
AWS
re:Invent
NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

API304

Amazon MWAA at scale

John Jackson (he/him)

Principal Product Manager, Amazon MWAA
AWS

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

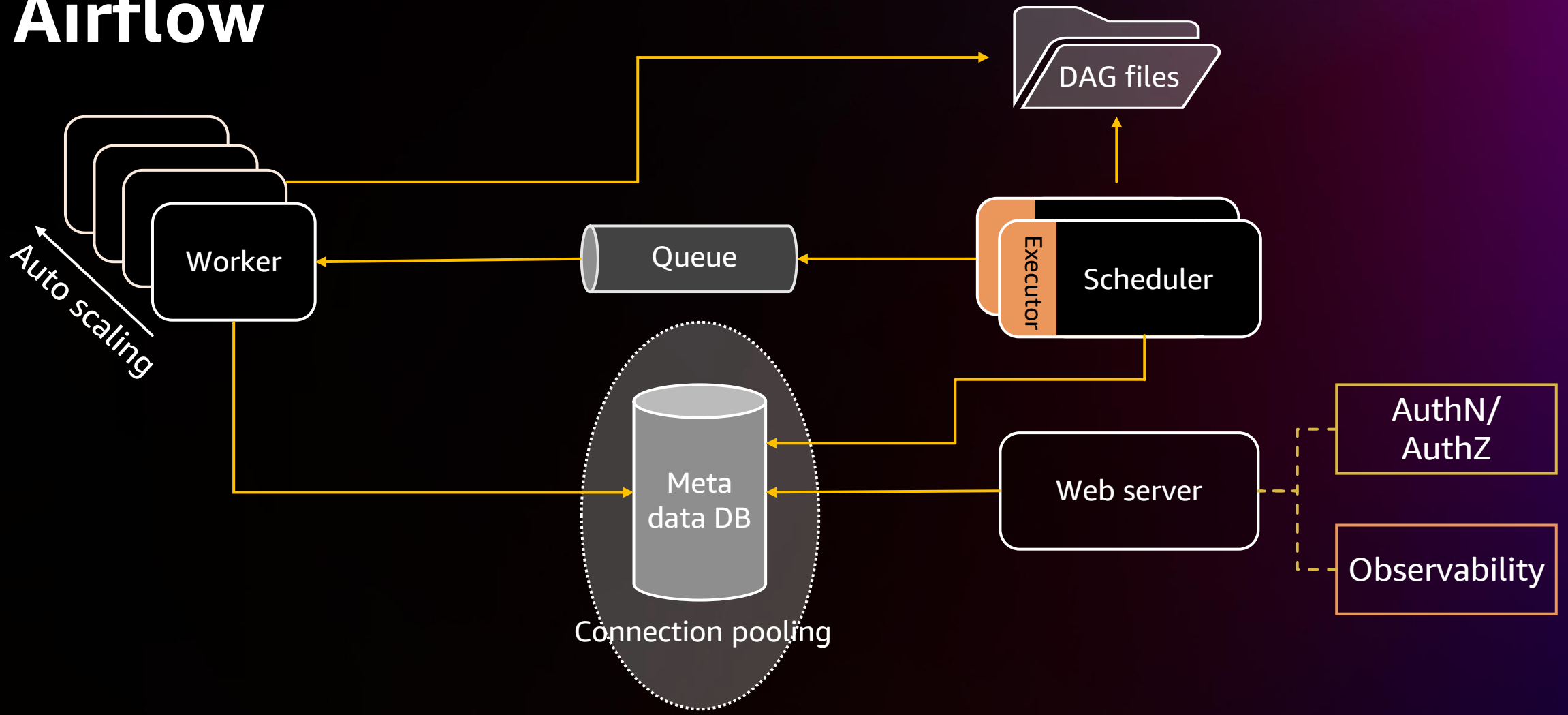
Agenda

- Introduction to Amazon MWAA deployment architecture
- Scheduler and worker processing
- Techniques to scale
- Demo

This chalk talk assumes audience is familiar with basic concepts of running DAGs in Amazon MWAA

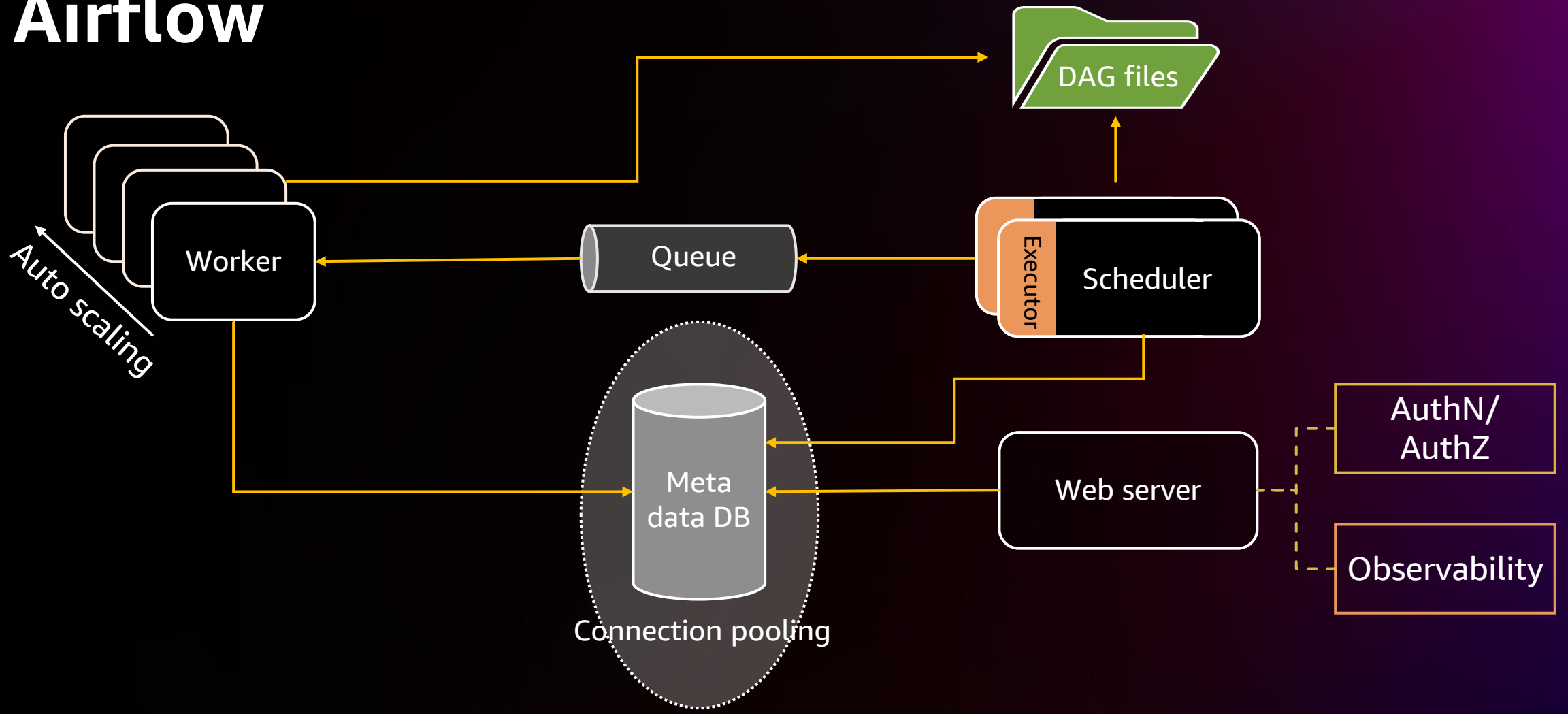
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

