Hochschule Karlsruhe
University of
Applied Sciences

Hochschule Karlsruhe – Data Engineering
WS 2021/2022
DSCB330 – Vorlesung 5 – ETL – Airflow

Foto: Quirin Leppert

Themenübersicht

Data Integration

- Data Formats (csv, XML, json)
- **Extract, Transform, Load**
- Object Relation Mapper (ORM)
- Staging

Data Processing

- Relationale Datenbanken
- nicht-relationale Datenbanken
- Resource Description Framework (RDF)
- Ontologien
- Data Warehouse

Data Modelling

- Serialisierung
- OPC UA
- MQTT
- Pub/Sub
- **Data pipelines**
  - **Apache Airflow**
  - gRPC

Web-Service Architektur

- Front-End
- Backend for Frontend (BFF)
- Micro Services
- Docker Container

Security

- Security ist wichtig in allen Phasen der Softwareentwicklung und Datenbereitstellung.

Übung

- Erstellung eines Daten-Modells einer prozesstechnischen Anlage
- Statische Daten
- Dynamische Daten
- Auswertung der Daten

Aufgaben eines Data Engineers

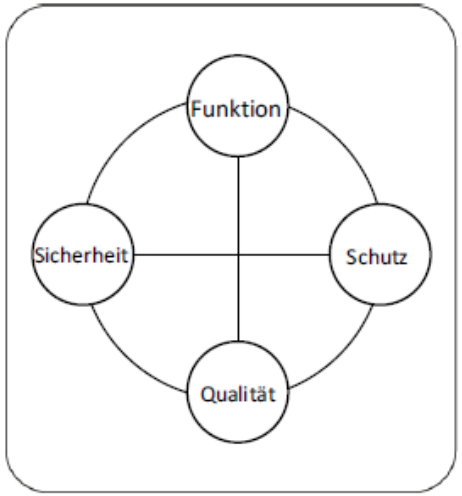
- Instandhaltung der Datenbank-Infrastruktur
- Extraktion, Zusammenführen und Bereitstellung von Daten in einer Pipeline (ETL)

Big Data

- Volume of data – Scale of Data
  Die Datenmenge hat in den letzten Jahren rapide zugenommen
- Variety – Different Forms of Data
  Daten aus verschiedenen Datenbanken in unterschiedlichen Formaten müssen vereinheitlicht werden
- Velocity – Analysis of Streaming Data
  Auswertung von Daten nahezu in Echtzeit
- Veracity – Uncertainty of Data
  Unsicherheit der in den Datenbanken abgespeicherten Informationen

Smart Data

- intelligente Daten
- haben Informationen über sich selbst
- kennen ihre Bedeutung und Deutung (Semantik)



Merkmale von Smart Data

Quellen: Data Engineering with python, Paul Crickard, Packt, Oktober 2020; Data Engineering 4.0, Kompositionale Informationsmodelle für industrielle Anwendungen, Herbert Weber, Springer, 2021

ETL – Extract, Transform, Load Transformationsprozess

Alle Aktivitäten zur Umwandlung der operativen Daten in fachlich interpretierbare, entscheidungsorientierte Daten

Teilprozesse

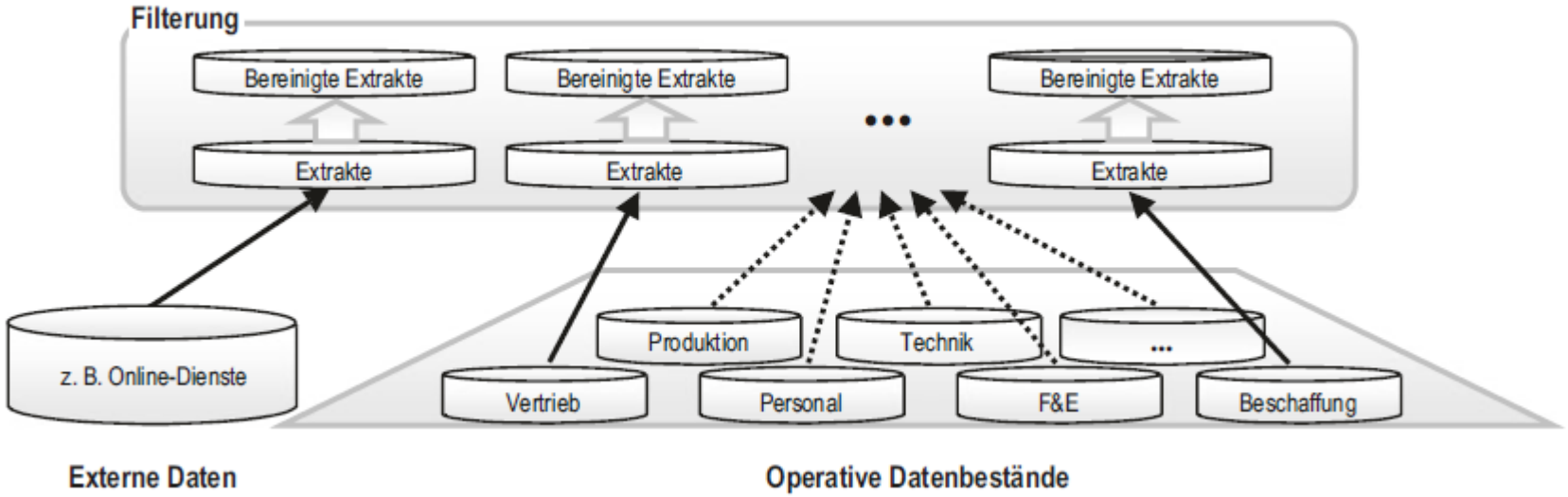
+ Filterung
- Unter der Filterung wird die Extraktion aus den operativen Daten und die Bereinigung syntaktischer oder inhaltlicher Defekte in den zu übernehmenden Daten verstanden.
+ Harmonisierung
- Die Harmonisierung bezeichnet den Prozess der betriebswirtschaftlichen Abstimmung gefilterter Daten.
+ Aggregation
- Die Aggregation ist die Verdichtung gefilterter und harmonisierter Daten.
+ Anreicherung
- Die Bildung und Speicherung betriebswirtschaftlicher Kennzahlen aus gefilterten und harmonisierten Daten wird als Anreicherung bezeichnet.

