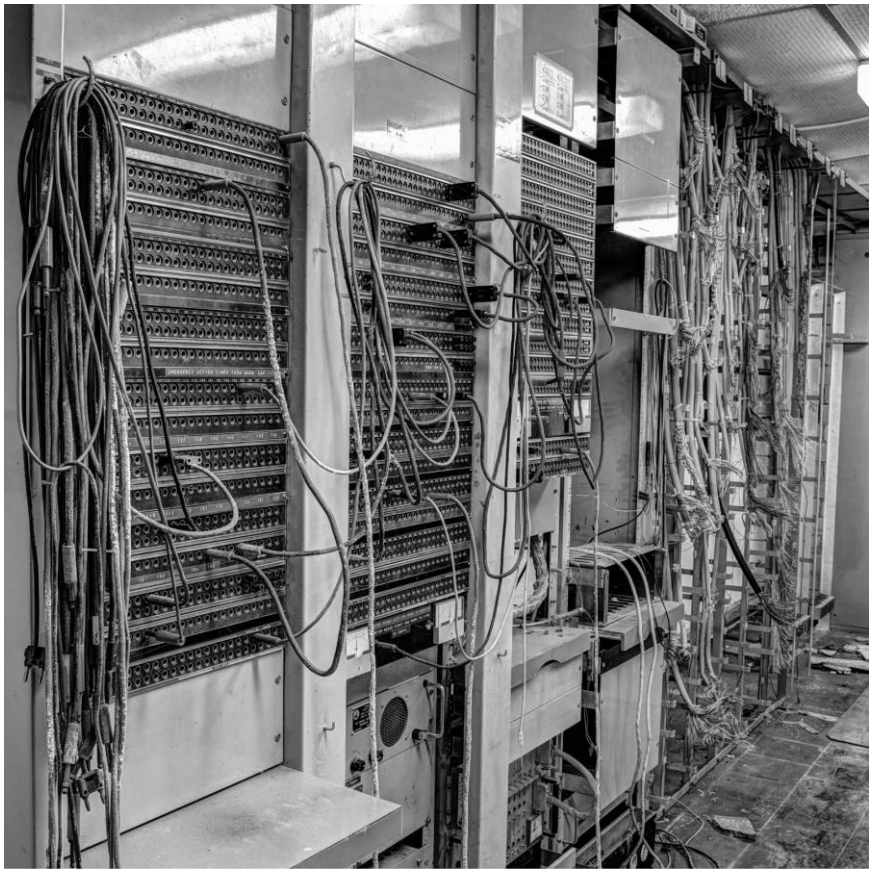
Apache Airflow

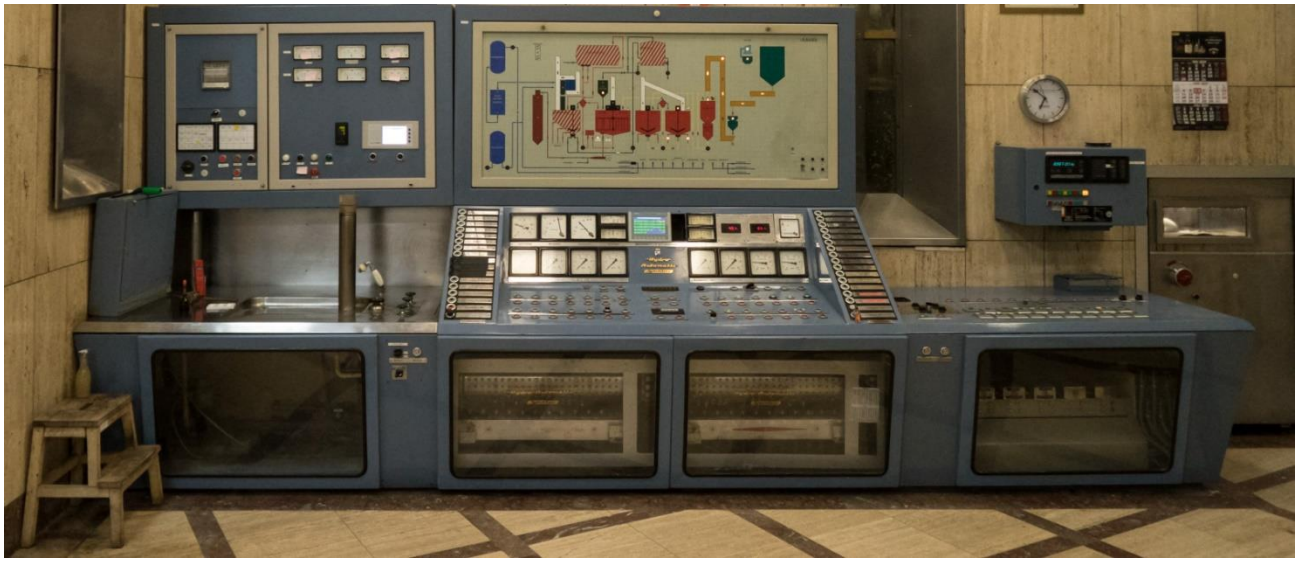
Airflow - Was ist das?

