04/07/redis_labs_doubles_value_to/
- O-ha!
- Was machen Databricks und Snowflake?
- Los geht's mit Aufgabe 1 in Moodle

A flurry of data warehouse activity surrounds Snowflake's staggering $120bn valuation

READ MORE →

The company was founded in 2011 by Ofer Bengal, now CEO, and Shoolman. It has doubled its nominal value in less than nine months. Last August, the firm secured $100m in a funding round that sets its valuation at an estimated $1bn. It has raised $347m investment since its inception.

The open-source Redis database was maintained by Salvatore Sanfilippo, better known by the nickname antirez, before he stepped down in June 2020. The project saw its first release in 2009.

Redis Labs will be hoping to follow in the footsteps of Databricks and Snowflake in balancing the data-plus-cloud-equals-megabucks equation. The former was valued a $28bn in a Series G funding round in February and the latter may well end up being written up as a hype-cycle case study after its post-IPO valuation hit $120bn, up from $1.8bn just three years earlier. ®

Was wollen wir mit unseren Daten machen?

- Speichern
- „Data Ingestion“
- Analytics
- Real-Time Analytics
- Machine Learning?
- Macht es Sinn eine „one-size-fits-all“-DB zu nutzen?
- (Hint: Nö!)

Typisches Szenario

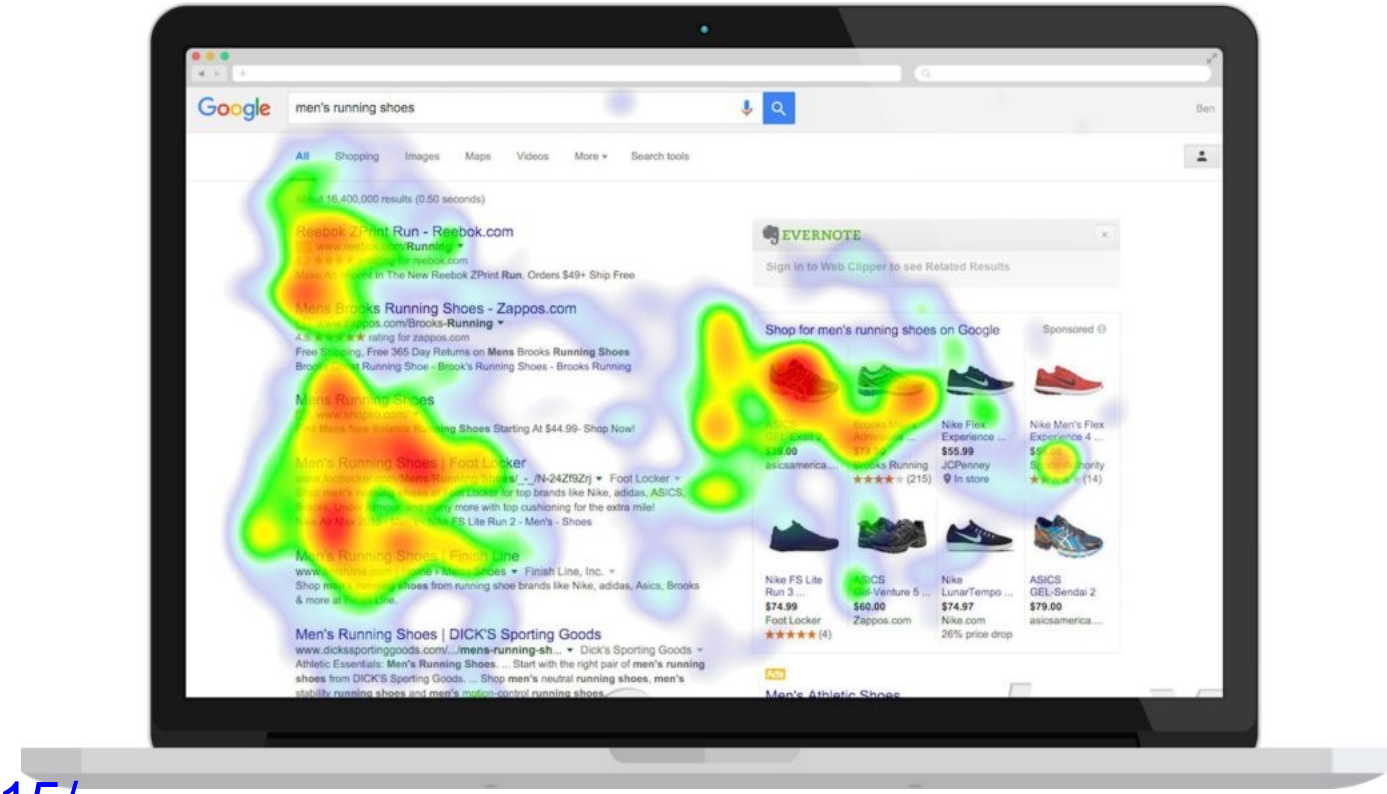
- Daten in „Monsterdatenbank“, oder…
- … auf mehrere Datenbanken verteilt
- Probleme:
  - Monster: Load issues (vertical scaling)
  - Verteilt: Security, Consistency, …
- Lösung: eine Monster-Monster-(verteilte)-Datenbank?
- Nein! Ein Data-Lake ist nicht nur eine „große“ Datenbank, sondern…

Was ist ein Data-Lake?

- Ein System, dass bei den folgenden Dingen hilft:
  - Ingest and Store:
    - Data Ingestion
    - (Pre/Post)Processing (incl z.B. Cleansing, Aggregating, Merging)
    - Storage
  - Catalog and Search (der „SQL“-Teil)
    - Schnell die gesuchten Daten finden
    - Egal ob SQL oder NoSQL Datenbanken: wenn ich nicht an die Daten komme (z.B. weil ich gar nicht weiß, wo sie sind), dann habe ich einen Data Swamp, keinen Data Lake
  - Process and Serve:
    - Z.B. Hadoop-Cluster, der sich hier aber NUR um „Berechnungen“ kümmert (also NICHT das HDFS für das Speichern nutzt)
    - Machine Learning
    - Visualisierung
  - Protect and Secure:
    - Data Encryption
    - Data Retention
    - Access Control
- Aber je nachdem, wo der Fokus liegt, haben auch Data-Lakes ihre Schwerpunkte!

Immer noch ganz schön abstrakt…

- Änderungen im Kaufverhalten?
- Vielleicht durch Änderungen auf der Webseite?
- Daten für Data-Lake:
  - Transaktionen
  - Weblogs
  - Clickstreaming
  - Social Media
  - IoT
  - Flugzeuge (warum?) https://www.forbes.com/sites/sap/2015/02/19/how-big-data-keeps-planes-in-the-air/



Quelle: https://www.datahash.com/services/google-analytics-heatmap/

Eigenschaften von Data Lakes

- Data Agnostic: egal was reinkommt, es bleibt im „rohen“ Format (keine bestimmten Datentypen oder Schema)
- Future-Proof: selbst wenn ich die Daten jetzt nicht brauche, u.U. brauche ich sie in der Zukunft.
  - Bsp: https://www.roche.com/media/releases/med-cor-2018-04-06.htm

DHBW Loerrach
Baden-Wuerttemberg
Cooperative State University

Datenbanken II: MongoDB

Prof Dr Veit U.B. Schenk, schenkv@dhbw-loerrach.de

Document Databases

- MongoDB
  - Strukturiert, aber nicht so arg strukturiert (Tables, Rows, Columns…)
  - Statt dessen: JSON (ok, ok, BSON)
  - Bsp: https://www.mongodb.com/developer/article/atlas-sample-datasets/#std-label-atlas-sample-data-local-installation (Achtung: ATLAS!!!)
- Installation:
  - https://www.mongodb.com/try/download/community
  - Oder Docker: `docker run --name brandspankingnewMongodb -p 27017:27017 -d mongo:latest`
  - Aber Compass brauchen wir auf jeden Fall
  - Falls es gar nicht geht (WTB???) dann Atlas: https://www.mongodb.com/cloud/atlas/register

Interaktion von der Shell aus

- mongo "mongodb://localhost/meineErsteMongoDB„
- Für Atlas:

```
mongo "mongodb+srv://sandbox.[redacted].mongodb.net/myFirstDatabase" --username m001-student
```

- Show dbs
- Use xxx
- Show collections
- db.rentals.find({"start_station_id": "5847.08" }).count()
- db.rentals.find({"start_station_id": "5847.08", "end_station_id": "5788.13" }).count()
- db.rentals.find({"start_station_id": "5847.08", "end_station_id": "5788.13" }).**pretty()**

Aufgabe

- Was passiert hier (Beschreibe GENAU…)
- use jabbathehut
- db.bobafett.insert([{"_id": 4711, "name": "Boba", "surname": "Fett" }, { "this": 99 }])
- db.bobafett.insert([{"_id": 4711, "name": "Boba", "surname": "Fett" }, { "this": 99 }])
- db.bobafett.insert([{"_id": 4712, "name": "Boba", "surname": "Fett" }, { "this": 99 }])

Compass

- Let’s city bike
- https://account.citibikenyc.com/access-plans
- https://ride.citibikenyc.com/system-data
- Let‘s import some data

Collections, Documents, Fields

- Collection: wie eine Datenbank in SQL
- Document: wie ein Eintrag in einer Tabelle in SQL
- Field: besteht immer aus Name und Value (wie eine Spalte in einer Tabelle)


```
{
"account_id": 794875,
"transaction_count": 6,
"bucket_start_date": {"$date": 693792000000},
"bucket_end_date": {"$date": 1473120000000},
"transactions": [
   {
      "date": {"$date": 1325030400000},
      "amount": 1197,
      "transaction_code": "buy",
      "symbol": "nvda",
      "price": "12.733002429934103361119923647493124008178710937 5",
      "total": "15241.40390863112172326054861"
   },
   {
      "date": {"$date": 1465776000000},
      "amount": 8797,
      "transaction_code": "buy",
      "symbol": "nvda",
      "price": "46.538731724063914896305504953488707542419433593 75",
      "total": "409401.222976590259342799527 1"
   },
   {
      "date": {"$date": 1472601600000},
```

JSON: JavaScript Object Notation

- Anfang und Ende mit {}
- „Key“: Value
- „Key1“:value1, key2: value2, key3:value3,….
- „key“ (aber bitte richtige Anführungszeichen, nicht die „“, die Powerpoint mir hier aufzwingt)
- Keys sind in MongoDB „fields“
- In Wirklichkeit BSON (Binary JSON) – siehe Datenimport in Compass!