Quelle: Henning Baars, Hans-Georg Kemper; Business Intelligence & Analytics – Grundlagen und praktische Anwendungen; Springer 2021

ETL – Extract, Transform, Load
Transformationsprozess

Filterung

- Erste Schicht der Transformation
- Heterogene, interne und externe Datenquellen
- Im Rahmen der Extraktion können die Daten in Extraktionsbereiche (Staging Areas) evtl. in-memory bereitgestellt werden.
- Bei der Bereinigung können syntaktische Mängel (z.B. falsche Formate, Sonderzeichen) automatisiert entfernt.
- Semantische Mängel können z.T. automatisiert entfernt. Manche Fehler sind nur manuell durch Experten identifizierbar.

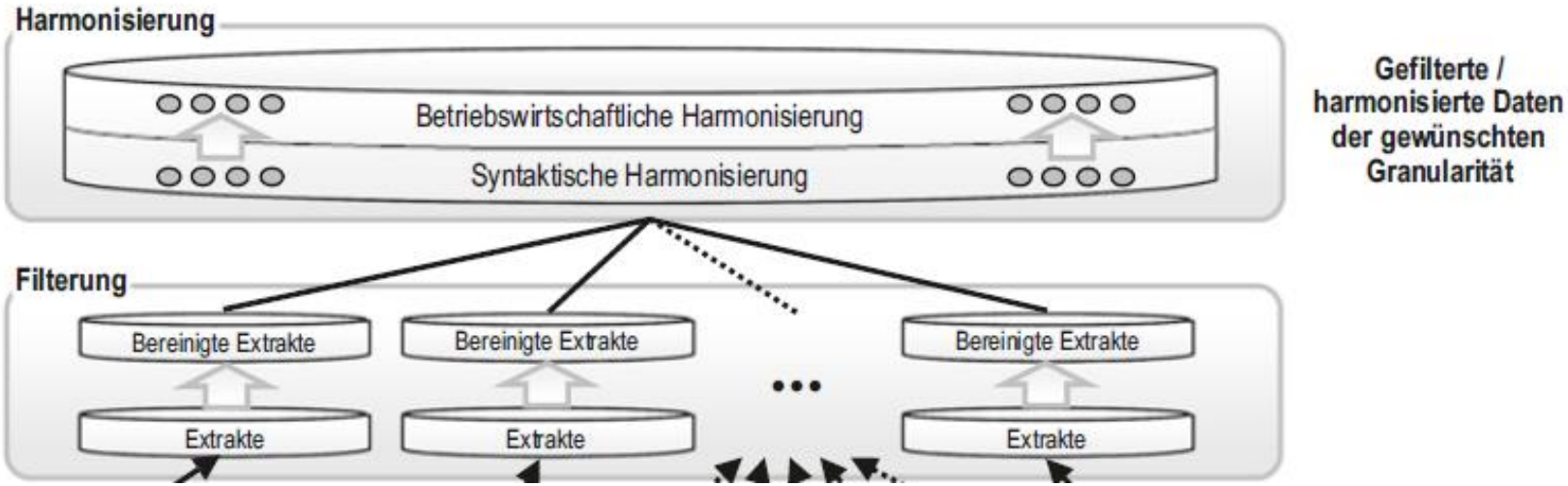


Quelle: Henning Baars, Hans-Georg Kemper; Business Intelligence & Analytics – Grundlagen und praktische Anwendungen; Springer 2021

ETL – Extract, Transform, Load Transformationsprozess

Harmonisierung

- Unterschiedliche Kodierung wird vereinheitlicht.
- Synonyme können z.B. durch Mapping-Tabellen vereinheitlicht werden.
- Homonyme haben unterschiedliche Bedeutungen
- Nach Abschluss der Transformationen der Filterungs- und der Harmonisierungsschicht liegt ein bereinigter und konsistenter Datenbestand vor.