- Software-Platform für Workflows
  - Programmatische Erstellung
  - Verwaltung
  - Monitoring und Alerting
  - airflow.apache.org



Facts to Airflow

- Gestartet 2014
  von Maxime Beauchemin bei Airbnb
- Seit 2015 Open Source
- Ab 2016 Teil von Apache Software Foundation

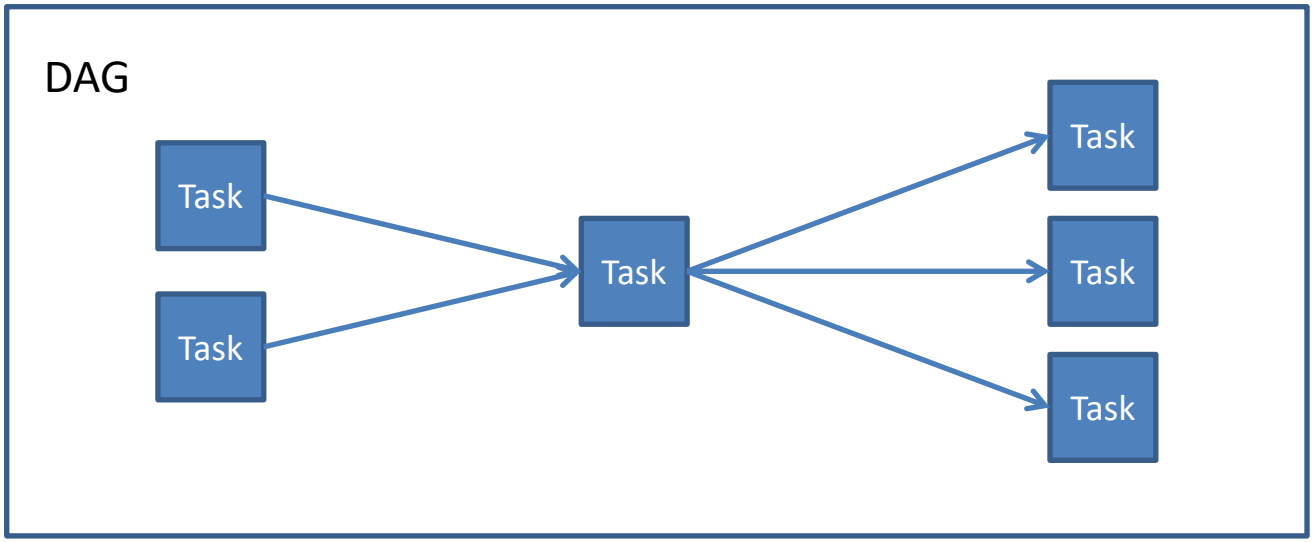


Was ist Airflow

- Basiert auf Python
- Job definitions in Python
- Xcom für operator-übergreifende Kommunikation
- Cool UI and Rich Command Line Interface
- Queues and Pools
- Backfilling
- Growing Community

Workflow in Airflow: DAG

- DAG - Direct acyclic Graph
- Series von Tasks verbunden durch Dependencies
- Eingängiger: Pipeline oder Workflow



Operatoren - Was startet Airflow

So ziemlich alles!

- BashOperator
- PythonOperator
- HiveOperator
- MySqlOperator
- CascadingOperator
- SparkOperator
- DummyOperator
- EmailOperator
- ExternalTaskSensor
- HdfsSensor
- Hive2SambaOperator
- HivePartitionSensor
- MySqlToHiveTransfer
- PostgresOperator
- PrestoCheckOperator
- PrestoIntervalCheckOperator
- PrestoValueCheckOperator
- HiveToMySqlTransfer
- S3KeySensor
- S3ToHiveTransfer
- SqlSensor
- TimeSensor

Airflow –CLI – Command Line Interface

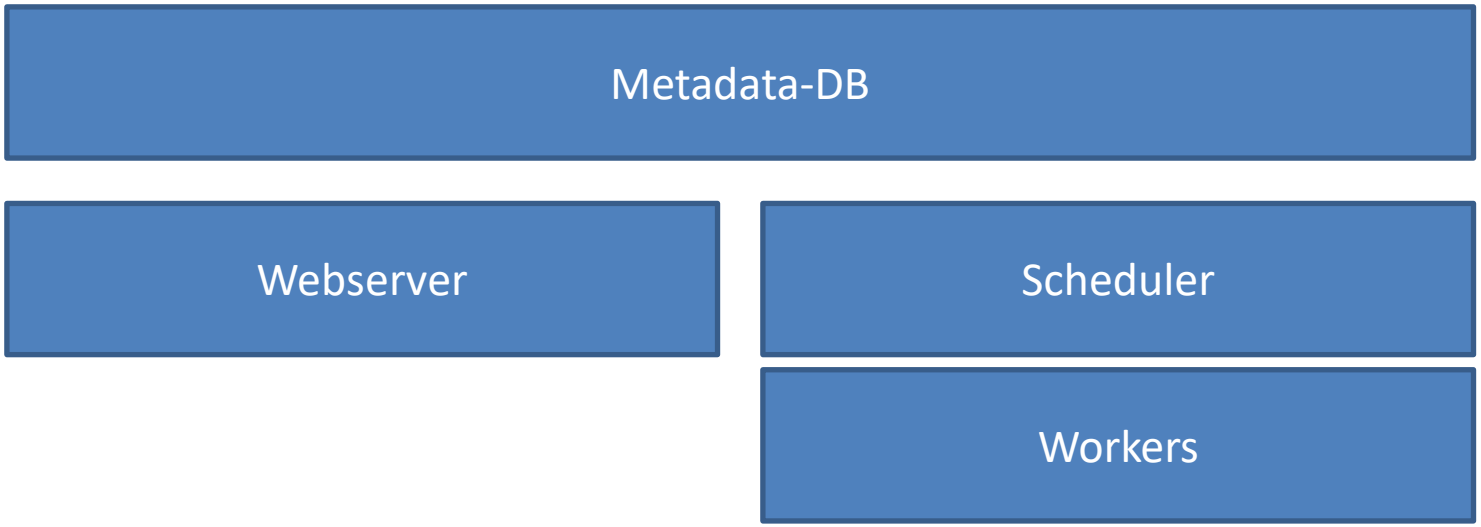
resetdb	Burn down and rebuild the metadata database
render	render a task instance's template(s)
create_user delete_user	Create or delete an account for the Web UI
Pause / unpause	Pause a DAG
task_failed_deps	Returns the unmet dependencies for a task instance from the perspective of the scheduler.
trigger_dag	Trigger a DAG run
initdb	Initialize the metadata database
test	Test a **task** instance. This will run a task without checking for dependencies or recording its state in the database.
list_dag_runs	List dag runs given a DAG id.
run	Run a single task instance

