Choosing Apache Airflow over other Proprietary Tools for your Data Orchestration needs

Parnab Basak
Solutions Architect, AWS
About me

- Over 18+ of IT experience in Energy, Utilities, FinTech & GovTech...
- ~2 Years in AWS as an SA serving our WWPS customers
- Specialist SME for Amazon Managed Workflow for Apache Airflow (Amazon MWAA)
- Airflow (MWAA) Blogpost Author

In my spare time: I evolve to be a Movie Buff

(Have Amazon Prime, Netflix, HBO Max, Apple TV+, Peacock, Disney+ AND MORE....)

** Wish to become a movie critic some day **
With Apache Airflow

01 Data Orchestration
Definition & Benefits

02 Tools
Proprietary and OSS

03 Compare
Between my Pick of 4

04 Similarities
All are good tools

05 Differences
There are a few

06 Unique Features
Of Apache Airflow

07 Get Started
With Apache Airflow

08 Migrate
To Apache Airflow

09 Q and A
You ask, I answer
Data **orchestration** is the process of taking siloed data from multiple data storage locations, combining and organizing it, and making it available for data analysis tools. It can also include tasks like provisioning resources and monitoring.

* Pic: https://www.bmc.com/blogs/workflow-orchestration/
Benefits

- Reduced human error
- Faster response to mission critical system problems
- More efficient allocation of resources

* As per EMA research
Persona based requirements

**Business**
- Availability
- Reliability
- Quality
- Democratization
- Simplicity
- Cost Effective

**Developer**
- Programmable/ APIs
- Testable – Local & Unit Test
- Reusable/ Extensible
- Integrations
- CI/CD
- Documentation/ Training

**Operations**
- Admin
- Reporting/ Predictions
- Alerting
- Scalability
- HA, DR
- Infrastructure As Code / CM

**Security**
- Compliance
- Access Control
- At-Rest
- In-Transit
- Audit
- Mitigation
The tool should

### Triggers
- Schedule
- Event
- Dependency
- Adhoc

### Task Definitions
- Start
- Branch
- Group
- Task
- Dependent
- Parallel
- Task
- Task
- Task
- Skip
- Task
- Loop
- Condition
- End

### Integrations
- Database
- Data Warehouse
- Data Lakes
- Files Stores
- Compute
- 3P Tools
- APIs
- More..
### Available Orchestration tools

<table>
<thead>
<tr>
<th>Proprietary Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Licensed from a vendor to install on a definite number of machines</td>
</tr>
<tr>
<td>• Source codes are <strong>not available publicly</strong></td>
</tr>
<tr>
<td>• Customizations may come at an <strong>extra cost</strong></td>
</tr>
<tr>
<td>• Specialized technical support, especially for enterprise clients</td>
</tr>
<tr>
<td>• Reliance on the vendor to continue to debug and improve the product</td>
</tr>
</tbody>
</table>

![Proprietary Tools Logos](image-url)
## Available Orchestration tools

### Open Source Software Tools

- **Free** to try, use, modify, redistribute
- **Free** community forums that offer support
- **Open standards** that increase transparency
- **No vendor lock-in, No IP restrictions**
- **Easily scaled and extended**

![Apache Airflow](image1)
![Luii](image2)
![Azkaban](image3)
![Oozie](image4)

