Spinner

Next Gen Workflow Engine on K8s at Pinterest

July 2020 **Workflow Platform Team**

Agenda

1. WHY
2. HOW
3. WRAP-UP
4. Q/A

01

WHY

WHY (Pain Points)

- **Performance and scalability issues**
  - Schedule / job start delay
  - Stateful component not being able to scale.
  - Centralized MetaStore service as SPOF
- **High maintenance cost**
  - Infra upgrades (ubuntu etc)
  - Multiple clusters (due to load partitioning)
  - Supervised worker slot scale up.
- **Features missing**
  - ACL and Auditing.
  - Execution stats virtualization
  - Code isolation.
- **Deep learning overhead**
  - Mostly in house project with own DSL
  - Insufficient doc

WHY (Why Airflow)

- **Industry standard**
  - Widely adopted
  - Good document support
  - Active community

- **Good feature alignment**
  - Python based.
  - Contains a lot of demanding features out of box.
  - Modularized and has standardized interfaces.
- **Able to scale**
  - Stateless UI.
  - Partitionable Scheduler (with modification)
  - Able to leverage kubernetes to scale task load.

- **Good reputation :)**

02

HOW

Workflow Scale @ Pinterest

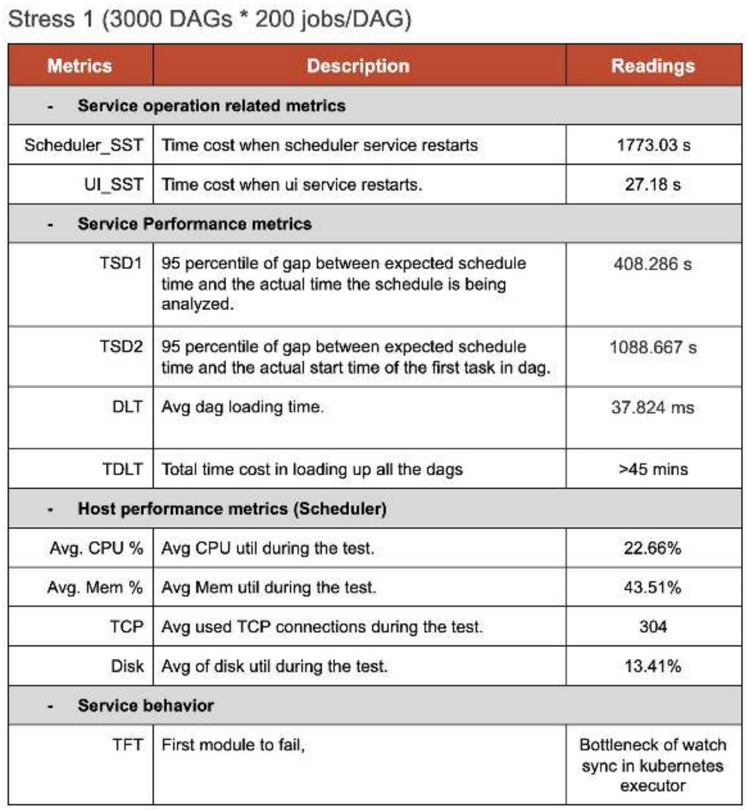
10+
CLUSTERS

4K+
TOTAL FLOWS

10K+
DAILY FLOW EXECUTION

38K+
DAILY JOB EXECUTION

HOW (POC and Test)



Stress 1 (3000 DAGs * 200 jobs/DAG)

Metrics	Description	Readings
- **Service operation related metrics**		
Scheduler_SST	Time cost when scheduler service restarts	1773.03 s
UI_SST	Time cost when ui service restarts.	27.18 s
- **Service Performance metrics**		
TSD1	95 percentile of gap between expected schedule time and the actual time the schedule is being analyzed.	408.286 s
TSD2	95 percentile of gap between expected schedule time and the actual start time of the first task in dag.	1088.667 s
DLT	Avg dag loading time.	37.824 ms
TDLT	Total time cost in loading up all the dags	>45 mins
- **Host performance metrics (Scheduler)**		
Avg. CPU %	Avg CPU util during the test.	22.66%
Avg. Mem %	Avg Mem util during the test.	43.51%
TCP	Avg used TCP connections during the test.	304
Disk	Avg of disk util during the test.	13.41%
- **Service behavior**		
TFT	First module to fail,	Bottleneck of watch sync in kubernetes executor

- **MVP / POC cluster setup**
  - Branched from 1.10.stable (1.10.4.rc3).
  - Two UI node one Scheduler (c5.2xlarge).
  - MySQL(1 master/1 slave, c5d.4xlarge).
  - Customized POC k8s Executor.
  - Worker pod with 300 Mhz CPU, 300MB requested mem/1G max.

- **Perf / load / stress test**
  - Measures performance in light load.
  - Checks to which extent scale is needed.
  - Estimate the cluster scope we need.
  - Identify the bottleneck and try to address it.