Airflow - Inbetriebnahme

- Install from pypi using pip
  `pip install apache-airflow`
- Initialize the database
  `airflow initdb`
- Start the web server, default port is 8080
  `airflow webserver -p 8080`
- Start the scheduler
  `airflow scheduler`

Airflow -Architektur / Komponenten

- Local Configuration



Local Executor

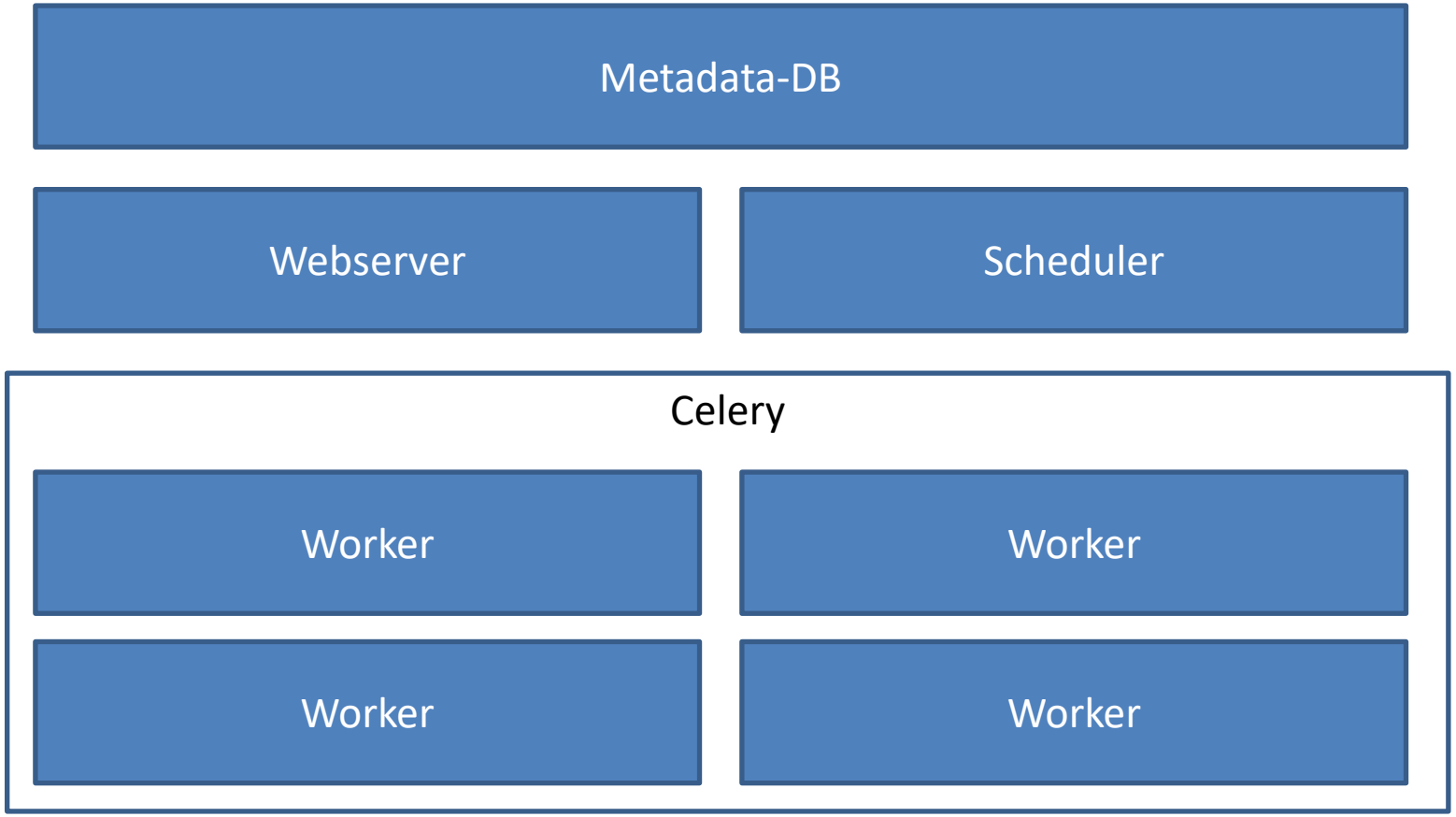
- Durch den Scheduler werden mehrere Processe erzeugt
- Vertical Scalable
- Auch für Produktionsumgebungen
- Braucht keine Broker

Sequential Executor

- Default Modus
- Minimales Setup funktionier auch mit SQLite
- Nur eine Task auf einmal
- Für Demo und Einarbeitung

Airflow -Architektur / Komponenten

- Distributed Configuration



Celery Executor

- Vertical and Horizontal Scalable
- Kann monitored werden (mit Flower)
- Supports Pools and Queues