and more..
Let's Compare between Apache Airflow, Broadcom AutoSys, bmc Control-M and Spotify Luigi
<table>
<thead>
<tr>
<th>History</th>
<th>Apache Airflow</th>
<th>Broadcom AutoSys</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Started in October 2014 by Maxime Beauchemin at Airbnb.</td>
<td>• First developed by William Arntz and Walter Goodwin who created AutoSystems Ltd.</td>
</tr>
<tr>
<td></td>
<td>• Open source from the very first commit and officially brought under the Airbnb GitHub and announced in June 2015</td>
<td>• Sold to Platinum Technology International in 1995</td>
</tr>
<tr>
<td></td>
<td>• Joined the Apache Software Foundation’s Incubator program in March 2016</td>
<td>• Bought by Computer Associates in 1999</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Broadcom acquired CA in 2018 and known today as “AutoSys Workload Automation”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>bmc Control-M</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Originally developed for scheduling jobs on mainframe computer systems, by an Israel-based company - New Dimension Software</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• BMC software acquired New Dimension Software in 1999</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• “BMC Helix Control-M” - SaaS solution was released in December - 2020</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Spotify Luigi</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Created by Spotify mainly by Erik Bernhardsson and Elias Freider</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Initial commit on github/spotify/luigi on Nov 17, 2011</td>
<td>• Open sourced in 2012</td>
</tr>
<tr>
<td></td>
<td>• “BMC Helix Control-M” - SaaS solution was released in December - 2020</td>
<td>• Spotify's Data Team maintains Luigi</td>
</tr>
</tbody>
</table>
Similarities

**OS Support**
- Multi-platform
- Cloud & On-prem, Hybrid

**Availability**
- HA, DR

**Security**
- SSO (LDAP/AD), RBAC, SSL

**Interaction**
- Rich UI, REST API, CLI

**Insights**
- History, Audit Trails, Dependency Graph

**Integration**
- DBs, Data Warehouses, Data Lakes, Hadoop, 3Ps

**Job Constructs**
- Sequential, Parallel, Branching, Loops, Skips

**Scheduling**
- Event, Schedule, Dependency, Manual

**Jobs**
- Groups/Nesting, Metadata, Dynamic Jobs
Apache Airflow Architecture

- User Interface
- Webserver
- Scheduler Executor
- Workers
- Metadata Database (PostgreSQL/MySQL/MSSQL)
- Authored DAGs
- DAG Directory
- TaskQueue
Control-M Architecture

User → Control-M Client (Web/Mobile)

Control-M Client (Web/Mobile) → GUI

Control-M Client (Web/Mobile) → Gateway

Gateway → EM DB

EM DB → Control-M Server for Distributed Systems

Control-M Server for Distributed Systems → Server DB

Server DB → Control-M Agents

Control-M Agents → Agentless Remote Hosts

Agentless Remote Hosts → Control-M for z/OS

Control-M for z/OS → Workload Archive Server

Workload Archive Server → Gateway

Gateway → Control-M for Distributed Systems

Control-M for Distributed Systems → Application plug-ins
Luigi Architecture

Network

Central Scheduler
- Register Tasks
- Next Task?

Task Tree (Workers)
- Load Dependencies
- Submit as job

Remote

Workers
- Load
- Poll status

Software & Images
- Load

Local

Analysis & Task Classes

Input/Output Targets

Remote

Software & Images

Remote

Network

Local

Remote

Remote

Network

Remote

Remote

Local

Remote

Network

Remote

Remote

Remote

Remote

Local

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote

Remote
<table>
<thead>
<tr>
<th>Apache Airflow</th>
<th>Broadcom AutoSys</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent-less.</td>
<td>Must install an agent (ex: Workload</td>
</tr>
<tr>
<td>Use Connections, Hooks &amp; Operators (Core</td>
<td>Automation Agent for UNIX, Linux, or</td>
</tr>
<tr>
<td>and part of community provider packages)</td>
<td>Windows) and agent plug-ins (ex: Database</td>
</tr>
<tr>
<td></td>
<td>Agent plug-in) on target system</td>
</tr>
<tr>
<td>bmc Control-M</td>
<td>Spotify Luigi</td>
</tr>
<tr>
<td>Agent based.</td>
<td>Agent-Less.</td>
</tr>
<tr>
<td>Involves installing a Server, Agent and client</td>
<td>pip install luigi</td>
</tr>
<tr>
<td>installs along with Application Plug-ins and</td>
<td>Create a physical connection in python to</td>
</tr>
<tr>
<td>add-ons</td>
<td>connect to DBs</td>
</tr>
</tbody>
</table>
## Job Definition

