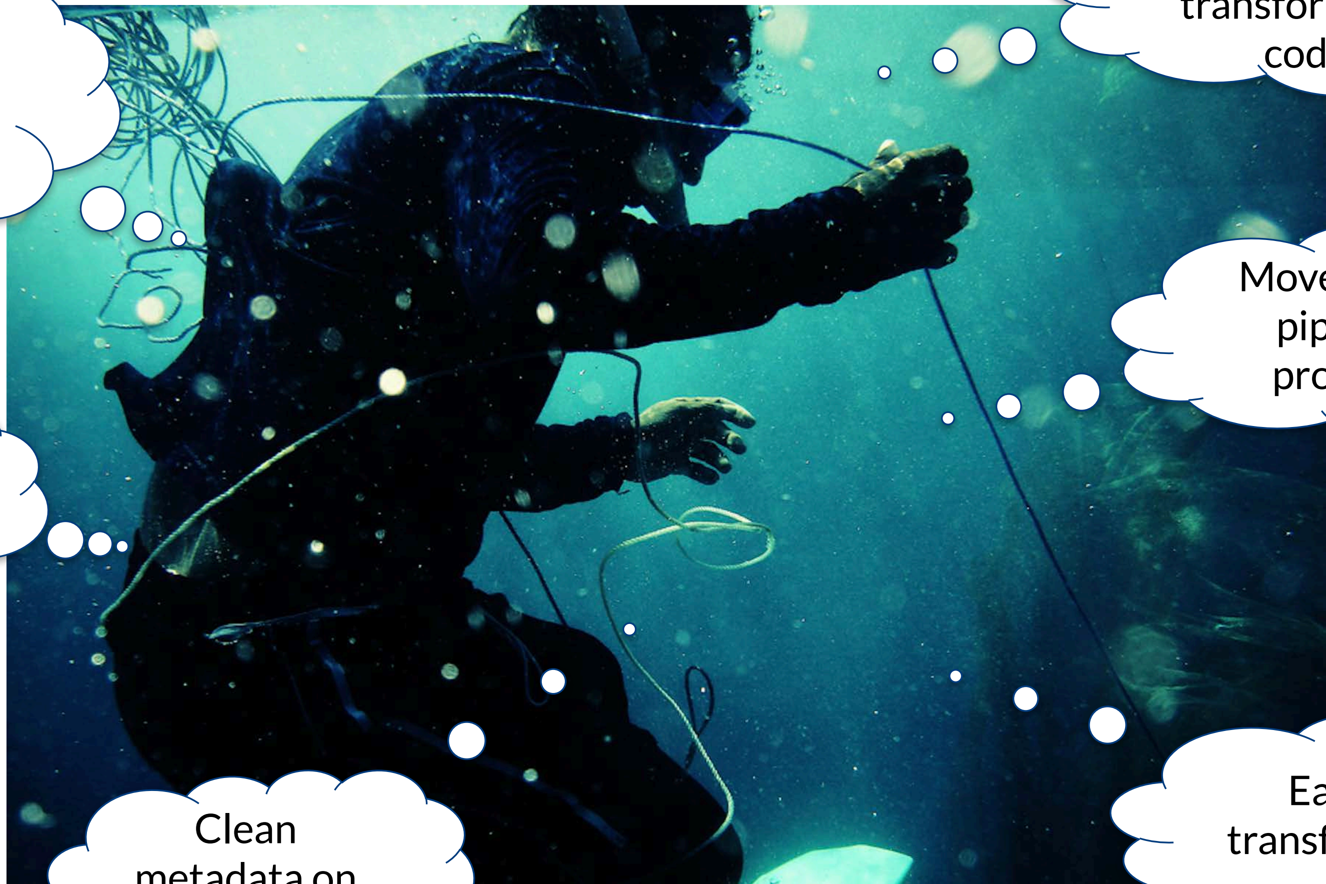
Flow is in the Air: Best Practices of Building Analytical Data Pipelines with Apache Airflow

Dr. Dominik Benz, inovex GmbH

PyConDe Karlsruhe, 27.10.2017

Diving deep in the analytical data lake?



Dependencies between jobs?

Avoid boilerplate data transformation code?

Move analytical pipelines to production?

Overview of failed jobs?