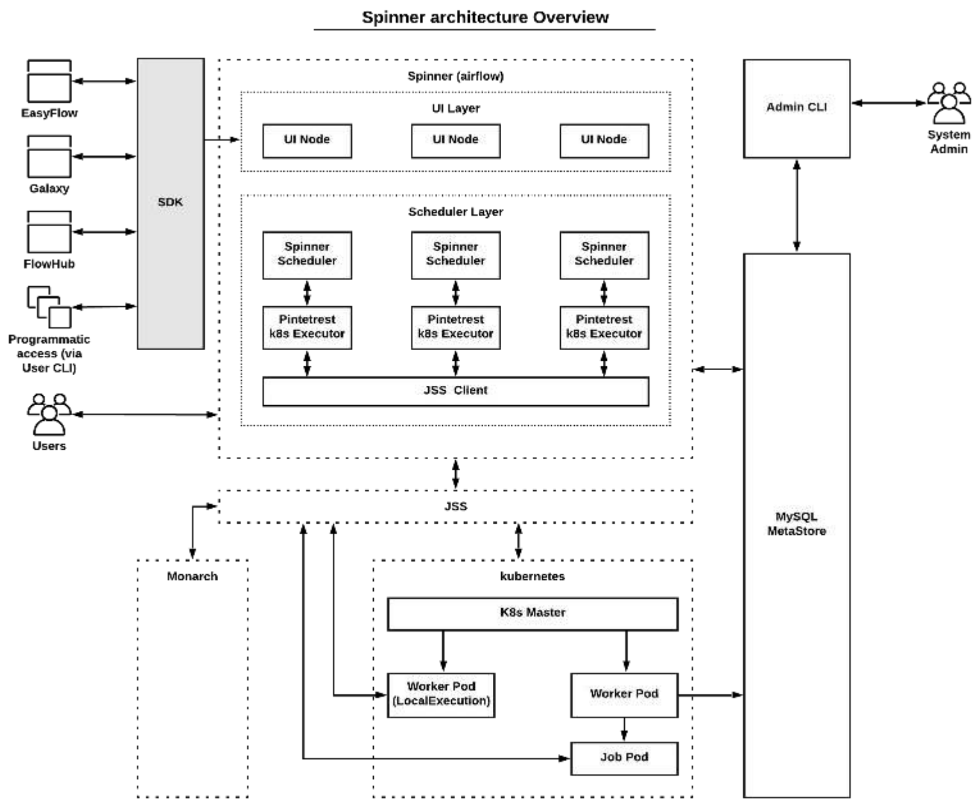
HOW (Design)

- **Single entry point w/ Multiple schedulers.**
  - Unified UI contains all the dags.
  - Multi schedulers providing load partitioning and tier support.

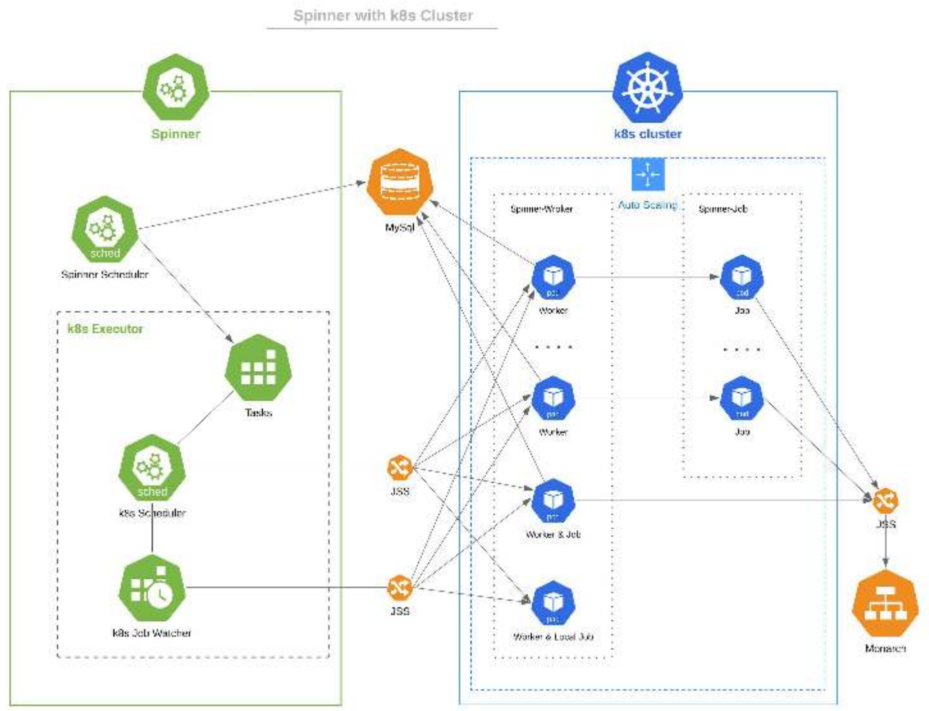
- **K8s Integration for execution scaling.**
  - Rewrite/redesign k8s Executor to work with the Pinterest internal k8s setup.