Quelle: Henning Baars, Hans-Georg Kemper; Business Intelligence & Analytics – Grundlagen und praktische Anwendungen; Springer 2021

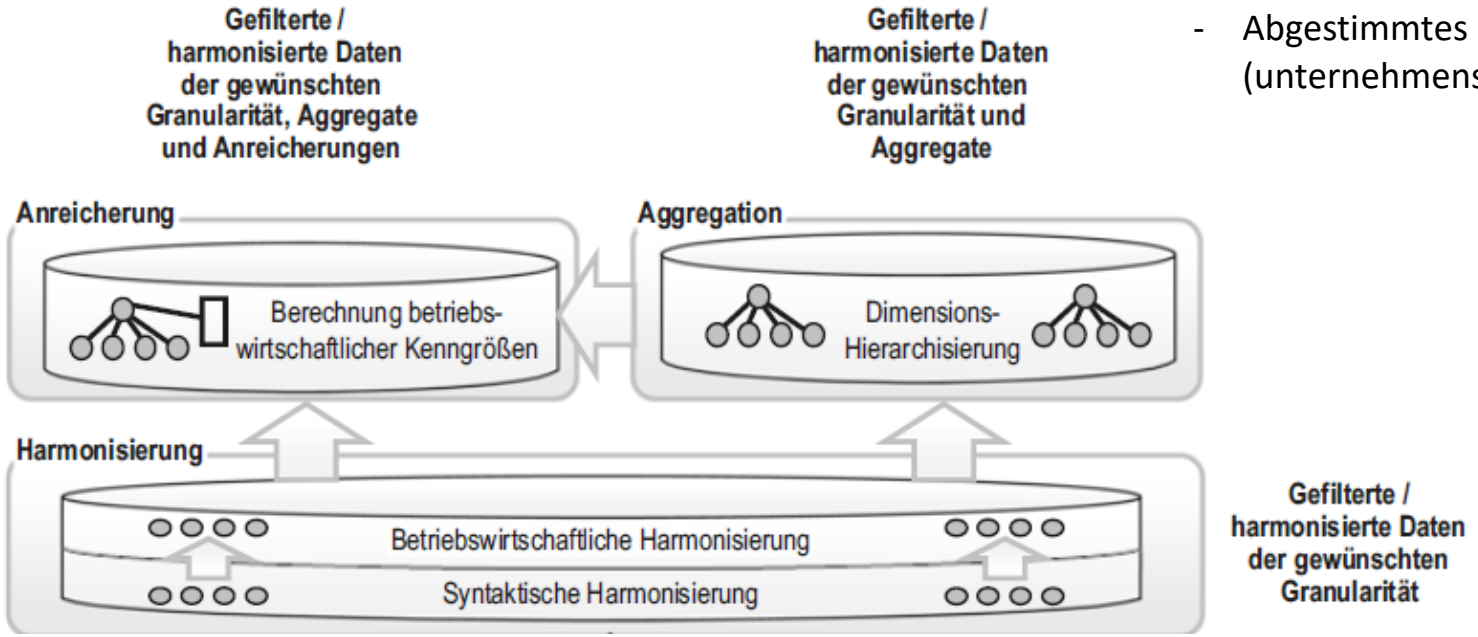
ETL – Extract, Transform, Load Transformationsprozess

Aggregation

- \+ Verdichtung der Daten
- \+ Nutzung von Hierarchien,
  z.B. Kunde → Kundengruppe → Gesamt
  Stadt → Bundesland → Land → Vertriebsregion → Gesamt

Anreicherung

- \+ Berechnung von Kennzahlen
- \- Vorberechnung reduziert Antwortzeitverhalten
- \- Garantierte Konsistenz der kalkulierten Werte (anwendungsübergreifend einmal berechnet, „Single Point of Truth“)
- \- Abgestimmtes betriebswirtschaftliches Instrumentarium (unternehmensweit)