PyMongo

- https://www.w3schools.com/python/python_mongodb_getstarted.asp
- https://towardsdatascience.com/using-mongo-databases-in-python-e93bc3b6ff5f

- https://www.mongodbtutorial.org/

DHBW Loerrach
Baden-Wuerttemberg
Cooperative State University

Jupyter Notebook Installieren

Prof Dr Veit U.B. Schenk, schenkv@dhbw-loerrach.de

DHBW Loerrach
Baden-Wuerttemberg
Cooperative State University

Juypter Notebook mit Anaconda

- Anaconda runterladen und installieren
- Datenstruktur anlegen:
- `/Users/veitschenk/DHBW/mongodb-analytics/intro-to-mongodb/notebooks`
- `conda create -n intro-to-mongodb (alt: Anaconda)`
- `conda activate intro-to-mongodb`
- pip install pymongo
- `ipython kernel install --user --name=mongodb-intro-env`
- `https://github.com/gusutabopb/imongo`

DHBW Loerrach
Baden-Wuerttemberg
Cooperative State University

Datenbanken II: Schema Design I in MongoDB

Prof Dr Veit U.B. Schenk, schenkv@dhbw-loerrach.de

Schema Design

- Ist wichtig:
  - Performance
  - Scalability
- Relationale Datebanken:
  - Design basiert auf entities. (“… alles was zum Auto gehört…”)
  - Datendesign UNABHÄNGIG von den Queries
  - Datendesign folgt bestimmtem Schema (1., 2., 3…. Normalform)
- Document/NoSQL Datenbanken:
  - Schema deutlich “näher” an den Queries
  - Kein Standard-Ansatz

Embedding vs Referencing

- Embedding: Bsp: Order info zusammen mit Kunden-Info
- Vorteile:
  - Ein fetch für “alle” Infos
  - Keine (teuren) Joins
  - Updates an einer Stelle (kein teurers (b)locking)
- Nachteile
  - “Verschwenderisch” wenn ich nur wenig Info brauche (muss dennoch alles lesen)
  - U.U. maximale Dokumentengröße: https://www.mongodb.com/docs/manual/core/document/
  - Was ist maximal Größe, warum, was kann man machen wenn > 16mb? (gridfs)

Referencing

- Kunden in 1 Collection, Orders in anderer Collection
- Eigentlich wie relational: keine Duplication, kleinere Docs (< 16MB), aber “teurer” da ”joins” (mehrere Queries), und updates nicht mehr atomic.

Wann benutze ich was?

- Wenn ich alle Infos brauche? **Embed**
- 1/many-many: **Reference** (z.B. 1 Auto besteht aus mehreren Teilen, diese Teile können auch in anderen Autos verwendet werden)
- 1-infinity: server logs. Wenn die alle in einem Array wären, dann wären die 16MB schnell voll
- 2-way **referencing**: main record array of IDs, …
- Und jetzt in der Praxis … timing von embedding vs referencing: Daten aus citibike auseinanderklamüsern und in referencing model umwandeln.
- Read-write ratio: wie sieht es da aus mit timing?

Hybrid

- Z.B. nur die letzten x reviews embedded, der rest referenced.

Aufgaben

- Installiere Jupyter Notebook
- Evaluiert in Gruppen, wie/ob der MongoDB-kernel https://github.com/gusutabopb/imongo gut funktioniert (an Hand von z.B. dem CRUD-Tutorial: https://www.w3schools.com/python/python_mongodb_getstarted.asp
- Bonus: das Gleiche für den JavaScript kernel: https://github.com/n-riesco/ijavascript
- Findet jeweils 3 Beispiele für “embedding”, ”referencing” und “hybrid” aus der ersten Vorlesung (Bild in Moodle)
- In Jupyter Notebook, evaluiert die Performance von embedding vs referencing
- Argumentieren können welche Art von Referenz (https://www.mongodb.com/docs/manual/reference/database-references/)

DHBW Loerrach

Baden-Wuerttemberg
Cooperative State University

Datenbanken II: (MongoDB) Indexing

Prof Dr Veit U.B. Schenk, schenkv@dhbw-loerrach.de

DHBW Loerrach
Baden-Wuerttemberg
Cooperative State University

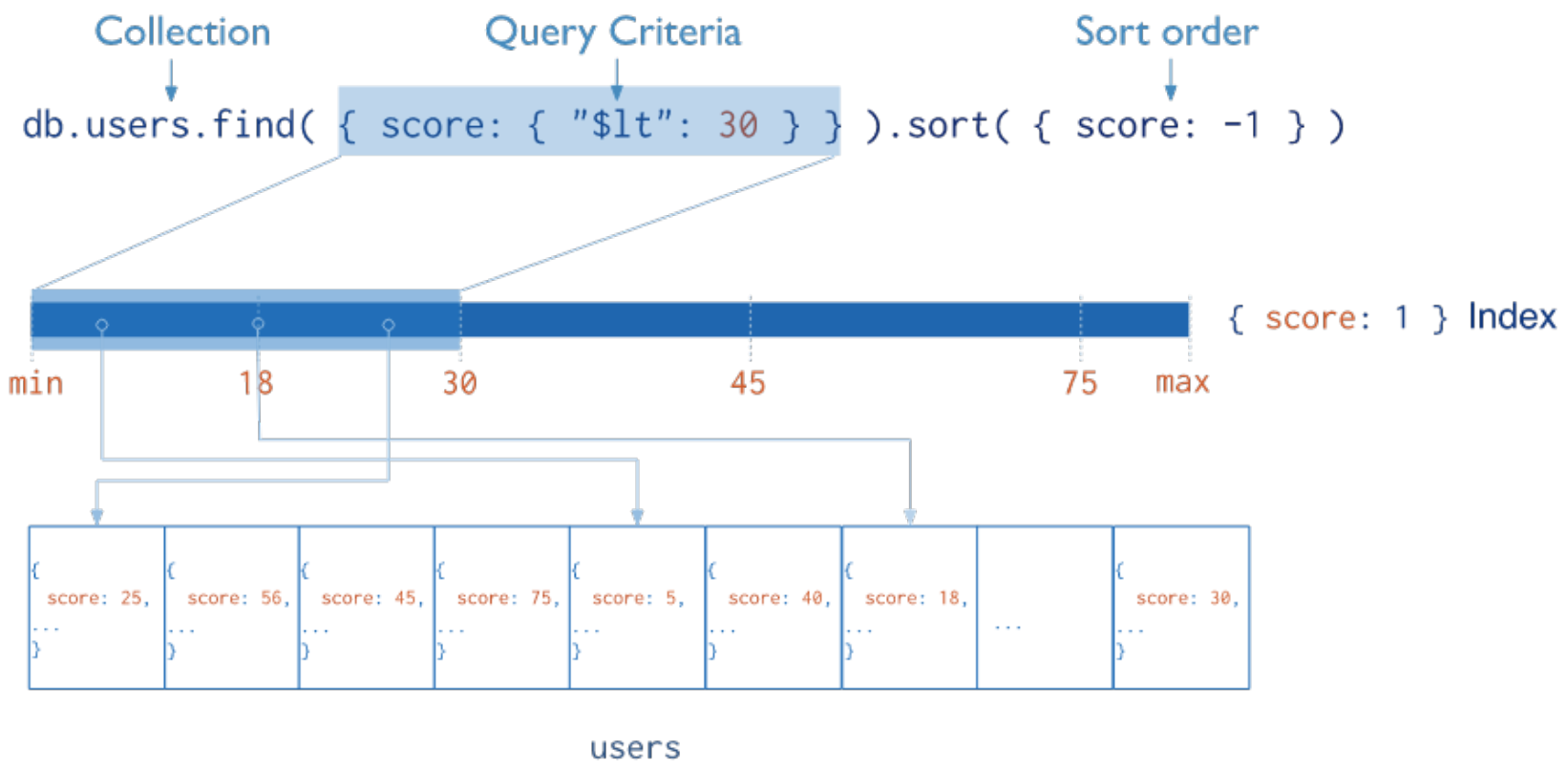
Aufweckaufgabe

- Erstelle ein kleines Python-Programm, das
  - Eine MongoDB Datenbank mit dem Namen “Kontakte”, und darin
  - Eine Collection mit dem Namen “Telefonnummern” erstellt.
  - Danach erstellt das Programm 100000 Documents mit jeweils unterschiedlichen Telefonnummern (z.B. 0761 000 000 bis 0761 100 000)
- In COMPASS:
  - Suche nach einer beliebigen Nummer und schaue, wie lange es dauert.
  - Erstelle einen Index für die Telefonnummer
  - Suche wieder, wie lange dauert es diesmal?

Der Grund für Indexes (Indices?)

- Ohne Index muss MongoDB ALLE Datensätze anschauen! (der sogenannte „Collection Scan“)
- Ein Index speichert ein/mehrere bestimmte Felder, egal wo im Dokument (d.h. auch subdocuments)
- Das Format ist (i.d.R) ein B-Tree (https://de.wikipedia.org/wiki/B-Baum)
- Der Index ist **sortiert**.
- Dadurch kann man „**equality**“ matches UND „**range-based**“ queries machen
- Außerdem kann das Ergebnis sortiert sein
- Ein Index ist auf **collection-level definiert**
- Default Index: `_id`

Beispiel: https://docs.mongodb.com/manual/indexes/



DHBW Loerrach
Baden-Wuerttemberg
Cooperative State University

Order matters

Order of Fields in a Compound Index

The order of the fields matters when creating the index and the sort order. It is recommended to list the fields in the following order: Equality, Sort, and Range.

- Equality: field/s that matches on a single field value in a query
- Sort: field/s that orders the results by in a query
- Range: field/s that the query filter in a range of valid values

The following query includes an equality match on the active field, a sort on birthday (descending) and name (ascer and a range query on birthday too.

```
db.customers.find({
  birthdate: {
    $gte:ISODate("1977-01-01")
    },
    active:true
    }).sort({
      birthdate:-1,
      name:1
      })
```

Here's an example of an efficient index for this query:

```
db.customers.createIndex({
  active:1,
  birthdate:-1,
  name:1
})
```

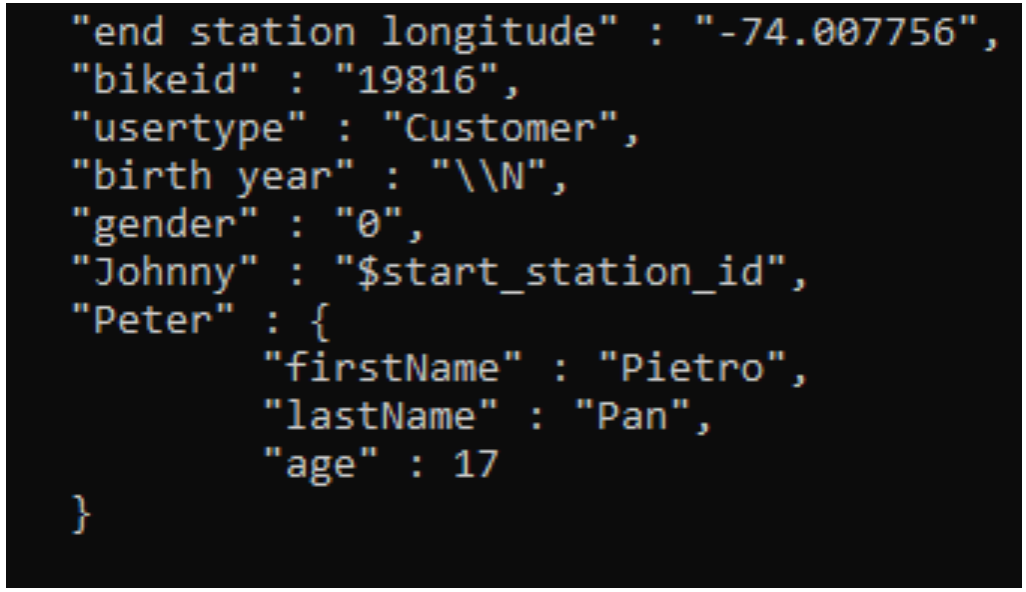
Index in Python

- In MongoDB: createIndex, in Python: create_index
- `db.collection.create_index( [(<key and index type specification>)], <options> )`
- Bsp:
- `collection.create_index([("name", pymongo.DESCENDING)])`
- In der Shell super-einfach:
- `db.verleih.createIndex({"name": -1 })`
- Aufgabe: erstelle einen Index in Python, und schau dann in Compass nach, ob es geklappt hat wie erwartet

Index auf subdocuments

- db.verleih.createIndex({ "Peter.firstName": 1 })

- Aufgabe: Teste dies an Hand eines eigenen Beispiels, und schaue dann in Compass, dass/wie der Index tatsächlich genutzt wird



Dies ist ein völlig schwachsinniges Beispiel …

Single Field vs Compound Index

- Single Field Index: nur ein einzelnes Feld (doh…;-)
- Compound Index:
  - bis zu 32 Felder in einem Index
  - Alle mit eigenem sorting.
  - Bsp: `{ userid: 1, score: -1 }`

Aufgaben

- Führe für den vorhin erstellten Single Field Index eine Suche aus:
  - Einmal aufsteigend, einmal absteigend sortiert
- Erstelle einen Compound Index:
  - Überlege zuerst, welche Art von Sortierung hier “typischerweise” in der Praxis sinnvoll wäre
  - Erstelle den Index
  - Überprüfe was passiert, wenn Du auf/absteigend sortiert suchst (natürlich mit einem Compound Field)
  - Geht das? (warum sollte es u.U. nicht gehen?)