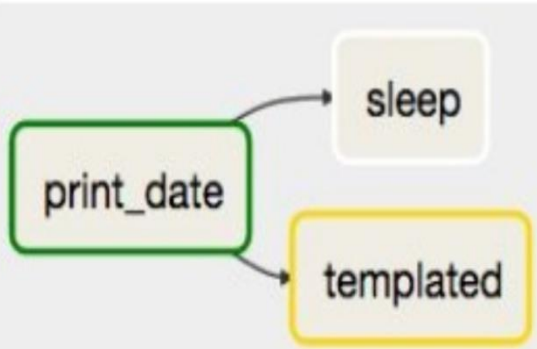
Erstellung von Workflows (DAGS)

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

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

```
t2.set_upstream(t1)
t3.set_upstream(t1)
```

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



Airflow – Erstellung DAGs - 1

```
# IMPORTS
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime, timedelta

# DEFAULT ARGUMENTS
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2019, 3, 21),
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'end_date': datetime(2020, 1, 1),
}
# DAG-Instantiation
dag = DAG('tutorial', default_args=default_args, schedule_interval=timedelta(days=1))
```

Airflow – Erstellung DAGs - 2

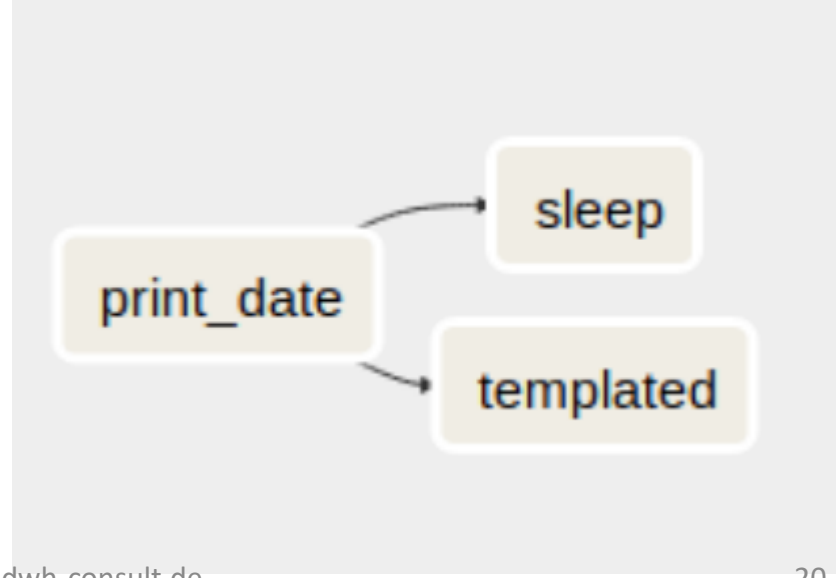
```
# task-definition
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag)

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

Airflow – Erstellung DAGs - 3

```
# Task based on a Jinja-Template
templated_command = """
    {% for i in range(5) %}
        echo "{{ ds }}"
        echo "{{ macros.ds_add(ds, 7)}}"
        echo "{{ params.my_param }}"
    {% endfor %}
"""

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

Airflow – Erstellung DAGs - 4

```
# dependencies
t2.set_upstream(t1)
t3.set_upstream(t1)

# oder
t1.set_downstream(t2)
t1.set_downstream(t3)

# oder:
t1 >> t2
t1 >> t3
```