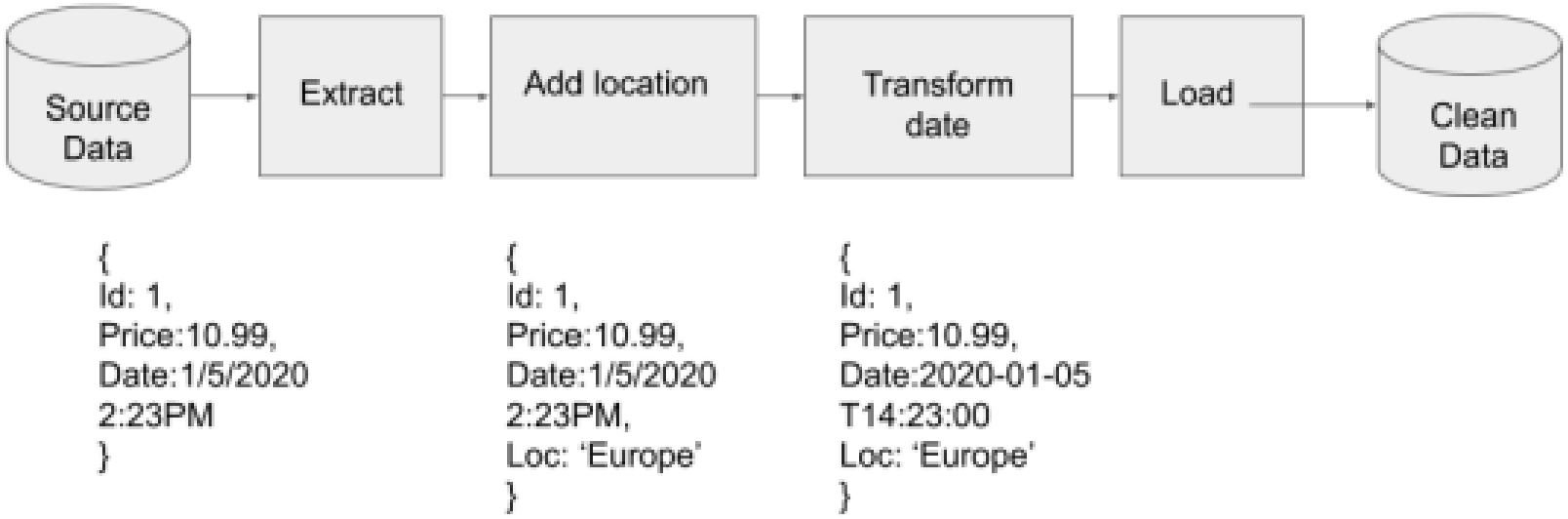


Quelle: Henning Baars, Hans-Georg Kemper; Business Intelligence & Analytics – Grundlagen und praktische Anwendungen; Springer 2021

ETL – Extract, Transform, Load

Beispiel für eine ETL-Pipeline

- Auslesen der Daten aus den Datenbanken
- Hinzufügen der Region der Datenquelle
- Formatierung der Zeit im ISO 8601-Format
- Laden (speichern) der Daten im einem Data Warehouse



Quelle: Data Engineering with python, Paul Crickard, Packt, Oktober 2020

Data pipelines mit Apache Airflow

Apache Airflow

+ entwickelt von Airbnb
+ beinhaltet
  - Web Server
  - Scheduler
  - Metastore
  - Queuing System
  - Executors
+ Zur Konfiguration (Orchestrierung) der Workflows werden Directed Acyclic Graphs (DAGs) verwendet.
+ Verteilung auf viele Nodes möglich
+ Workflows werden mit python scripts erstellt.
+ Apache Airflow wird auf der Google Cloud Platform und Amazon Web Services angeboten.

Motivation

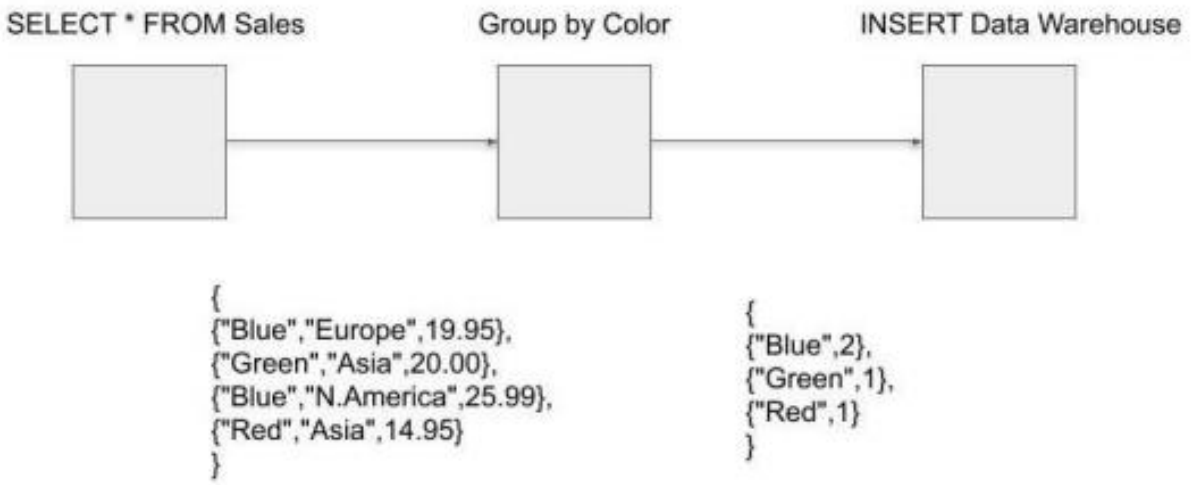
+ Verwaltung und effiziente Ausführung von ETL-Prozessen

Quelle: Data Engineering with python, Paul Crickard, Packt, Oktober 2020

Data pipelines mit Apache Airflow

Apache Airflow

+ Pipeline zum Extrahieren, Gruppieren und Speichern von Daten



Quelle: Data Engineering with python, Paul Crickard, Packt, Oktober 2020

Data pipelines mit Apache Airflow
Installation

Windows

- lokale Installation nicht möglich
- Docker-Installation fehlgeschlagen

Linux (ubuntu 21.04)

