Airflow

Семинар 7

Batch Pattern & Online Pattern

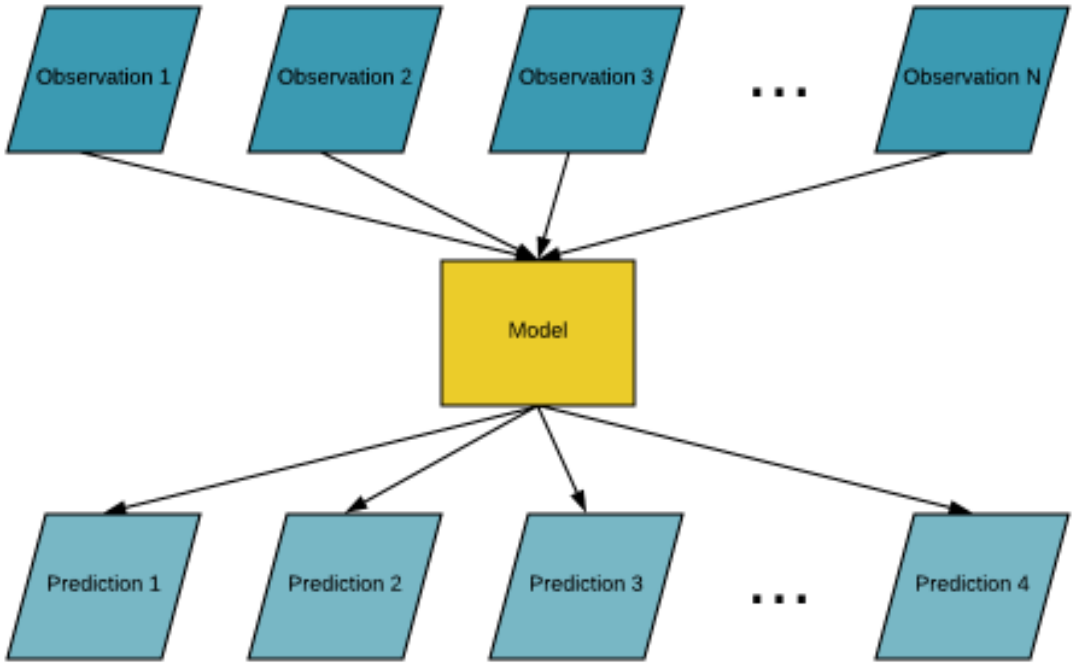


Figure 1. Batch Inference

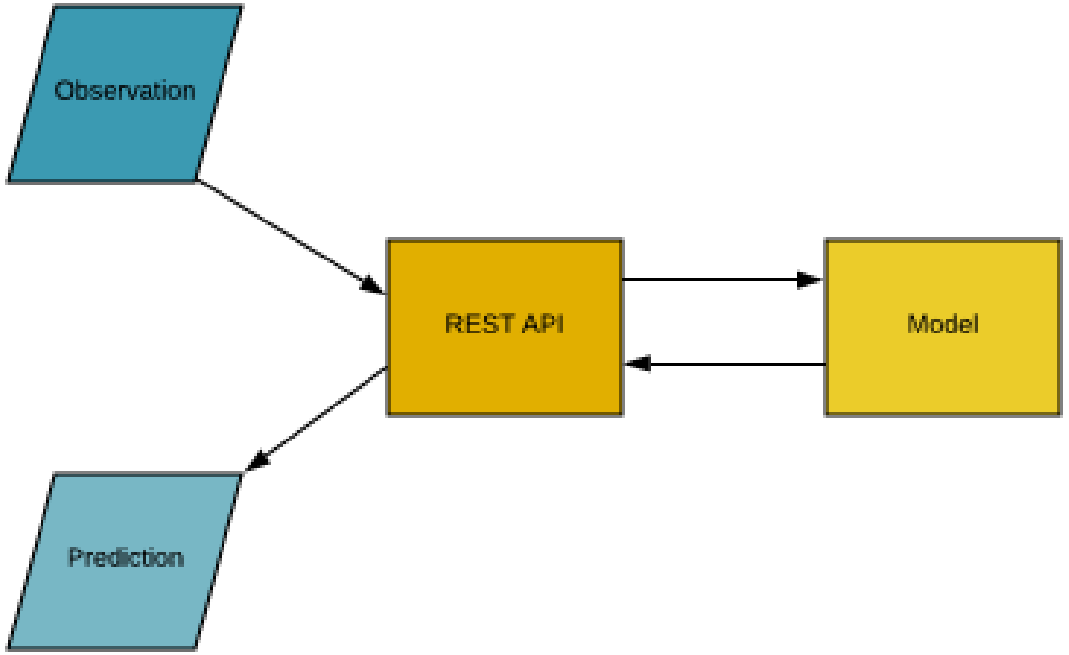


Figure 2. Online Inference

Cron

```
# * * * * *  command to execute
# ┬ ┬ ┬ ┬ ┬
# │ │ │ │ │
# │ │ │ │ └──── day of week (0 - 7)
# │ │ │ └─────────── month (1 - 12)
# │ │ └──────────────── day of month (1 - 31)
# │ └──────────────────── hour (0 - 23)
# └───────────────────────── min (0 - 59)
```