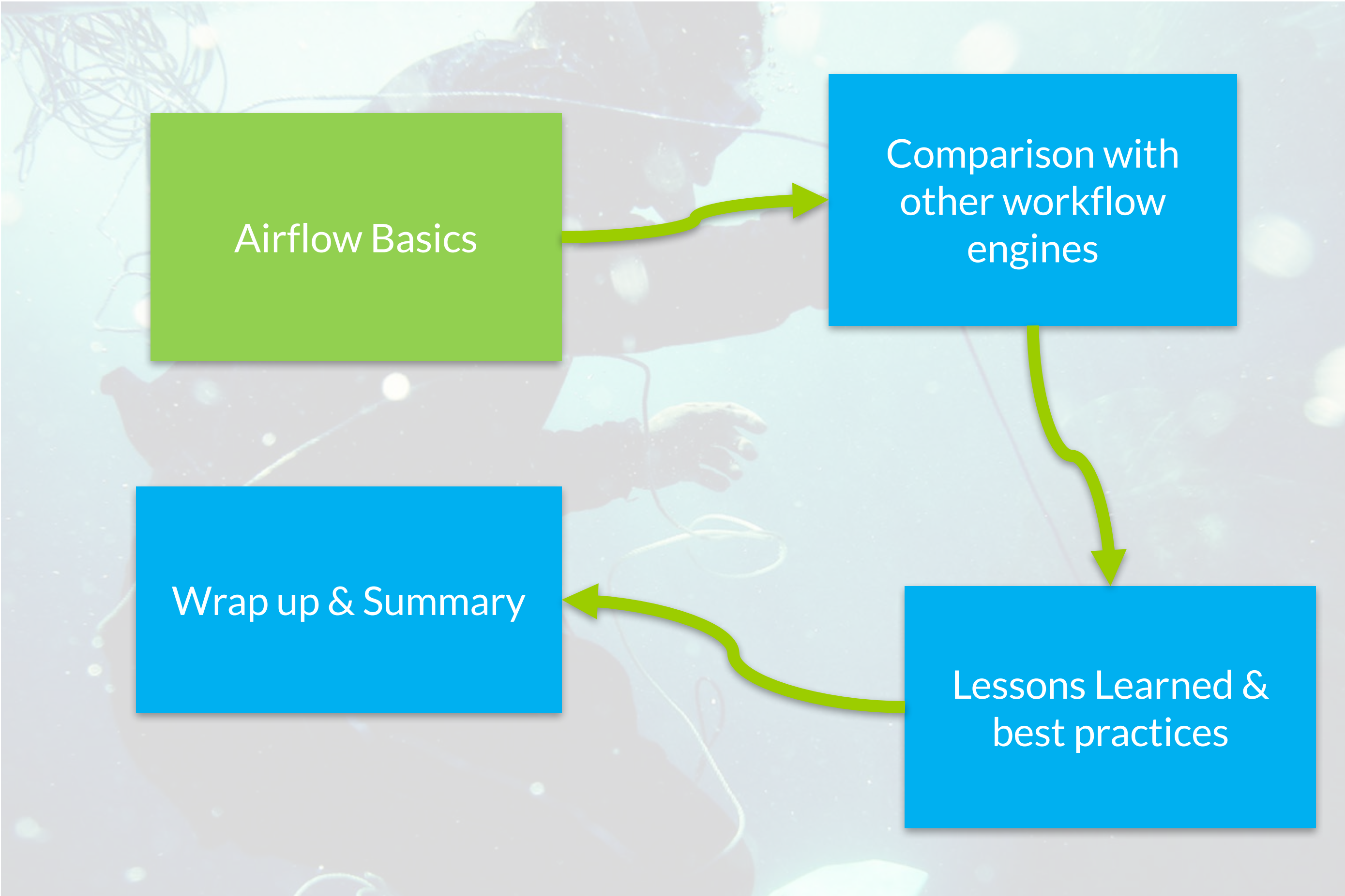
Clean metadata on job runs?

Easily test transformations?