<table>
<thead>
<tr>
<th>Apache Airflow</th>
<th>Broadcom AutoSys</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Python</strong></td>
<td><strong>Job Information Language</strong></td>
</tr>
<tr>
<td>• Default Args = a constructor</td>
<td>• job_type</td>
</tr>
<tr>
<td>• DAG instantiate = id, schedule</td>
<td>• command, machine</td>
</tr>
<tr>
<td>• Tasks = Operators</td>
<td>• owner</td>
</tr>
<tr>
<td>• Dependencies = upstream, downstream</td>
<td>• date_conditions, days_of_week, start_times</td>
</tr>
<tr>
<td>• optional input() = input from other tasks.</td>
<td>• alarm_if_fail</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bmc Control-M</th>
<th>Spotify Luigi</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JSON/XML</strong></td>
<td><strong>Python</strong></td>
</tr>
<tr>
<td>• Defaults = define a parameter once</td>
<td>• param = parameters for the task</td>
</tr>
<tr>
<td>• Job/When = scheduling criteria</td>
<td>• requires() = specify task dependencies</td>
</tr>
<tr>
<td>• Job = Tasks</td>
<td>• run() = logic for execution</td>
</tr>
<tr>
<td>• Flow = define order dependency</td>
<td>• output() = returns the artifacts generated</td>
</tr>
<tr>
<td></td>
<td>• optional input() = input from other tasks.</td>
</tr>
</tbody>
</table>
Apache Airflow Code Example

```python
from airflow.contrib.sensors.file_sensor import FileSensor
from airflow.operators.dummy_operator import DummyOperator

import datetime
from datetime import date, timedelta
import airflow

default_args = {
    "depends_on_past": False,
    "start_date": airflow.utils.dates.days_ago(1),
    "retries": 1,
    "retry_delay": datetime.timedelta(hours=5),
}
today = datetime.datetime.today()
yesterday = date.today() - timedelta(days=1)

with airflow.DAG("file_sensor_example", default_args=default_args, schedule_interval="*/5 * * * ") as dag:
    start_task = DummyOperator(task_id="start")
    stop_task = DummyOperator(task_id="stop")
    sensor_task = FileSensor(task_id="file_sensor_task",
                               poke_interval=30,
                               fs_conn_id="<path>",
                               filepath="<file or directory name>")

start_task >> sensor_task >> stop_task
```

- **Imported Libraries**
- **Default Arguments/dictionary of default parameters**
- **Unique identifier + schedule interval**
- **Task definitions**
- **Dependency**
AutoSys JIL example

```plaintext
[File: autosys.jil]

1. insert_job: file-watcher-job
2. job_type: FW
3. machine: localhost
4. owner: me@aws
5. permission: mx
6. date_conditions: 1
7. days_of_week: all
8. start_times: “15:00, 14:00”
9. watch_file: <Some file path location>
10. watch_interval: 10
11. description: “Some description of the job”
12. std_out_file: /tmp/std_out
13. std_err_file: /tmp/std_err
14. alarm_if_fail: 1
15. profile: /tmp/.profile
```
Control-M Example

```json
{
"Defaults": {
"Application": "SampleApp",
"SubApplication": "SampleSubApp",
"RunAs": "USERNAME",
"Host": "HOST",
"Job": {
  "When": {
    "Months": ["JAN", "OCT", "DEC"],
    "MonthDays": ["22", "1", "11"],
    "WeekDays": ["MON", "TUE", "WED", "THU", "FRI"],
    "FromTime": "0300",
    "ToTime": "2100"
  }
},
"AutomationAPI SampleFlow": {
  "Type": "Folder",
  "Comment": "Code reviewed by John",
  "CommandJob": [],
  "Job:Command": {
    "Type": "Job:Command",
    "Command": "echo my 1st job"
  },
  "ScriptJob": {
    "Type": "Job:Script",
    "FilePath": "SCRIPT_PATH",
    "FileName": "SCRIPT_NAME"
  },
  "Flow": {
    "Type": "Flow",
    "Sequence": ["CommandJob", "ScriptJob"]
  }
}
}
```
Luigi Code Example

```python
from datetime import date, timedelta
import luigi

class HelloWorld(luigi.Task):
    date = luigi.DateParameter(default=date.today() - timedelta(days=1))

    # no upstream requirements at all
    def requires(self):
        return None

    # creates a local file as output
    def output(self):
        return luigi.LocalTarget('helloworld.txt')

    # the actual job to perform
    def run(self):
        with self.output().open('w') as outfile:
            outfile.write('Hello World!
')

if __name__ == '__main__':
    luigi.run()
```
### File System Events