Airflow – Pro und Kontra

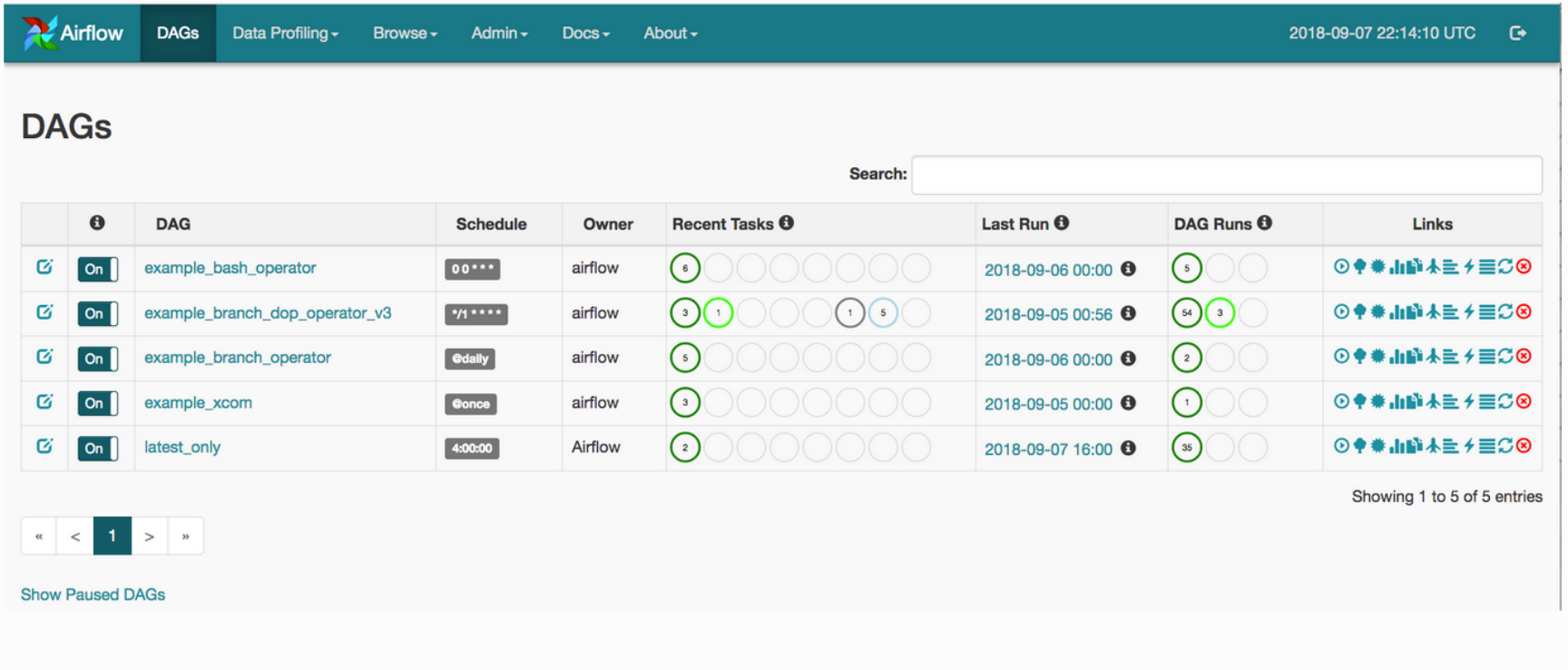
Pro

- Feine Konfigurationsmöglichkeit
- Viel unterschiedliche Zielsysteme
- Erleichtert komplexe Workflows
- Scheduling, Monitoring und Alerting in einem
- Große Community und gute Doku
- Automatisierte metadatengetriebene DAG und Taskerstellung mit Python möglich