MWAA deployment architecture of Apache Airflow



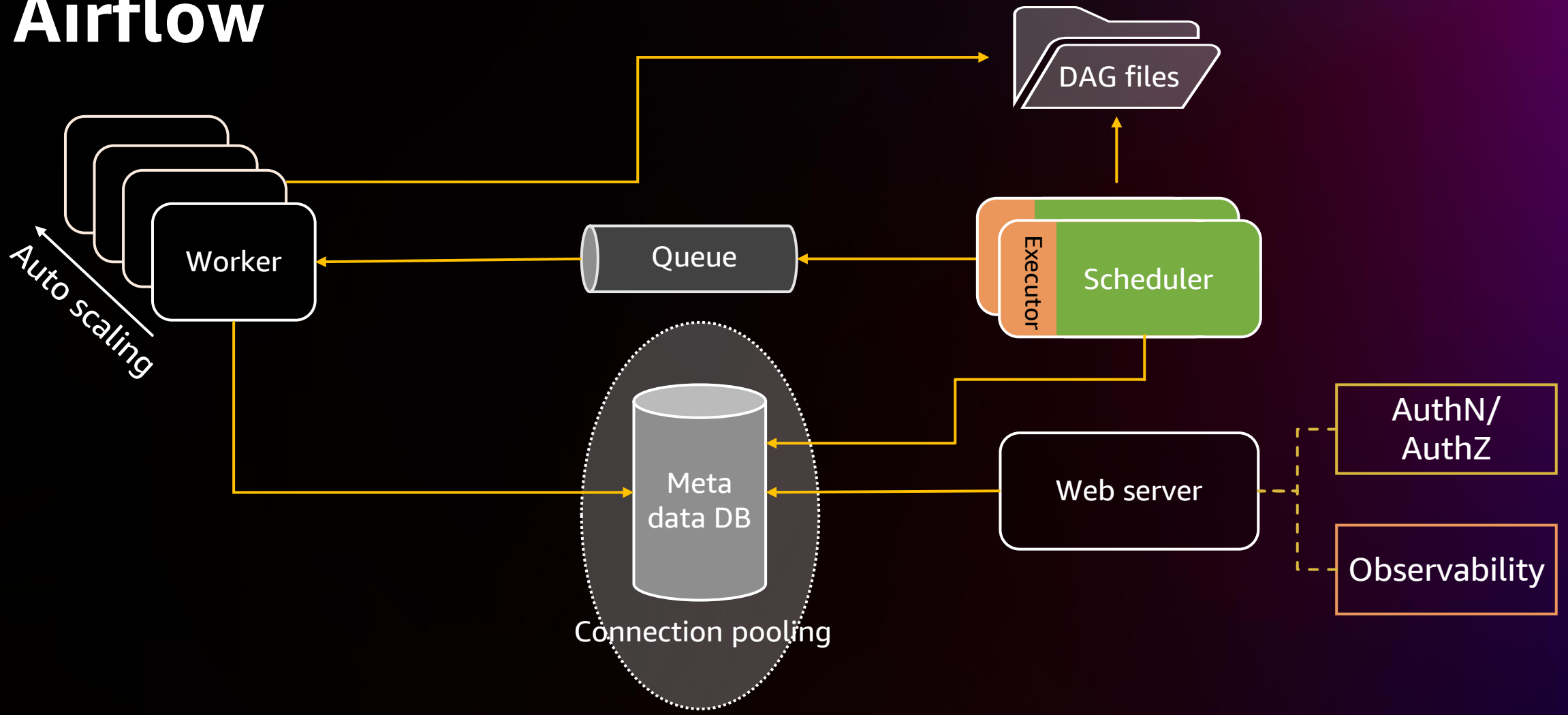
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

MWAA deployment architecture of Apache Airflow



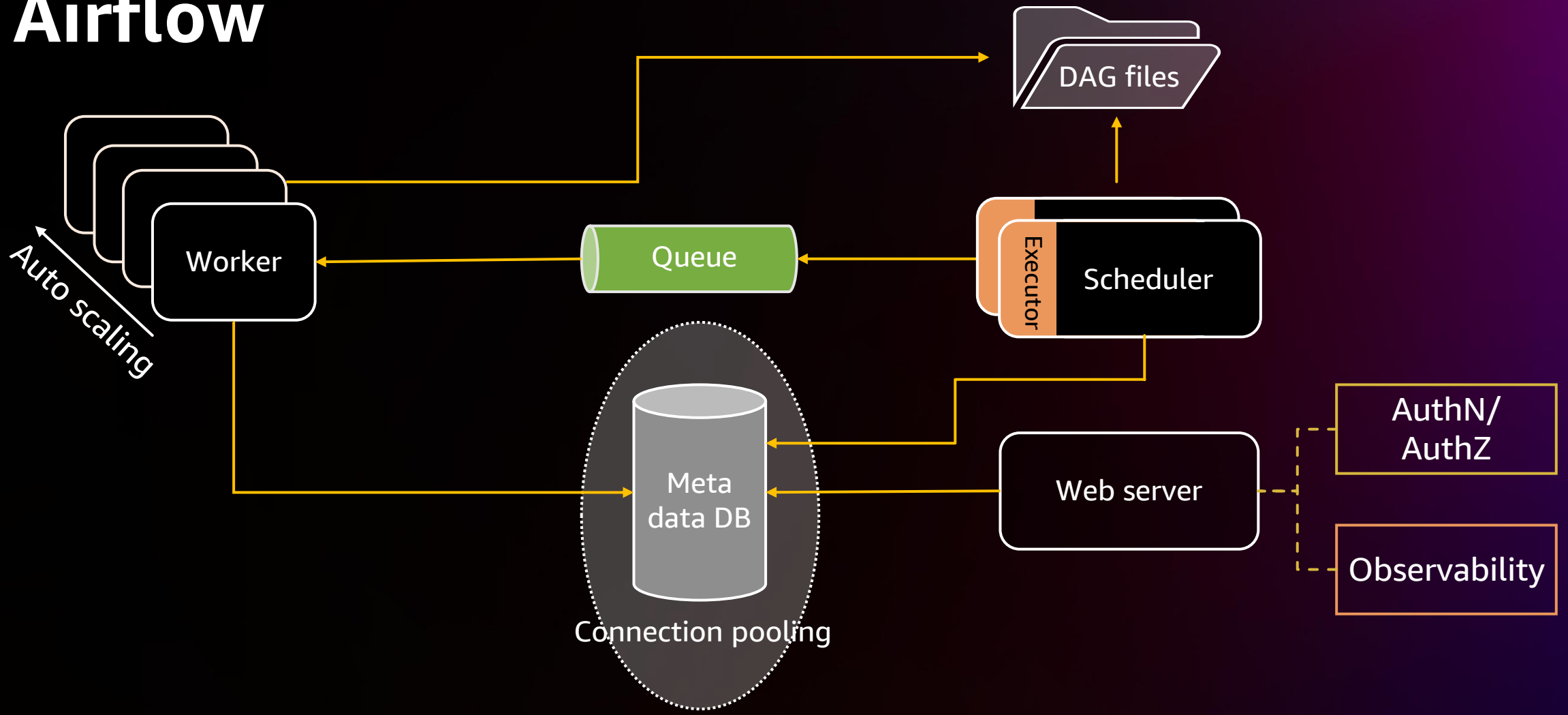
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

MWAA deployment architecture of Apache Airflow



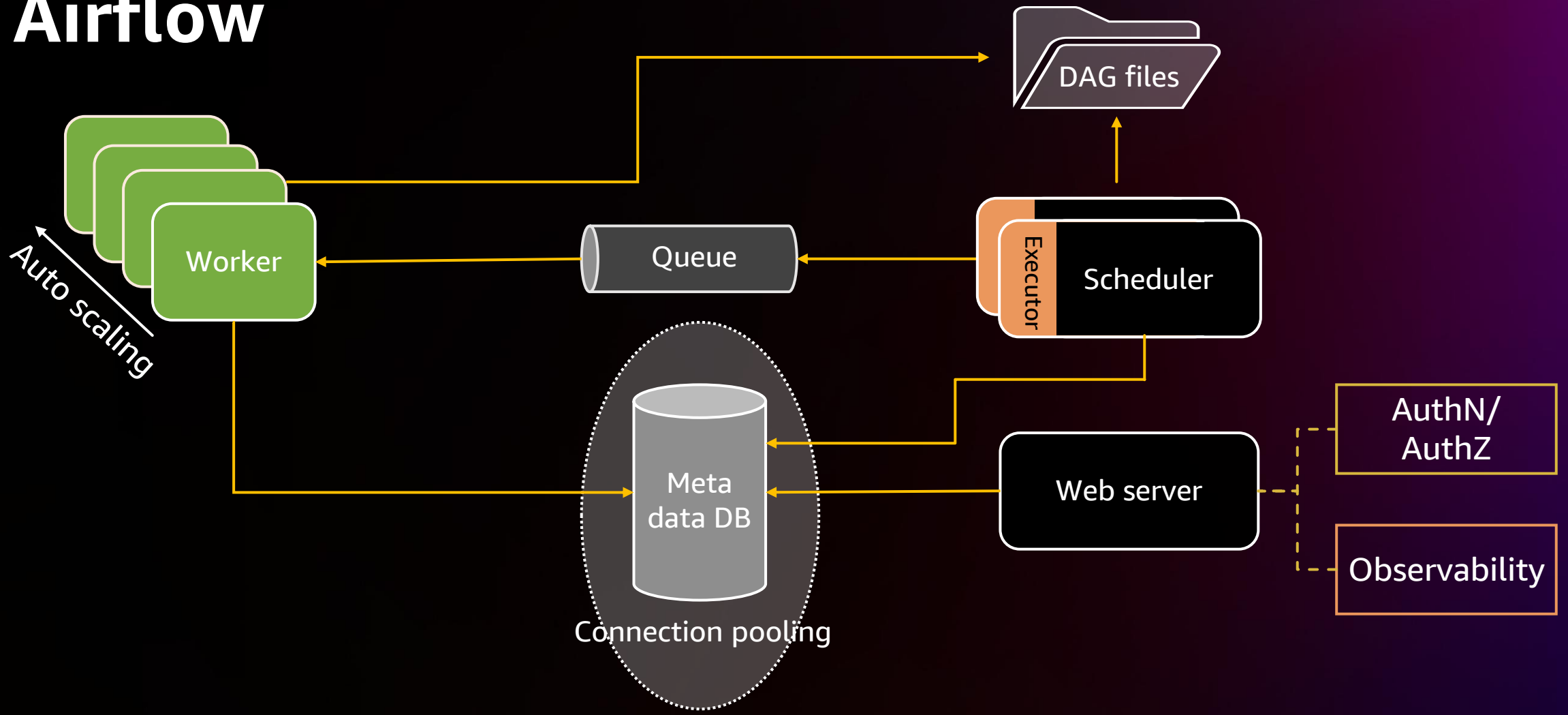
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