<table>
<thead>
<tr>
<th>Apache Airflow</th>
<th>Broadcom AutoSys</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FileSensor</strong></td>
<td><strong>File Watcher/File Trigger Job</strong></td>
</tr>
<tr>
<td>- <em>FileSensor</em> - Waits for a file or folder to land in a filesystem</td>
<td>- The file reaches the minimum file size that is specified in the <code>watch_file_min_size</code> attribute.</td>
</tr>
<tr>
<td>- <em>S3KeySensor</em> - AWS</td>
<td>- The file reaches a “steady state” during the polling interval.</td>
</tr>
<tr>
<td>- <em>WasbBlobSensor</em> - Azure</td>
<td></td>
</tr>
<tr>
<td>- <em>GCSObjectExistenceSensor</em> - GCP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bmc Control-M</th>
<th>Spotify Luigi</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File Watcher (ctmfw)</strong></td>
<td>None available natively</td>
</tr>
<tr>
<td>- File transfer activity (min size)</td>
<td>Have to be triggered externally based on file event</td>
</tr>
<tr>
<td>- File Creation</td>
<td></td>
</tr>
<tr>
<td>- File Deletion</td>
<td></td>
</tr>
</tbody>
</table>
## Autoscaling

<table>
<thead>
<tr>
<th><strong>Apache Airflow</strong></th>
<th><strong>Broadcom AutoSys</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Workers Autoscale</td>
<td>Agent based. Container scaling</td>
</tr>
<tr>
<td>• <em>Parallelism</em></td>
<td>• Agents can run in docker containers</td>
</tr>
<tr>
<td>• <em>Concurrency (max_active_tasks)</em></td>
<td>• Run jobs on the container, autoscale tasks</td>
</tr>
<tr>
<td>• <em>min-workers</em></td>
<td>• Auto register/un-register the Agent container from the server</td>
</tr>
<tr>
<td>• <em>max-workers</em></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>bmc Control-M</strong></th>
<th><strong>Spotify Luigi</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent based. Container scaling</td>
<td>Workers Autoscale</td>
</tr>
<tr>
<td>• Agents can run in docker containers</td>
<td>Based on PENDING tasks count as determined by the Central Scheduler</td>
</tr>
<tr>
<td>• Run jobs on the container, Autoscale tasks</td>
<td></td>
</tr>
<tr>
<td>• Auto register/un-register the Agent container from the server</td>
<td></td>
</tr>
</tbody>
</table>
Calendar

**Apache Airflow**
- Cron expression
- Time deltas - Waits for a timedelta after the run's data interval (*TimeDeltaSensor*)
- Timetables - custom schedules using Python

**Broadcom AutoSys**
- Standard Calendar - Lists fixed dates
- Cycle - Lists date ranges (periods)
- Extended Calendar - Specifies complex criteria to generate a schedule based on logic

**bmc Control-M**
- Regular Calendar - specific dates, such as, days of the month, and days of the week in a selected year, holidays
- Periodic Calendar - different calendar periods other than months and days (quarter)
- Rule Based Calendar - specific complex rules. (3 days before the end of the month)

**Spotify Luigi**

No native concept of scheduling

Luigi does not include its own triggering, so you have to rely on an external scheduler such as crontab to actually trigger the workflows.
## Job Queue Priority

### Apache Airflow

- **Pool + priority_weights parameter**
  - `default_pool = 128 slots`
  - `priority_weights = any arbitrary integer` (default is 1). Higher values get higher priority in the executor queue.