Single Field vs Compound Index

- Obacht: Sortierte Suchen!
  - Beim Single Field Index ist es egal, wie er sortiert ist, das Ergebnis ist immer sortierbar (auf- oder absteigend)
  - (Angeblich….;-) Beim Compound Index hingegen kann man dann nur noch in der Gleichen Richtung wie im Index sortiert ausgeben.
  - Präziser: der **Index** kann nur in der Richtung genutzt werden, wie er definiert ist.
  - Wenn er nicht definiert ist, dann wird halt Chuck Norris Style von Hand gesucht!
  - DESIGN DECISION!
  - Bsp: { userid: 1, score: -1 } – hier geht ein `db.events.find().sort( { userid 1, score: 1 } )` NICHT! (d.h. nutzt den Index nicht!)

Indexing for Sorting

- Wenn MongoDB sortierte Ergebnisse ausgeben soll …
- … aber kein passender Index vorhanden ist, dann …
- … wird die Datenbank gelockt („Blocking Sort“)
- D.h.: DESIGN DECISION: man muss wissen, welche Art von Sortierungen in der Praxis genutzt werden!

Prefix

- Ein Prefix ist/sind alle „beginning subsets“.
- Bsp: `{ "item": 1, "location": 1, "stock": 1 }`
- Hat die Prefixes:
  - `{ item: 1 }` und
  - `{ item: 1, location: 1 }`
- MongoDB unterstützt ein Compound Index Queries für
  - alle prefixes und (hier: item, oder item UND location)
  - den kompletten Index (item UND location UND Stock)
- Was NICHT geht für diesen Index: queries nach location und oder stock
- DESIGN DECISION: welcher Compound Index deckt viele Queries ab?

„Sonder“-Indices: Multikey, Geospatial, Text und Hashed

- **Multikey**: Ist ein Index für Werte von Arrays!
  - MongoDB erstellt diese automatisch;-)
- **Geospatial**: 2D und 3D Daten
- **Text:** speichert einen Index auf ***root-words*** in Text-Fields. Wichtig hierbei: die Sprache. Brauchen „Collations“.
- **Hash(ed):** für Sharding. Können nur equality (kein Range)

Text Index

- https://docs.mongodb.com/manual/core/index-text/
  - Was macht ein Text-Index?
  - Wie viele Felder sind im Text-Index enthalten?
  - Wie kann man die einzelnen Felder unterschiedlich gewichten?
  - Wie verwendet man einen Text-Index in einem Compound Index? (was für Besonderheiten sind zu beachten?)
  - Wie viele Text-Indices kann eine Collection haben?
  - Diskutiere die folgende Aussage: „Ein Text-Index ist im Prinzip wie ein Multi-key Index“
  - Wie wirkt sich die Nutzung eines Text-Index auf die Performance aus?

Die Nutzung des Text-Index

- https://docs.mongodb.com/manual/text-search/
- https://docs.mongodb.com/manual/reference/operator/query/text

DHBW Loerrach
Baden-Wuerttemberg
Cooperative State University

Datenbanken 2: Apache Airflow

Automatische Workflows/Pipelines

Prof Dr Veit U.B. Schenk, schenkv@dhbw-loerrach.de

Automatisch Befehle/Scripts regelmässig ausführen lassen?

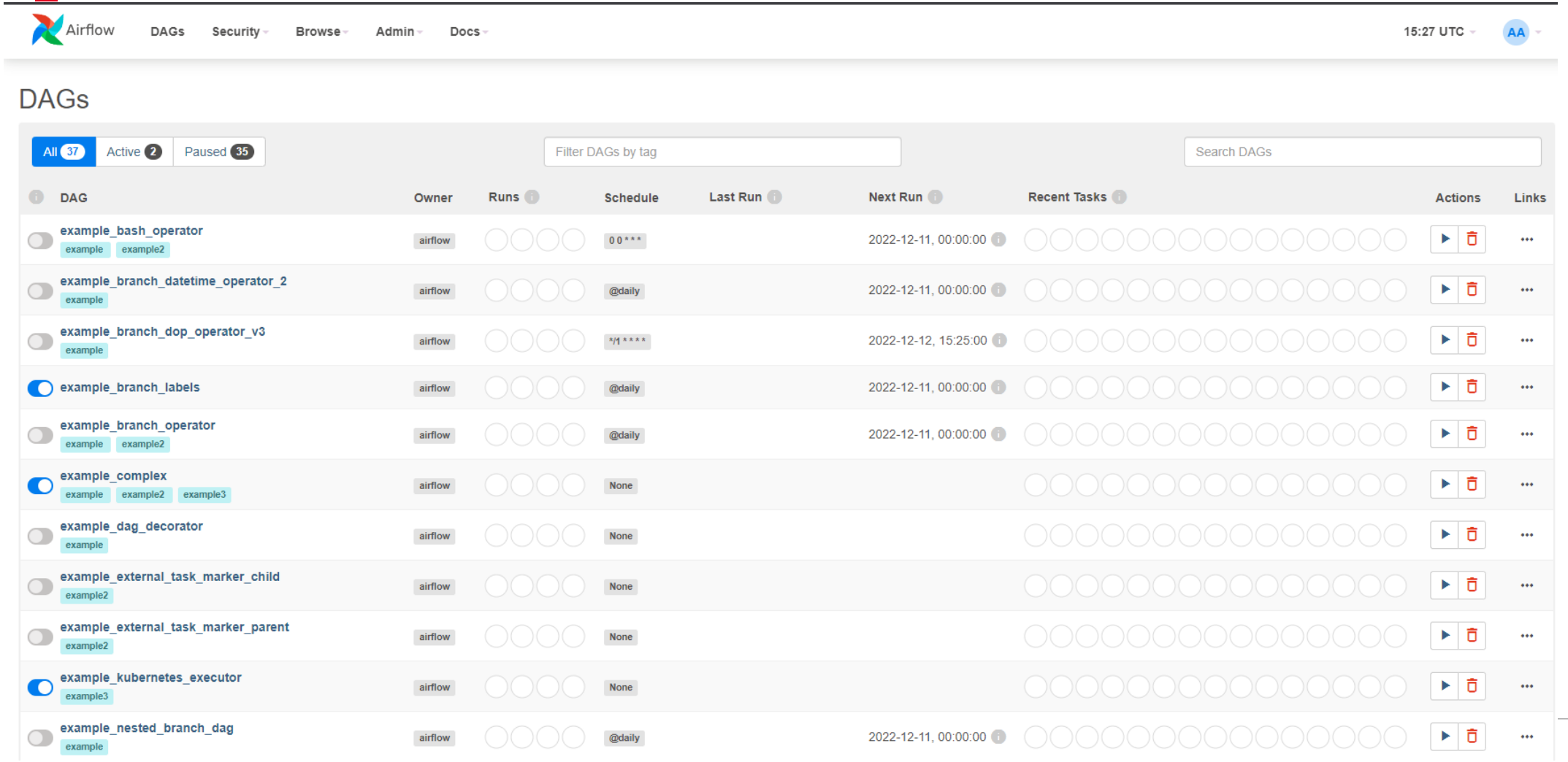
- Unix: Cron (command run on notice)-jobs
- https://www.ionos.de/digitalguide/hosting/hosting-technik/cronjob/
- https://en.wikipedia.org/wiki/Cron#CRON_expression

airflow.apache.org

- Batch-orientierter Pipeline Builder
- Airflow is NICHT das Processing tool, sondern der Koordinator
- Processing mit Python Code,…
- Kann aber viele andere Technologien einbinden (z.B. shell-scripts)
- UI: volle Übersicht über Workflows
- Installation: https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html

DHBW Loerrach
Baden-Wuerttemberg
Cooperative State University

Das Web-Interface

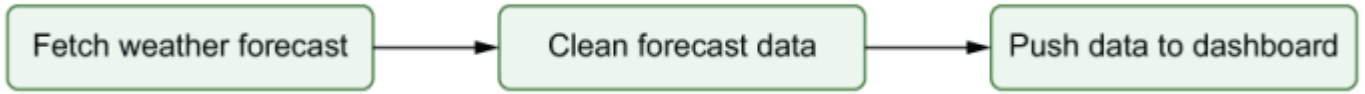


Literatur

- “Data Pipelines with Apache Airflow”, Julian de Ruiter, Bas Harenslak, Manning Verlag (suche nach „apache airflow“ in EDS)

Ein Wetter Dashboard

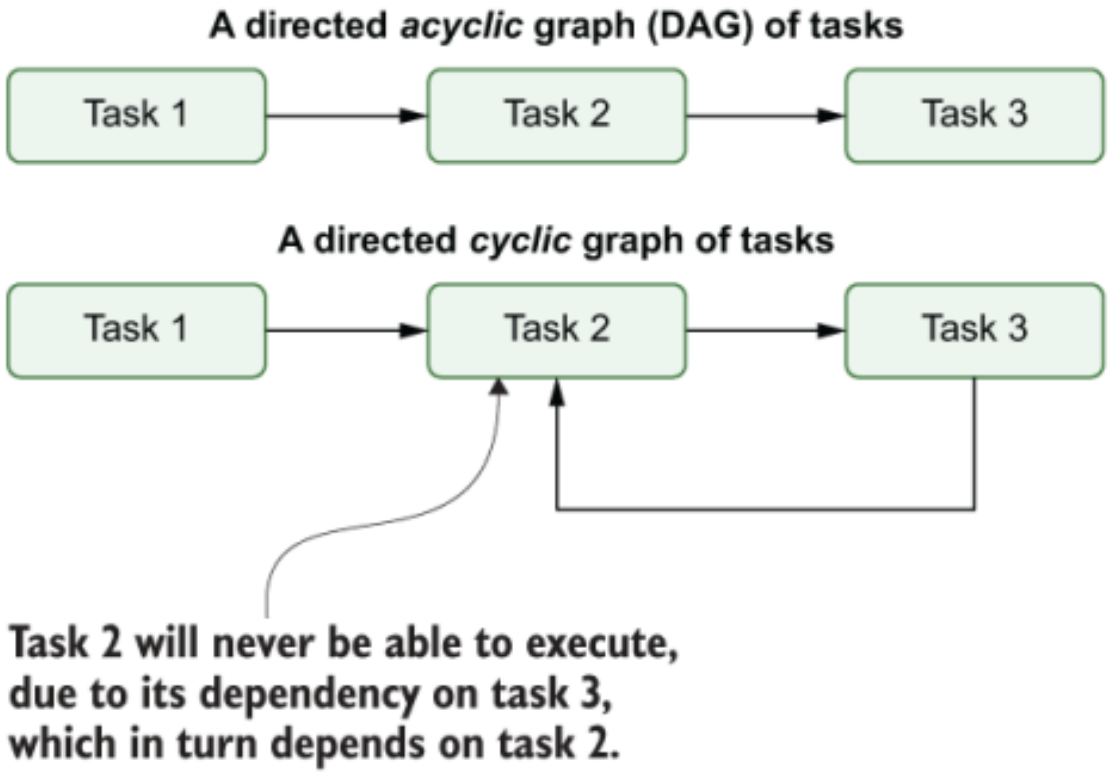
- Welche Komponenten brauchen wir um ein Wettervorhersage-Dashboard zu bauen?
  - Wettervorhersage-Daten aus einem API
  - Daten vorbereiten (u.U. extrahieren, säubern, filtern, konvertieren…)
  - Gesäuberte Daten an das Dashboard schicken
- Ist die Reihenfolge wichtig?
- Natürlich, und genau das ist der Sinn und Zweck der Pipelines!