https://flic.kr/p/64gmbY

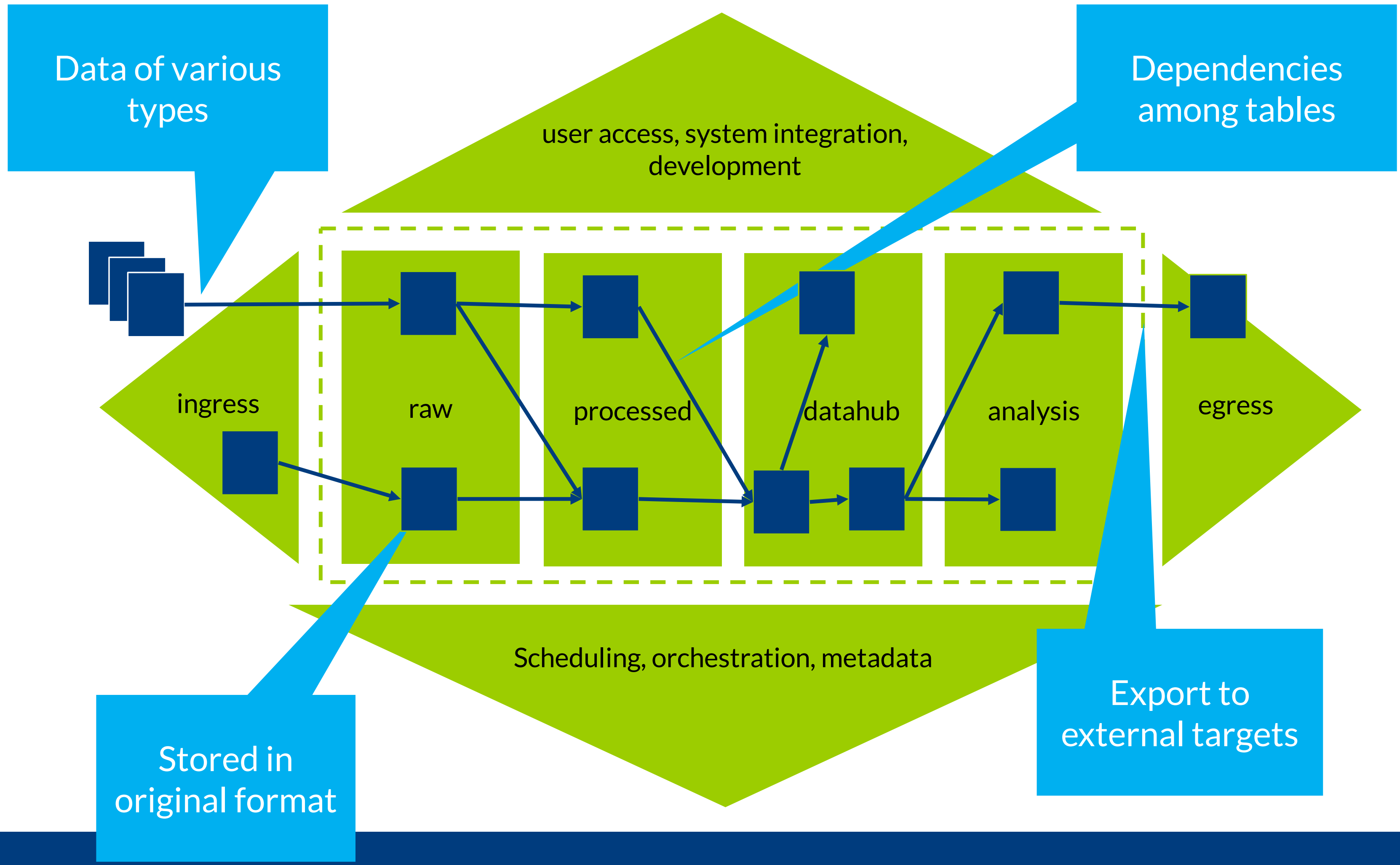
The Flow in Airflow

inovex



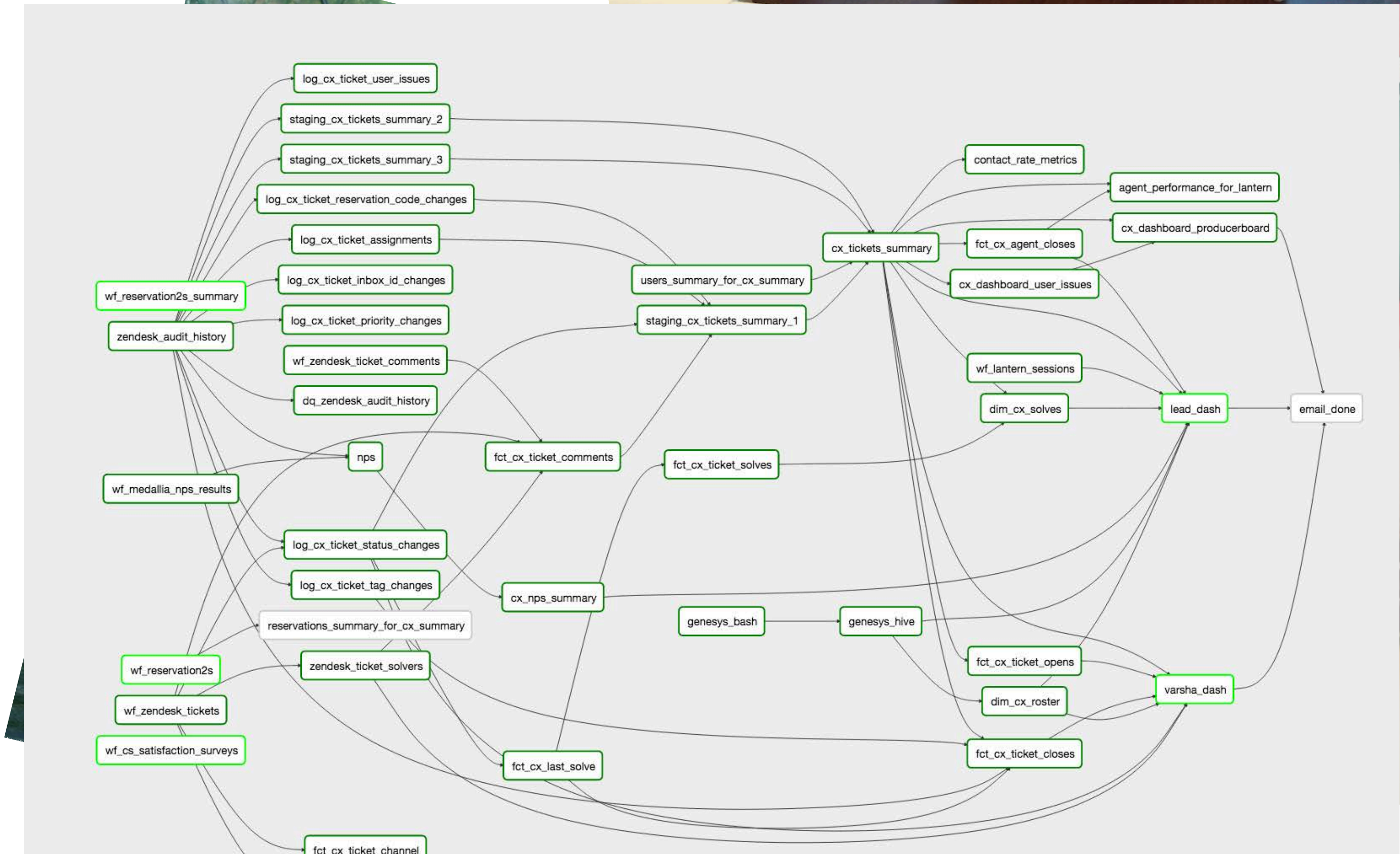
A typical data lake

inovex



Airflow: let‘s DAG!

inovex



› Workflow is (python) code

› Specify tasks & dependencies programmatically

Manages workflow metadata

Nice GUI ☺

Brief History

inovex

- developed at AirBnB by Maxime Beauchemin (former Yahoo / Facebook)
- open-sourced June 2015
- ~4200 commits, 81 releases, 332 contributors
- in Apache Incubator starting 2016/05
- used in several projects since 09/2015 ☺