MWAA deployment architecture of Apache Airflow



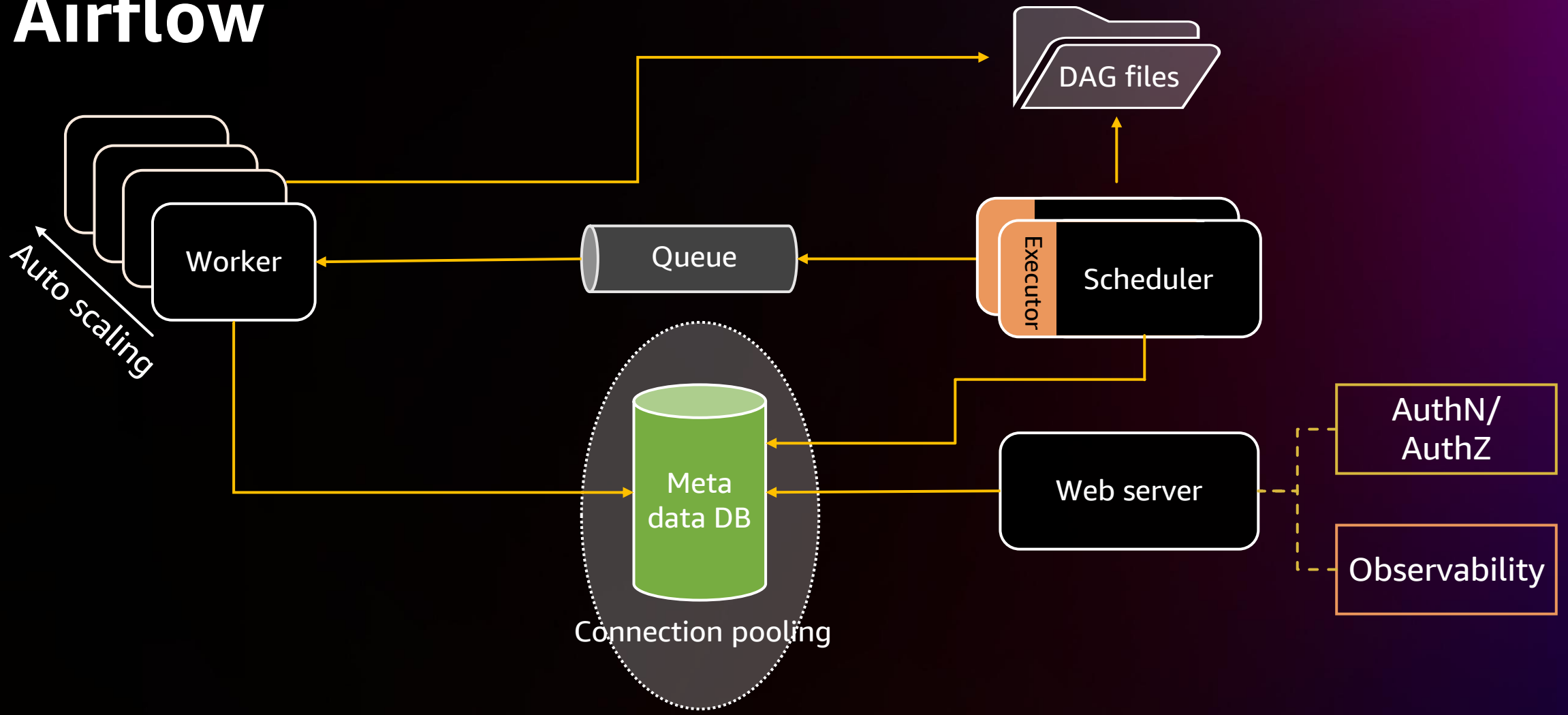
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

MWAA deployment architecture of Apache Airflow



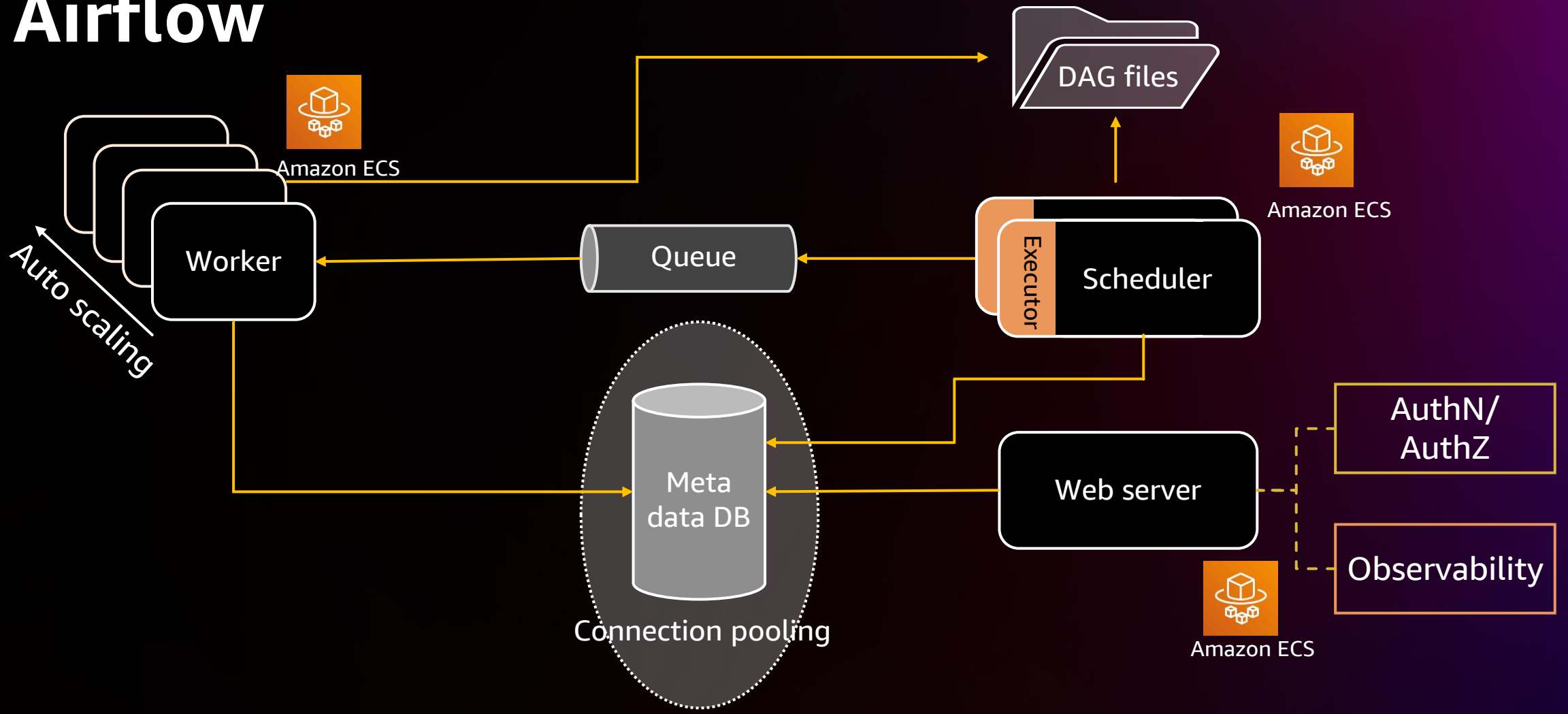
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