  .
- **CI/CD pipeline provides infra/user code isolation**
  - All modules dockerlized.
  - Dags/infra code in different repro.

- **One-script-away test env setup**
- **DLAC based access control and backup from UI**
- **Automated Job migration from old system**

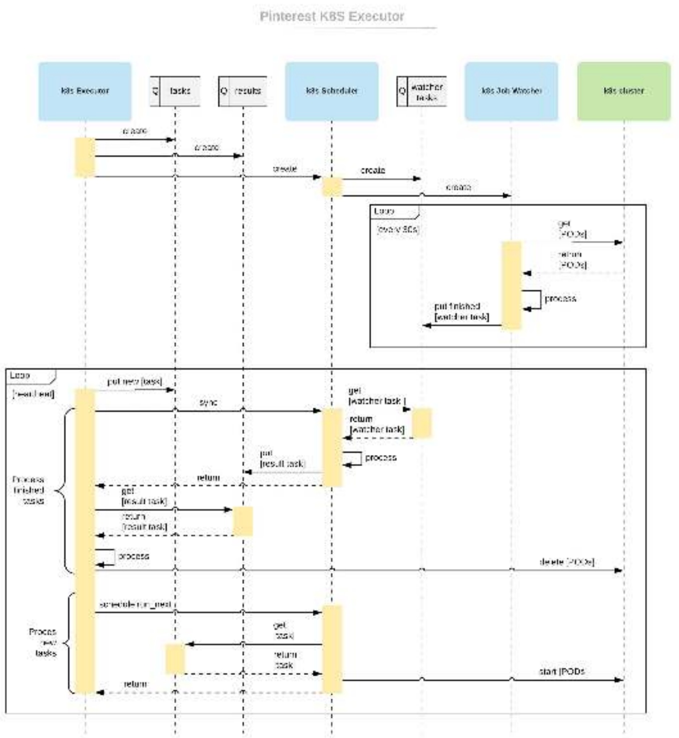


HOW (Onboarding to K8s)



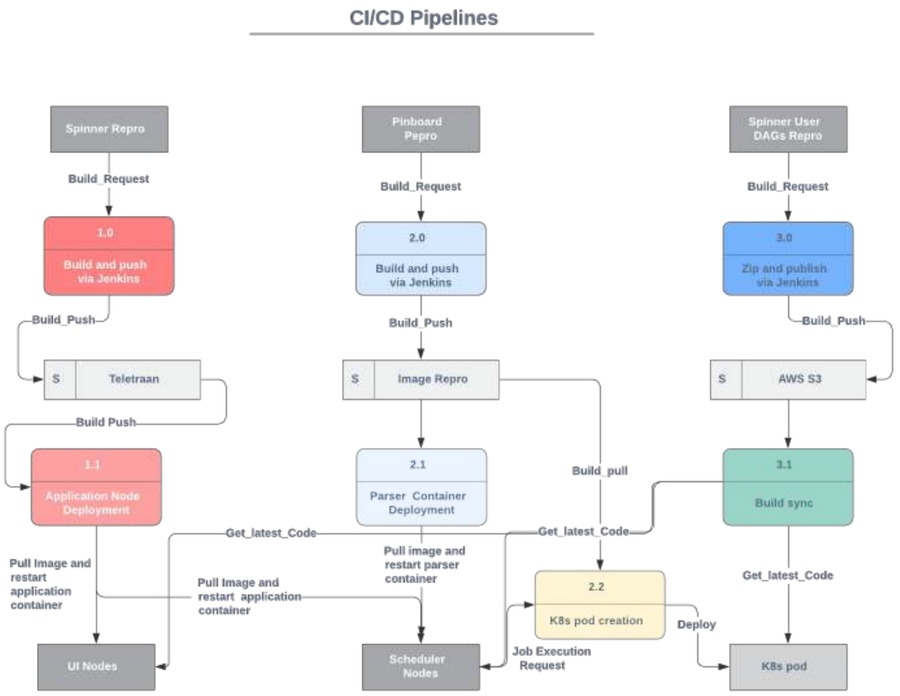
- **Rebuild the K8s Executor to accommodate the Pinterest K8s Setup.**
  - Rewrite all the k8s APIs to interact with Pinterest inhouse K8s setup.
  - Retain the basic business logic from open source K8s Executor.
- **Full Runtime isolation and ability to scale up/down**
  - Each task runs in their own pod (migration case takes two pods).
  - Pods share k8s nodes with other services.
  - Pods get removed right after task execution.
- **Rebuild K8s Operator.**
  - Support Pod over Pod scenario (mainly used by migration cases). User can choose to bring their own image to execute jobs.