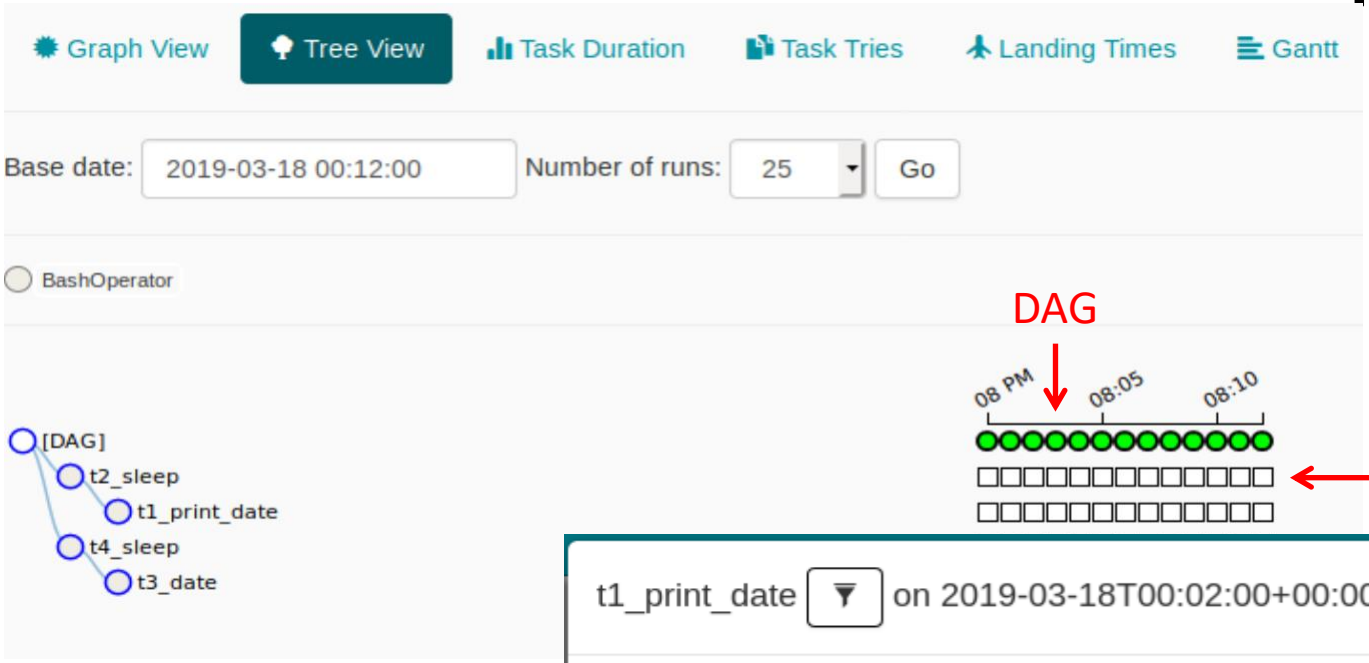
Kontra

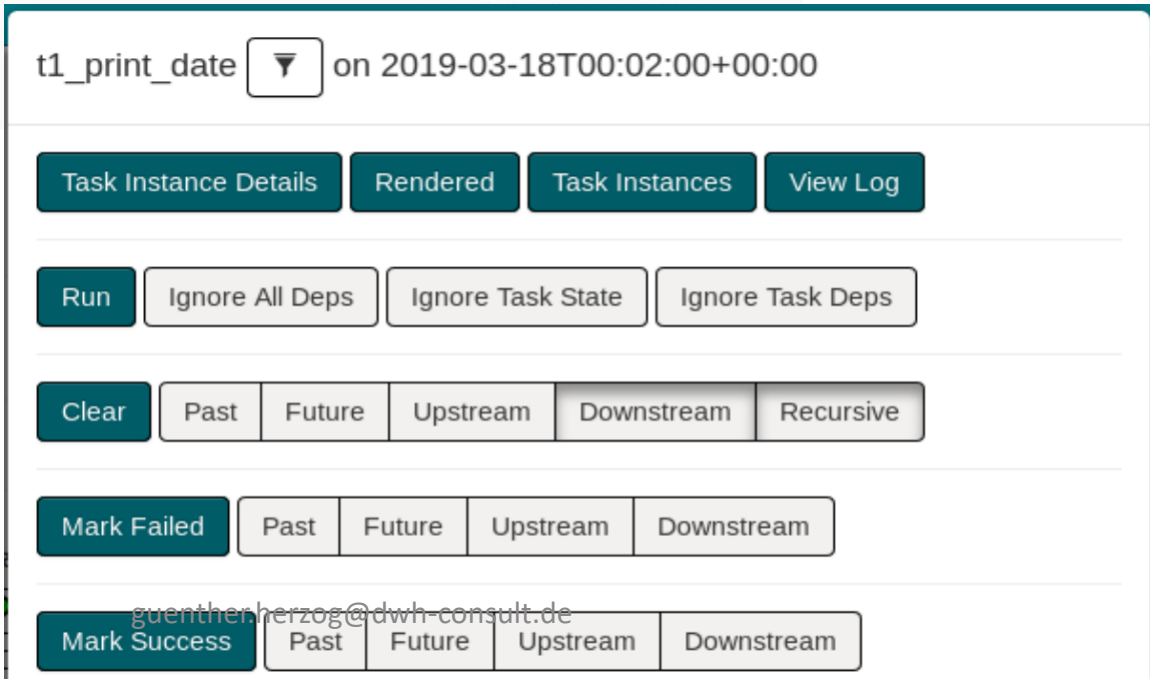
- Komplexe Konfiguration
- Infrastrukturelle Abhängigkeiten
- Programmierkenntnisse oder eigenes Framework für DAG-Erstellung notwendig