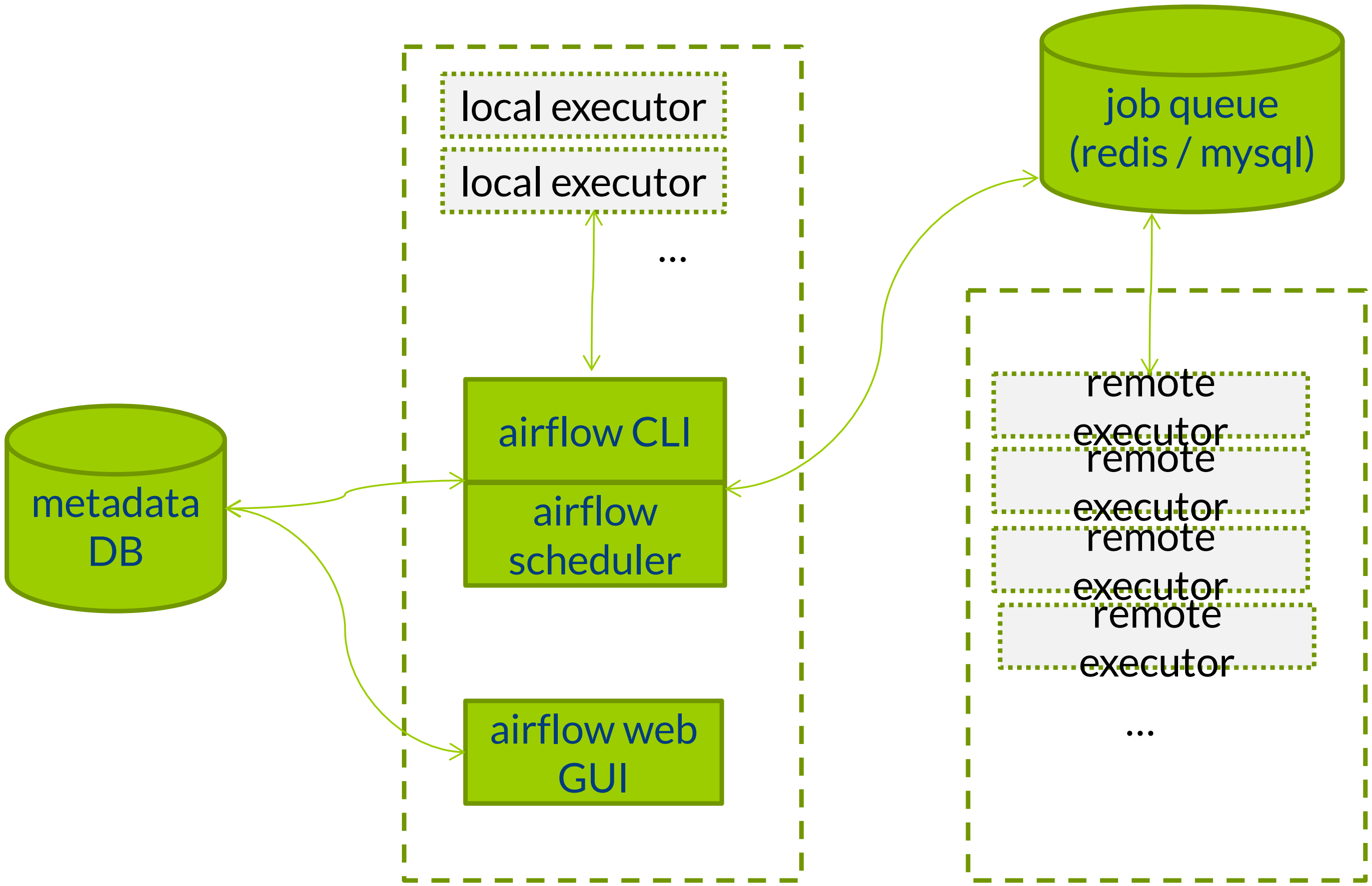
Gimme some code ..

inovex

```
from airflow import DAG

default_args = { 'owner': 'airflow',
  'retries': 2,
  ...
}

dag = DAG('tutorial', default_args=default_args)

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

t2 = HiveOperator( task_id='make_query',
  sql='select x from y where z group by k',
  dag=dag)

t2.set_upstream(t1)
```

Airflow: Architectural components

inovex



Basic Concepts

- **DAG**: graph of operator usages (=tasks)

- **Operator**: "Transformation" step
  - **Sensor**: Operator which polls with frequency / timeout (e.g. LocalFileSensor)
  - **Executor**: Trigger operation (e.g. HiveOperator, BashOperator, PigOperator, …)

- **Task**: Usage of Operator in DAG
  - **Task Instance**: run of a Task at a point in time

- **Hook**: Interface to external System (JDBCHook, HTTPHook, …)