MWAA deployment architecture of Apache Airflow



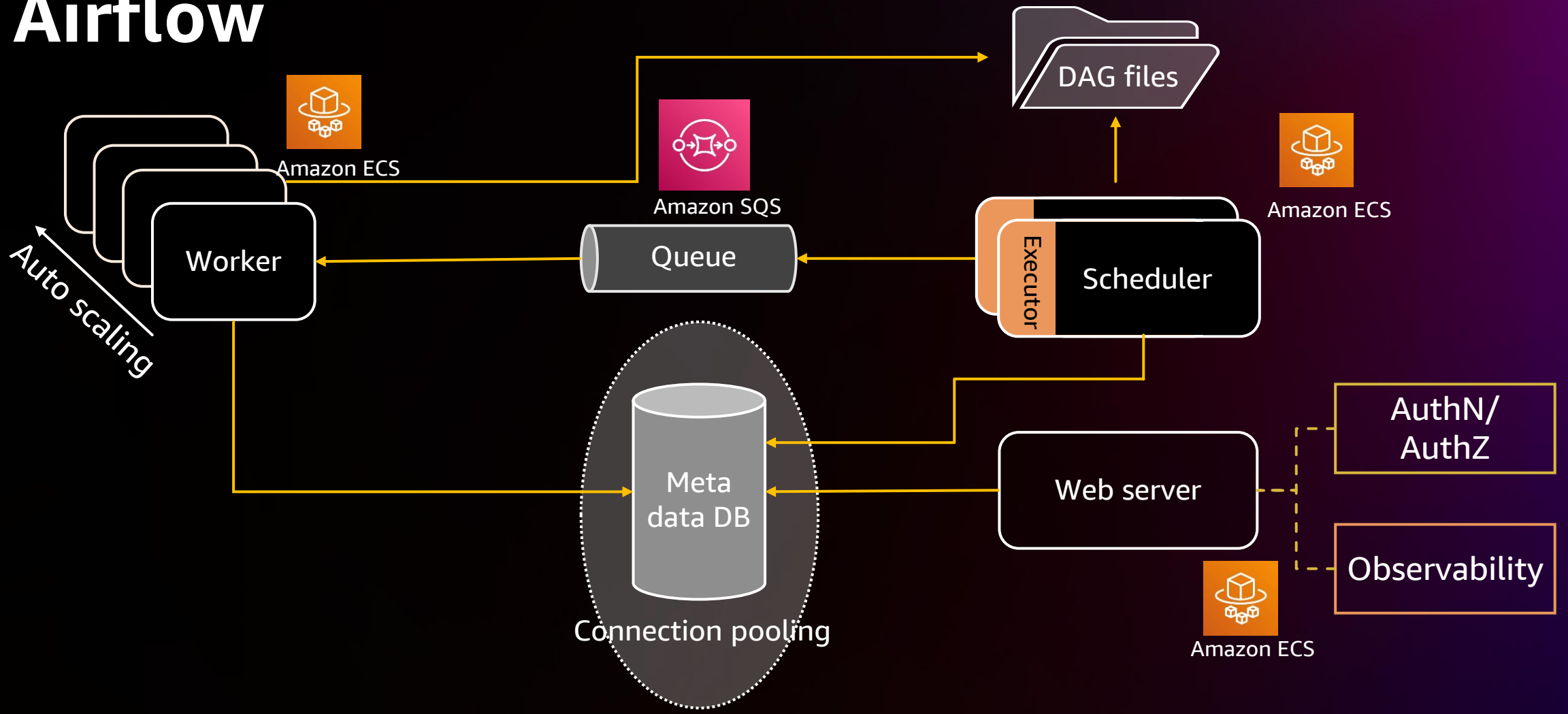
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

MWAA deployment architecture of Apache Airflow



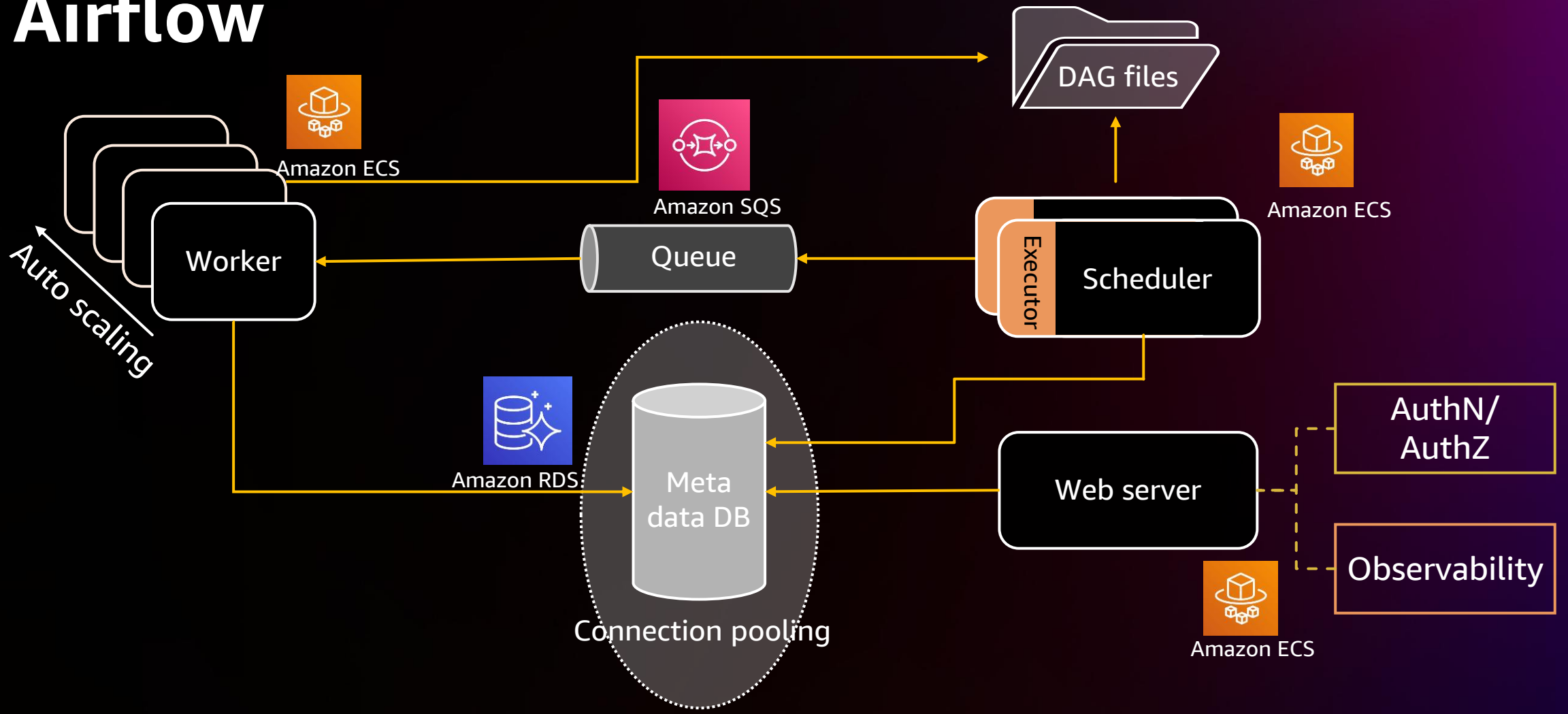
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

MWAA deployment architecture of Apache Airflow



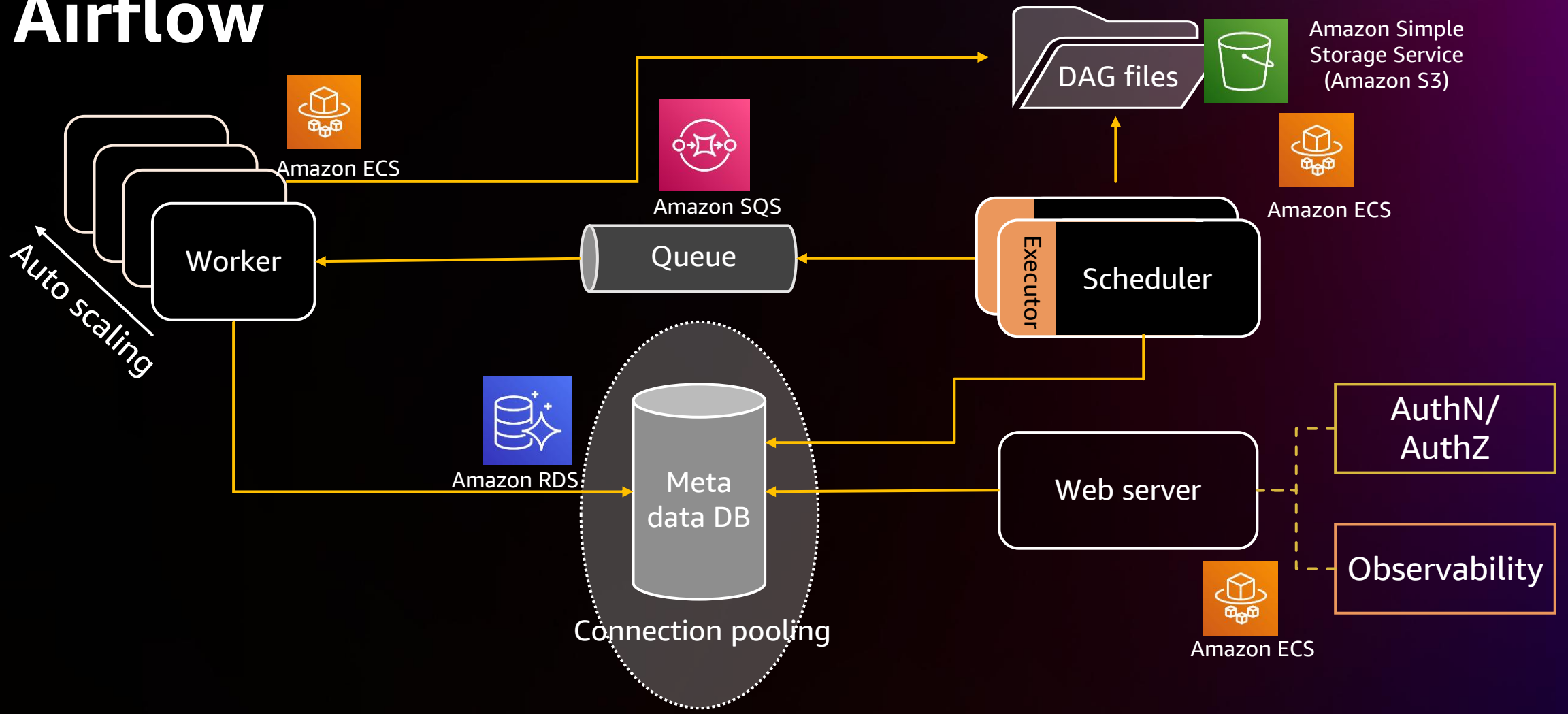
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