Airflow – User Interface



Airflow – Tree View Report

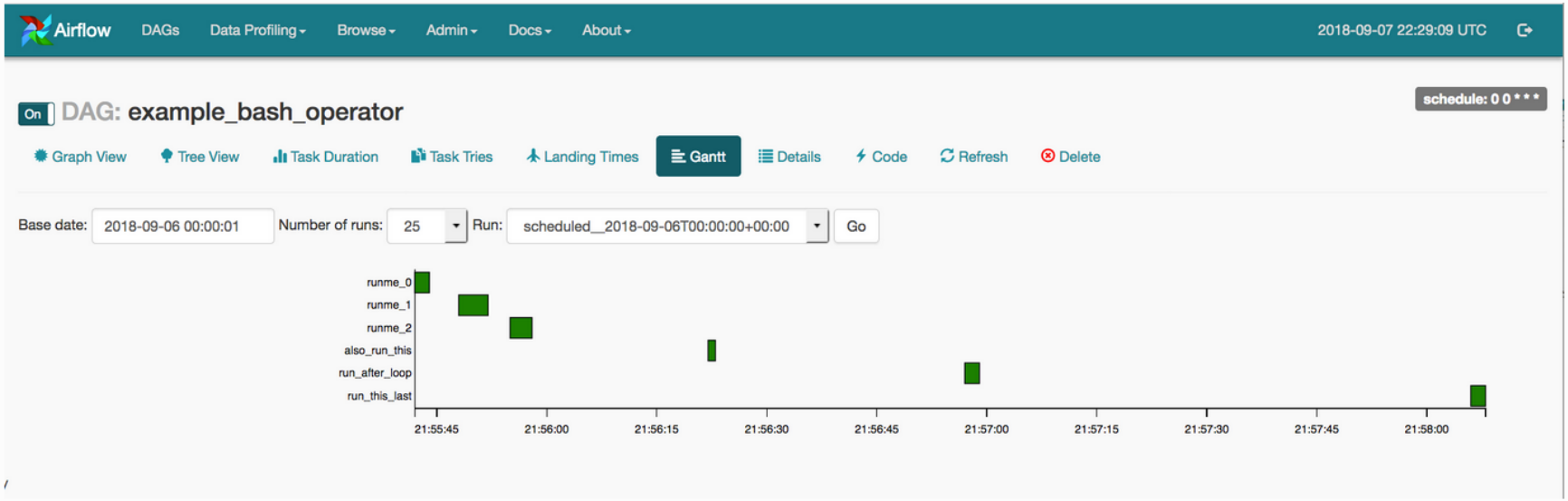




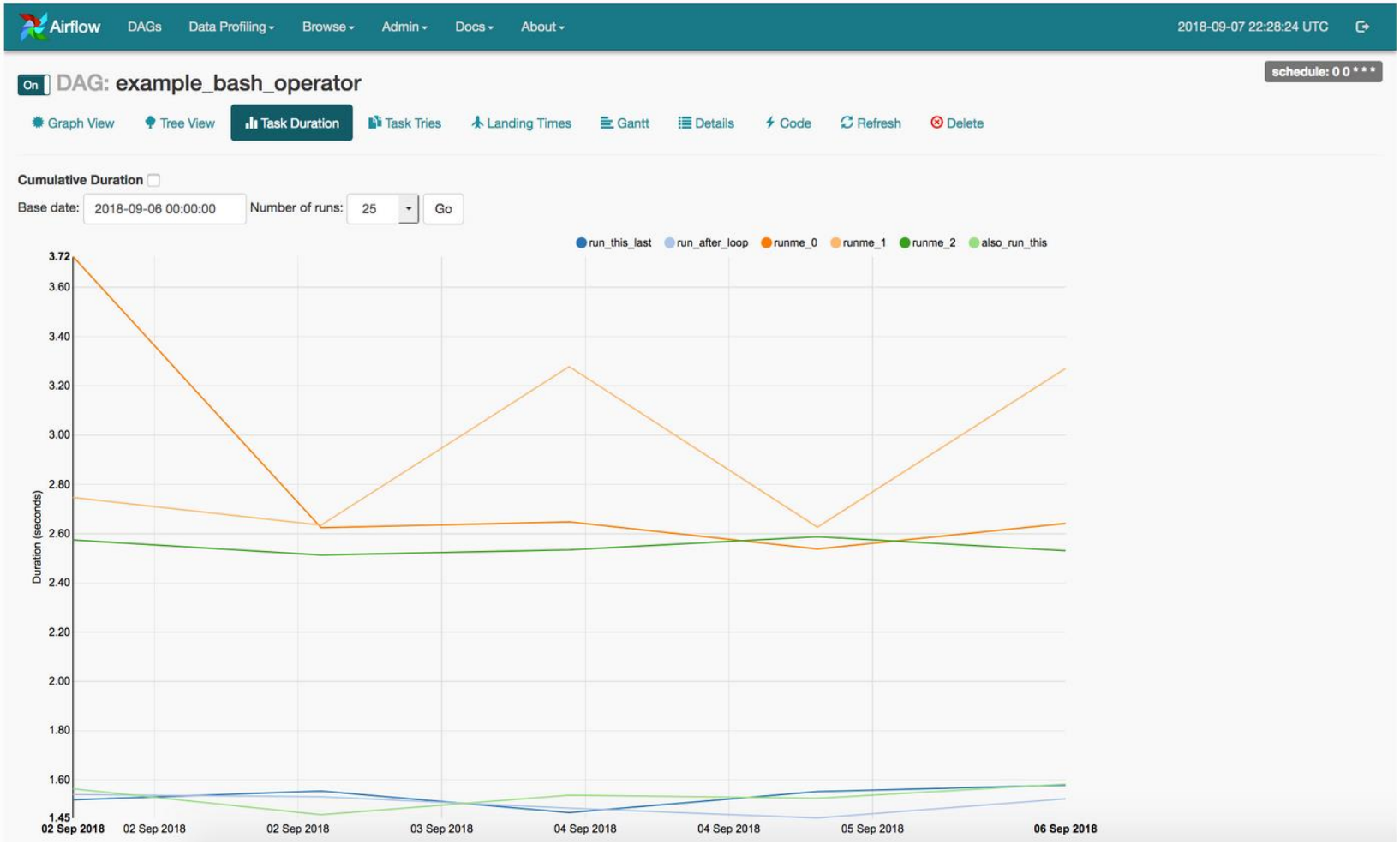
guenther.herzog@dwh-consult.de

Airflow - Gantt Chart

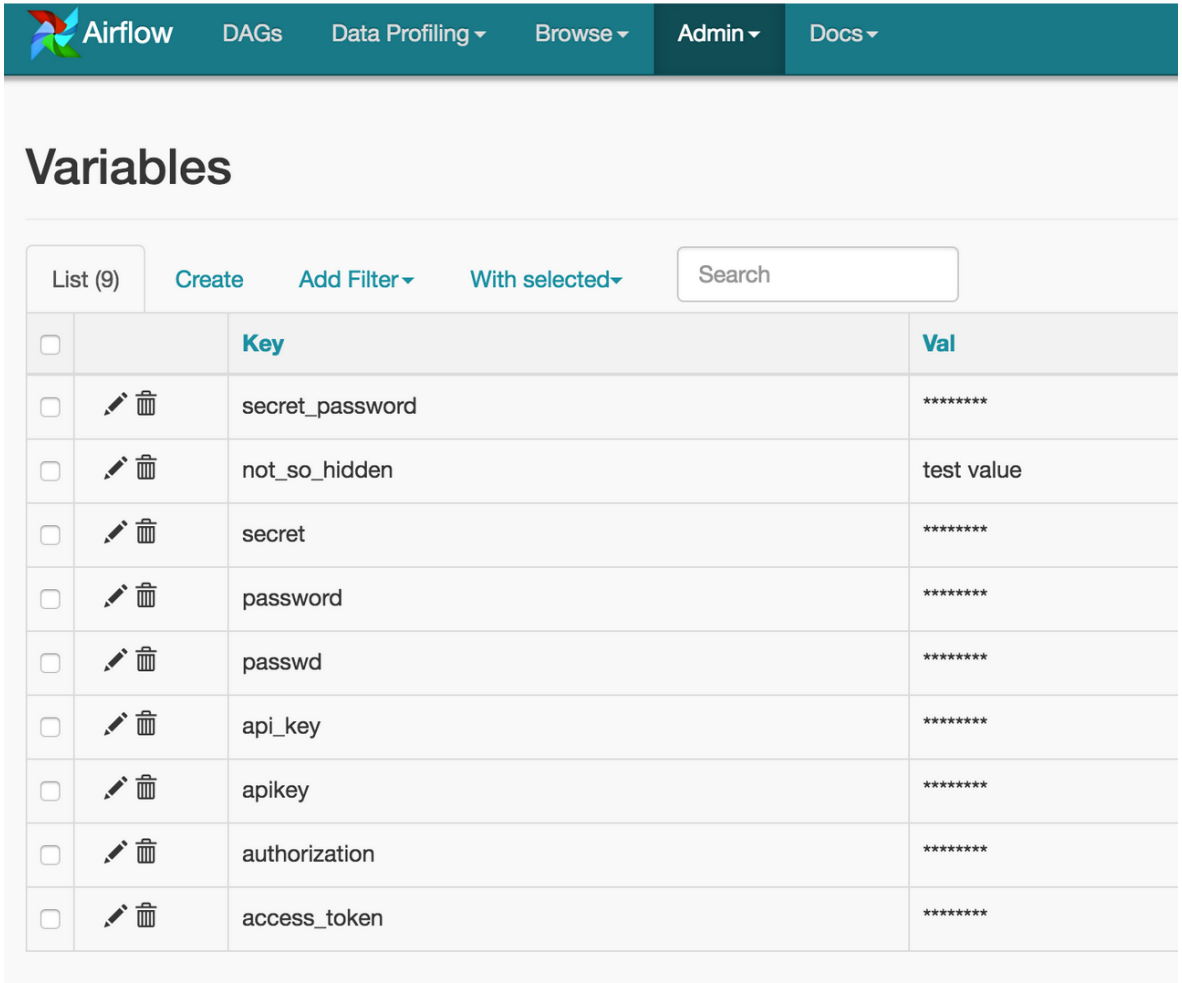
- Analyse der Ausführungszeiten der Tasks und Taskü̈berlappungen