Quelle: de Ruiter & Harenslak

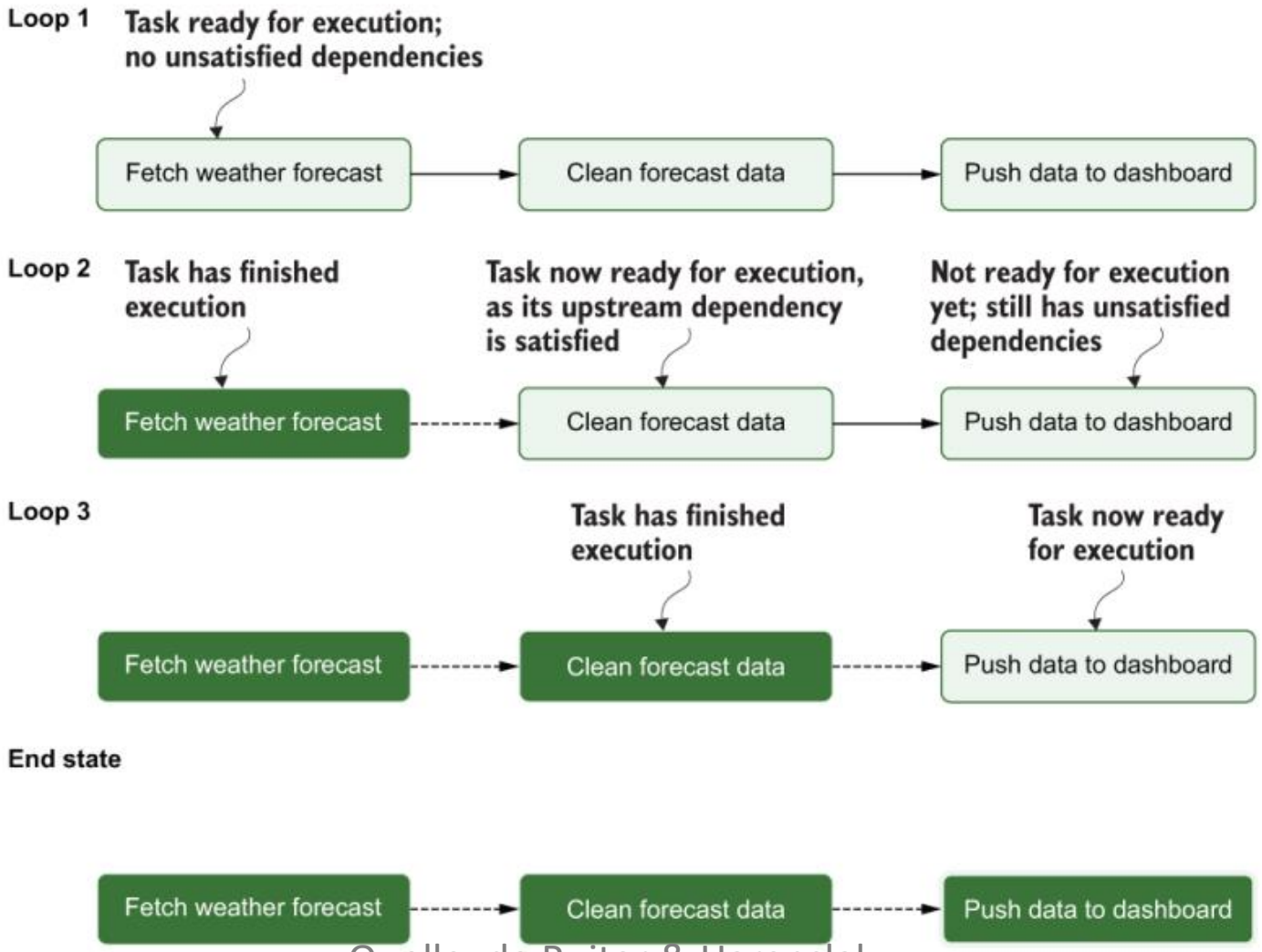
DAG

- Keine „loops“ möglich, …
- Weil wir sonst keine Reihenfolge festlegen können (was muss zuerst ausgeführt werden)
- Was sagt uns das über die Möglichkeit in Airflow Dinge Mehrfach auszuführen?
- A: keine Iterationen möglich!



Quelle: de Ruiter & Harenslak

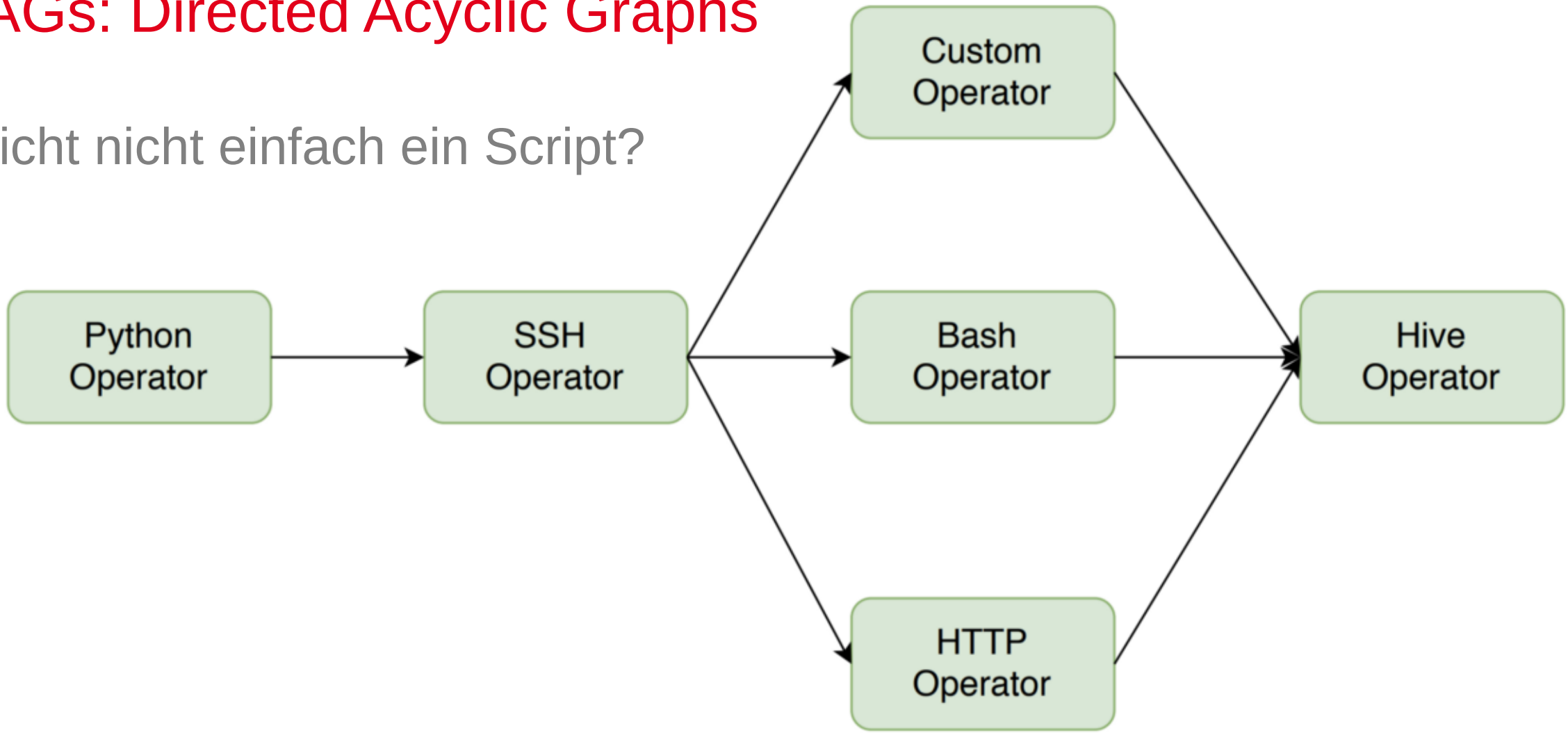
DAG Execution in der Praxis



Quelle: de Ruiter & Harenslak

DAGs: Directed Acyclic Graphs

- Reicht nicht einfach ein Script?



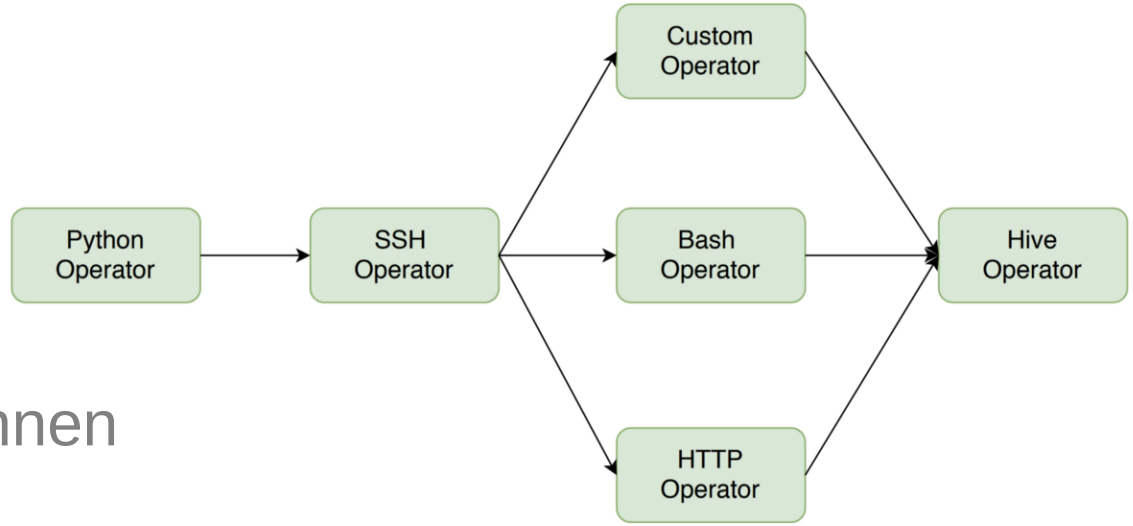
Quelle: https://miro.medium.com/max/1200/1*CtoqTsvlAuXU_4V-W5VMyQ.png

Aufgabe: Partnerarbeit

- Erstellt ein DAG für eine Badeanzugherstellerin: Sie möchte ein ML-Model erstellen basierend auf Wetterdaten und Verkaufsdaten, so dass sie ein Modell hat, mit dem sie die Verkäufe von Badeanzügen mit Hilfe der aktuellen 14-Tage Wettervorhersage erstellen kann.