MWAA deployment architecture of Apache Airflow



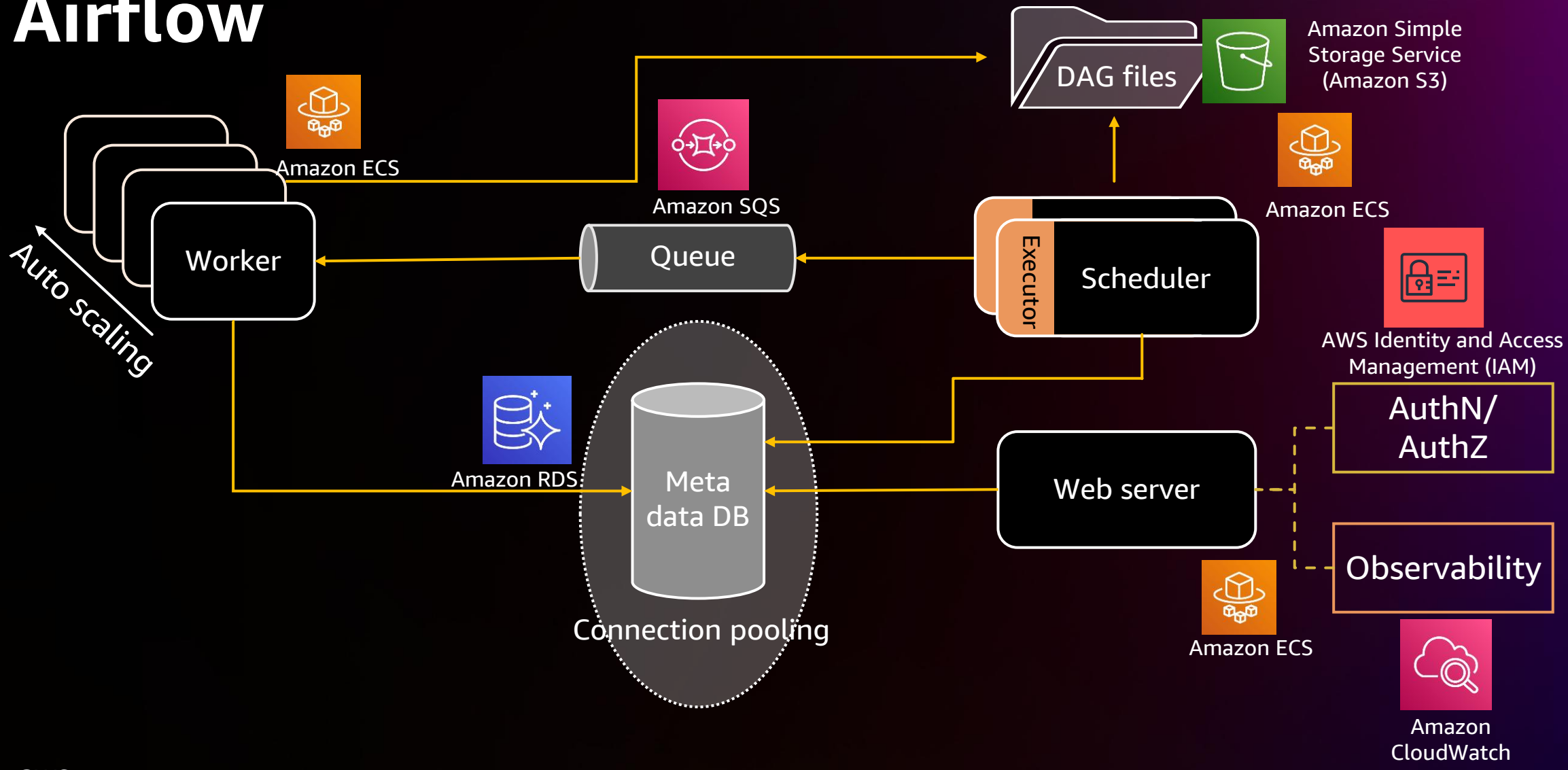
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

MWAA deployment architecture of Apache Airflow



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

MWAA deployment architecture of Apache Airflow



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon MWAA environment class

Class	Scheduler vCPU	Worker vCPU	Web server vCPU	Worker RAM (GB)	Task concurrency number
mw1.small	1	1	1	2	5
mw1.medium	2	2	1	4	10
mw1.large	4	4	2	8	20

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon MWAA environment class

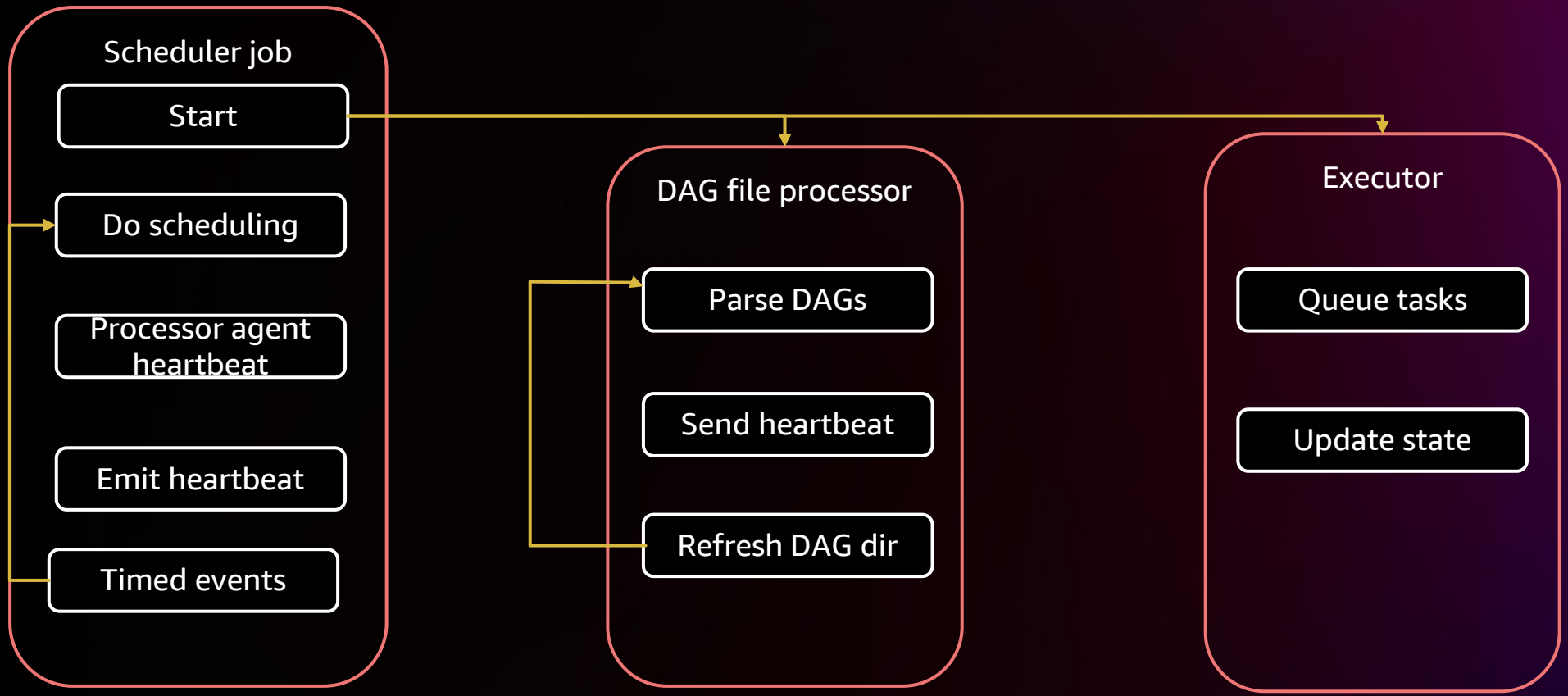
Class	Scheduler vCPU	Worker vCPU	Web server vCPU	Worker RAM (GB)	Task concurrency number
mw1.small	1	1	1	2	5
mw1.medium	2	2	1	4	10
mw1.large	4	4	2	8	20

