Well-Architected Workflows - Resiliency

Uma Ramadoss
Sr. Specialist Solutions Architect, Integration
Amazon Web Services
What is Resiliency?

Capability of the workload to recover from infrastructure or service disruptions
<table>
<thead>
<tr>
<th>Reliability</th>
<th>Resiliency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø Ability of the workload to perform its</td>
<td>Ø Ability of the system to recover from failures</td>
</tr>
<tr>
<td>intended function correctly and consistently.</td>
<td></td>
</tr>
<tr>
<td>Ø Reliability is impacted by operational</td>
<td>Ø Resiliency is the component of Reliability</td>
</tr>
<tr>
<td>practices, performance efficiency, security</td>
<td></td>
</tr>
<tr>
<td>etc. including Resiliency</td>
<td></td>
</tr>
</tbody>
</table>
What are Resilient workflows?
Redriveable workflows with retrievable atomic tasks.
Build

Fail

Analyze causes of failure

Change practices
“Everything fails, all the time.”

Werner Vogels
CTO, amazon.com
Build

Fail

Analyze causes of failure

Change practices
Build

Analyze causes of failure

Fail

Change practices
Resiliency in Airflow Architecture
Understanding Main Components of Apache Airflow

- Scheduler
- Worker
- Web Server
- Meta Database
Production suitable implementation
Resilient Design Principles
All or None - Atomicity

Extraction
- Extract data from sources
- Store metadata
- Process file by file type

Data aggregation

Update external API
All or None - Atomicity

1. Store metadata
2. Choose filetype
3. Process image file
4. Data aggregation
5. Update external API

Extraction:
- Extract data1
- Extract data2
All or None - Atomicity

This is an atomic operation

Store metadata

Choose filetype

Process image file

Data aggregation

Process csv file

Extract data1

Extract data2

Get API Key

Update API
All or None - Atomicity

Store metadata

Choose filetype

Process image file

Data aggregation

Process csv file

GetAPIKey and Update external API

Extract data1

Extract data2
Make failures safe - Idempotency

- Store metadata
- Choose filetype
- Process image file
- Data aggregation
- Process csv file
- Store in Database

Extract data1
Extract data2
Make failures safe - Idempotency

Store metadata

Choose filetype

Process image file

Data aggregation

Store in Database

Process csv file

Extract data1

Extract data2

INSERT INTO TABLE...
Make failures safe - Idempotency

- Store metadata
  - Extract data1
  - Extract data2
- Choose filetype
- Process image file
  - Process csv file
- Data aggregation
  - UPSERT
- Store in Database
task_transfer_s3_to_redshift = S3ToRedshiftOperator(
    s3_bucket=S3_BUCKET_NAME,
    s3_key=S3_KEY,
    schema='PUBLIC',
    table=REDSHIFT_TABLE,
    copy_options=['csv'],
    method='UPSERT',
    task_id='transfer_s3_to_redshift',
)
Make failures safe - Idempotency

- Store metadata
- Choose filetype
- Process image file
- Data aggregation
- Process csv file
- Store in Database

UPSERT / STAGING TABLE/ VALIDATE EXISTENCE

Extract data1
Extract data2
Make failures safe - Idempotency

- Store metadata
- Choose filetype
- Process image file
- Data aggregation
- Process csv file
- Call external Service
  - Extract data1
  - Extract data2
Make failures safe - Idempotency

1. Store metadata
2. Choose filetype
3. Process image file
4. Data aggregation
5. Process csv file
6. Call external Service if not called before
7. Store metadata
8. Extract data1
9. Extract data2
10. STORE STATE
Protect downstream – retries, backoff

```python
args=
{
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': True,
    'email_on_retry': False,
    'retries': 3,
    'retry_delay': timedelta(seconds=5),
    'retry_exponential_backoff': True
}
```
Protect downstream – max active runs

Legend:
- Green: Dag
- Orange: Backfill
- Gray: Current date

Timeline:
- Dag Start date
- Current date

Timeline Details:
- May 20, 21:30
- May 20, 21:55
- May 20, 22:20
- May 20, 22:45
- May 20, 23:10
Protect downstream – Preemptive load shedding

**Airflow pools** can be used to limit the execution parallelism on arbitrary sets of tasks. Typically this is done to limit downstream impact, for example putting all database tasks in an “RDS” pool that has a limit based upon the connection limit of the DB.
Fail fast and fail forward – SLA and Timeout

- Leverage SLA and sla_miss_callback for awareness
- Use execution timeout for cancellation of tasks
- Raise AirflowSkipException, AirflowFailException to fail fast on obvious errors
- Checkpoint/validate data

```python
@task(sla=timedelta(seconds=60),
      execution_timeout=timedelta(seconds=70)
)
def long_running_task():
    blah = call_external_service()
    if blah == "foo":
        raise AirflowSkipException

    ....

Validate_data = SQLCheckOperator
    (task_id=validate, ...)
```
Resiliency Design Principles – Recap

- All or None - Atomicity
- Make failures safe – Idempotency
- Protect Downstream
- Fail fast and fail forward
Some best practice implementations
Use Airflow as an orchestration tool

- Externalize compute/memory-intensive work to purpose built services.
- Leverage community offered operators.
Operation as code & small reversible changes

- Add controls to verify integrity, scope, and usage of DAGs before deploying
- Test locally for faster feedback cycle
- Update plugins and requirements programmatically

Pipeline:
- Local dev test
- Source Control
- Create environment [first time]
- Deploy/Update env if needed
- Build/plugins
- Sanity check
- Unit test
- Build/docker image
- Dagbag/parse time metric
- Run Airflow docker image
Monitoring Workflow

- Build dashboards with relevant metrics like parse time, scheduling delays, queued/running tasks etc.
- Send notification when thresholds are exceeded.
- Leverage dashboards like Landing times, Gantt chart to troubleshoot performance issues.
Testing in Airflow – Medium article

https://bit.ly/3w0PpeA
Data validation

https://www.youtube.com/watch?v=6ib2gH4A0rI
Airflow best practices

Q/A
Thank you!

https://www.linkedin.com/in/umamaheswari-r-96578910/