DAGs: Directed Acyclic Graphs

- Reicht nicht einfach ein Script?
- Nein, denn:
  - Von einander unabhängige Aktionen können ausgeführt werden, und dennoch später „zusammen“ genutzt werden
  - Statt eines riesen Scripts, haben wir schön kleine, separat zu testende Units.

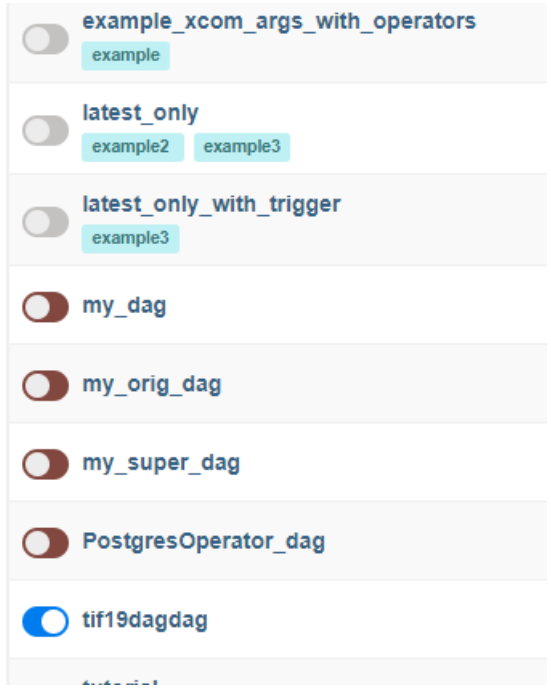


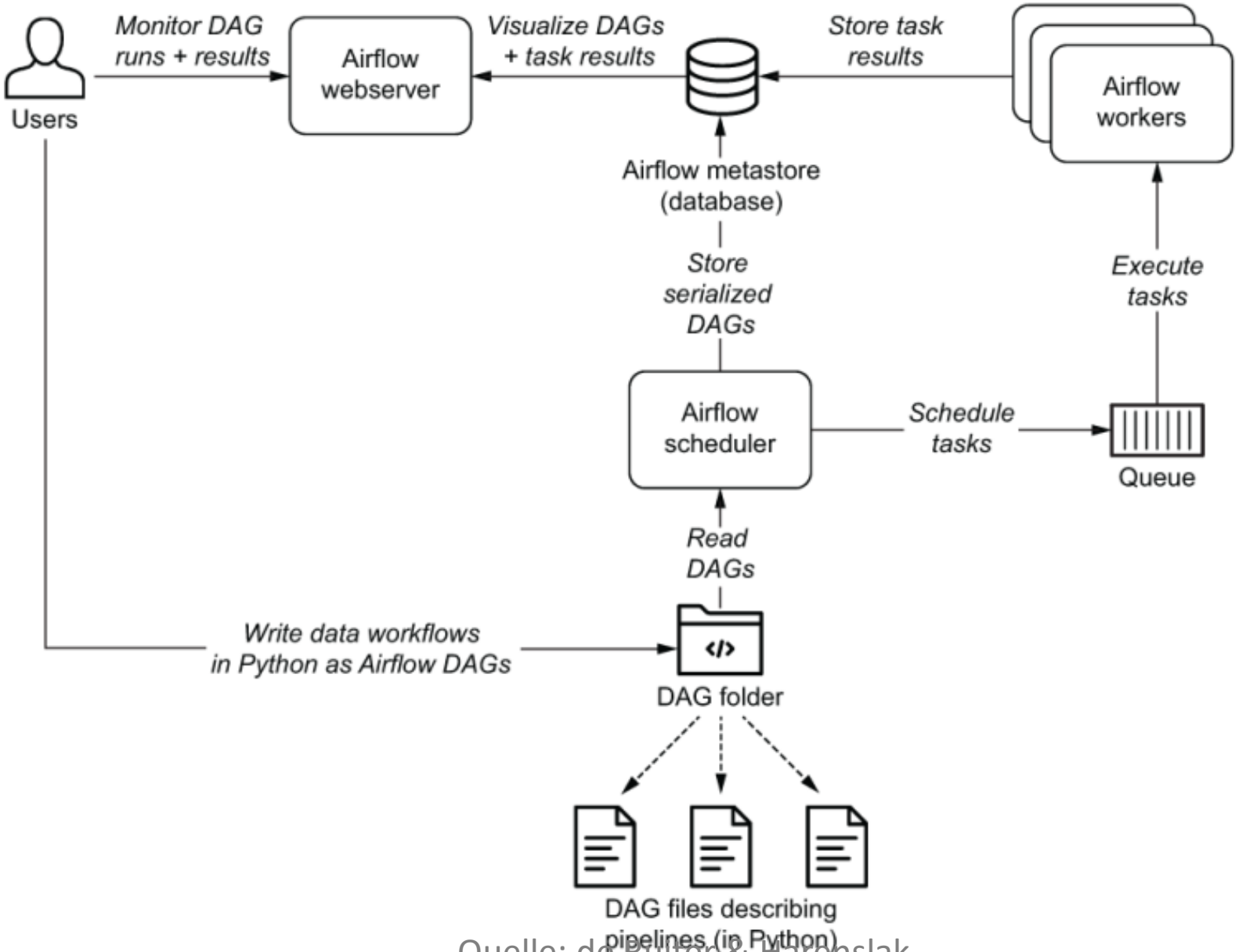
Quelle: https://miro.medium.com/max/1200/1*CtoqTsvlAuXU_4V-W5VMyQ.png

Warum nun Airflow und nicht einer von X anderen?

- Oozie: Hadoop basiert, und spricht Java und XML (statisch, booo!)
- Luigi (Spotify): Python, aber kein Scheduling (brauche immer noch cron)
- Apache Airflow Super-leicht (;-) erweiterbar:
  - https://github.com/airflow-plugins/
  - https://airflow.apache.org/docs/apache-airflow/stable/plugins.html

Apache Airflow: Workflow Manager





Quelle: de Ruiter & Harenslak

Status

- https://towardsdatascience.com/airflow-state-101-2be3846d4634
- Idealerweise:
  - No Status
  - Scheduled
  - Queued
  - Running
  - Success

Incremental Loading and Backfilling

- Läuft für bestimmten ZeitRAUM.
- Kann ganz gezielt Daten für diesen Zeitraum “berechnen”
- Bsp: https://docs.firebolt.io/loading-data/incrementally-loading-data.html
- Oder mit Backfilling für jede beliebige Anzahl von Zeiträumen “nachholen”

Anwendungen, für die Airflow nicht taugt

- ???
- Streaming. Airflow ist für Batch-Processing
- Pipelines, die sich oft/dynamisch ändern (Reporting messed up)
- No Python, No Luck
- Sehr komplexe DAGS (es sei denn, man ist echter Software-Engineer…)

DAG Grundstruktur

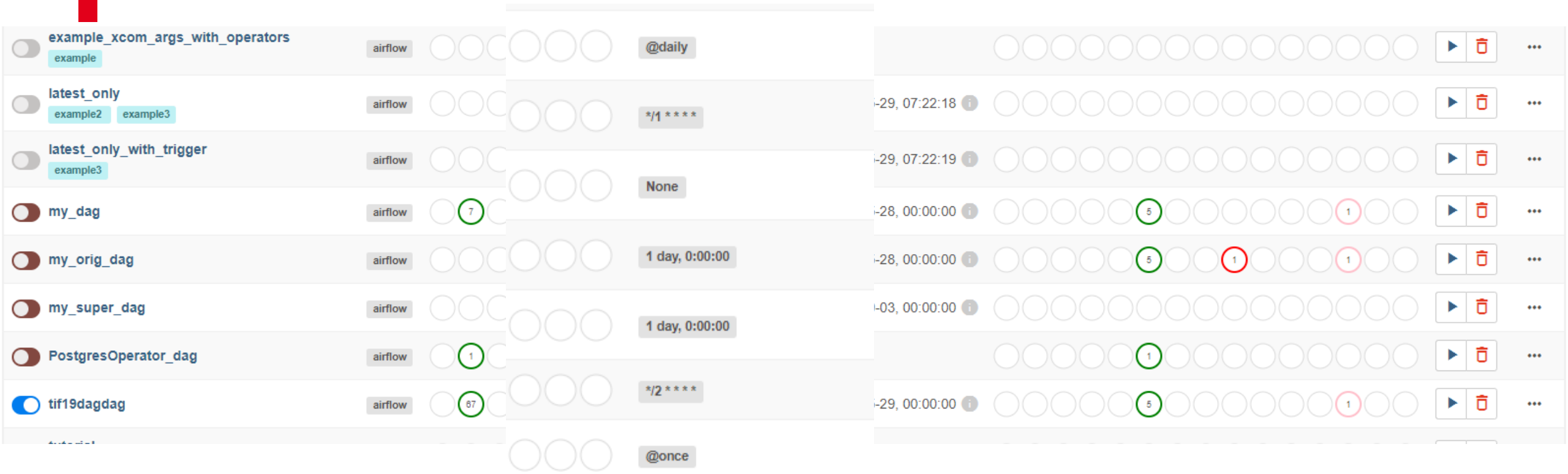
- https://towardsdatascience.com/de-mystifying-an-airflow-dag-script-ff4f267edf34
- Library imports block
- DAG argument block
- DAG definition block
- Task definition block
- Task pipeline block

Task vs Operator

- Im Operator beschreiben WIR was gemacht werden soll
- Im Task wird sichergestellt, das der Operator korrekt ausgeführt wird (Argumente übergeben, Ergebnisse abholen, Reihenfolge, usw)
- Ein Task ist wie ein Wrapper um einen Operator
- https://airflow.apache.org/docs/apache-airflow/stable/_api/airflow/operators/index.html
- In der Praxis werden die Begriffe aber oft vertauscht

```
training_model_A = PythonOperator(
    task_id="training_model_A",
    python_callable=_training_model
)
```

Das Web-Interface



DAG: my_dag

Tree | Graph | Calendar | Task Duration | Task Tries | L

Details | Code

Tree | Graph | Calendar | Task Duration

2022-03-10T09:33:36Z | Runs | 25 | Updat

2022-03-10T09:33:37Z | Runs | 25 | Run | manual__2022-03-10T

Layout | Left > Right | Update

Find Task...