Large environment, min workers = 9, max workers = 15

Min concurrency = 9 * 20 = 180
Max concurrency = 15 * 20 = 300

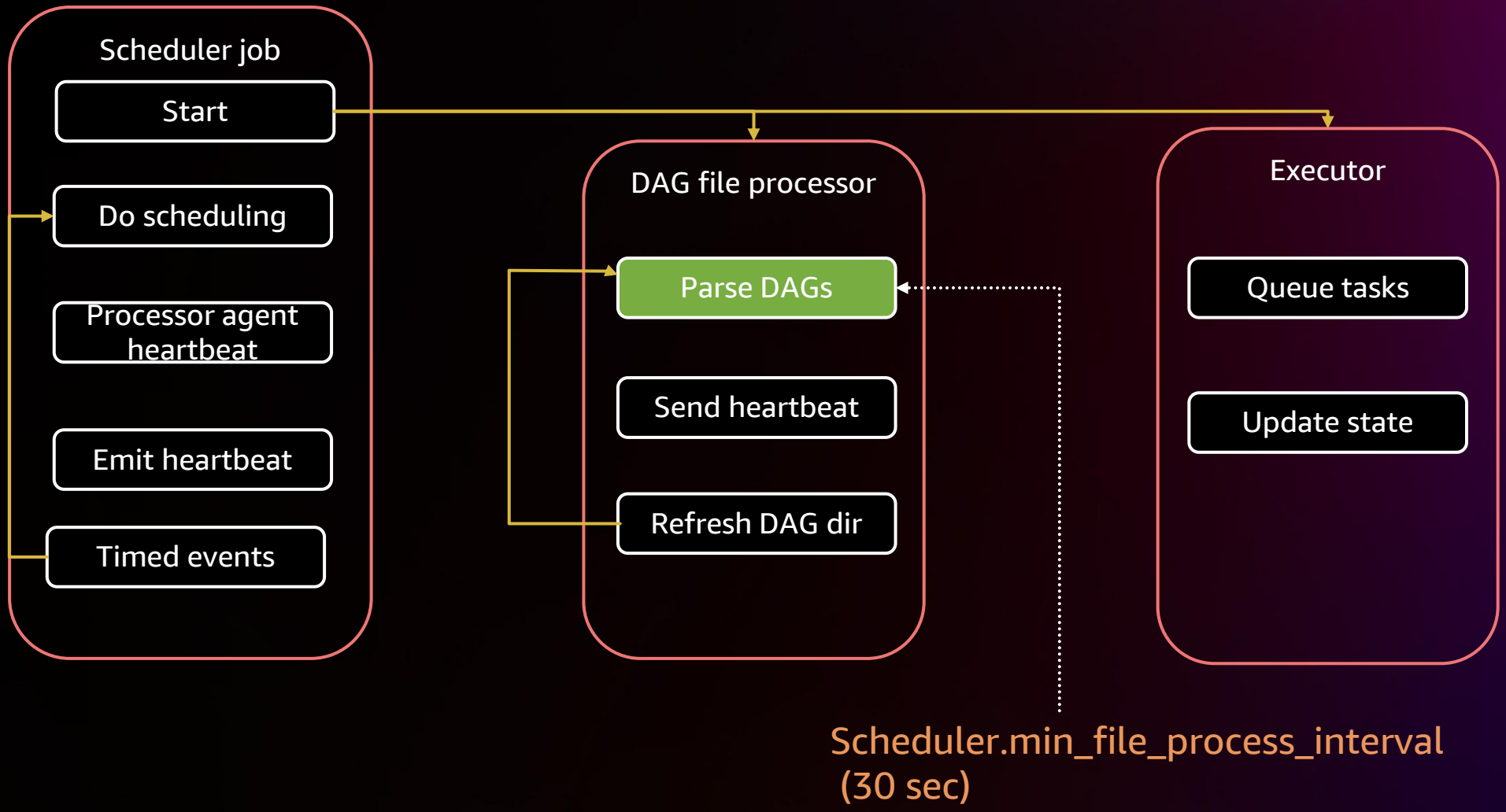
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Understanding scheduler loop



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Understanding scheduler loop



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Top-level code

```python
import numpy as np  # <-- THIS IS A VERY BAD IDEA! DON'T DO THAT!

with DAG(
    dag_id="example_toplevelcode",
    schedule=None,
    start_date=pendulum.datetime(2022, 1, 1, tz="UTC"),
    catchup=False,
    tags=["example"],
) as dag:

    def getVariables():
        cluster = Variable.get("sandbox_cluster")
        email = Variable.getCluster("email")
        region = Variable.getCluster("region")
        print(cluster)
        return cluster,email,region

    @task()
    def print_array():
        """Print Numpy array."""
        a = np.arange(15).reshape(3, 5)
        print(a)
        return a

    print_array(getVariables())
```

Top-level import

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Top-level code

```python
import numpy as np  # <-- THIS IS A VERY BAD IDEA! DON'T DO THAT!

with DAG(
    dag_id="example_toplevelcode",
    schedule=None,
    start_date=pendulum.datetime(2022, 1, 1, tz="UTC"),
    catchup=False,
    tags=["example"],
) as dag:

    def getVariables():
        cluster = Variable.get("sandbox_cluster")
        email = Variable.getCluster("email")
        region = Variable.getCluster("region")
        print(cluster)
        return cluster,email,region

    @task()
    def print_array():
        """Print Numpy array."""
        a = np.arange(15).reshape(3, 5)
        print(a)
        return a

    print_array(getVariables())
```

- Top-level import
- Top-level code with multiple calls to DB

> Time python3 toplevel.py

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Top-level code

```python
with DAG(
    dag_id="JINJA_TEMPLATE",
    schedule_interval=None,
    start_date=days_ago(2),
    catchup=False,
    tags=["example"],
) as dag:

    @task()
    def print_array():
        """Print Numpy array."""
        import numpy as np
        a = np.arange(15).reshape(3, 5)

    @task()
    def print_variables(var=None):
        print(var.value.sandbox_cluster)
        print(var.value.email)
        print(var.value.region)

    print_array()
    print_variables()
```

- Local import
- Jinja template

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Top-level code

```python
with DAG(
    dag_id="JINJA_TEMPLATE",
    schedule_interval=None,
    start_date=days_ago(2),
    catchup=False,
    tags=["example"],
) as dag:

    @task()
    def print_array():
        """Print Numpy array."""
        import numpy as np
        a = np.arange(15).reshape(3, 5)

    @task()
    def print_variables(var=None):
        print(var.value.sandbox_cluster)
        print(var.value.email)
        print(var.value.region)

    print_array()
    print_variables()
```

```python
task1 = PythonOperator(
            task_id='load_date',
            python_callable=load_date_fn,
            op_kwargs={
                        # evaluated at parse time
                        'date': str(datetime.datetime.today())
            }
)

task2 = PythonOperator(
            task_id='load_date',
            python_callable=load_date_fn,
            op_kwargs={
                         # evaluated at task run time
                        'date': '{{ ds }}'
            }
)
```

Jinja template

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

DAG processing results

```
================================================================================
DAG File Processing Stats

File Path                                  PID    Runtime      # DAGs    # Errors  Last Runtime    Last Run
-----------------------------------------  -----  ---------  --------  ----------  --------------  -------------------
/usr/local/airflow/dags/topvariable.py                             1           0  0.92s           2022-10-13T13:04:12
/usr/local/airflow/dags/toplevel.py                                1           0  0.63s           2022-10-13T13:04:02
/usr/local/airflow/dags/notoplevel.py                              1           0  0.39s           2022-10-13T13:04:07
/usr/local/airflow/dags/jinjavariable.py                           1           0  0.44s           2022-10-13T13:04:09
================================================================================
```

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Worker DAG parsing

From worker logs:

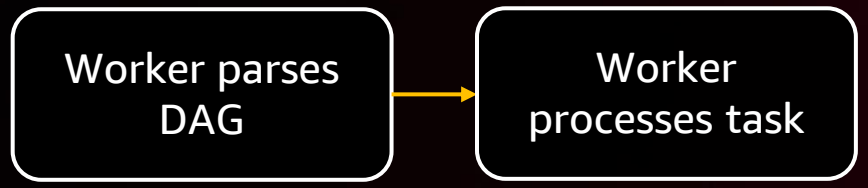
```
Executing command in Celery: ['airflow', 'tasks', 'run', 'scale_db_direct_users42',
'query_table', 'scheduled__2022-10-14T16:00:00+00:00', '--local', '--subdir',
'DAGS_FOLDER/scale_db_direct.py
```

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Worker DAG parsing

From worker logs:

```
Executing command in Celery: ['airflow', 'tasks', 'run', 'scale_db_direct_users42',
'query_table', 'scheduled__2022-10-14T16:00:00+00:00', '--local', '--subdir',
'DAGS_FOLDER/scale_db_direct.py
```