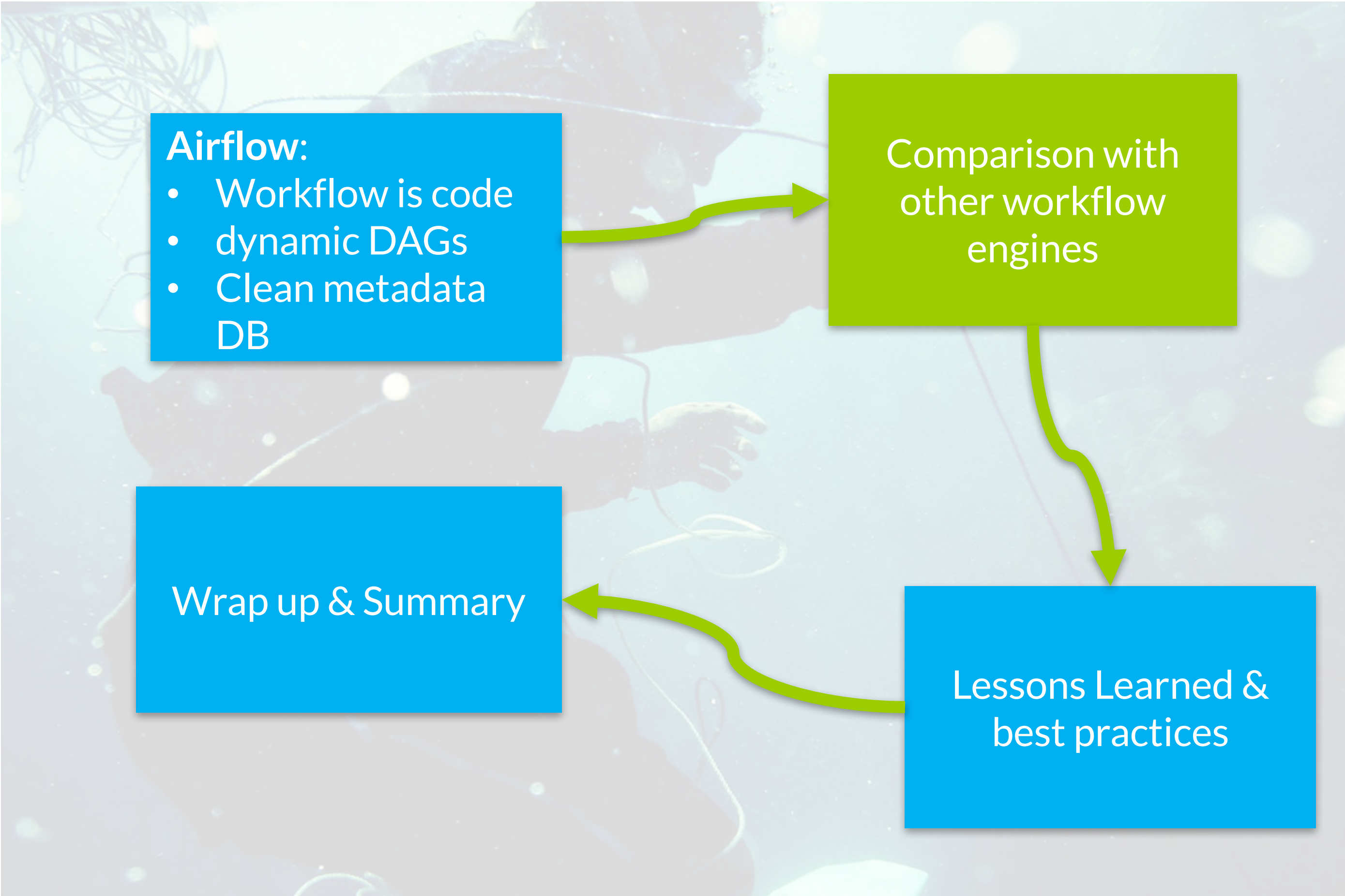
Most popular airflow CLI commands

command	does
`airflow initdb`	initialize metadata DB schema
`airflow test <dag> <task> <date>`	test task of a dag (shows command only)
`airflow run <dag> <task> <date>`	run task of a dag
`airflow backfill <dag> -s <start_date> -e <end_date>`	reload / backfill dag
`airflow clear <dag> -s <start_date> -e <end_date> -t <task_regex>`	clear state of dag / tasks
`airflow backfill <dag> -s <start_date> -e <end_date> -m true`	mark dag runs as success without running

Advanced Concepts

- **XCom**: send „messages“ between tasks
- **Trigger Rules**: specify handling for multiple upstream dependencies (e.g. all_success, one_success, ..)
- **Variables**: define key/value mappings in airflow metadata DB (value can be nested JSON as well)
- **Branching**: Define python function to choose which downstream path to follow
- **SubDAGs**: encapsulate repeating functionality

The Flow in Airflow



What else is out there?

inovex

	Oozie	Azkaban	Luigi	Airflow	Schedoscope
Language	Java	Java	Python	Python	Scala
WF specification	static (XML)	static (.job file)	static (task = extend class)	dynamic (task = instantiate operator)	dynamic
Schema / Change Management	no	no	no	no	yes
Test Framework	no	no	no	no	yes
WF trigger	data, time	time	data, time	data (sensors), time	goal

Other voices ..

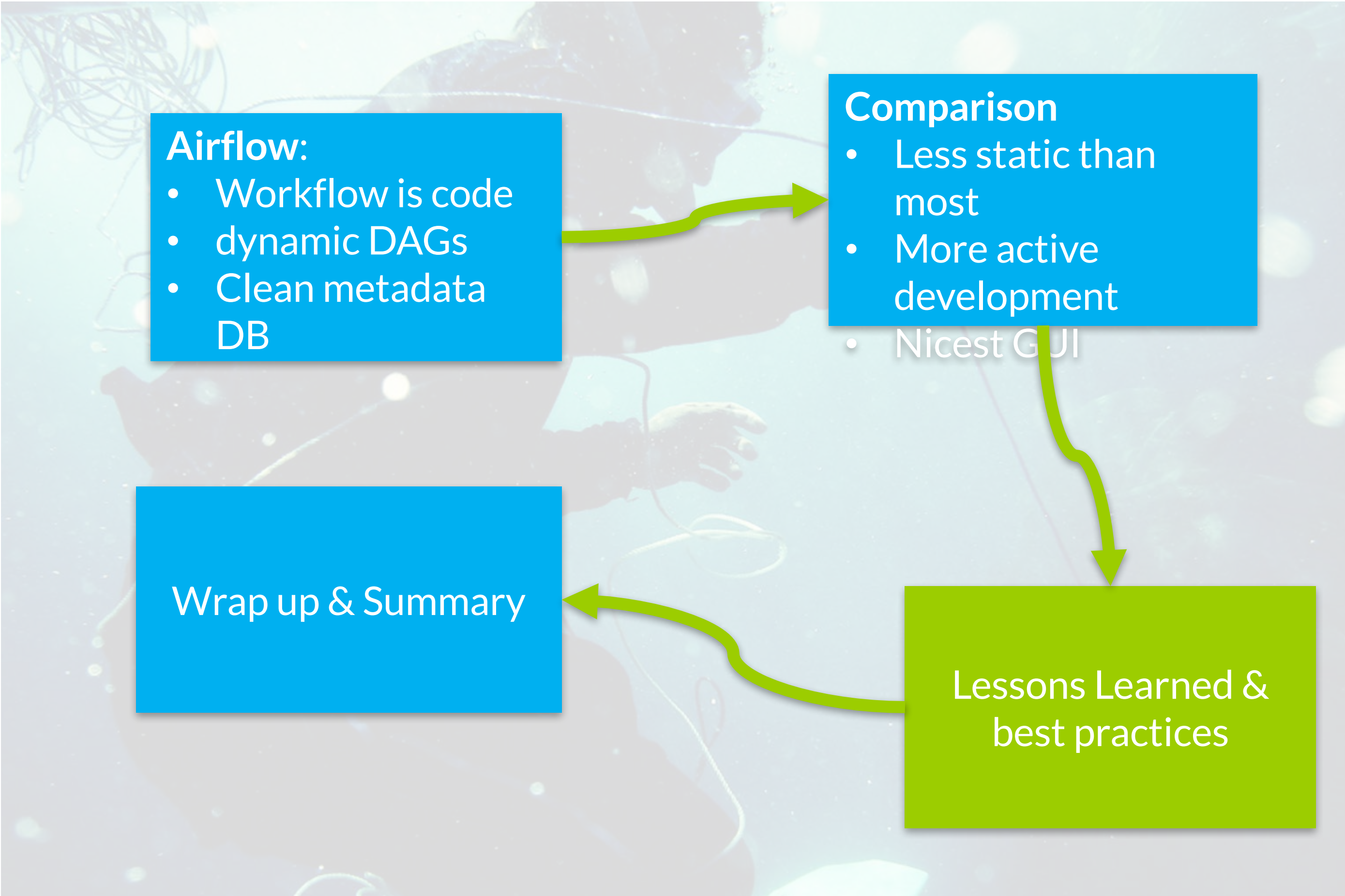
- Comparison (2016) of Airflow, Luigi, Pinball by Marton Trencseni (data science manager at Facebook) http://bytepawn.com/luigi-airflow-pinball.html :