BashOperator | BranchPythonOperator | PythonOperator

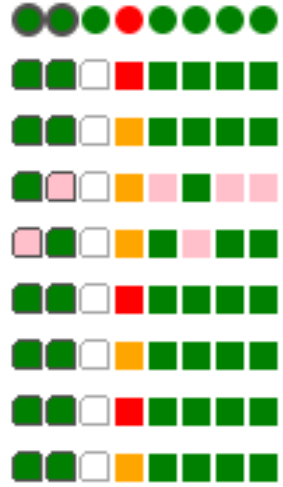
BashOperator | BranchPythonOperator | PythonOperator

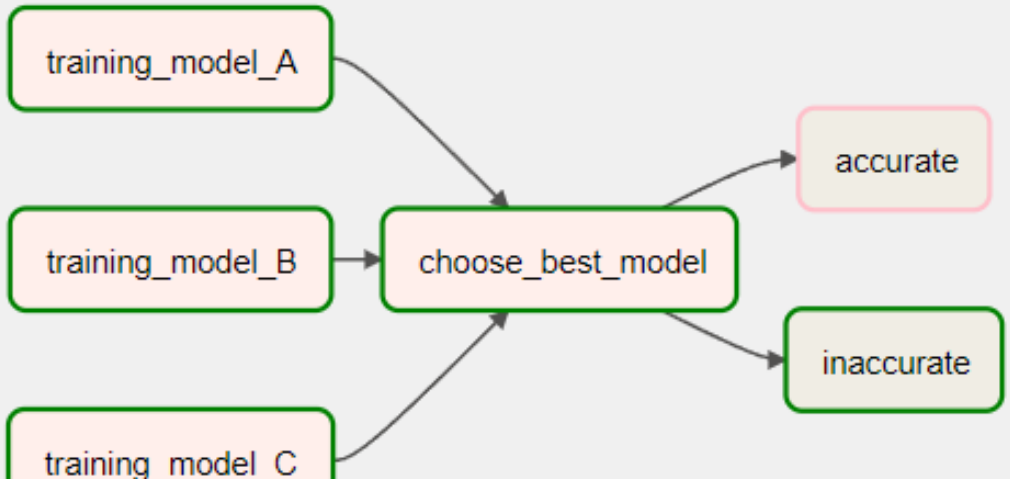
queued | running | success | failed | up_for_retry | up_for_resc

scheduled | deferred | no_status

Mar 08, 01:00 — Mar 10, 10:3

- [DAG]
  - training_model_A
    - choose_best_model
      - accurate
      - inaccurate
  - training_model_B
    - choose_best_model
  - training_model_C
    - choose_best_model





Task Instance: inaccurate at 2022-03-10, 09:33:36

Task Instance Details | <> Rendered Template | Log | XCom

Log by attempts

1

```
*** Reading local file: /opt/airflow/logs/my_dag/inaccurate/2022-03-10T09:33:36.414435+00:00/1.log
[2022-03-10, 09:33:40 UTC] {taskinstance.py:1037} INFO - Dependencies all met for <TaskInstance: my_dag.inaccurate manual__2022-03-10T09:33:36.414435+00:00 [queued]>
[2022-03-10, 09:33:40 UTC] {taskinstance.py:1037} INFO - Dependencies all met for <TaskInstance: my_dag.inaccurate manual__2022-03-10T09:33:36.414435+00:00 [queued]>
[2022-03-10, 09:33:40 UTC] {taskinstance.py:1243} INFO -
--------------------------------------------------------------------------------
[2022-03-10, 09:33:40 UTC] {taskinstance.py:1244} INFO - Starting attempt 1 of 1
[2022-03-10, 09:33:40 UTC] {taskinstance.py:1245} INFO -
--------------------------------------------------------------------------------
[2022-03-10, 09:33:40 UTC] {taskinstance.py:1264} INFO - Executing <Task(BashOperator): inaccurate> on 2022-03-10 09:33:36.414435+00:00
[2022-03-10, 09:33:40 UTC] {standard_task_runner.py:52} INFO - Started process 925 to run task
[2022-03-10, 09:33:40 UTC] {standard_task_runner.py:76} INFO - Running: ['***', 'tasks', 'run', 'my_dag', 'inaccurate', 'manual__2022-03-10T09:33:36.414435+00:00', '--job-id', '4
[2022-03-10, 09:33:40 UTC] {standard_task_runner.py:77} INFO - Job 41: Subtask inaccurate
[2022-03-10, 09:33:40 UTC] {logging_mixin.py:109} INFO - Running <TaskInstance: my_dag.inaccurate manual__2022-03-10T09:33:36.414435+00:00 [running]> on host f83458a54475
[2022-03-10, 09:33:40 UTC] {taskinstance.py:1431} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_OWNER=***
AIRFLOW_CTX_DAG_ID=my_dag
AIRFLOW_CTX_TASK_ID=inaccurate
AIRFLOW_CTX_EXECUTION_DATE=2022-03-10T09:33:36.414435+00:00
AIRFLOW_CTX_DAG_RUN_ID=manual__2022-03-10T09:33:36.414435+00:00
[2022-03-10, 09:33:40 UTC] {subprocess.py:62} INFO - Tmp dir root location:
 /tmp
[2022-03-10, 09:33:40 UTC] {subprocess.py:74} INFO - Running command: ['bash', '-c', "echo 'inaccurate &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&'"]
[2022-03-10, 09:33:40 UTC] {subprocess.py:85} INFO - Output:
[2022-03-10, 09:33:40 UTC] {subprocess.py:89} INFO - inaccurate &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
[2022-03-10, 09:33:40 UTC] {subprocess.py:93} INFO - Command exited with return code 0
[2022-03-10, 09:33:40 UTC] {taskinstance.py:1282} INFO - Marking task as SUCCESS. dag_id=my_dag, task_id=inaccurate, execution_date=20220310T093336, start_date=20220310T093340, e
[2022-03-10, 09:33:40 UTC] {local_task_job.py:154} INFO - Task exited with return code 0
[2022-03-10, 09:33:40 UTC] {local_task_job.py:264} INFO - 0 downstream tasks scheduled from follow-on schedule check
```

```python
from socket import INADDR_ALLHOSTS_GROUP
from airflow import DAG
from airflow.operators.python import PythonOperator, BranchPythonOperator
from airflow.operators.bash import BashOperator
from random import randint
from datetime import datetime


def _training_model():
    print("Ich bin der Python code der _training_model heisst")
    return randint(1,10)


def _choose_best_model(ti):
    accuracies = ti.xcom_pull(task_ids=[
        'training_model_A',
        'training_model_B',
        'training_model_C'
    ])
    print(f"_choose_best_accuracies von {accuracies}")
    best_accuracy = max(accuracies)
    if (best_accuracy > 8):
```

Aufgaben

- Arbeite das offizielle Apache Airflow Tutorial durch (jetzt sind wir nämlich in der Lage es zu verstehen und auszuführen)
  - https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html

    1. erstelle auf der command-line eine Liste der Tasks des TIF20-Dags in Baumform
    2. teste einen der "training_model" tasks mehrfach um zu sehen, ob wirklich Zufallszahlen herauskommen (auch auf der command-line)
    3. Füge Dokumentation zu Deinem DAG hinzu und überprüfe, ob diese auch richtig im System erscheint
- Erstelle ein DAG, das die Badeanzugsituation darstellt. Die Daten dürfen/sollen per Zufallsgenerator generiert werden (erstmal einfach machen…)
- https://xnuinside.medium.com/short-guide-how-to-use-postgresoperator-in-apache-airflow-ca78d35fb435
- https://www.mongodb.com/developer/products/mongodb/mongodb-apache-airflow/

DHBW Loerrach
Baden-Wuerttemberg
Cooperative State University

Datenbanken 2: Kafka: Von State zu Events

Prof Dr Veit U.B. Schenk, schenkv@dhbw-loerrach.de

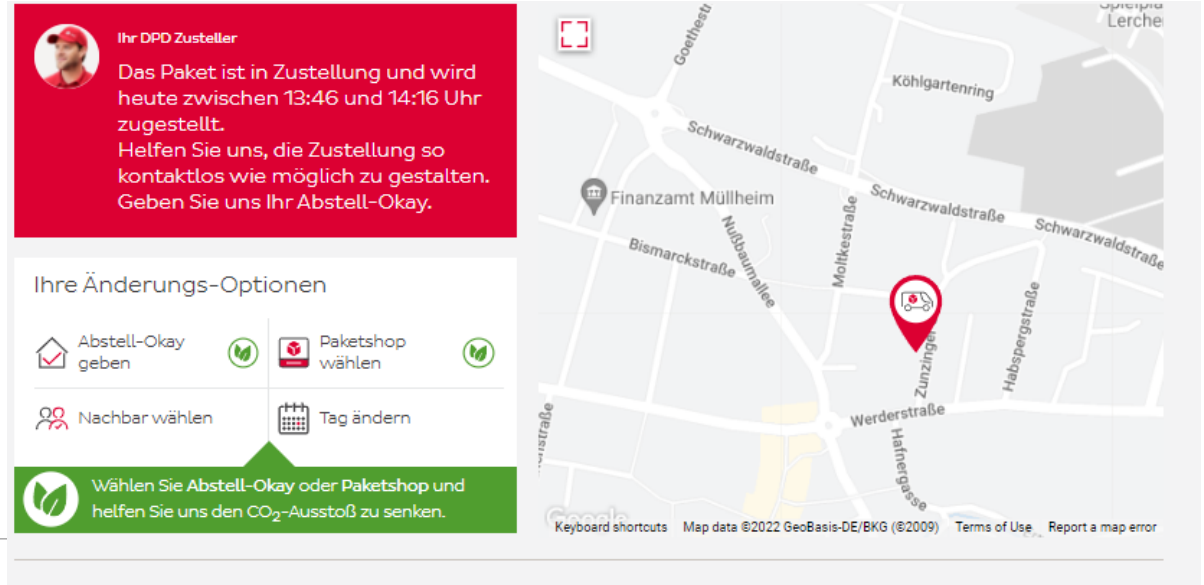
Der große Umbruch

- Von statischem „**State**“ …
- … zu (endlosem Stream von) „**Events**“.
- Zeitung: statisch. Newsfeed: real-time
- Anruf: statischer „Update“ von „wie geht es“ zu dauernden live-updates auf social media
- Der wirkliche Unterschied: „Events“ sind ALLE Dinge, die passieren, State ist die „Zusammenfassung“ der Events.
- Der Grund: fast immer ist „real time“ data *besser*(???) als „slow“ data
  - Fraud Detection, Real-time Inventory, Real time recommendations, sensors, DHL sagt mir wann mein Paket (nicht) kommt…

ESP: Event Streaming Platform

- Events:
  - Irgendetwas „wichtiges“ passiert
  - „an entity‘s observable state over time“
- Beispiele?
  - GPS Koordinaten während der Fahrt
  - Temperatur im Rechnerraum
  - Blutdruck
  - RAM Nutzung im Rechner
  - ….

- Event key: "Alice"
- Event value: "Made a payment of $200 to Bob"
- Event timestamp: "Jun. 25, 2020 at 2:06 p.m."



Aufgabe: Was sind/wer braucht EVENTS?