- lokale Installation siehe https://airflow.apache.org/docs/apache-airflow/stable/start/local.html

```
# Airflow needs a home. `~/airflow` is the default, but you can put it # somewhere else if you prefer (optional)
export AIRFLOW_HOME=~/airflow

# Install Airflow using the constraints file
AIRFLOW_VERSION=2.2.0 PYTHON_VERSION="$(python --version | cut -d " " -f 2 | cut -d "." -f 1-2)“
# For example: 3.9
CONSTRAINT_URL=https://raw.githubusercontent.com/apache/airflow/constraints-${AIRFLOW_VERSION}/constraints-${PYTHON_VERSION}.txt
# For example: https://raw.githubusercontent.com/apache/airflow/constraints-2.2.0/constraints-3.9.txt
pip install "apache-airflow==${AIRFLOW_VERSION}" --constraint "${CONSTRAINT_URL}“

# The Standalone command will initialise the database, make a user,
# and start all components for you.
airflow standalone

# Visit localhost:8080 in the browser and use the admin account details
# shown on the terminal to login.
# Enable the example_bash_operator dag in the home page
```

Quelle: https://airflow.apache.org/docs/apache-airflow/stable/start/local.html

Data pipelines mit Apache Airflow Einführung

Start

+ airflow standalone

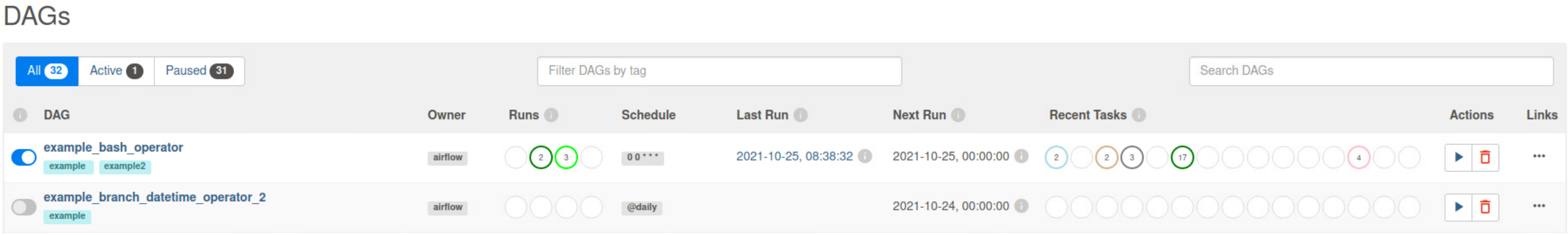
oder

Start einzelner Komponenten

+ airflow webserver
+ airflow scheduler

Der Webserver ist nach Start von airflow unter http://localhost:8080 erreichbar.

