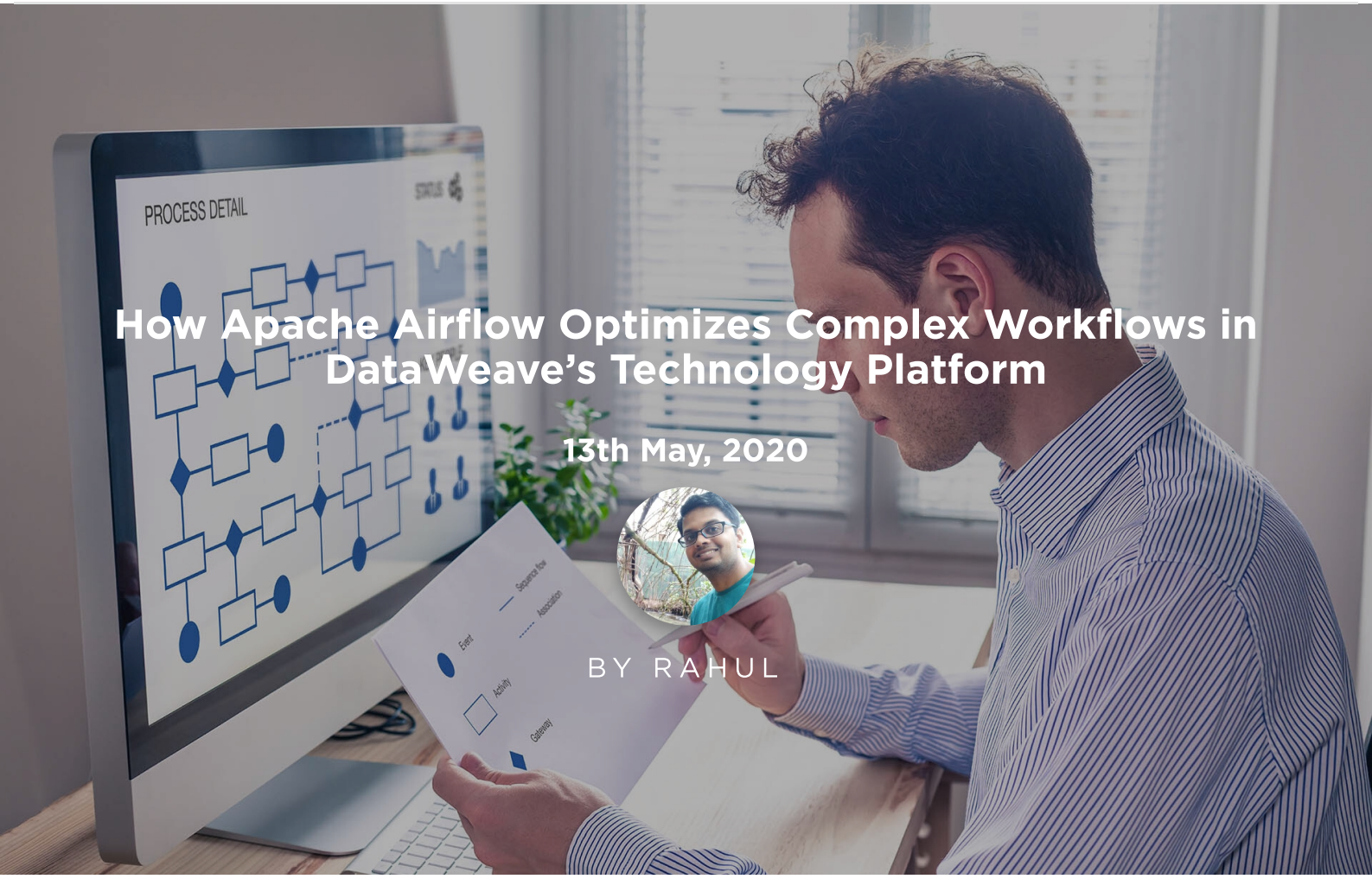


How Apache Airflow Optimizes Complex Workflows in DataWeave's Technology Platform

13th May, 2020

BY RAHUL

As successful businesses grow, they add a large number of people, customers, tools, technologies, etc. and roll out processes to manage the ever-increasingly complex landscape. Automation ensures that these processes are run in a smooth, efficient, swift, accurate, and cost-effective manner. To this end, Workflow Management Systems (WMS) aid businesses in rolling out an automated platform that manages and optimizes business processes at large scale.

While workflow management, in itself, is a fairly intricate undertaking, the eventual improvements in productivity and effectiveness far outweigh the effort and costs.

At DataWeave, on a normal day, we collect, process and generate business insights on terabytes of data for our retail and brand customers. Our core data pipeline ensures consistent data availability for all downstream processes including our proprietary AI/ ML layer. While the data pipeline itself is generic and serves standard workflows, there has been a steady surge in customer-specific use case complexities and the variety of product offerings over the last few years.