HOW (K8s Executor continued)



- **Rebuild the k8s pod status pulling mechanism**
  - Create Pinterest Watcher to pull pod status due to Pinterest k8s setup.
  - Use labels to isolate PODs started by different schedulers.
- **Pod customization**
  - Combine dag_id, task_id, execution date, try number info to create meaningful Pod name.
  - Pod customization with different requirements (Pod Env, Image, resources, packages.)
- **Pod readiness check**
  - S3, Mysql, Knox
- **Rebuild/Refine the log pulling mechanism**
  - Upload logs to s3 after task completion.
  - Runtime logs get synced to Pinterest ES.
  - UI pulls runtime logs from Pods or Pinterest ES. (Works for Pod failed case before executing task as well)

HOW (CI/CD Pipeline)



- **Separate repositories for infra and user code**
  - Decouple the deployment cycle.

- **Infra CI/CD**
  - **Infra code changes trigger a Jenkins job to make build and push the new image to registry.**
  - **Jenkins job notifies Teletraan to refresh the image/containers on the VM nodes.**
  - **K8s Pods will use the same image as scheduler to execute new tasks.**
  - **User doing local test will be able to load the latest image to test out their dag.**

- **Dag CI/CD**
  - **User code changes trigger a Jenkins job to sync dags to S3.**
  - **Daemon runs on the VM nodes (scheduler) syncs the file from S3 in a very short interval (30 sec).**
  - **Worker pod also syncs latest dags from S3.**

HOW (Enforced DLAC)

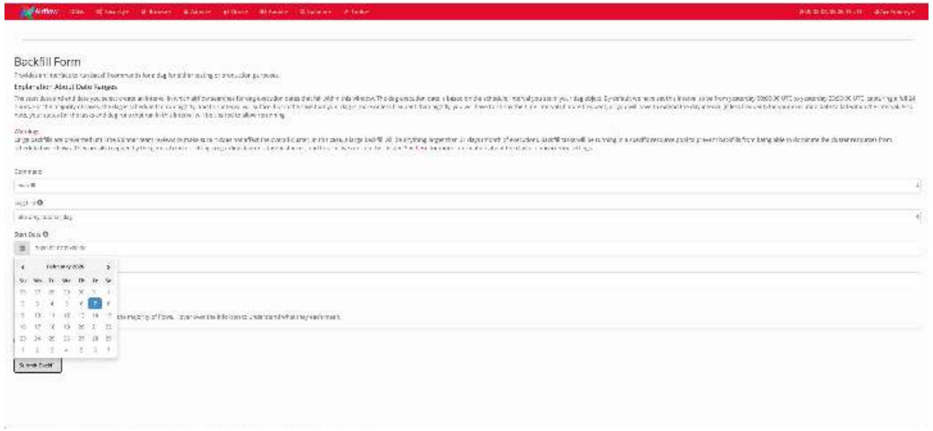
```
dag = DAG('pinterest_hadoop_dag_spinner_tier_3_0',
        catchup=False,
        default_args=default_args,
        schedule_interval=timedelta(minutes=60),
        access_control={'spinner-users':
{'can_dag_read','can_dag_edit'}},
        doc_md=doc_md,
        project='workflow')
```

```
# The default user self registration role
AUTH_USER_REGISTRATION_ROLE = "spinner-users"
```

```
# When set to True the access_control field of the DAG will be
analyzed and synced to the DB when the
# DAG object is constructed, otherwise the sync will be deferred to
when of the following happens:
# 1. the web app is reconstructed (usually happens when web server
restarts)
# 2. dag manually refreshed in UI (via the refresh button)
# 3. sync_perm call via CLI.
early_dag_perm_sync = False
```

- **Enforced DLAC for every dag**
  - Access_control field is mandatory.
  - Default role is assigned during users first login.
  - Pre-populated roles for different team/org.
  - Manual approval user-role assignment (automation pending)
- **Early dag perm sync.**
  - Dag access info is synced to DB when the dag is parsed by scheduler.
- **Filtered dag list for individuals.**
  - Can only see dags opened to “spinner-users”.
  - Can only see dags granted access to a role user is assigned.
- **Action Auditing.**

HOW (Backfill from UI)



Backfill Form

[illegible]