Cron

https://crontab.guru

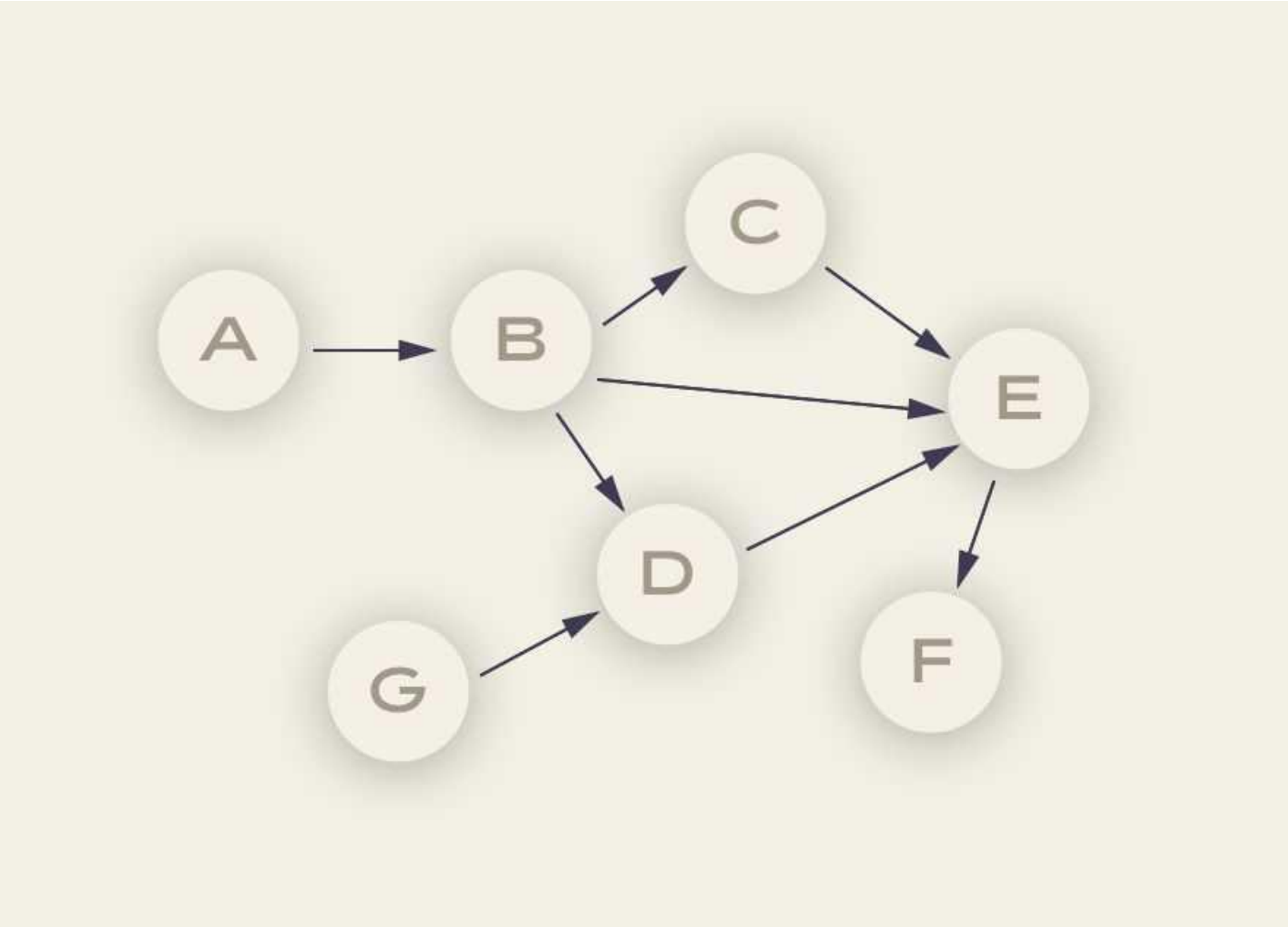
```
francis@variety_village: ~
  GNU nano 2.9.3                    /etc/crontab

# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/$
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/$
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/$
#

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line
```