A few months ago, we recognized the need for an orchestration engine. This engine would serve to manage the vast volumes of data received from customers, capture data from competitor websites (which can range in complexity and from 2 to 130+ in number), run the required data transformations, execute the product matching algorithm through our AI

systems, process the output through a layer of human verification, generate actionable business insights, feed the insights to reports and dashboards, and more. In addition, this engine would be required to help us manage the diverse customer use cases in a consistent way.

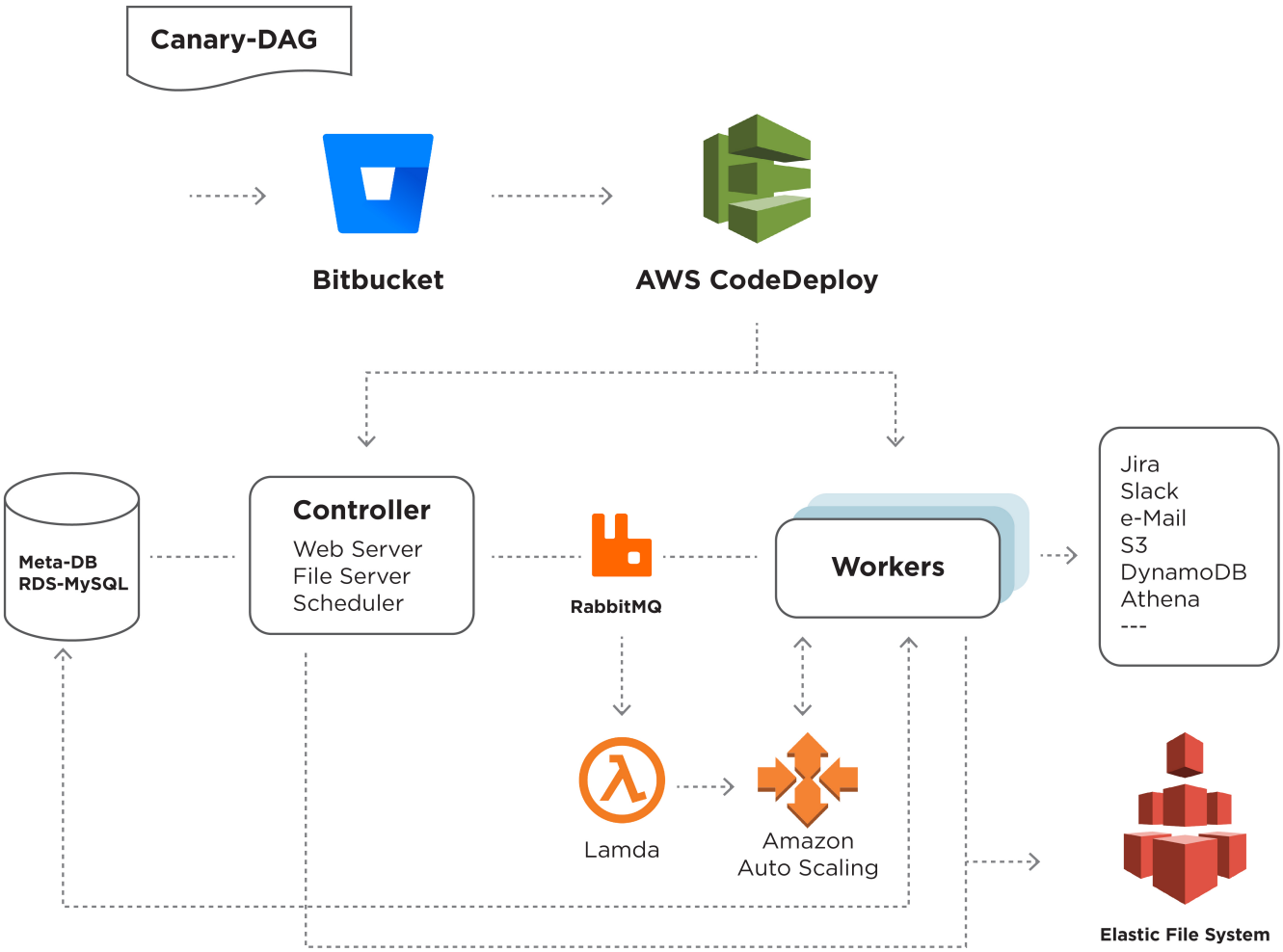
As a result, we launched a hunt for a suitable WMS. We needed the system to satisfy several criteria:

- Ability to manage our complex pipeline, which has several integrations and tech dependencies
- Extendable system that enables us to operate with multiple types of databases, internal apps, utilities, and APIs
- Plug and play interfaces to execute custom scripts, and QA options at each step
- Operates with all cloud services
- Addresses the needs of both 'Batch' and 'Near Real Time' processes
- Generates meaningful feedback and stats at every step of the workflow
- Helps us get away with numerous crontabs, which are hard to manage
- Execute workflows repeatedly in a consistent and precise manner
- Ability to combine multiple complex workflows and conditional branching of workflows
- Provides integrations with our internal project tracking and messaging tools such as, Slack and Jira, for immediate visibility and escalations
- A fallback mechanism at each step, in case of any subsystem failures.
- Fits within our existing landscape and doesn't mandate significant alterations
- Should support autoscaling since we have varying workloads (the system should scale the worker nodes on-demand)

On evaluating several WMS providers, we zeroed in on Apache Airflow. Airflow satisfies most of our needs mentioned above, and we've already onboarded tens of enterprise customer workflows onto the platform.