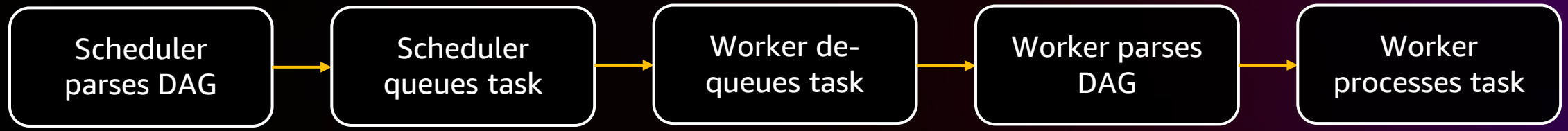


© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Worker DAG parsing

From worker logs:

```
Executing command in Celery: ['airflow', 'tasks', 'run', 'scale_db_direct_users42',
'query_table', 'scheduled__2022-10-14T16:00:00+00:00', '--local', '--subdir',
'DAGS_FOLDER/scale_db_direct.py
```

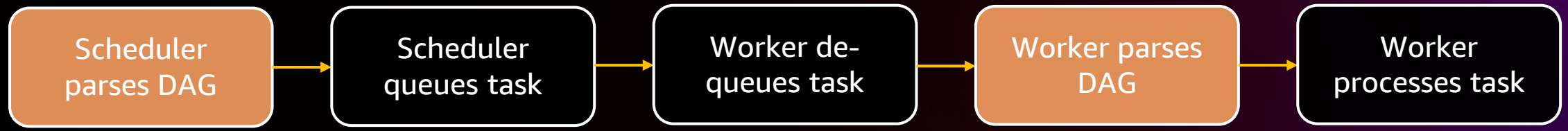


© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Worker DAG parsing

From worker logs:

```
Executing command in Celery: ['airflow', 'tasks', 'run', 'scale_db_direct_users42',
'query_table', 'scheduled__2022-10-14T16:00:00+00:00', '--local', '--subdir',
'DAGS_FOLDER/scale_db_direct.py
```



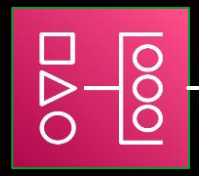
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What inefficient DAGs actually cost – 20 sec processing time

350 hourly DAGs

10 tasks per DAG

Estimated task time 60 sec



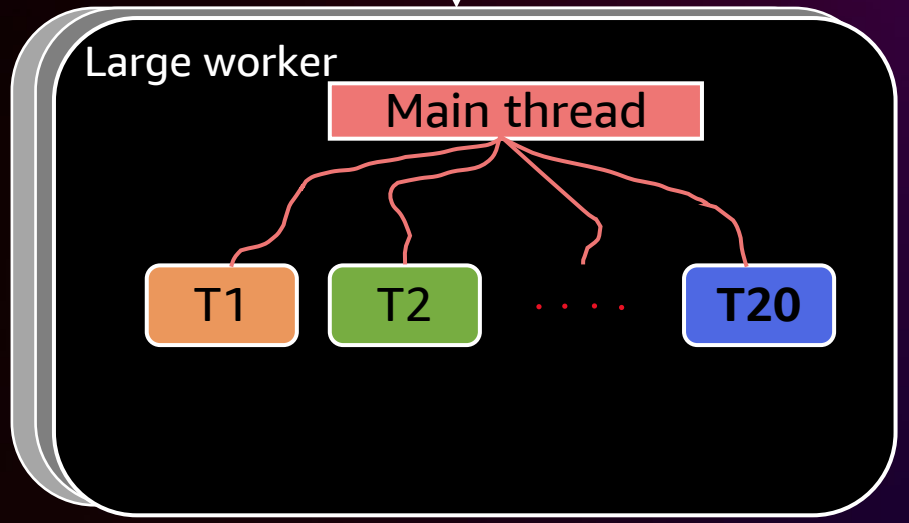
Amazon MWAA

Total task time/worker/hour – 72000 sec

Total task time 60 + 20 sec

Total task per hour – 280000

4 workers



Monthly worker cost = **$633.60**

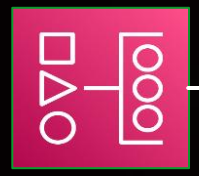
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What inefficient DAGs actually cost – 60 sec processing time