Оркестратор данных

Фреймворк, который умеет:

1. Планирование задач (когда запустить)
2. Управление зависимостями (что за чем запустить)
3. Репроцессинг (легко перезапускать упавшее)
4. Мониторинг (рассылка уведомлений в случае падения)

DAG



Apache Airflow

https://github.com/apache/airflow

Создание DAG

```python
import datetime

from airflow import DAG
from airflow.operators.empty import EmptyOperator

with DAG(
    dag_id="my_dag_name",
    start_date=datetime.datetime(2021, 1, 1),
    schedule="@daily",
):
    EmptyOperator(task_id="task")
```

```python
import datetime

from airflow import DAG
from airflow.operators.empty import EmptyOperator

my_dag = DAG(
    dag_id="my_dag_name",
    start_date=datetime.datetime(2021, 1, 1),
    schedule="@daily",
)
EmptyOperator(task_id="task", dag=my_dag)
```

```python
import datetime

from airflow.decorators import dag
from airflow.operators.empty import EmptyOperator


@dag(start_date=datetime.datetime(2021, 1, 1), schedule="@daily")
def generate_dag():
    EmptyOperator(task_id="task")


generate_dag()
```

Интерфейс DAG

Airflow DAGs Cluster Activity Datasets Security Browse Admin Docs 19:42 UTC AA

DAGs

All 60 | Active 2 | Paused 58 | Running 0 | Failed 0 | Filter DAGs by tag | Search DAGs | Auto-refresh

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
01_example myexample	airflow		0 0 * * *		2024-10-17, 00:00:00			
conditional_dataset_and_time_based_timetable dataset-time-based-timetable	airflow		Dataset or 0 1 * * 3		2024-10-16, 01:00:00			
consume_1_and_2_with_dataset_expressions	airflow		Dataset		0 of 2 datasets updated			
consume_1_or_2_with_dataset_expressions	airflow		Dataset		0 of 2 datasets updated			
consume_1_or_both_2_and_3_with_dataset_expressions	airflow		Dataset		0 of 3 datasets updated			
dataset_consumes_1 consumes dataset-scheduled	airflow		Dataset		On s3://dag1/output_1.txt			
dataset_consumes_1_and_2 consumes dataset-scheduled	airflow		Dataset		0 of 2 datasets updated			
dataset_consumes_1_never_scheduled consumes dataset-scheduled	airflow		Dataset		0 of 2 datasets updated			

Airflow Task

Базовая единица DAG.

Может представлен в виде:

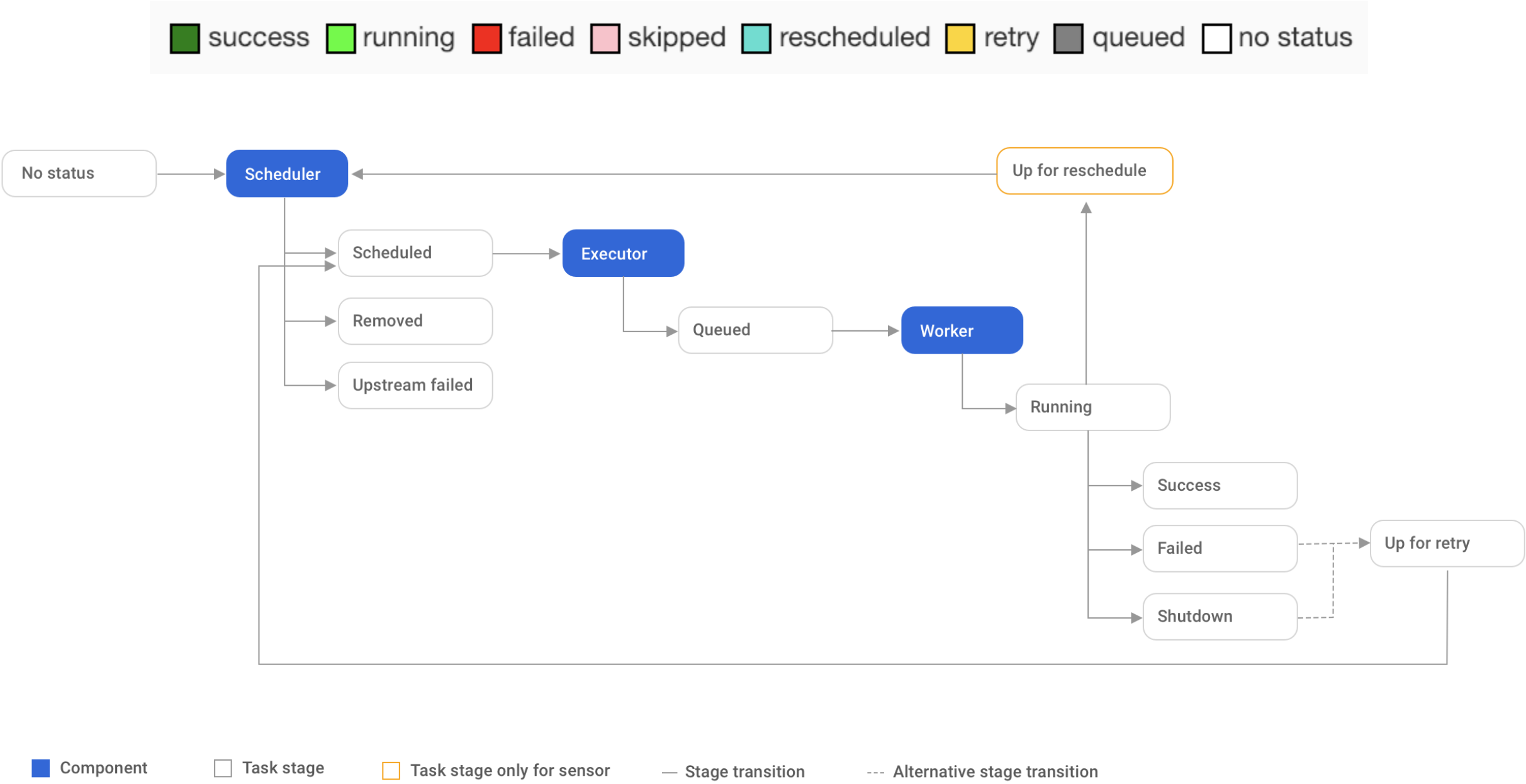
1. Оператор
2. Сенсор
3. TaskFlow декоратор @task

```
first_task >> second_task >> [third_task, fourth_task]
```