Auf der Webseite sind Beispiele sichtbar, z.B. example_bash_operator

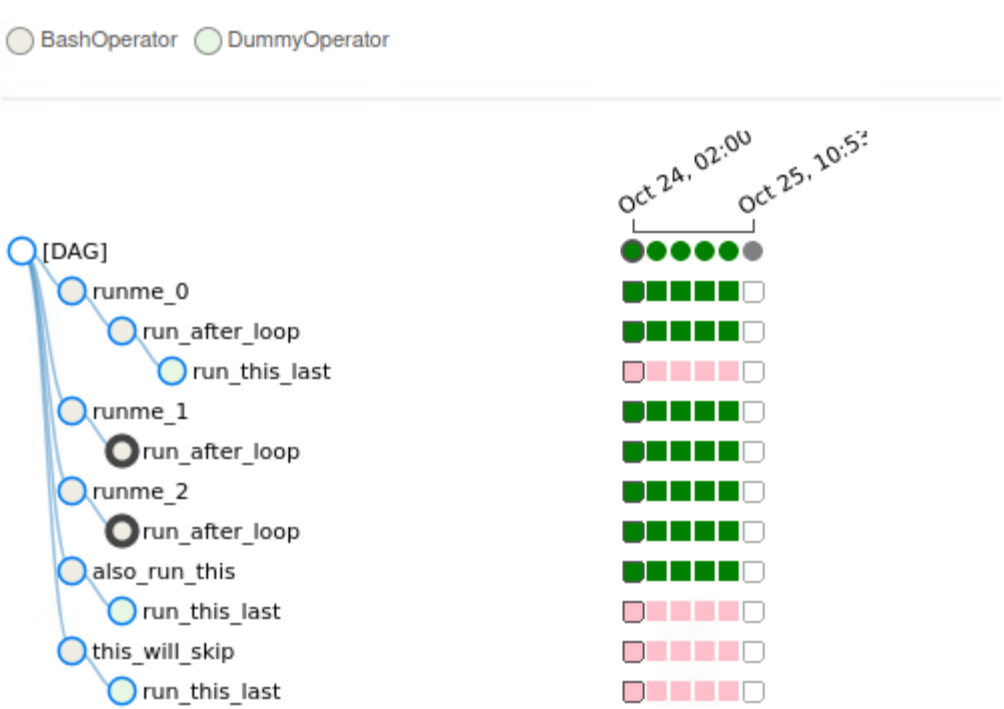


Quelle: https://airflow.apache.org/docs/apache-airflow/stable/start/local.html

Data pipelines mit Apache Airflow
Einführung

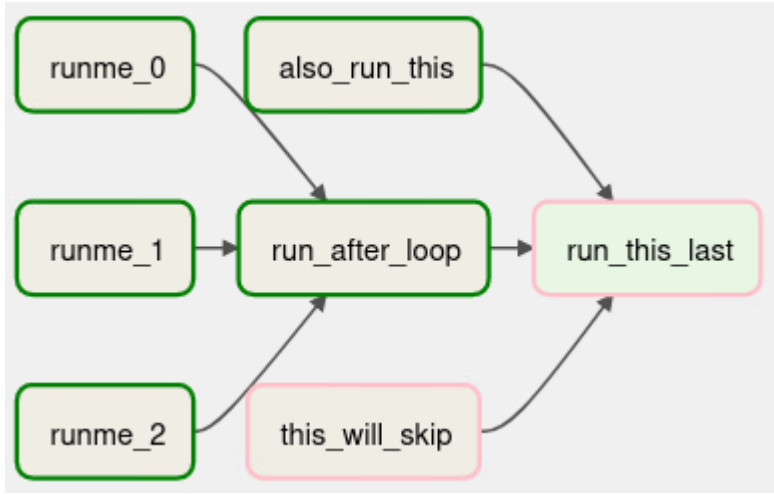
Darstellung der Tasks als Baumstruktur

- \+ Baumstruktur zeigt einzelne Tasks
- \+ Ausführungszustand wird dargestellt



Graph Struktur

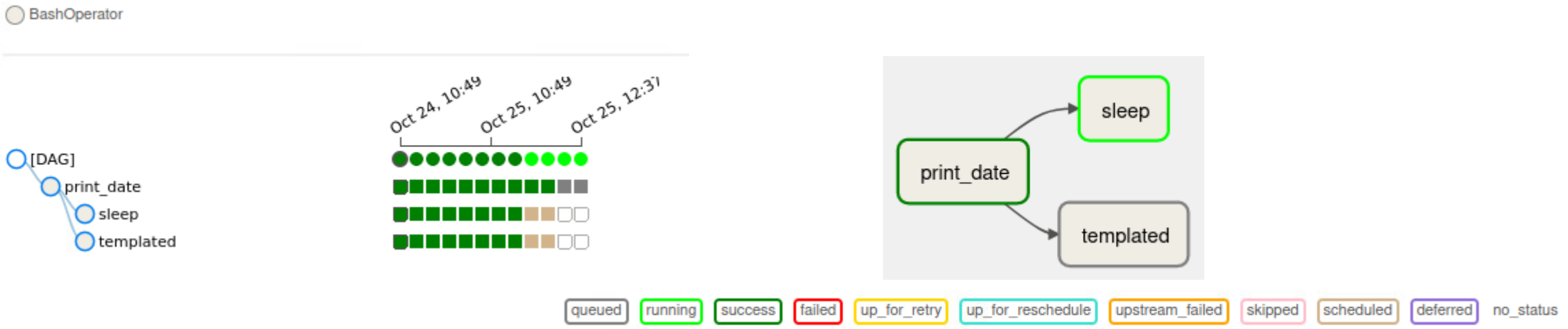
- \+ Ablauf und Zusammenhang der einzelnen Tasks



Data pipelines mit Apache Airflow
Einführung tutorial.py

Einführungsbeispiel tutorial.py

- 18-Airflow-tutorial
- Definition der Pipeline erfolgt als python-Skript
- Configuration file specifying the DAG's structure
- Drei Tasks, print_date, sleep und templated



Quelle: https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html

Data pipelines mit Apache Airflow
Einführung tutorial.py

Einführungsbeispiel tutorial.py

- \+ 18-Airflow-tutorial
- \+ Importing Modules

```
from datetime import datetime, timedelta
from textwrap import dedent
# The DAG object; we'll need this to instantiate a DAG (directed acyclic graph)
from airflow import DAG
# Operators; we need this to operate!
from airflow.operators.bash import BashOperator
```

Quelle: https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html

Data pipelines mit Apache Airflow
Einführung tutorial.py

Einführungsbeispiel tutorial.py

- 18-Airflow-tutorial
- Default Arguments (as dictionary)

```
# These args will get passed on to each operator
# You can override them on a per-task basis during operator initialization
default_args = {
                'owner': 'airflow‘,
                'depends_on_past': False,
                'email': ['airflow@example.com‘],
                'email_on_failure': False,
                'email_on_retry': False,
                'retries': 1,
                'retry_delay': timedelta(minutes=5)
}
```