Explanation: About Date Ranges

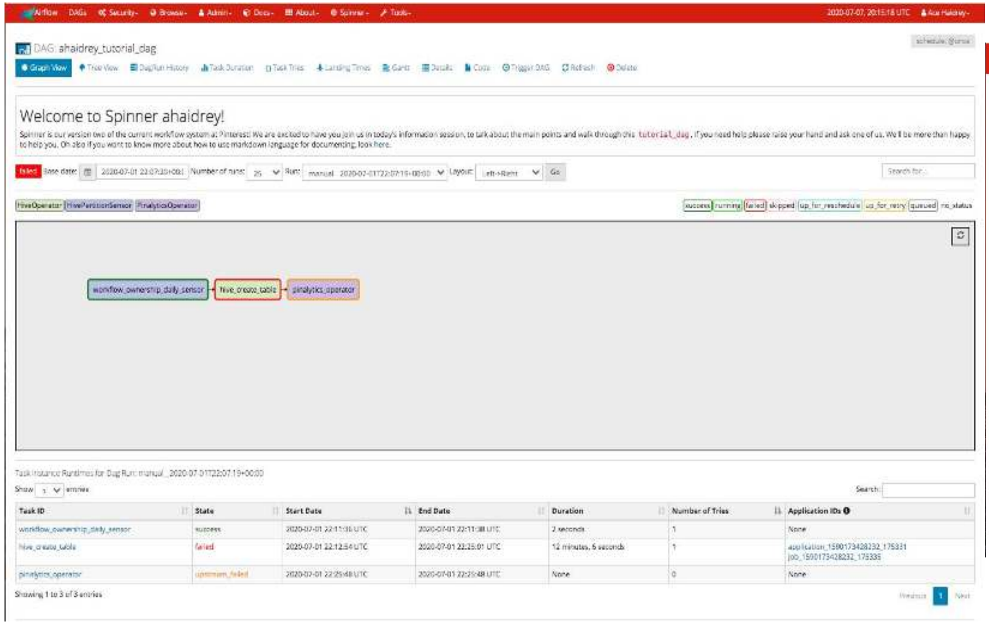
[illegible]

Warnings

[illegible]

- **One Stop Service Portal**
    - No learning cost to user as compared to how to use the CLI tool.
    - No concerns regarding releasing DB connection credentials to user if backfill can only be performed via CLI.
    - Fully utilizing the RBAC/DLAC from UI:
        - Admin controls who can access backfill page.
        - Users can only backfill for DAGs they have write access to.
    - Utilize different pool to separate the backfill traffic out from the production.
    - Set limits on the max sized backfill a user can run (overridable by admins)
    - Uses cluster configs for resources (same regulations as scheduled runs)

HOW (Dag & Task Run Info UI)



Welcome to Spinner ahaidrey!

Spinner is our version two of the current workflow system at Pinterest! We are excited to have you join us in today's information session, to talk about the main points and walk through this `tutorial_dag`. If you need help please raise your hand and ask one of us. We'll be more than happy to help you. Oh also if you want to know more about how to use markdown language for documenting, look here.

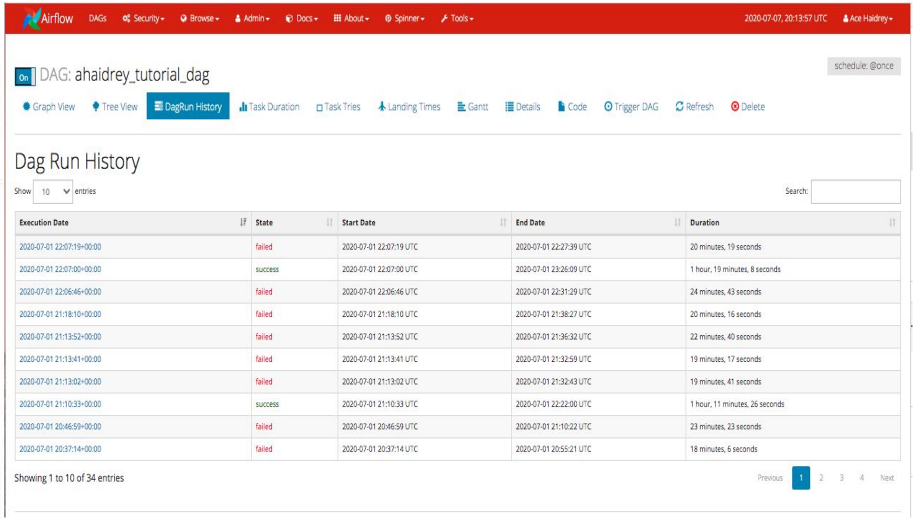
Task Instance Runtimes for Dag Run: manual__2020-07-01T22:07:19+00:00

Show 5 entries Search:

Task ID	State	Start Date	End Date	Duration	Number of Tries	Application IDs
workflow_ownership_daily_sensor	success	2020-07-01 22:11:36 UTC	2020-07-01 22:11:38 UTC	2 seconds	1	None
hive_create_table	failed	2020-07-01 22:12:54 UTC	2020-07-01 22:25:01 UTC	12 minutes, 6 seconds	1	application_1590173428232_175331 job_1590173428232_175335
pinalytics_operator	upstream_failed	2020-07-01 22:25:48 UTC	2020-07-01 22:25:48 UTC	None	0	None

Showing 1 to 3 of 3 entries

Previous 1 Next



Dag Run History

Show 10 entries Search:

Execution Date	State	Start Date	End Date	Duration
2020-07-01 22:07:19+00:00	failed	2020-07-01 22:07:19 UTC	2020-07-01 22:27:39 UTC	20 minutes, 19 seconds
2020-07-01 22:07:00+00:00	success	2020-07-01 22:07:00 UTC	2020-07-01 23:26:09 UTC	1 hour, 19 minutes, 8 seconds
2020-07-01 22:06:46+00:00	failed	2020-07-01 22:06:46 UTC	2020-07-01 22:31:29 UTC	24 minutes, 43 seconds
2020-07-01 21:18:10+00:00	failed	2020-07-01 21:18:10 UTC	2020-07-01 21:38:27 UTC	20 minutes, 16 seconds
2020-07-01 21:13:52+00:00	failed	2020-07-01 21:13:52 UTC	2020-07-01 21:36:32 UTC	22 minutes, 40 seconds
2020-07-01 21:13:41+00:00	failed	2020-07-01 21:13:41 UTC	2020-07-01 21:32:59 UTC	19 minutes, 17 seconds
2020-07-01 21:13:02+00:00	failed	2020-07-01 21:13:02 UTC	2020-07-01 21:32:43 UTC	19 minutes, 41 seconds
2020-07-01 21:10:33+00:00	success	2020-07-01 21:10:33 UTC	2020-07-01 22:22:00 UTC	1 hour, 11 minutes, 26 seconds
2020-07-01 20:46:59+00:00	failed	2020-07-01 20:46:59 UTC	2020-07-01 21:10:22 UTC	23 minutes, 23 seconds
2020-07-01 20:37:14+00:00	failed	2020-07-01 20:37:14 UTC	2020-07-01 20:55:21 UTC	18 minutes, 6 seconds

Showing 1 to 10 of 34 entries

Previous 1 2 3 4 Next

Task Instance Table

- Added under graph view to give clear metadata info of the executions.
- Application ids of cluster submissions (extra links)

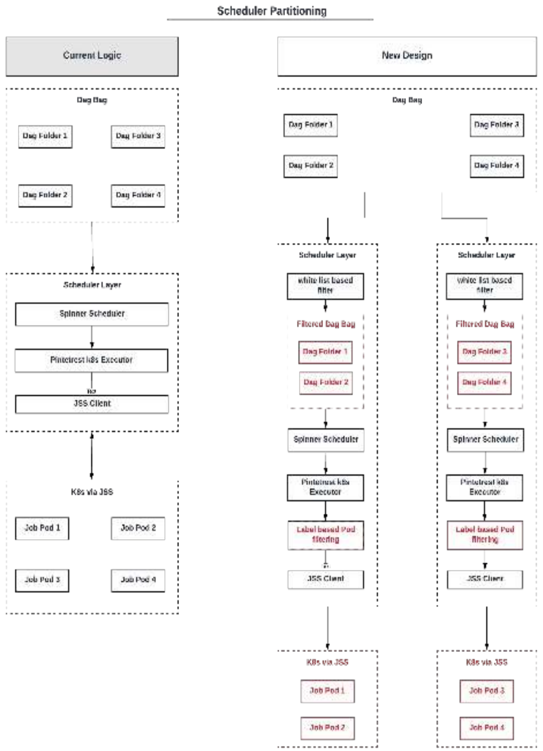
Dag Run History View

- Added as new tab to give dag run history. Current Browse->DR view not easy to use & missing duration info

HOW (Multi-Scheduler)

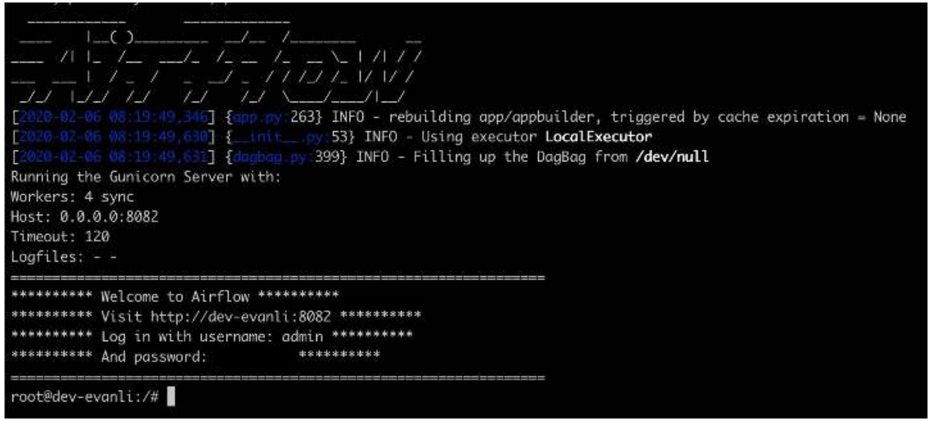
- **Rule Based Dag Partitioning**
  - Customized rule based partitioner maps a given dag to a specific partition (currently is based on the dag file path)
  - Multi scheduler nodes, with each monitoring a specific partition and processing DAGs belong to its assigned partition.
  - UI will render all DAGs despite the partition, when task is reset manually from UI, UI will make sure the TI is submitted to the correct partition so the corresponding scheduler can process it.