350 hourly DAGs

10 tasks per DAG

Estimated task time 60 sec



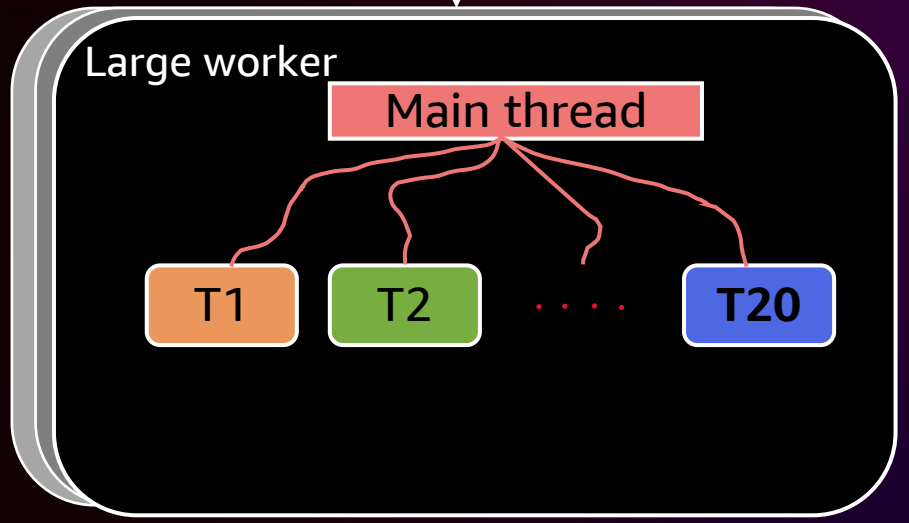
Amazon MWAA

Total task time/worker/hour – 72000 sec

Total task time 60 + 60 sec

Total task per hour – 420000

6 workers



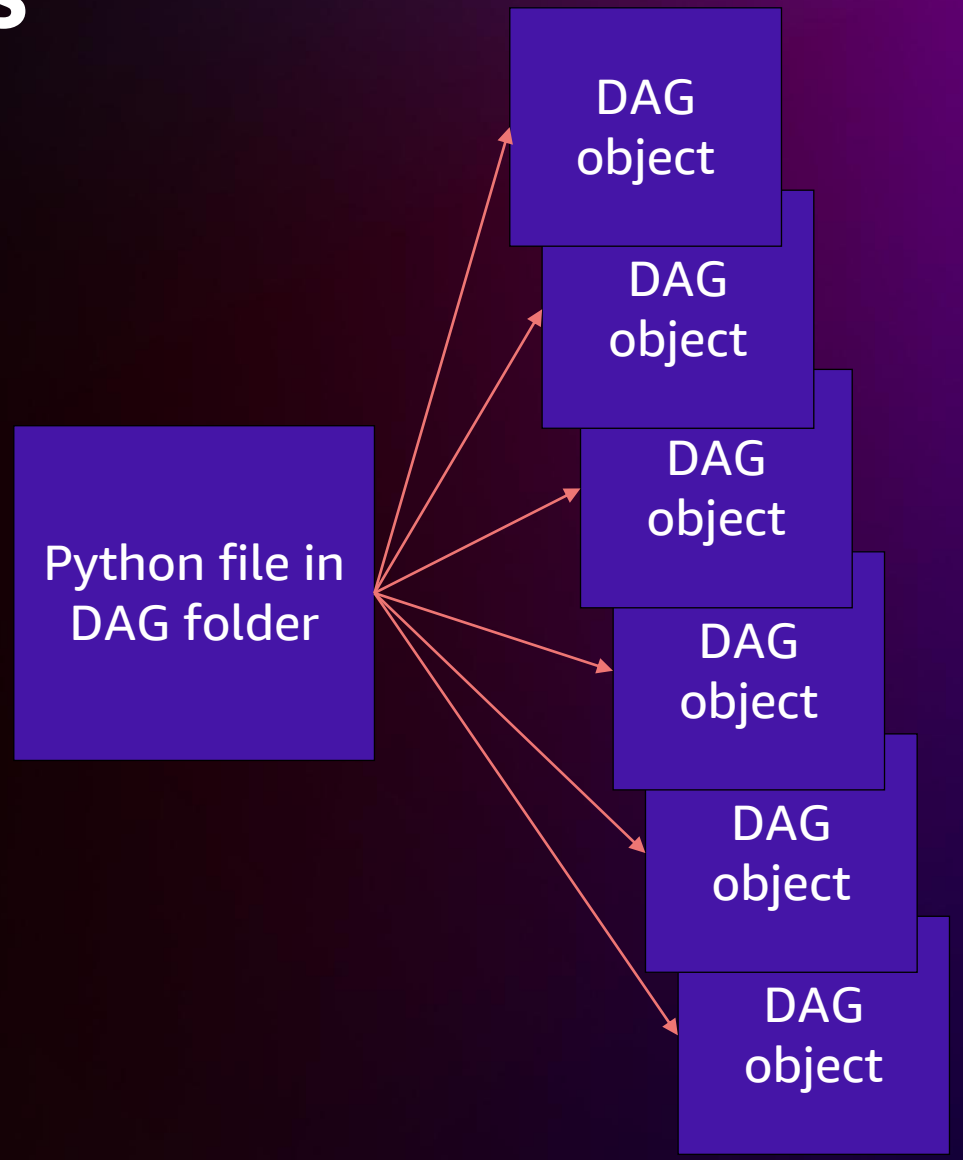
Monthly worker cost = **$950.40**

50% increase

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Dynamic DAGs/DAG Factories

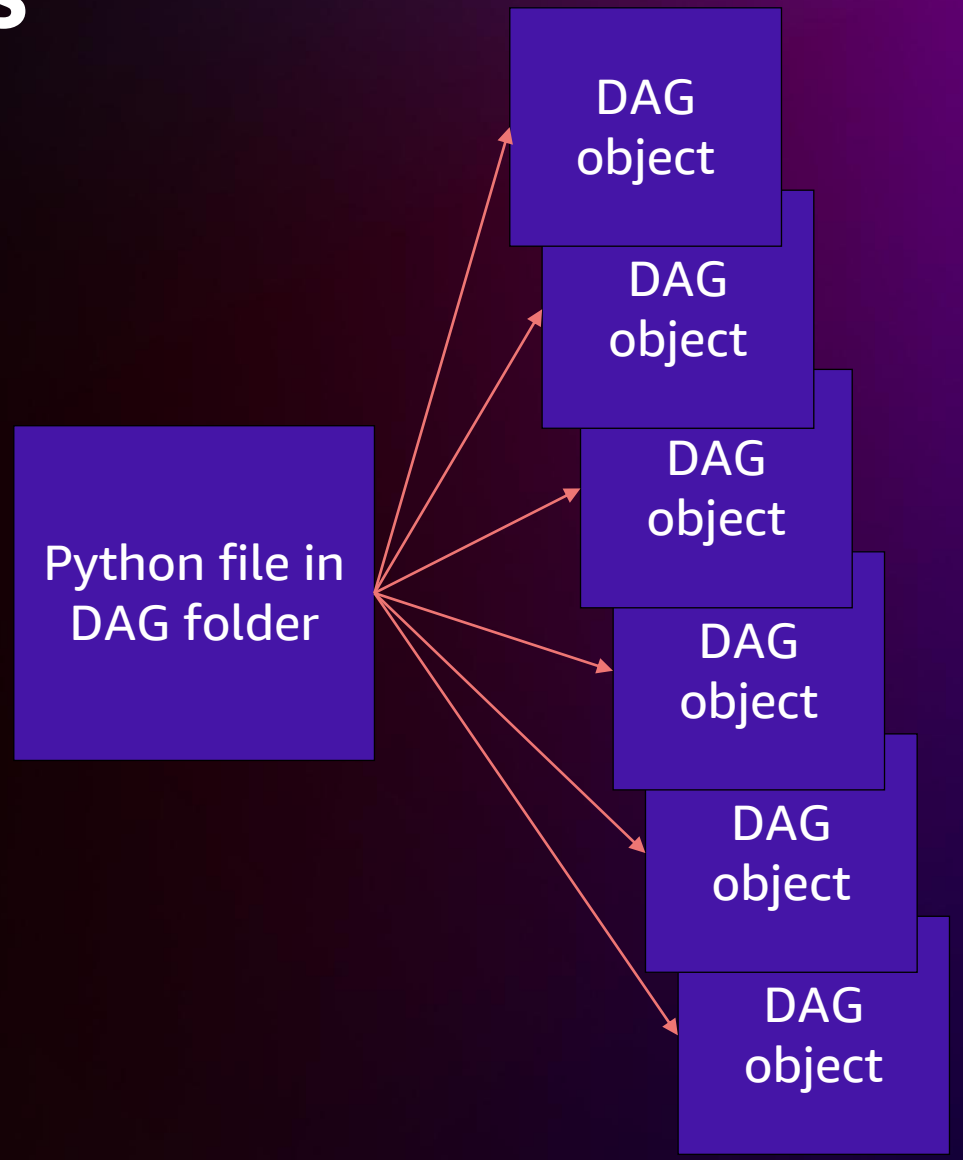
- Dynamic DAG creation
    - File in the DAGS folder (tables, JSON, variables)
    - Build time generation with multiple files (DAG Factory: https://github.com/ajbosco/dag-factory)



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Dynamic DAGs/DAG Factories

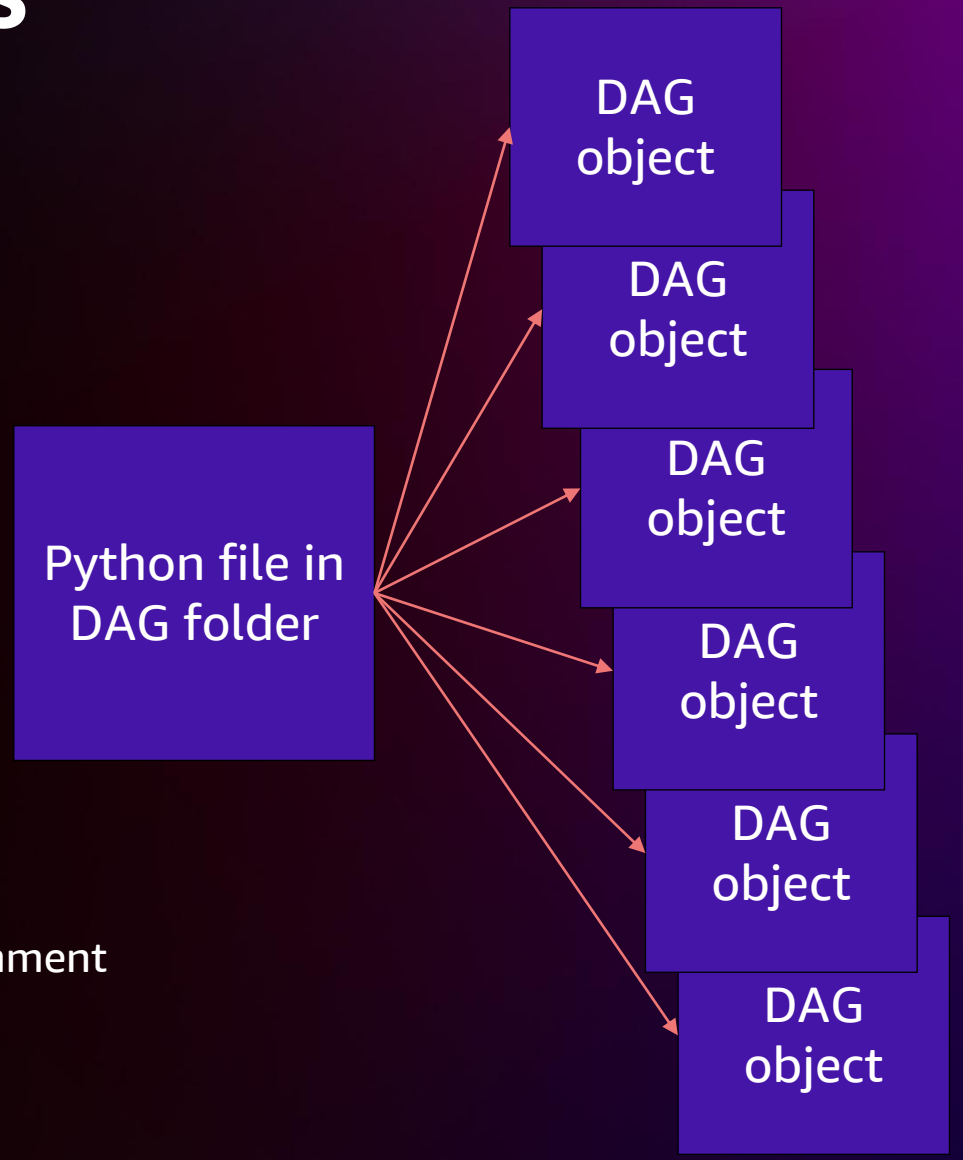
- Dynamic DAG creation
    - File in the DAGS folder (tables, JSON, variables)
    - Build time generation with multiple files (DAG Factory: https://github.com/ajbosco/dag-factory)
- Advantages
    - Automates building 100s of DAGs with repeated logic
    - Better access control with just operators



aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Dynamic DAGs/DAG Factories

- Dynamic DAG creation
  - File in the DAGS folder (tables, JSON, variables)
  - Build time generation with multiple files (DAG Factory: https://github.com/ajbosco/dag-factory)
- Advantages
  - Automates building 100s of DAGs with repeated logic
  - Better access control with just operators
- Disadvantages
  - Visibility to the code is limited
  - Scheduler parsing time may cause performance issues
  - Care should be taken to size the DAGs/tasks with the environment



aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Q&A

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Key takeaways

DESIGNING WORKFLOWS FOR MASSIVE SCALE



Deep dive into your workflows



Explore metrics and configurations



Apply the techniques

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Resources

- Airflow best practices – https://s12d.com/hcXhMmNy
- Airflow scheduler – https://s12d.com/w5fkcblR
- Airflow at scale – https://s12d.com/fEAy2hrf
- Improve worker performance - https://s12d.com/mwaa-get-dagid

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

John Jackson
jacnjoh@amazon.com



Please complete the session survey in the **mobile app**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.