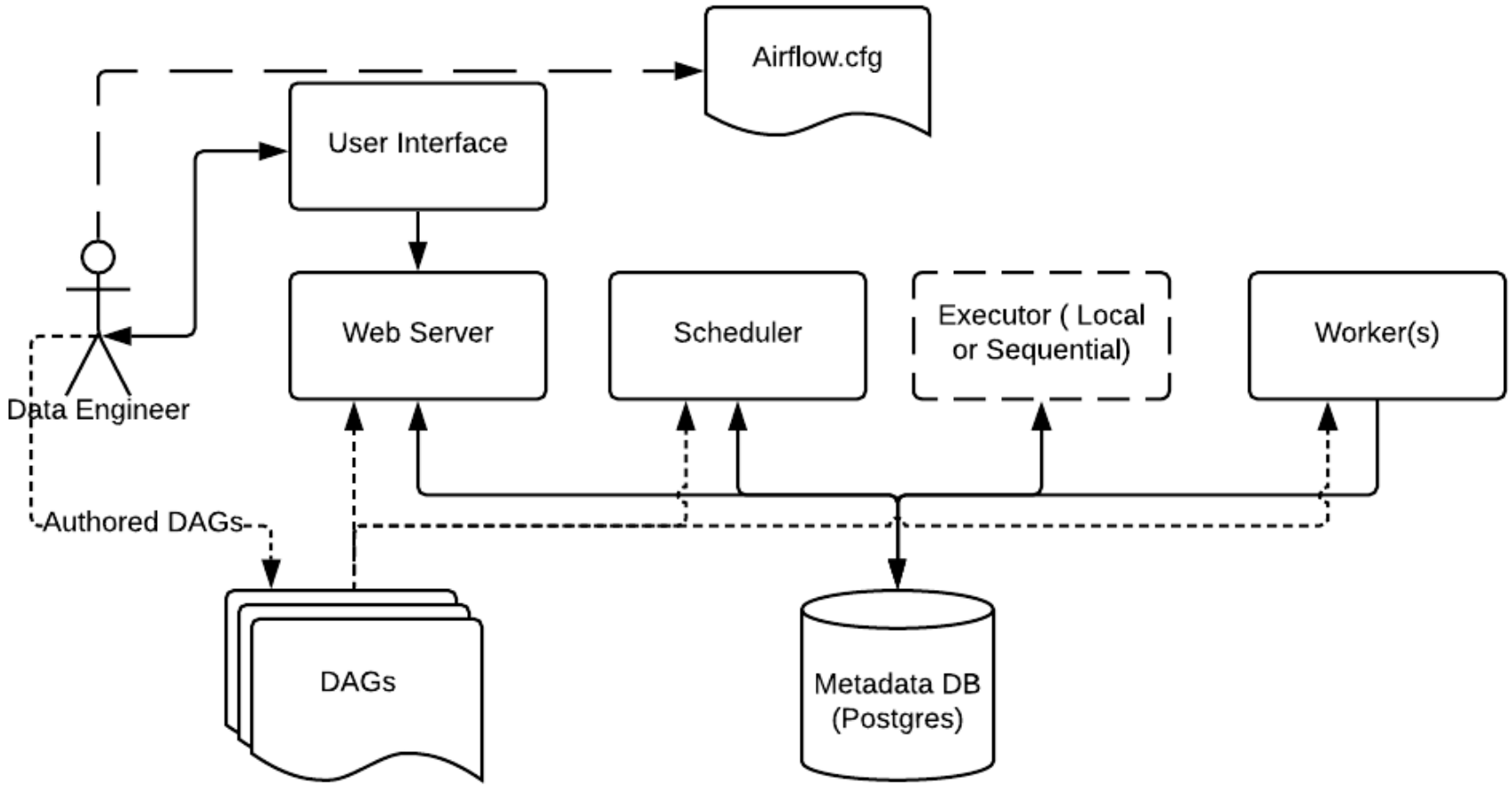
```
first_task.set_downstream(second_task)
third_task.set_upstream(second_task)
```

Airflow Task Состояния

- `none` : The Task has not yet been queued for execution (its dependencies are not yet met)
- `scheduled` : The scheduler has determined the Task's dependencies are met and it should run
- `queued` : The task has been assigned to an Executor and is awaiting a worker
- `running` : The task is running on a worker (or on a local/synchronous executor)
- `success` : The task finished running without errors
- `restarting` : The task was externally requested to restart when it was running
- `failed` : The task had an error during execution and failed to run
- `skipped` : The task was skipped due to branching, LatestOnly, or similar.
- `upstream_failed` : An upstream task failed and the Trigger Rule says we needed it
- `up_for_retry` : The task failed, but has retry attempts left and will be rescheduled.
- `up_for_reschedule` : The task is a Sensor that is in `reschedule` mode
- `deferred` : The task has been deferred to a trigger
- `removed` : The task has vanished from the DAG since the run started


success
running
failed
skipped
rescheduled
retry
queued
no status
No status
Scheduler
Up for reschedule
Scheduled
Executor
Removed
Queued
Worker
Upstream failed
Running
Success
Failed
Up for retry
Shutdown
Component
Task stage
Task stage only for sensor
Stage transition
Alternative stage transition