Pools are meant to control parallelism for Task Instances.

### Broadcom AutoSys

- **priority attribute**
  - `priority_level` = Defines the queue priority of the job. The lower the value, the higher the priority; 0 signifies to run the job immediately, regardless of the current machine load

  Default: 0

  ```bash
  sendevent -E CHANGE_PRIORITY -q queue_priority
  ```

### bmc Control-M

- **PRIORITY* property**
  - Available `priority levels` are Critical, High, Normal, Low, and Lowest. The default value for the PRIORITY* property is Normal.

  ```bash
  pause a running job > Update priority > resume
  ```

### Spotify Luigi

- **priority**

Tasks with a higher priority value will be picked before tasks with a lower priority value. No predefined range of priorities, you can choose whatever (int or float) values. The default value is 0.
### Alerting

<table>
<thead>
<tr>
<th>Apache Airflow</th>
<th>Broadcom AutoSys</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Notifications</strong></td>
<td>send_notification attribute</td>
</tr>
</tbody>
</table>
| • default_args/BaseOperator  
  `email_on_failure`: True  
  `email`: ['noreply@aws.com']  
  • Custom Notifications (DAG or task level)  
  on_failure_callback, on_success_callback  
  • SlackWebhookOperator | set the send_notification attribute value to y and specify the notification_template and the notification_emailaddress, notification_emailaddress_on_success, or notification_emailaddress_on_terminated attributes in your job definition. |
| bmc Control-M | Spotify Luigi |
| Shout for job + Destination | luigi.notifications module |
| • sent out before a job ends  
  • sent out after a job ends | [email] receiver=foo@bar.baz |
| | send_email, send_error_email,  
  send_email_smtp, send_email_ses,  
  send_email_sns |
## SLA Management

### Apache Airflow

**SLA**
- Define a callback method
- Pass the callback method to DAG
  ```python
sla_miss_callback
  ```
- Define the SLA duration on task(s)/DAG
  ```python
  sla=timedelta(seconds=5)
  ```
- Monitor SLA miss in the Airflow UI

### Broadcom AutoSys

**CA Workload Automation iDash SLAs**
- A separate web-based solution to install
- Generates alerts for SLA deadlines that are at risk of being missed, are predicted to be missed, or have been missed
- Executes automated recovery actions in response to alerts

### bmc Control-M

**Batch Impact Manager**
- Separate utility that needs to be installed
- Alerts on potential delays
- Can set deadline to finish, should be completed by since start, notification action
- Analyze why by Filtering on Critical Path using Analysis Viewpoint option