  *„If I had to build a new ETL system today from scratch, **I would use Airflow**“*

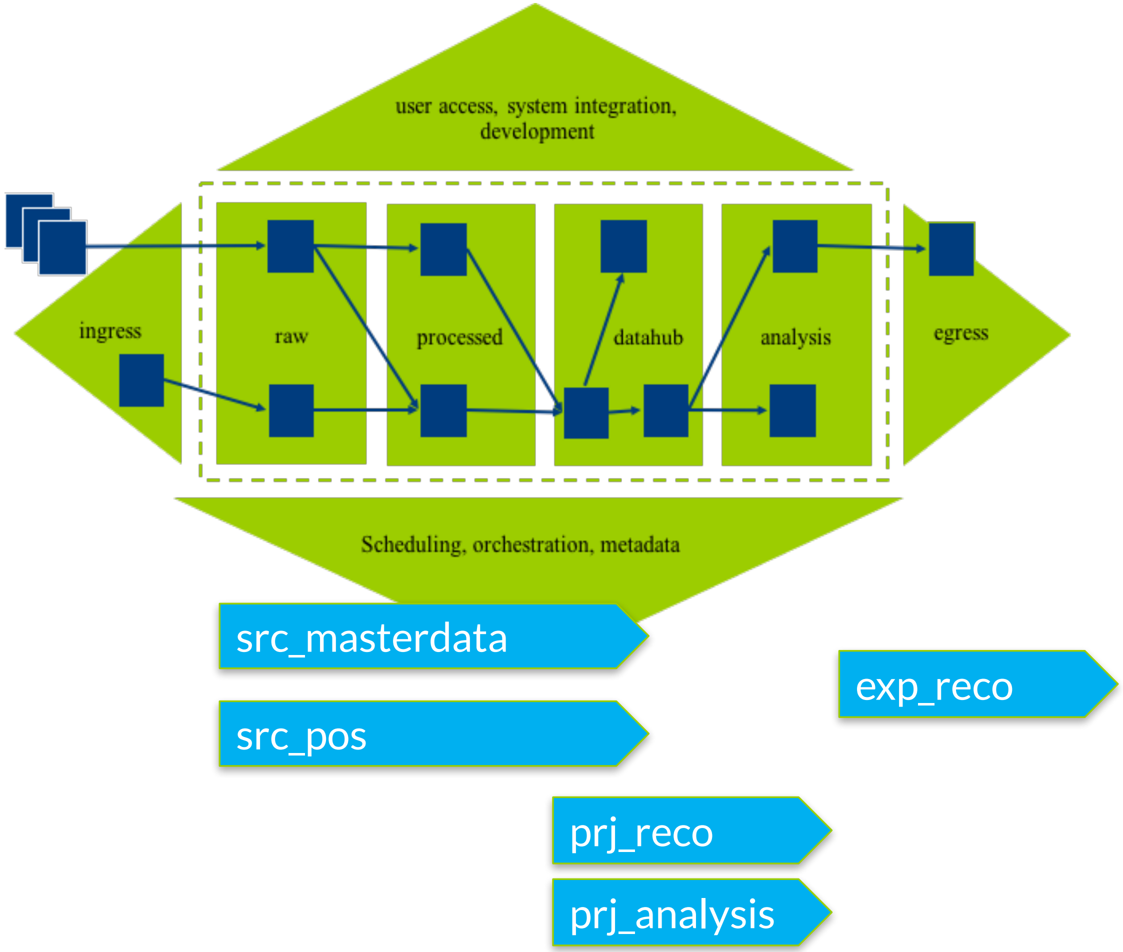
- Databricks Airflow integration https://databricks.com/blog/2017/07/19/integrating-apache-airflow-with-databricks.html :

  “We implemented an Airflow operator called DatabricksSubmitRunOperator, enabling a smoother integration between Airflow and Databricks“

The Flow in Airflow



Structuring / Cutting DAGs



- one DAG per **data source**
- one DAG per **„project“**
- one DAG per **data sink**

Structuring DAG resources

- $PROJECT_BASE
  - workflows
    - src_pos
      - hive
        - insert.sql
      - sqoop
        - trans.sqoop
    - src_pos.py
    - prj_reco
      - hive
        - compute.sql
      - pyspark
        - analyze.py
    - prj_reco.py

- keep code in template files
- for hive templates: use `hiveconf_jinja_translate`
- use template searchpath (see next slide)
- keep template files „airflow agnostic“ (if possible)