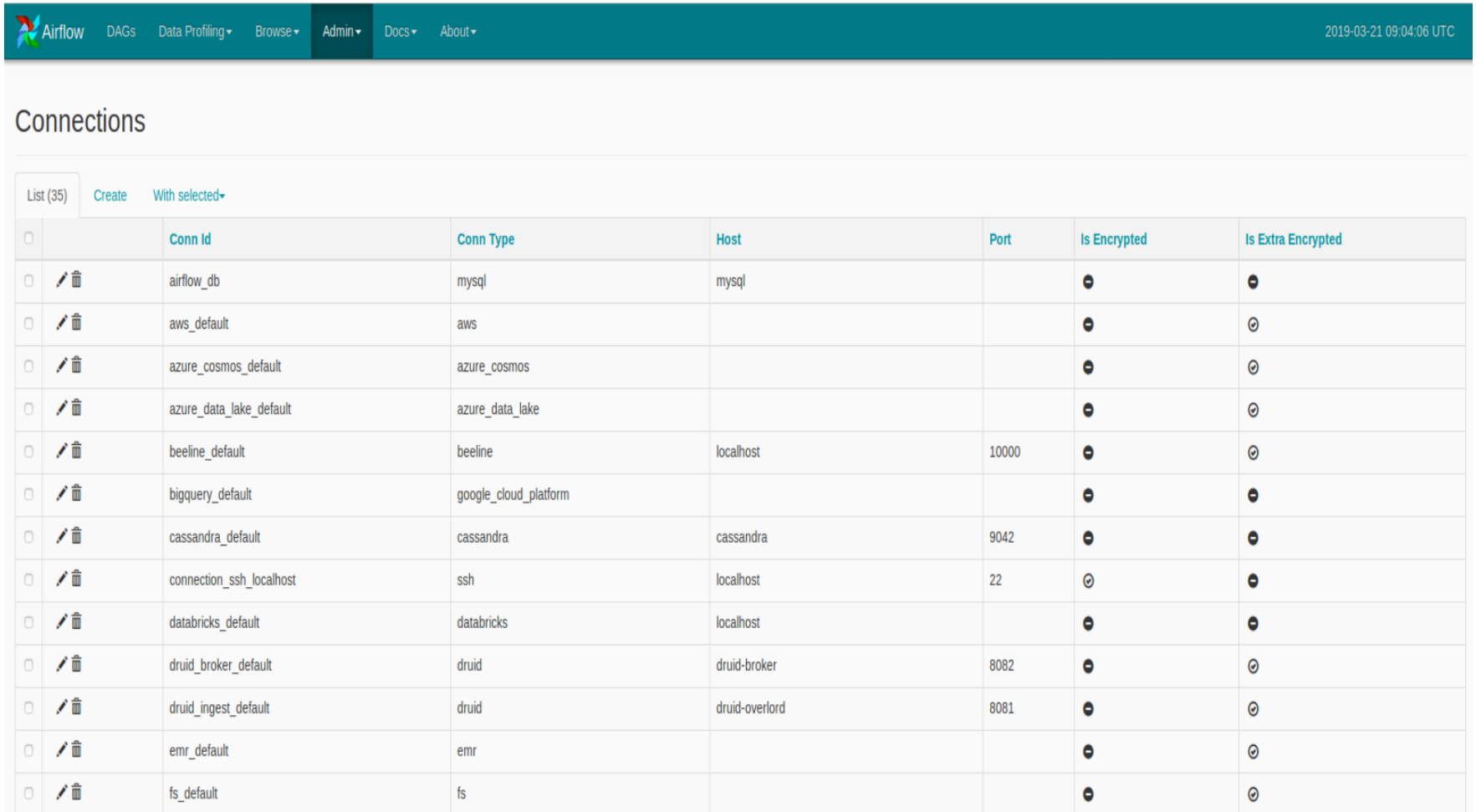
Airflow – Task Ausführungsdauer



Airflow – UI - Variablen



Airflow Connections



Conn Id	Conn Type	Host	Port	Is Encrypted	Is Extra Encrypted
airflow_db	mysql	mysql		⊖	⊖
aws_default	aws			⊖	⊘
azure_cosmos_default	azure_cosmos			⊖	⊘
azure_data_lake_default	azure_data_lake			⊖	⊘
beeline_default	beeline	localhost	10000	⊖	⊘
bigquery_default	google_cloud_platform			⊖	⊖
cassandra_default	cassandra	cassandra	9042	⊖	⊖
connection_ssh_localhost	ssh	localhost	22	⊘	⊖
databricks_default	databricks	localhost		⊖	⊖
druid_broker_default	druid	druid-broker	8082	⊖	⊘
druid_ingest_default	druid	druid-overlord	8081	⊖	⊘
emr_default	emr			⊖	⊘
fs_default	fs			⊖	⊘

Fragen?