### Spotify Luigi

**No out-of-the-box solution**
- Have to be custom-built using Python
<table>
<thead>
<tr>
<th><strong>Forecast Reports</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Apache Airflow</strong></td>
</tr>
<tr>
<td>Insights only for past runs</td>
</tr>
<tr>
<td>• Calendar View - see trends of the overall success/failure rate of runs over time.</td>
</tr>
<tr>
<td>• Gantt Chart - analyse task duration and overlap</td>
</tr>
<tr>
<td>• Task Duration - duration of your different tasks over the past N runs</td>
</tr>
<tr>
<td><strong>Broadcom AutoSys</strong></td>
</tr>
<tr>
<td>Forecast Reports</td>
</tr>
<tr>
<td>• Displays information about predicted workflow</td>
</tr>
<tr>
<td>• Forecast reports help you identify problems with the predicted workflow to resolve them before they occur or to plan changes in the workflow.</td>
</tr>
<tr>
<td><strong>bmc Control-M</strong></td>
</tr>
<tr>
<td>Forecast</td>
</tr>
<tr>
<td>• Add-on component</td>
</tr>
<tr>
<td>• A visual calendar that displays all the dates on which the job will be scheduled.</td>
</tr>
<tr>
<td>• Estimated time execution window for each and every job</td>
</tr>
<tr>
<td>• Trend analysis displayed as a histogram</td>
</tr>
<tr>
<td><strong>Spotify Luigi</strong></td>
</tr>
<tr>
<td>No out-of-the-box solution</td>
</tr>
<tr>
<td>Have to be custom-built using Python</td>
</tr>
<tr>
<td>• Luigi Visualizer shows basic info (Status, Priority, Time)</td>
</tr>
<tr>
<td>• Luigi Task status shows dependency graph</td>
</tr>
<tr>
<td>Job Source Versioning</td>
</tr>
<tr>
<td>------------------------</td>
</tr>
<tr>
<td><strong>Apache Airflow</strong></td>
</tr>
<tr>
<td><strong>No Built-in Support</strong></td>
</tr>
<tr>
<td>• Have to use external SCM tools like Github, BitBucket, AWS Code Commit etc. for DAG code versioning</td>
</tr>
<tr>
<td>• Have to use other CD tools like Jenkins, CircleCI, Github Actions etc. for deployment to <code>{AIRFLOW_HOME}/dags</code></td>
</tr>
<tr>
<td><strong>bmc Control-M</strong></td>
</tr>
<tr>
<td><strong>Built in Check-in/Check-out</strong></td>
</tr>
<tr>
<td>• Display changes between Job versions</td>
</tr>
<tr>
<td>• Restore a previous version of a job</td>
</tr>
<tr>
<td>• Restore a deleted job</td>
</tr>
<tr>
<td>• Audit Report – User that changed the job</td>
</tr>
<tr>
<td><strong>Workload Change Manager</strong></td>
</tr>
</tbody>
</table>

*Airflow Summit 2022*
Features Unique to Apache Airflow

- Extensible
- Run Locally
- Catchup & Backfill
- Swappable Executors
- Deferrable Operators & Triggers
- Unit Tests
- Custom Secrets Backend
- Data Lineage
- Templating
How to get started

On-Premises
- Src/PyPi
- Container
- Self Manage
- Custom
- Isolation

Cloud
- Managed Services
  - Amazon Managed Workflow for Apache Airflow
  - Google Cloud Composer
  - Modern Data Orchestration
  - Quube

Auto Setup
Autoscaling
Security
Metrics
Cost
How to Migrate

AutoSys
- JIL
- Custom Calendars
- Scripts

Control-M
- Job Definitions
- Custom Calendars
- Scripts

Luigi
- Tasks

Mappings

File Parser (XML or JSON)

Apache Airflow
- DAGs
- Timetables
- Plug-ins

Rewrite

+ Recreate Airflow Connection to data sources & servers
In Summary

- Use a data workflow orchestration tool to easily **Build, Define, Schedule, Manage, and Monitor** production workflows, ensuring visibility, reliability, and improving SLAs.

- Use Apache Airflow to:
  - **Programmatically** author, schedule and monitor workflows.
  - It is **Easy to Use, Scalable, Dynamic, Extensible & Elegant**.
  - **Open Source**, so no punitive licensing fees and you can **Customize** it.
  - Provides **Robust Integrations** to current infrastructure and **Extend** to next-gen technologies.
  - **Comparable features** (and more) than proprietary tools.

- Use a **Managed service** to focus on authoring, scheduling, and monitoring your workflows as opposed to provisioning resources.

- **Refactor/re-architect** from existing proprietary tools to improve operating cost, agility, performance, and scalability.
Questions for Me?

Post them in the Chat
Feel free to voice them as well

- And If you don’t remember them now:
- https://www.linkedin.com/in/parnab-basak/
Thank You!

Enjoy the other summit talks as well.