In the following sections, we will cover our Airflow implementation and some of the best practices associated with it.

DataWeave's Implementation



Components

Broker: A 3 node Rabbit-MQ cluster for high availability. There are 2 separate queues maintained, one for SubDags and one for tasks, as SubDags are very lightweight processes. While they occupy a worker slot, they don't do any meaningful work apart from waiting for their tasks to complete.

Meta-DB: MetaDB is one of the most crucial components of Airflow. We use RDS-MySQL for the managed database.

Controller: The controller consists of the scheduler, web server, file server, and the canary dag. This is hosted in a public subnet.

Scheduler and Webserver: The scheduler and webserver are part of the standard airflow services.

File Server: Nginx is used as a file server to serve airflow logs and application logs.

Canary DAG: The canary DAG mimics the actual load on our workers. It runs every 30 minutes and checks the health of the scheduler and the workers. If the task is not queued at all or has spent more time in the queued state than

expected, then either the scheduler or the worker is not functioning as expected. This will trigger an alert.

Workers: The workers are placed in a private subnet. A general-purpose AWS machine with two types of workers is configured, one for sub-DAGs and one for tasks. The workers are placed in an EC2-Autoscaling group and the size of the group will either grow or shrink depending on the current tasks that are executed.

Autoscaling of workers

Increasing the group size: A lambda is triggered in a periodic interval and it checks the length of the RMQ queue. The lambda also knows the current number of workers in the current fleet of workers. Along with that, we also log the average run time of tasks in the DAG. Based on these parameters, we either increase or decrease the group size of the cluster.

Reducing the group size: When we decrease the number of workers, it also means any of the workers can be taken down and the worker needs to be able to handle it. This is done through termination hooks. We follow an aggressive scale-up policy and a conservative scale-down policy.

File System: We use EFS (Elastic File System) of AWS as the file system that is shared between the workers and the controller. EFS is a managed NAS that can be mounted on multiple services. By using EFS, we have ensured that all the logs are present in one file system and these logs are accessible from the file server present in the controller. We have put in place a lifecycle policy on EFS to archive data older than 7 days.

Interfaces: To scale up the computing platform when required, we have a bunch of hooks, libraries, and operators to interact with external systems like Slack, EMR, Jira, S3, Qubole, Athena, and DynamoDB. Standard interfaces like Jira and Slack also help in onboarding the L2 support team. The L2 support relies on Jira and Slack notifications to monitor the DAG progress.

Deployment

Deployment in an airflow system is fairly challenging and involves multi-stage deployments.

Challenges:

- If we first deploy the controller and if there are any changes in the DAG, the corresponding tasks may not be present in workers. This may lead to a failure.
- We have to make blue-green deployments as we cannot deploy on the workers where tasks may still be running. Once the worker deployments are done, the controller deployment takes place. If it fails for any reason, both the deployments will be rolled back.

We use an AWS code-deploy to perform these activities.

Staging and Development

For development, we use a docker container from Puckel-Airflow. We have made certain modifications to change the user_id and also to run multiple docker containers on the same system. This will help us to test all the new functionality at a DAG level.

The staging environment is exactly like the development environment, wherein we have isolated our entire setup in separate VPCs, IAM policies, S3-Buckets, Athena DBs, Meta-DBs, etc. This is done to ensure the staging environment doesn't interfere with our production systems. The staging setup is also used to test the infra-level changes like autoscaling policy, SLAs, etc.

In Summary

Following the deployment of Airflow, we have onboarded several enterprise customers across our product suite and seen up to a 4X improvement in productivity, consistency and efficiency. We have also built a sufficient set of common libraries, connectors, and validation rules over time, which takes care of most of our custom, customer-specific needs. This has enabled us to roll out our solutions much faster and with better ROI.
As Airflow has been integrated to our communications and project tracking systems, we now have much faster and better visibility on current statuses, issues with sub processes, and duration-based automation processes for escalations.
At the heart of all the benefits we've derived is the fact that we have now achieved much higher consistency in processing large volumes of diverse data, which is one of DataWeave's key differentiators.
In subsequent blog posts, we will dive deeper into specific areas of this architecture to provide more details. Stay tuned!

- Rahul Ramesh

Technical Architect at DataWeave, 13th May, 2020

DATA ENGINEERING E COMMERCE