- **Tiering Support**
  - Flows with different priority will be put into different partitions managed by different scheduler nodes



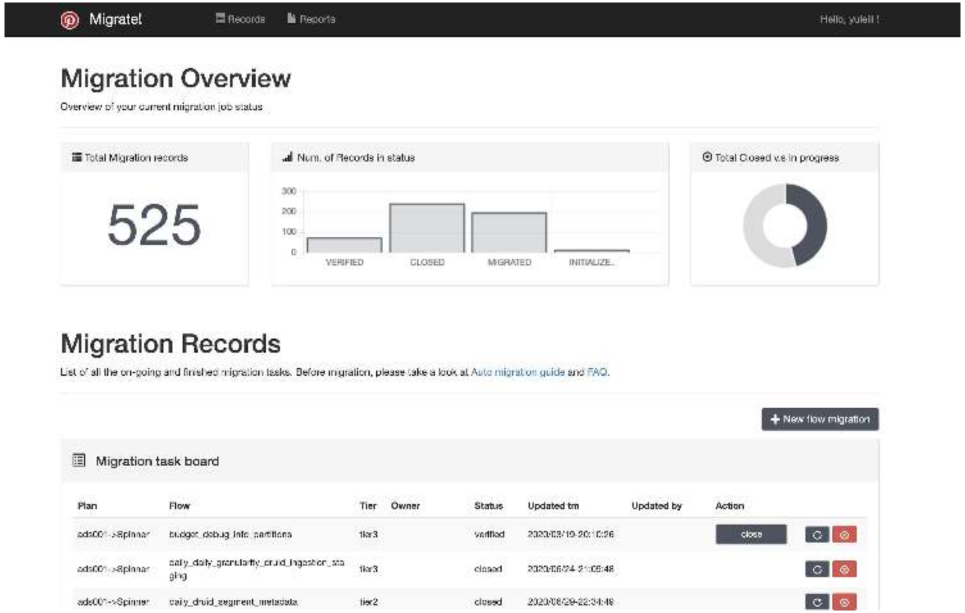
HOW (Testing)

- **One-script-away test environment setup**
    - Script included in the user dag repro
    - One bash command to pull the latest image, setup UI and scheduler on local box.
    - View DAG layout in UI and test out DAG execution via localExecutor.



```
  ____________       _____________
 ____    |__( )_________  __/__  /________      __
____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
___  ___ |  / _  /   _  __/ _  / / /_/ /_ |/ |/ /
 _/_/  |_/_/  /_/    /_/    /_/  \____/____/|__/
[2020-02-06 08:19:49,346] {app.py:263} INFO - rebuilding app/appbuilder, triggered by cache expiration = None
[2020-02-06 08:19:49,630] {__init__.py:53} INFO - Using executor LocalExecutor
[2020-02-06 08:19:49,631] {dagbag.py:399} INFO - Filling up the DagBag from /dev/null
Running the Gunicorn Server with:
Workers: 4 sync
Host: 0.0.0.0:8082
Timeout: 120
Logfiles: - -
=================================================================
********** Welcome to Airflow **********
********** Visit http://dev-evanli:8082 **********
********** Log in with username: admin **********
********** And password:           **********
=================================================================
root@dev-evanli:/#
```

HOW (Automated Migration)



Migration Overview

Overview of your current migration job status

Total Migration records

525

Num. of Records in status

Total Closed v.s in progress

Migration Records

List of all the on-going and finished migration tasks. Before migration, please take a look at Auto migration guide and FAQ.

Migration task board

Plan	Flow	Tier	Owner	Status	Updated tm	Updated by	Action
ads001->Spinner	budget_debug_info_partitions	tier3		verified	2020/03/19-20:10:26		close
ads001->Spinner	daily_daily_granularity_druid_ingestion_staging	tier3		closed	2020/06/24-21:05:48		
ads001->Spinner	daily_druid_segment_metadata	tier2		closed	2020/08/29-22:34:48		

- **Not a direct translation to Airflow DSL**
  - User logic and dependencies made direct DSL translation impossible.
  - Customized scheduler to extract flow layout from existing pinball workflows.
  - Rebuild Airflow Dag on the fly.
  - Utilize dag serialization so UI and worker doesn't need physical Dag files for rendering and execution.

- **Automated migration tool with "click to migrate"**
  - UI based tool for user to easily migrate existing flows on pinball to spinner.
  - Easy rollback if things doesn't workout for the first attempt.