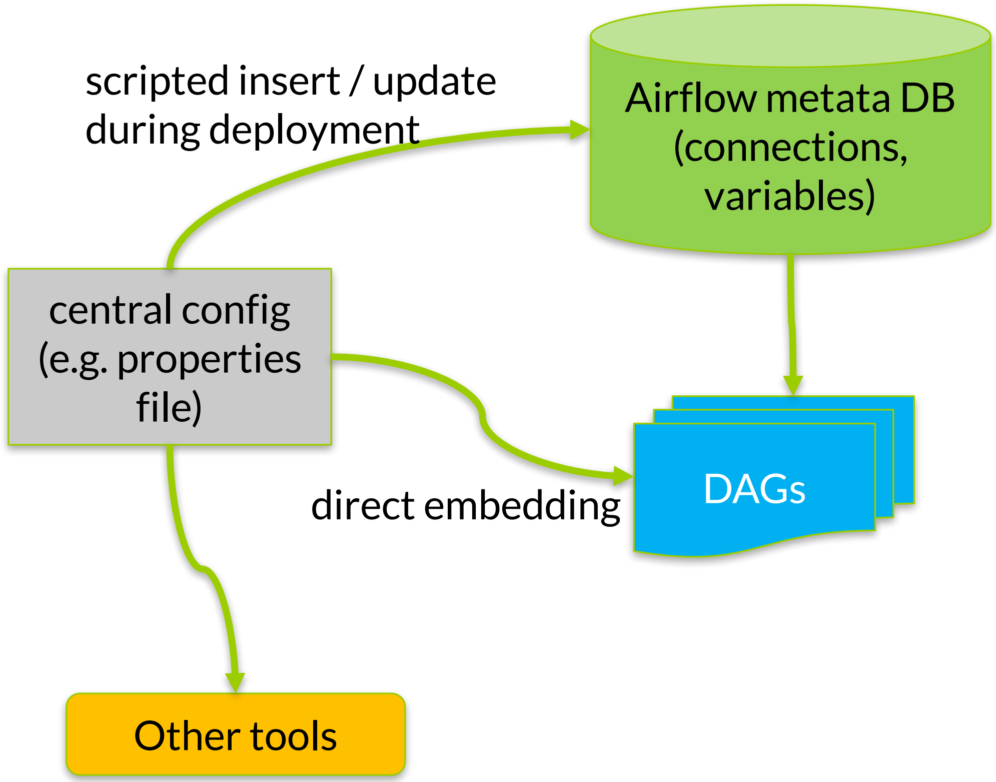
Structuring DAG resources (ctd.)

```
from airflow import DAG

default_args = { 'owner': 'airflow',
  'retries': 2,
  ...
}

dag = DAG('src_pos', default_args=default_args,
 template_searchpath=(
    '/base/workflows/src_pos/hive',
    '/base/workflows/src_pos/sqoop'))


insert = HiveOperator( task_id='insert',
  sql='insert.sql',
  dag=dag)

...
```

Configuration Management

inovex

- Built-in: definition of
  - Connections (e.g. to Hive, DBs, ..)
  - Variables (key/value)
- Often other (non-python) tools present
  - e.g. Realtime tools, ..
- Goal: **single source** of configuration
  - inject e.g. via “user_defined_macros“



Configuration Management (ctd.)

conf.py

```
ENV_NAME="prod"
PRJ_NAME="myprj"
...
```

insert.sql

```
INSERT INTO TABLE
${ENV_NAME}_${PRJ_NAME}_target
SELECT .. FROM ..
```

dag definition

```
from airflow import DAG
import conf
...

dag = DAG('src_pos',
default_args=default_args,
 user_defined_macros=conf.__dict__)


insert = HiveOperator(
task_id='insert',
   sql='insert.sql',
   dag=dag)
...
```

Develop & Deploy Workflows

inovex

file-based DB

PERSONAL

- data engineers **or data scientists**
- test via „airflow test“

LocalExecutor

GIT

RDBMS

INTEGRATION

- perfomance tests, integration tests
- fixes

LocalExecutor

jenkins

RDBMS

PRODUCTIVE

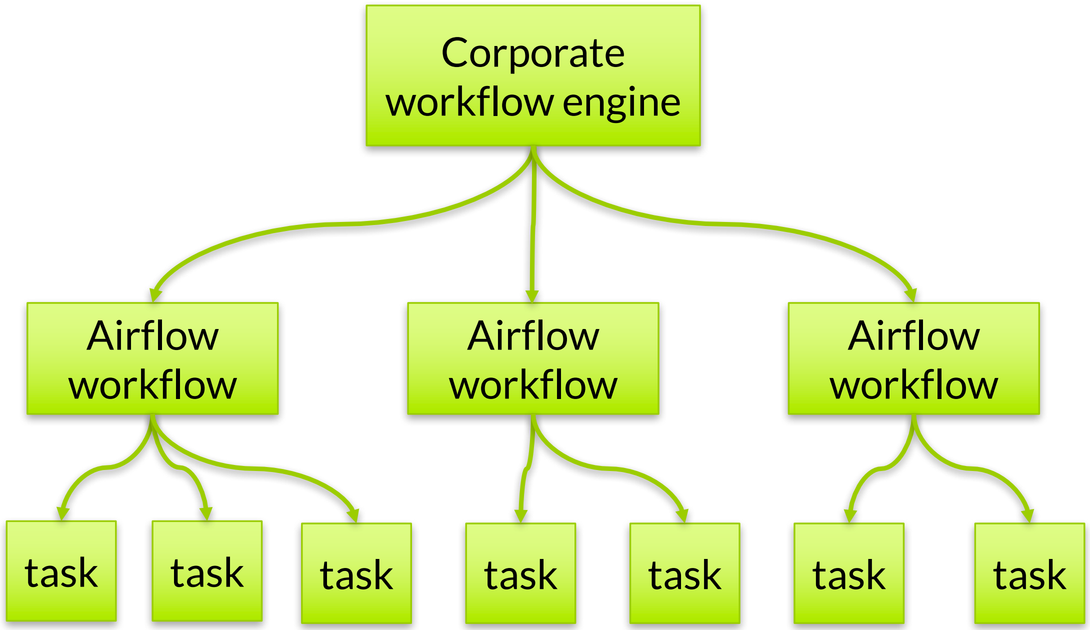
- monitoring
- backfilling of failed jobs