Airflow Concepts



Airflow Task Operator

```python
task_bash_op = BashOperator(
    task_id="show_files",
    bash_command="ls /opt/airflow/",
)

email = EmailOperator(
    task_id='send_email',
    to="email@gmail.com",
    subject="Update complete",
    html_content='Hello'
)
```

```python
def my_custom_function(input_value):
    print(f'Square of input value is: {input_value ** 2}')

task_python_op = PythonOperator(
    task_id="python_operator",
    python_callable=my_custom_function,
    op_kwargs={"input_value": 15},
)
```

Airflow Task Operator

Возможные операторы:

- SparkSubmitOperator
- PostgresOperator
- HiveOperator
- DockerOperator
- SSHOperator
- TelegramOperator

```python
from airflow.models.baseoperator import BaseOperator


class HelloOperator(BaseOperator):
    def __init__(self, name: str, **kwargs) -> None:
        super().__init__(**kwargs)
        self.name = name

    def execute(self, context):
        message = f"Hello {self.name}"
        print(message)
        return message
```

Providers

Active providers

- Airbyte
- Alibaba
- Amazon
- Apache Beam
- Apache Cassandra
- Apache Drill
- Apache Druid
- Apache Flink
- Apache HDFS
- Apache Hive
- Apache Iceberg
- Apache Impala
- Apache Kafka
- Apache Kylin
- Apache Livy
- Apache Pig
- Apache Pinot
- Apache Spark
- Apprise
- ArangoDB
- Asana
- Atlassian Jira
- Celery
- Cloudant
- CNCF Kubernetes
- Cohere
- Common Compat
- Common IO
- Common SQL
- Databricks
- Datadog
- dbt Cloud
- Dingding
- Discord
- Docker
- Elasticsearch
- Exasol
- FAB (Flask-AppBuilder)
- Facebook
- File Transfer Protocol (FTP)
- GitHub
- Google
- gRPC
- Hashicorp
- Hypertext Transfer Protocol (HTTP)
- IBM Cloudant
- Influx DB
- Internet Message Access Protocol (IMAP)
- Java Database Connectivity (JDBC)
- Jenkins
- Microsoft Azure
- Microsoft SQL Server (MSSQL)
- Microsoft PowerShell Remoting Protocol (PSRP)
- Microsoft Windows Remote Management (WinRM)
- MongoDB
- MySQL
- Neo4j
- ODBC
- OpenAI
- OpenFaaS
- OpenLineage
- Open Search
- Opsgenie
- Oracle
- Pagerduty
- Papermill
- PgVector
- Pinecone
- PostgreSQL
- Presto
- Qdrant
- Redis
- Salesforce
- Samba
- Segment
- Sendgrid
- SFTP
- Singularity
- Slack
- SMTP
- Snowflake
- SQLite
- SSH
- Tableau
- Telegram
- Teradata
- Trino
- Vertica
- Weaviate
- Yandex
- YDB
- Zendesk