03

WRAP-UP

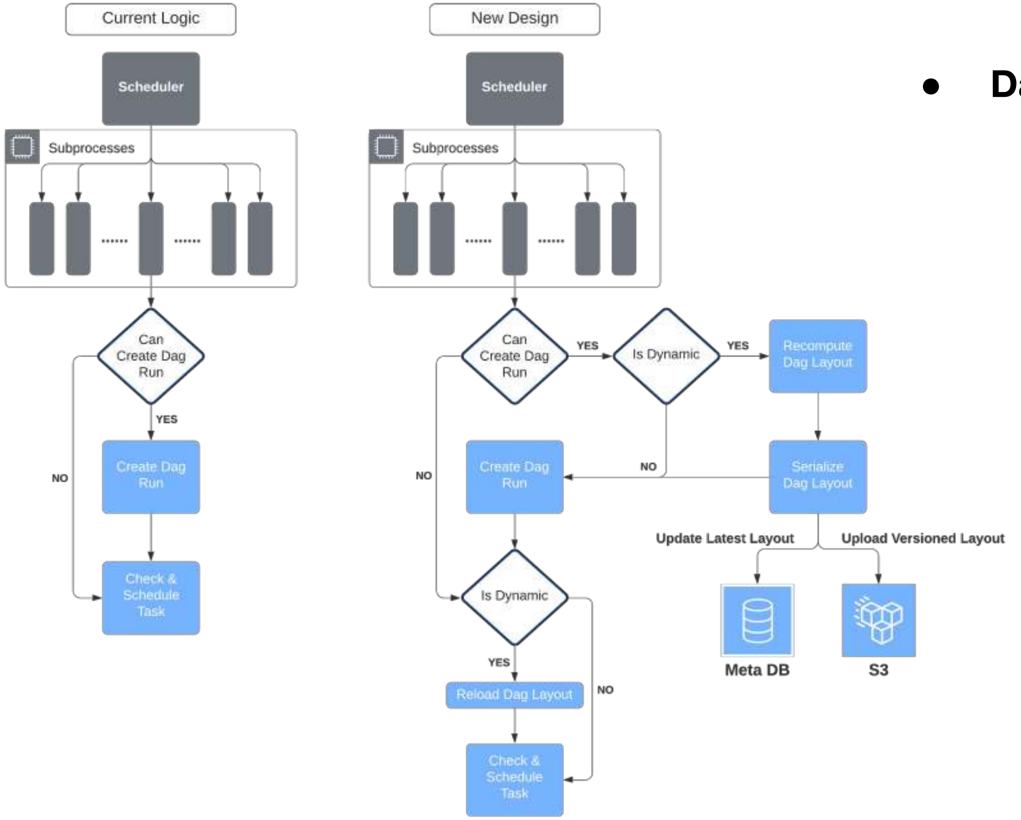
WRAP-UP (Where We Are)

- **Spinner GAed Late December 2019.**
  - Shipped Airflow 1.10.stable merged with pr_5743.
  - Two production envs each has 2 webserver nodes with 6 schedulers.
  - Running on k8s infrastructure.

- **Migration planned and ongoing.**
  - Migration from pinball is ongoing and gained user support.
  - 800+ production flows have been migrated to spinner (June 2020).

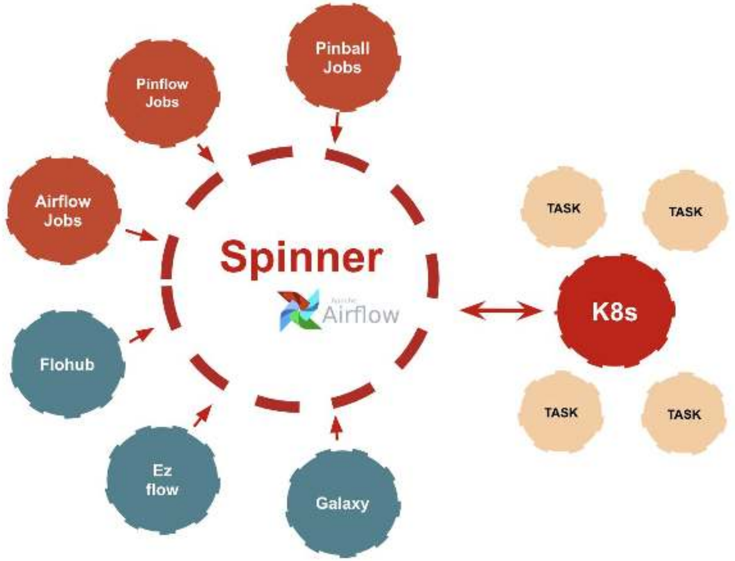
- **User trainings ongoing.**
  - ~30 new operators (June 2020).
  - +80 native flows onboard (June 2020).
  - ~600 registered internal users (June 2020).

WRAP-UP (Next Step)



- **Dag Versioning**
  - Support use case where dag layout is generated dynamically and may vary between scheduler scanning.
  - Not supported natively by Airflow.
  - Compute & serialize the dag layout each time a dagrun is created and uses execution_date as versioned dag layout id.
  - Modify scheduler/webserver/worker logic to fetch (deserialize) versioned dag layout when needed.

WRAP-UP (Next Step)



- **IDE Auto Templating**
  - Enable users to compose dag from templates
- **New Operators and feature pairing**
  - Support more user cases and business logic.
  - New features to ease user onboarding and improve system stability and scalability.
- **System Integration**
  - Support systems based on workflow engine.
  - SDK/API support
- **Contribute**

04

Q/A

THANK YOU