LocalExecutor / CeleryExecutor

Integrating with the Enterprise

inovex

- Often existing workflow / scheduling tools present (e.g. control M, ...)
- Existing integration in e.g. ticketing systems
- Idea: “Hierarchy“ of engines:
  - Enterprise engine: scheduling, coarse-grained
  - Airflow: workflow, fine-grained



Writing Plugins & Extensions

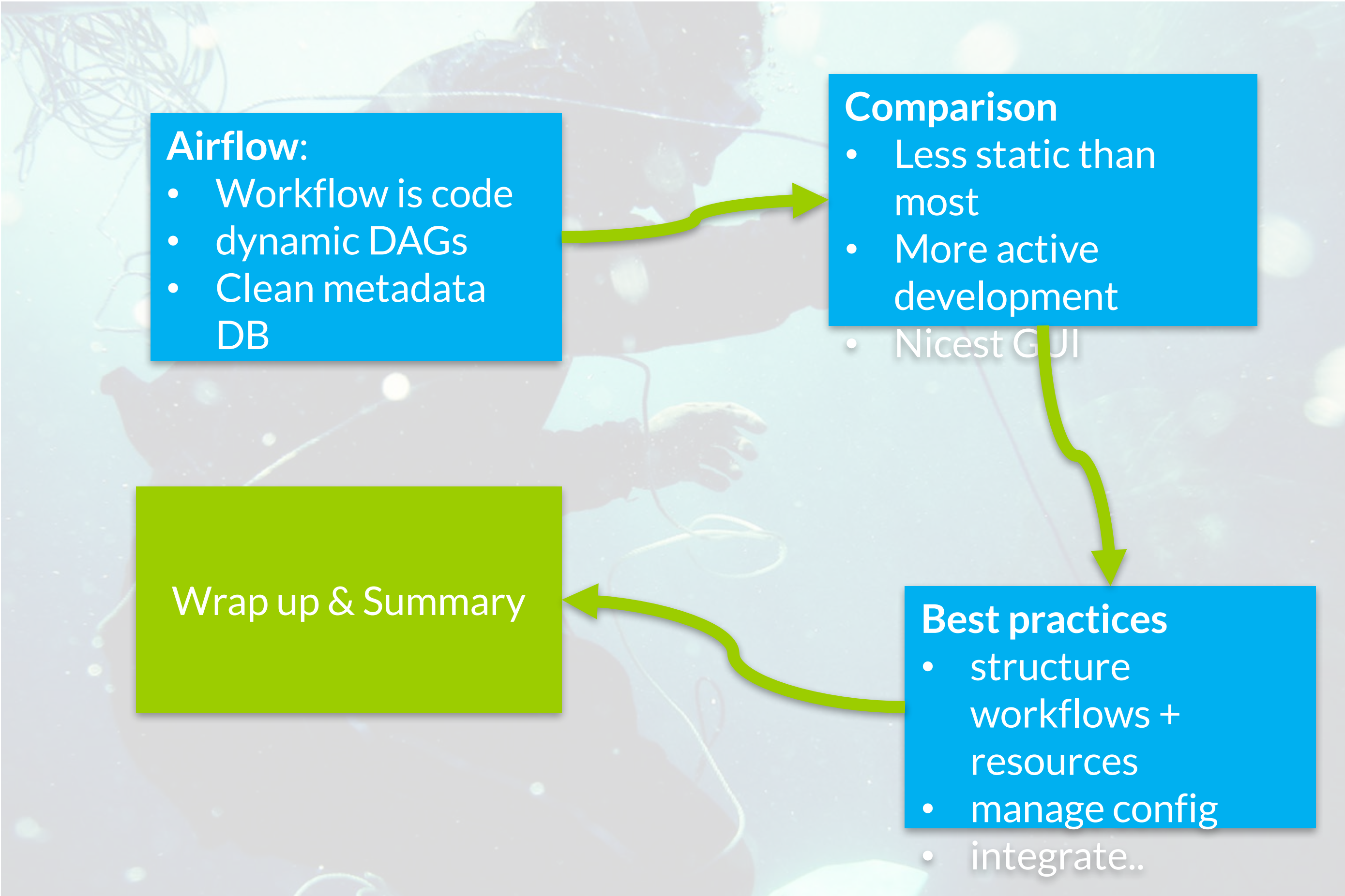
- Extension points:
  - **operators**
  - **hooks**
  - executors
  - macros
  - UI adaption (views, links)
- Easy – but also needed ☺
- Start from existing classes, adapt

- Developed so far:
  - SSHFileSensor
  - HiveServer2Operator (you have to!)
  - SparkSQLOperator
  - SparkOperator
  - ...
- integrate via **`airflow_plugin_directory`**

Configs, Gotchas, ..

config, topic	explanation
airflow.cfg: **`parallelism`**	max nr. of task instances to run in parallel (per metadata DB / installation)
airflow.cfg: **`dag_concurrency`**	how many parallel tasks are allowed **per dag** (attention: further tasks will **not be scheduled**!)
LDAP integration	works, but problems with LDAPs who implement another „memberOf" attribute (fixed in 1.9, see AIRFLOW-1095)
Kerberos	kerberos relogin process („airflow kerberos") broken; workaround = own BashOperator who does kinit with a keytab
Hive integration via impyla	Problems with 1.8.2 (thrift-sasl version mismatch); solution = downgrade thrift_sasl to 0.2.1 (instead of 0.3.0)
...	fore sure more to come ☺

The Flow in Airflow



Summary

What‘s the flow ..

- Airflow = workflow is code
- Programmatically define DAGs of Operators
- Integrates seamlessly into “pythonic“ data science stack
- Easily extensible (which is needed)
- Lowers the gap between data science + engineering
- Clean management of workflow metadata (state, runs, ...)
- Under active development
- Fun to use, & real-world project proven ☺

Vielen Dank

Dr. Dominik Benz
dominik.benz@inovex.de

inovex GmbH
Park Plaza
Ludwig-Erhard-Allee 6
76131 Karlsruhe