XCom

List XComs

Search

« < 1 2 > » Page size Actions

Record Count: 125

Key	Value	Timestamp	Dag Id	Task Id	Run Id	Map Index	Execution Date
return_value	plugins	2024-10-18, 09:04:03	01_example	show_files	scheduled__2024-10-17T00:00:00+00:00		2024-10-17, 00:00:00
return_value	Whatever you return gets printed in the logs	2024-10-18, 09:04:03	02_python_operator	print_the_context	scheduled__2024-10-17T00:00:00+00:00		2024-10-17, 00:00:00
return_value	less_0.5	2024-10-18, 09:04:03	03_branching	branching	scheduled__2024-10-17T00:00:00+00:00		2024-10-17, 00:00:00
return_value	Number is less than 0.5	2024-10-18, 09:04:09	03_branching	less_0.5	scheduled__2024-10-17T00:00:00+00:00		2024-10-17, 00:00:00
skipmixin_key	{'followed': ['less_0.5']}	2024-10-18, 09:04:03	03_branching	branching	scheduled__2024-10-17T00:00:00+00:00		2024-10-17, 00:00:00
return_value	Number is more than 0.5	2024-10-17, 21:00:28	03_branching	more_0.5	scheduled__2024-10-14T00:00:00+00:00		2024-10-14, 00:00:00
skipmixin_key	{'followed': ['more_0.5']}	2024-10-17, 21:00:20	03_branching	branching	scheduled__2024-10-14T00:00:00+00:00		2024-10-14, 00:00:00
return_value	more_0.5	2024-10-17, 21:00:20	03_branching	branching	scheduled__2024-10-14T00:00:00+00:00		2024-10-14, 00:00:00
return_value	less_0.5	2024-10-17, 21:00:21	03_branching	branching	scheduled__2024-10-13T00:00:00+00:00		2024-10-13, 00:00:00
return_value	Number is less than 0.5	2024-10-17, 21:00:27	03_branching	less_0.5	scheduled__2024-10-13T00:00:00+00:00		2024-10-13, 00:00:00
skipmixin_key	{'followed': ['less_0.5']}	2024-10-17, 21:00:21	03_branching	branching	scheduled__2024-10-13T00:00:00+00:00		2024-10-13, 00:00:00
skipmixin_key	{'followed': ['less_0.5']}	2024-10-17, 21:00:20	03_branching	branching	scheduled__2024-10-12T00:00:00+00:00		2024-10-12, 00:00:00
return_value	Number is less than 0.5	2024-10-17, 21:00:27	03_branching	less_0.5	scheduled__2024-10-12T00:00:00+00:00		2024-10-12, 00:00:00
return_value	less_0.5	2024-10-17, 21:00:20	03_branching	branching	scheduled__2024-10-12T00:00:00+00:00		2024-10-12, 00:00:00

Templating

Variable	Type	Description
`{{ data_interval_start }}`	pendulum.DateTime	Start of the data interval. Added in version 2.2.
`{{ data_interval_end }}`	pendulum.DateTime	End of the data interval. Added in version 2.2.
`{{ logical_date }}`	pendulum.DateTime	A date-time that logically identifies the current DAG run. This value does Use `data_interval_start` and `data_interval_end` instead if you want such as to get a slice of rows from the database based on timestamps.
`{{ ds }}`	str	The DAG run's logical date as `YYYY-MM-DD` . Same as `{{ logical_date | ds }}` .
`{{ ds_nodash }}`	str	Same as `{{ logical_date | ds_nodash }}` .
`{{ exception }}`	None | str | Exception KeyboardInterrupt	Error occurred while running task instance.
`{{ ts }}`	str	Same as `{{ logical_date | ts }}` . Example: `2018-01-01T00:00:00+00:00` .
`{{ ts_nodash_with_tz }}`	str	Same as `{{ logical_date | ts_nodash_with_tz }}` . Example: `20180101T000000+0000` .
`{{ ts_nodash }}`	str	Same as `{{ logical_date | ts_nodash }}` . Example: `20180101T000000` .
`{{ prev_data_interval_start_success }}`	pendulum.DateTime | `None`	Start of the data interval of the prior successful `DagRun` . Added in version 2.2.
`{{ prev_data_interval_end_success }}`	pendulum.DateTime | `None`	End of the data interval of the prior successful `DagRun` . Added in version 2.2.
`{{ prev_start_date_success }}`	pendulum.DateTime | `None`	Start date from prior successful `DagRun` (if available).
`{{ prev_end_date_success }}`	pendulum.DateTime | `None`	End date from prior successful `DagRun` (if available).
`{{ inlets }}`	list	List of inlets declared on the task.

Sensors

Примеры:

1. Появление файла
2. Появление файла на HDFS
3. Код запроса 200 от http запроса к сервису
4. Выполнение SQL-запроса
5. Создание таблицы

airflow.sensors

Sensors.

```
sphinx-autoapi-skip
```

Submodules

- airflow.sensors.base
- airflow.sensors.bash
- airflow.sensors.date_time
- airflow.sensors.external_task
- airflow.sensors.filesystem
- airflow.sensors.python
- airflow.sensors.time_delta
- airflow.sensors.time_sensor
- airflow.sensors.weekday

Домашнее задание №3

Реализовать

1. Пайплайн обучения модели в виде Airflow DAG
1.1 Генерация данных
1.2 Препроцессинг
1.3 Обучение модели

2. Пайплайн батч инференса обученной модели
2.1 Реализовать сенсор на файл /opt/data/new_data.csv
2.2 Если этот файл появляется, то необходимо его считать, получить предсказания модели, сохранить локально и удалить исходный файл /opt/data/new_data.csv