- Was für APIs, Services, User hätten denn gerne Zugriff auf ein real-time verteiltes Log (entweder zum Verteilen von Infos, oder zum Abholen)???
- Credit card fraud detection
- Autos
- Real time e-Commerce
- IoT, Edge Devices
- Medical Devices
- Mobile Devices
- SaaS Apps
- Datencenter
- Microservices
- Datenbanken, Data Lakes
- Messenger, besonders wenn offline geht
- Dashboards: Taskmanager/Ressourcen anschauen
- Warenbestände
- ML Modelle updaten
- Energie/Ressourcenverteilung
- Multiplayer Games

TIF Ideen

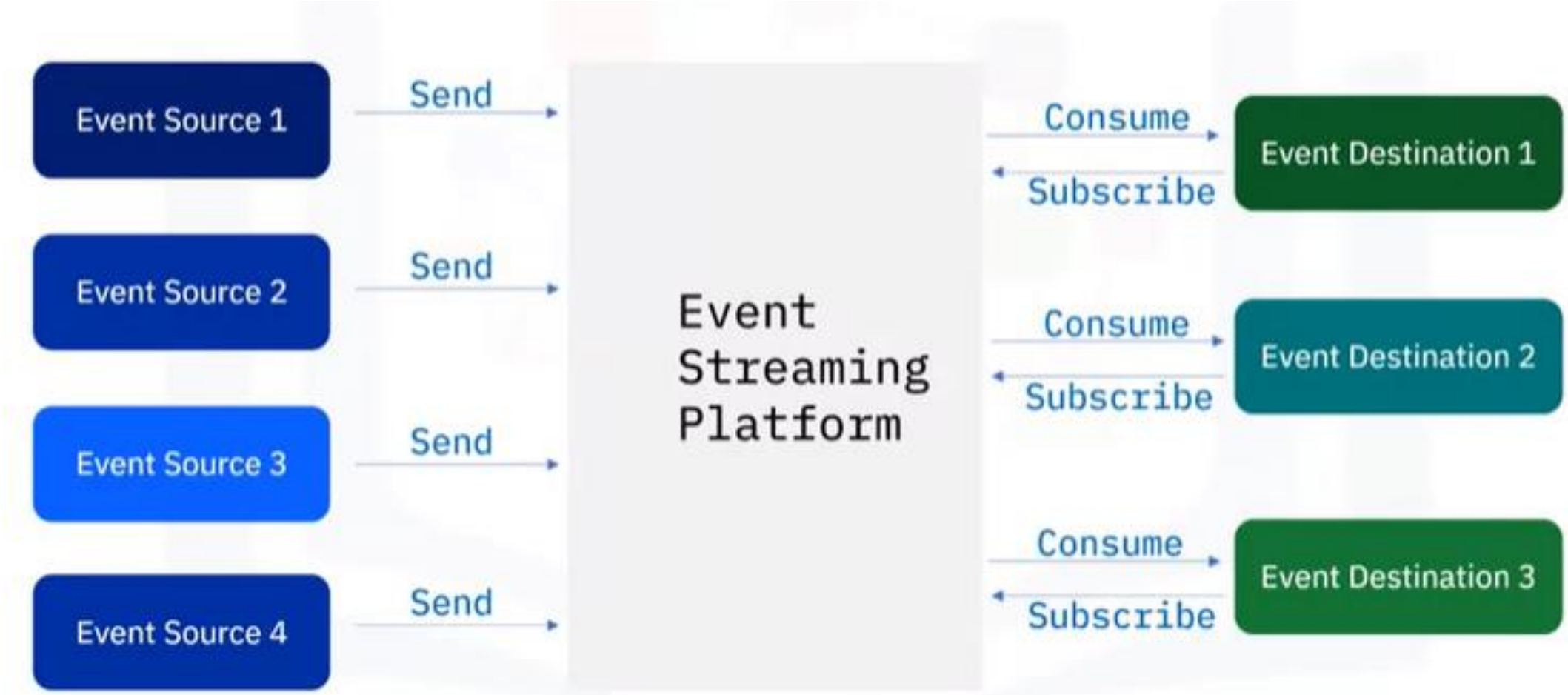
Typisches Format

- Primitive Daten (int, string, …)
- KV-Paar:
  - Key: „car_id_xyz“
  - Value: (44.87, -78.43)
  - (optional: mit Time-Stamp: 17.03.2022 14h17:22)

Event Streaming

- 1 Sender, 1 Empfänger
  - Sender: all die schönen Datenquellen, die beim E von ETL anstehen
  - Empfänger: all die schönen „L“-Destinations UND aber auch:
    - Applications
    - Als Sender für weitere Empfänger
  - Bisher eigentlich eine ETL-Pipeline
- Aber was passiert wenn
  - wir *n* Sender und *m* Empfänger haben? und
  - Empfänger auch Sender sein sollen? Und
  - Verschiedene Protokolle genutzt werden (FTP, HTTP, JDBC, SCP, …)
- Publish-Subscribe to the rescue

Quelle: IBM Developer Skills Network



Ziel

- Alle „Events“ für alle (Interessierten) zur Verfügung stellen
- Real-Time (nicht Batch wie in Airflow) …
- … aber auch „verlangsamen“ (handle back-pressure) wenn nötig
- Und wenn nötig, alle Events (für potentiell immer) aufbewahren, so dass man neue „Zusammenfassungen“ machen kann.
- Apache Kafka: verteilter real-time Log, der (bei Bedarf) die Events speichert
- Wie soweit immer der Fall: fault-tolerant, verteilt, APIs für andere Sprachen, Devices usw
- https://kafka.apache.org/powered-by (35% der Fortune 500 …)

Apache Kafka

APACHE KAFKA

*More than **80% of all Fortune 100 companies** trust, and use Kafka.*

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

Common use cases

ESP: Event Streaming Platform

- https://kafka.apache.org/documentation/

1. To **publish** (write) and **subscribe to** (read) streams of events, including continuous import/export of your data from other systems.
2. To **store** streams of events durably and reliably for as long as you want.
3. To **process** streams of events as they occur or retrospectively.

Kafka-Client-Server-Architektur

- Verteilte Server, auch „Broker“ genannt.
- Alle Broker werden (wurden;-) von Apache Zookeeper verwaltet
- Clients:
  - CLI
  - High-Level APIs (Java, Scala, Python)…
- Wunderbar Skalierbar
- Replication: Sicherheit
- Persistency: kann jederzeit auf alle Daten zugreifen (ist also NICHT Streaming im klassischen Sinn)

ESP Komponenten

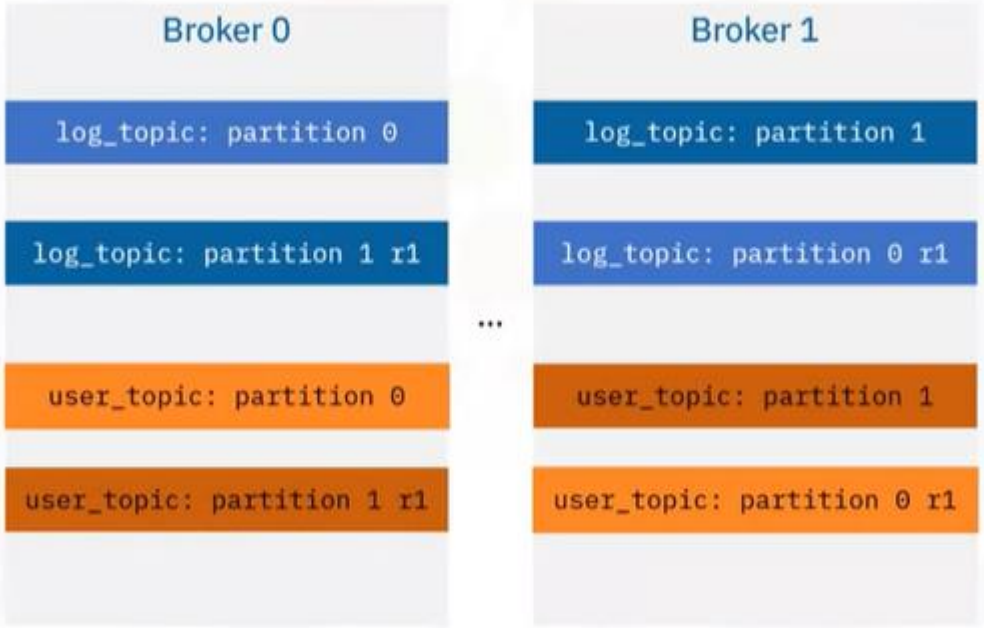
- Broker: einzelner „Rechner“ mit eigenem Storage (wo Daten „beliebig“ lang aufgehoben werden können). Viele Broker zusammen: Kafka Cluster
- Event Broker
  - Ingester
  - Processor (De/Serialize, De/Compress, De/Crypt, …)
  - Consumption (Distribution, genau so bescheuerter Begriff wie das L in ETL)
- Event Storage
- Analytic and Query Engine

DHBW Loerrach
Baden-Wuerttemberg
Cooperative State University

Broker und Topics

Broker and topic

Partition and replications



The World Produces Data

“Producer”

Connected Cars

Financial Transactions

Hospitals EMR

Logistics Shipping

Broker 1

Broker 2

....