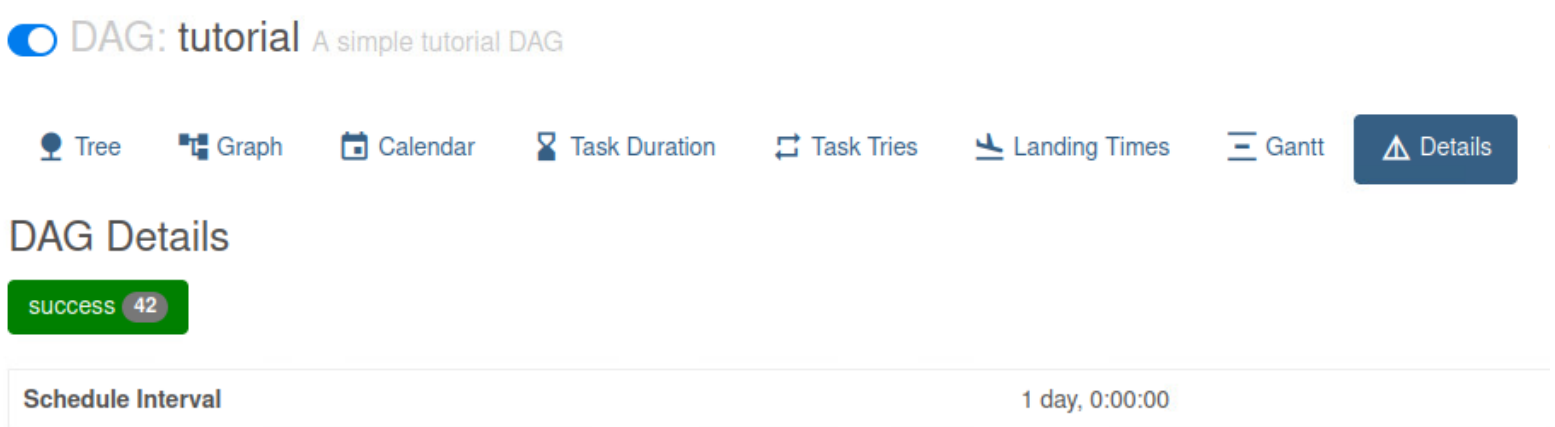
Quelle: https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html

Data pipelines mit Apache Airflow
Einführung tutorial.py

Einführungsbeispiel tutorial.py

- 18-Airflow-tutorial
- Instanziierung eines Directed Acyclic Graph

```
with DAG(
        'tutorial',
        default_args=default_args, description='A simple tutorial DAG',
        schedule_interval=timedelta(days=1),
        start_date=datetime(2021, 1, 1),
        catchup=False, tags=['example'],
) as dag:
```



Quelle: https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html

Data pipelines mit Apache Airflow
Einführung tutorial.py

Einführungsbeispiel tutorial.py

- 18-Airflow-tutorial
- Instanziierung eines Directed Acyclic Graph

```
with DAG(
        'tutorial‘,
        default_args=default_args, description='A simple tutorial DAG‘,
        schedule_interval=timedelta(days=1),
        start_date=datetime(2021, 1, 1),
        catchup=False, tags=['example‘],
) as dag:
```

Default Args	{'depends_on_past': False, 'email': ['airflow@example.com'], 'email_on_failure': False, 'email_on_retry': False, 'owner': 'airflow', 'retries': 1, 'retry_delay': datetime.timedelta(seconds=300)}
Tasks Count	3
Task IDs	['print_date', 'sleep', 'templated']

Quelle: https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html

Data pipelines mit Apache Airflow
Einführung tutorial.py

Einführungsbeispiel tutorial.py

- 18-Airflow-tutorial
- Tasks
- Bash: Bourne-again shell
- Alle default_args werden den Tasks t1, t2 und t3 übergeben.
- Task 1 löst das Shell-Kommando „date“ aus.
- Task 2 wartet für 5 Sekunden.

```
t1 = BashOperator( task_id='print_date', bash_command='date‘, )

t2 = BashOperator( task_id='sleep', depends_on_past=False, bash_command='sleep 5', retries=3, )
```

Quelle: https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html

Data pipelines mit Apache Airflow
Einführung tutorial.py

Einführungsbeispiel tutorial.py

- 18-Airflow-tutorial
- Task 3 wird über ein Jinja-Template definiert.

- {{ ds }} today‘s „date stamp“
- {% for i in range(5) %} code logic
- "{{ macros.ds_add(ds, 7) add 7 days
- params.my_param user-defined parameter

```
templated_command = dedent(
                """„
{% for i in range(5) %}
                echo "{{ ds }}„
                echo "{{ macros.ds_add(ds, 7)}}„
                echo "{{ params.my_param }}„
{% endfor %}
"""
)

t3 = BashOperator(
                task_id='templated‘,
                depends_on_past=False,
                bash_command=templated_command,
                params={'my_param': 'Parameter I passed in‘},
)
```

Quelle: https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html

Data pipelines mit Apache Airflow
Einführung tutorial.py

Einführungsbeispiel tutorial.py

- 18-Airflow-tutorial
- Tasks t1, t2 und t3 sind erstellt. Im nächsten Schritt werden Abhängigkeit zwischen ihnen definiert.
- Tasks t2 und t3 hängen von Task t1 ab.

```
t1 >> [t2, t3]
```

- t2 und t3 werden „downstream“ von t1 ausgeführt.
- Andere Schreibweise
  t1.set_downstream([t2, t3])

