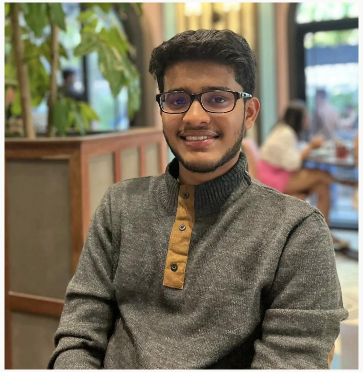
Overcoming Custom Python Package Hurdles in Airflow

Amogh Rajesh Desai

Shubham Raj

About Us



- Senior Software Engineer at Cloudera
    - Cloudera Data Engineering on Private Cloud
- Apache Airflow Committer
    - Contributing since 2023
    - Breeze
    - Helm Charts
    - CNCF, Hive Providers
- Coffee Connoisseur
- Loves Sports and Outdoor

- Software Engineer II at Cloudera
    - Cloudera Data Engineering on Private Cloud
- Apache Airflow Contributor
- Spoke at Airflow Summit 2023 and local summits
- Published author with a book chapter in Scrivener Publishing - Wiley
- Loves playing badminton and cricket

Context

What are we doing?

- Cloudera Data Engineering
- Airflow on Kubernetes
  - Kubernetes Executor
- Multiple Airflow instances
- One per Team
- Teams share DAG code, custom python packages
- … etc.

Example Use Case

Demo

Custom Python Packages

Why are they needed?

- DAG code = native airflow libraries + libraries over pip repos
- Custom Libraries may not be in public pip repos
- Corporate libraries + Airflow DAGs ❌
- Third party Airflow Operators ❌
- No direct way to do this in Airflow
  - When deployed in ***cloud native*** fashion

Custom Python Packages

What Airflow Offers

- Add modules to one of the folders that Airflow adds to its `PYTHONPATH` env
  - Custom code under `PYTHONPATH`
  - `dags`, `plugins`
- Add extra folders where you keep your code to `PYTHONPATH`
  - Extend `PYTHONPATH`
- Package your code into a Python package and install it with Airflow
  - Do not want to extend `PYTHONPATH`
  - Package as pip parcel, pip install it
  - Import and Use in DAGs

Custom Python Packages

Challenges for Airflow on K8s

- Add modules to one of the folders that Airflow adds to its \`PYTHONPATH\` env
  - \`dags\` and \`plugins\` don't reflect at runtime
  - Pods are ephemeral
- Add extra folders where you keep your code to \`PYTHONPATH\`
  - Changes to environment variables will be lost
  - Pods are ephemeral
- Package your code into a Python package and install it with Airflow
  - Baking in Airflow Dockerfile
  - Re-compiling for every change
  - Rolling out updates & repeating this

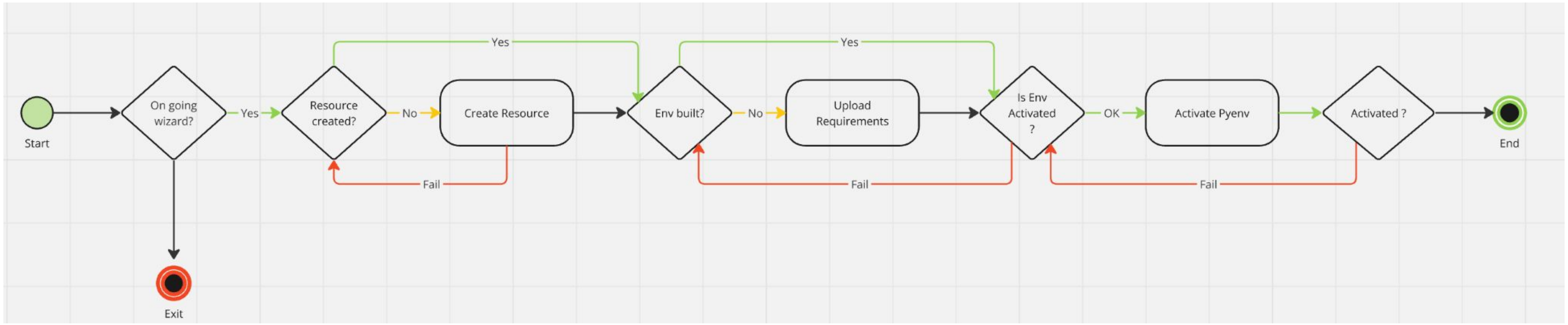
Custom Global Python Environment

What is it?

- Microservice that automates solution to earlier challenges
- Builds a *“global”* Python environment
  - Post Deployment of Airflow
- Works in three steps
  1. Environment Definition
  2. Building the environment
  3. Activating the environment

Custom Global Python Environment

Design



Demo 🤞

Challenges

No Path is Easy!

1. Stable system at all times
    a. Core components restarted
    b. Used maintenance model
        i. State machine approach
        ii. Prevents system crashes
        iii. Proper rollbacks
2. In Place Upgrades
    a. Python / Airflow version changes involved
    b. Can break custom library
    c. Might require updating / rewriting custom library

Remaining Work / Future Plans

Nothing is Perfect :)

1. Backup and Restore
2. Switching away from fileshare
   a. Cloud Native Approach
   b. Pre built docker runtimes
   c. Eliminates latency issues of fileshare
3. Extend custom environments
   a. Not just Python
   b. Extra dependencies like jars, binaries

That's all T

Questions?

@amogh-desai-385141157

@amoghrajesh

@shubhamrajofficial

@shubhamraj-git