Broker N

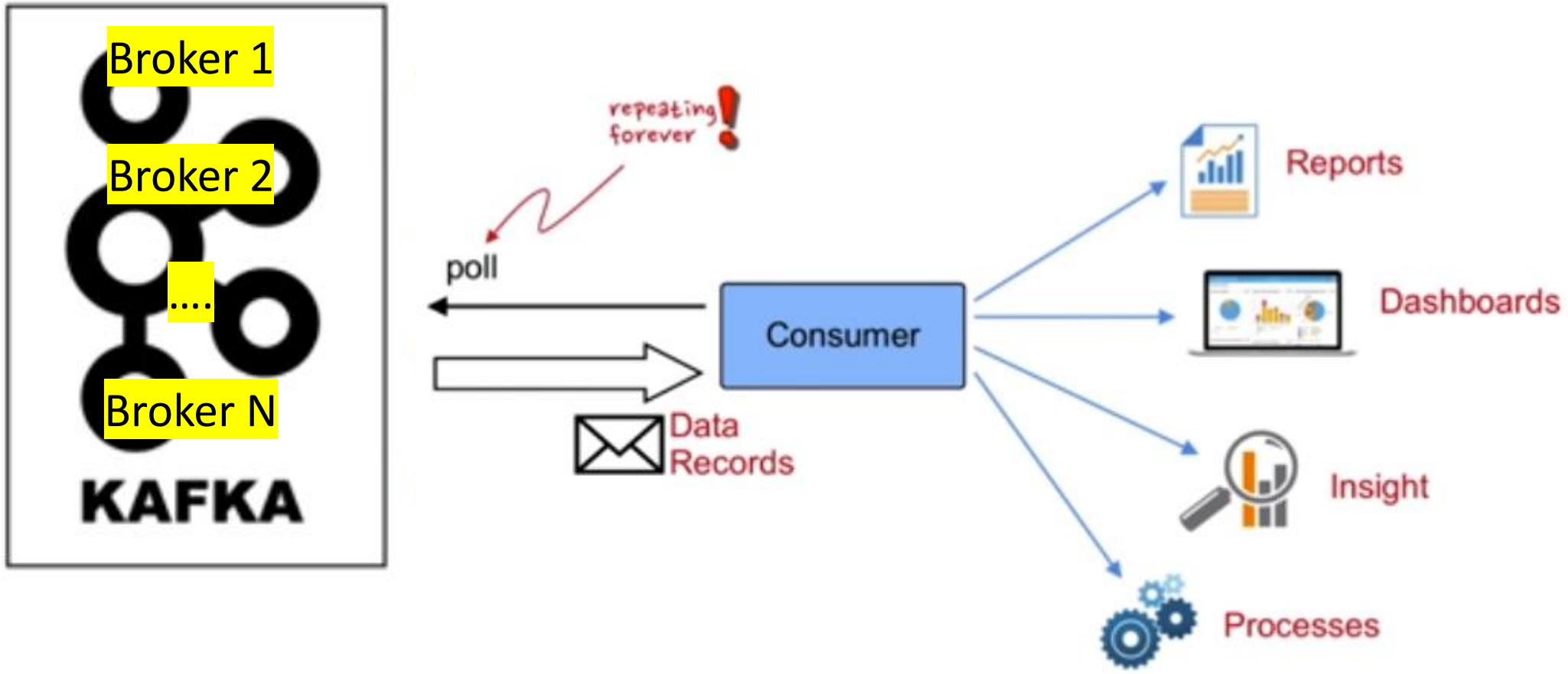
KAFKA

Manufacturing Assembly Lines

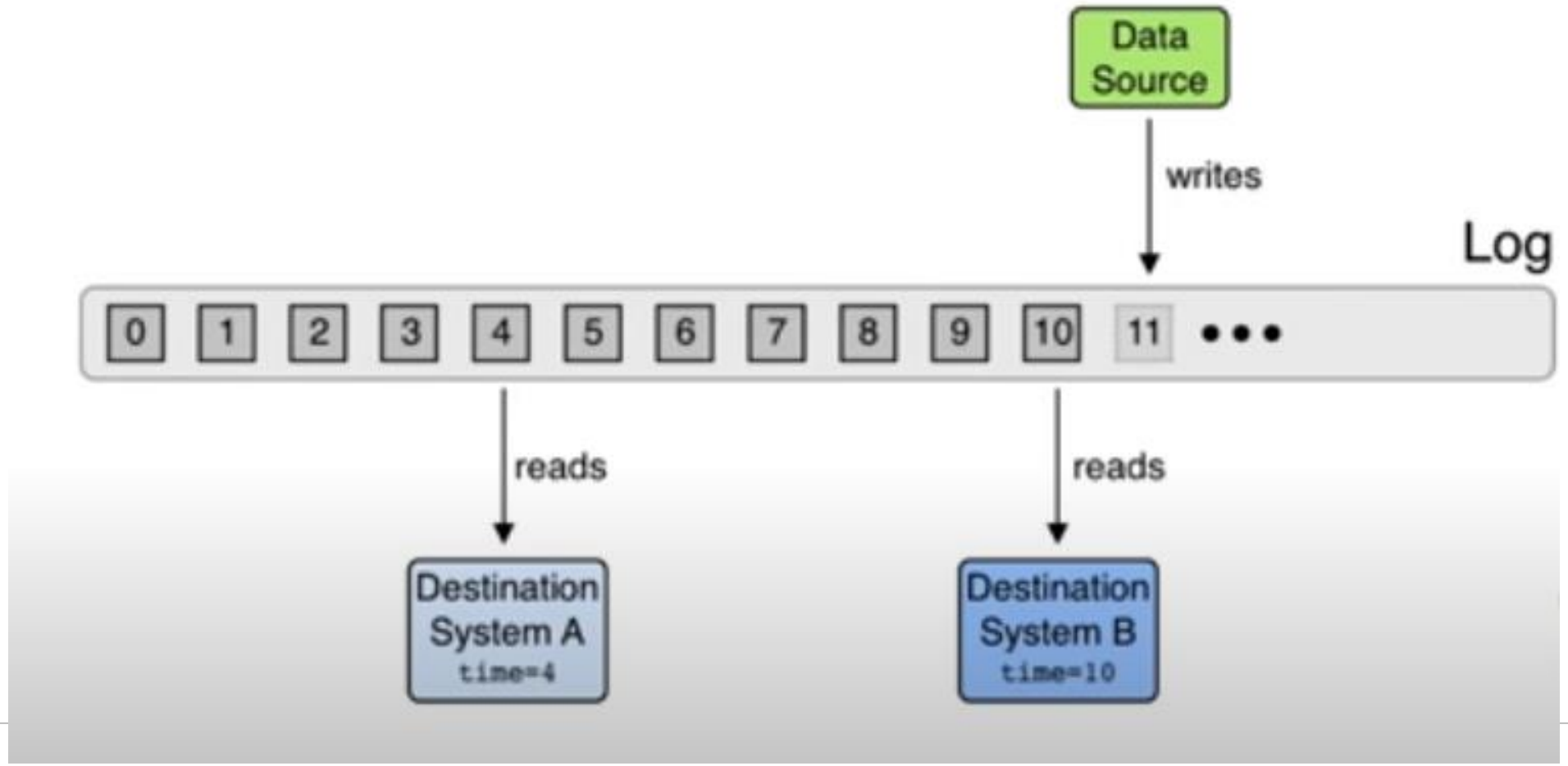
Mobility

eCommerce

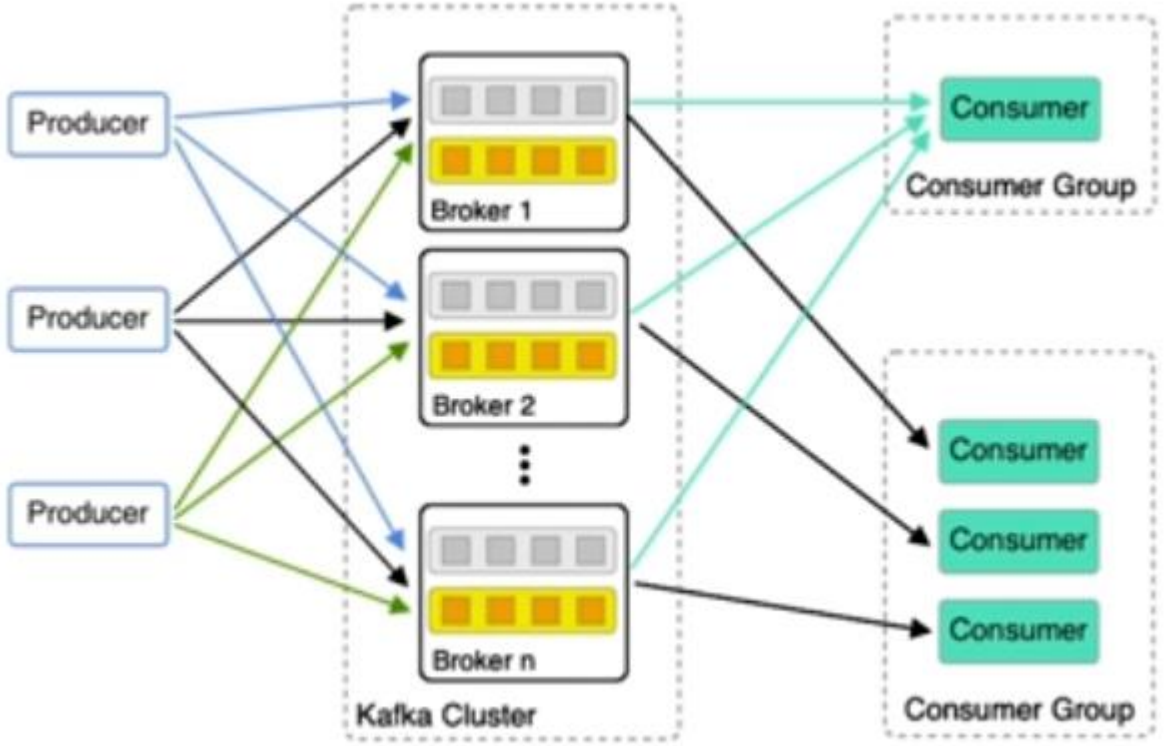
Insurances


Broker 1
Broker 2
....
Broker N
KAFKA
repeating forever
poll
Consumer
Data Records
Reports
Dashboards
Insight
Processes

Die Logs (oder Streams)



Das Kafka-Modell zusammengefasst



Aufgabe

- https://www.kai-waehner.de/blog/2021/07/19/kafka-automotive-industry-use-cases-examples-bmw-porsche-tesla-audi-connected-cars-industrial-iot-manufacturing-customer-360-mobility-services/
- https://www.kai-waehner.de/blog/2021/02/19/apache-kafka-aviation-airline-aerospaceindustry-airport-gds-loyalty-customer/
- https://www.kai-waehner.de/blog/2019/11/28/apache-kafka-industrial-iot-iiot-build-an-open-scalable-reliable-digital-twin/

Time to get hands dirty: Installation

- https://hub.docker.com/r/bitnami/kafka/
- https://www.baeldung.com/ops/kafka-docker-setup
- Wurstmeister: https://ertan-toker.de/apache-kafka-mit-docker-compose/
- Wir benutzen Heute die Confluent Installation: https://developer.confluent.io/quickstart/kafka-docker/

Hands-Dirty: wir spielen mit Topics

- https://towardsdev.com/apache-kafka-cli-commands-cheatsheet-584800de1477
- Aufgabe 1: zeige alle bestehenden Topics an
- **`kafka-topics \`**
  `--bootstrap-server localhost:9092 \`
  `--list`
- Aufgabe 2: Lege ein Topic mit dem Namen „tifdrinks“ an
- **`kafka-topics.sh --create --topic tifdrinks --bootstrap-server localhost:9092`**
- Auch gerne mit **`--partitions 1 --replication-factor 1`**

List and Describe

- Erstelle eine Liste aller Topics (wdh Aufgabe 1)
- Beschreibe das tifdrinks topic (nicht Du, sondern Kafka…)
-
```
kafka-topics \
--bootstrap-server localhost:9092 \
--topic tifdrinks \
--describe
```

Producer und Consumer

- Erstelle einen Producer für das neu erstellte Topic
- **`kafka-console-producer.sh --topic tifdrinks --bootstrap-server localhost:9092`**
- Erstelle ein paar Events mit dem Producer
- Öffne ein neues Terminal und erstelle einen Consumer
- **`kafka-console-consumer.sh --topic tifdrinks --bootstrap-server localhost:9092`**
- Ja und wo sind sie denn unsere schönen Daten?
- **`--from-beginning`**

KV Paare

- Angenommen wir haben ATM Daten:
  `{"atmid": 1, "transid": 100}`
- Erstelle ein Topic `banktransfer` und passende Producer mit 2 Partitionen
- füge im Producer die folgenden Transaktionen ein:
- **`{"atmid": 1, "transid": 100}`**
- **`{"atmid": 1, "transid": 101}`**
- **`{"atmid": 2, "transid": 200}`**
- **`{"atmid": 1, "transid": 102}`**
- **`{"atmid": 2, "transid": 201}`**
- Erstelle jetzt einen passenden Consumer, und zeige die 5 Transaktionen an. Fällt Dir etwas auf?
- Reihenfolge ist wahrscheinlich/unter Umständen anders!
- Weil sie im Zufallsprinzip auf verschiedene Partitions verteilt wurden

(nach Zeit) Geordnete KV Paare

- **`kafka-console-producer.sh --bootstrap-server localhost:9092 --topic banktransfer --property parse.key=true --property key.separator=:`**
- Füge jetzt diese Events ein:
- **`1:{"atmid": 1, "transid": 102}`**
- **`1:{"atmid": 1, "transid": 103}`**
- **`2:{"atmid": 2, "transid": 202}`**
- **`2:{"atmid": 2, "transid": 203}`**
- **`1:{"atmid": 1, "transid": 104}`**
- Füge einen neuen Consumer hinzu, mit den Properties „**`print.key=true`**“ und „**`key.separator=:`**“

Das Python Tutorial

- https://developer.confluent.io/get-started/python/
- (geht aber auch in vielen anderen Sprachen)