Quelle: https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html

Data pipelines mit Apache Airflow
Einführung tutorial.py

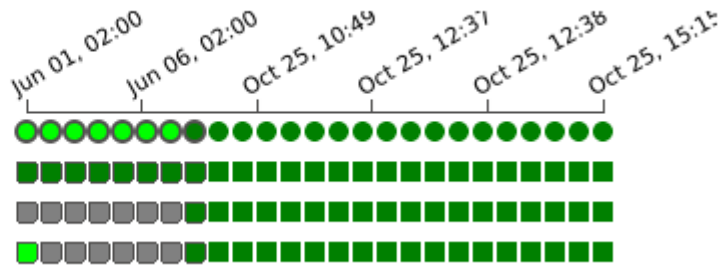
Einführungsbeispiel tutorial.py

+ 18-Airflow-tutorial

Ausführung des Scripts

+ Skript ist Teil der Examples
+ Ausführen des Skripts als Test auf korrekte Syntax (optional)
  - python PATH/tutorial.py
+ Initialisierung der Datenbank
  - airflow db init
+ Liste aktiver DAGs
  - airflow dags list
+ Liste aller Tasks im DAG „tutorial“
  - airflow tasks list tutorial
+ Hierarchie der Tasks im DAG „tutorial“
  - airflow tasks list tutorial –tree

```
# start your backfill on a date range
airflow dags backfill tutorial \
                --start-date 2015-06-01 \
                --end-date 2015-06-07
```



Quelle: https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html

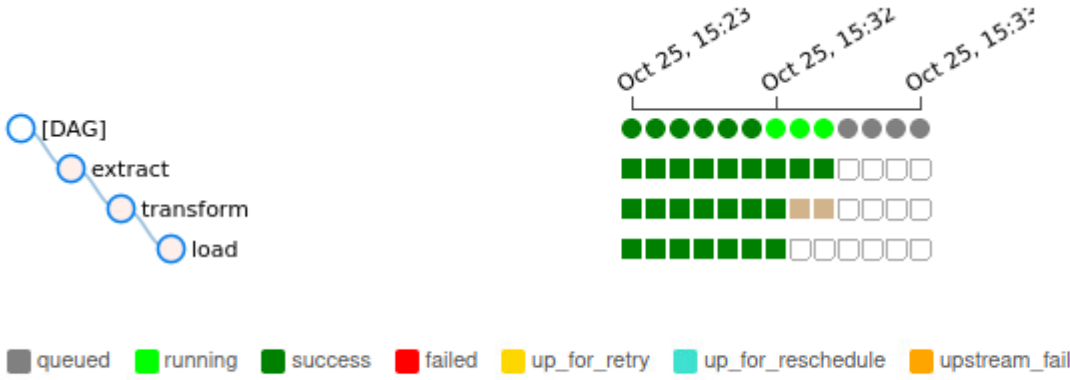
Data pipelines mit Apache Airflow
Einführung tutorial_taskflow_api_etl.py

Einführungsbeispiel tutorial_taskflow_api_etl.py

+ 19-Airflow-ETL
+ Airflow 2.0 DAG (einfachere Syntax als Airflow 1.x)

Einfacher ETL-Prozess (Extract, Transform, Load)

+ Drei Tasks
- Extract
- Transform
- Load



Quelle: https://airflow.apache.org/docs/apache-airflow/stable/tutorial_taskflow_api.html

Data pipelines mit Apache Airflow
Einführung CSVtoJSON.py

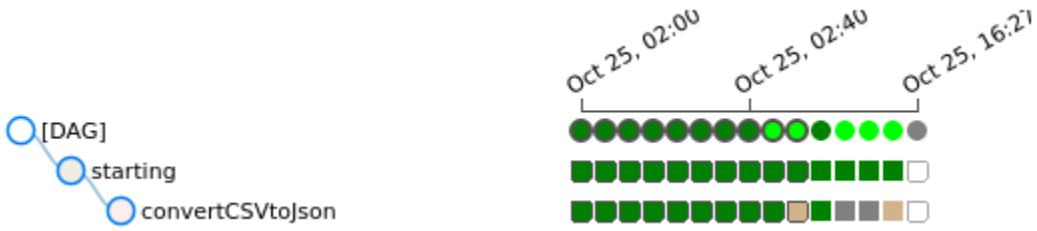
Beispiel CSVtoJSON.py

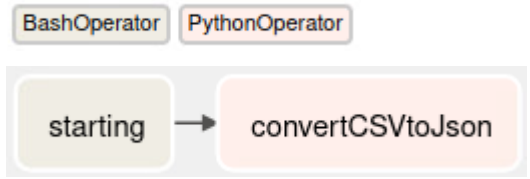
+ 20-Airflow-CSV-to-JSON

Convert CSV to JSON

+ Zwei Tasks
- starting
- convertCSVtoJson

+ Folgende Schritte werden ausgeführt:
- Einlesen der Datei data.csv
- Konvertierung der CSV-Daten in json
- Schreiben in die Datei fromAirflow.json





Quelle: Data Engineering with python, Paul Crickard, Packt, Oktober 2020

Hochschule Karlsruhe
University of
Applied Sciences

www.h-ka.de