Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2024, 11(9):17-21**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Orchestration Tool - A Holistic Comparison of GCC vs Astronomer Airflow

Rameshbabu Lakshmanasamy

Senior Data Engineer, Jewelers Mutual Group

Abstract: There is a comprehensive list of Orchestration tools both open source and proprietary options in the marketplace like Apache Airflow, Nifi, Oozie, etc. In this article, am going to discuss and compare about two of the very popular managed flavors of Apache Airflow – "Astronomer Airflow" & "Google Cloud Composer". Both are managed versions of Airflow by two popular providers. We will discuss about the pros and cons in each, and our experiences with both these flavors, and what would suit for different use cases. Let's dive in.

Keywords: Apache Airflow, Google Cloud Composer, Astronomer, Orchestration tool, GKE, Pricing, Auto Scaling, Managed airflow, Kubernetes executor, Celery executor

1. Introduction

Orchestration tools in IT are software solutions designed to automate and coordinate workflows, complex tasks, and handle processes across multiple interfacing applications / systems. In any modern IT infrastructure management, particularly in cloud computing and DevOps environments, this plays a very crucial role. IT teams need this for various use cases like (1) automating repetitive tasks/workflows (2) coordinating multiple services/systems/apps (3) improve effectiveness and reduce human error (4) enabling scalability and flexibility in IT ops. In this article, let us dive into comparing two such orchestration tools that are market leaders and weigh it's pros and cons.

2. Background

In our datalake, all our workflows were originally orchestrated in Google Cloud Composer (GCC), the managed airflow from Google's GCP. It was basically hosted in "Cloud Composer 2" version of GCC. Looking at the humongous monthly costs from GCC usage, our team decided to explore other vendors and narrowed down on Astronomer Airflow, another popular managed version. And after lots of brainstorming /analysis with endless discussions, we migrated to Astronomer and we were right to switch. Astronomer came with more flexibility for our use cases, and our cost savings was 48% post migration, for idential number of workflows/DAG(s).

Though both are to help create airflow environments quickly and gives us access to powerful airflow native tools of UI & Command Line tool (so the development team can put complete focus on DAGs and not on the infrastructure), there are many differences in what they offer in terms of infrastructure to choose from. Am going to list some of those features, and I hope this helps in teams that have similar dilemma on what to pick among these two – Astronomer Airflow or Google Cloud Composer?

3. Key factors

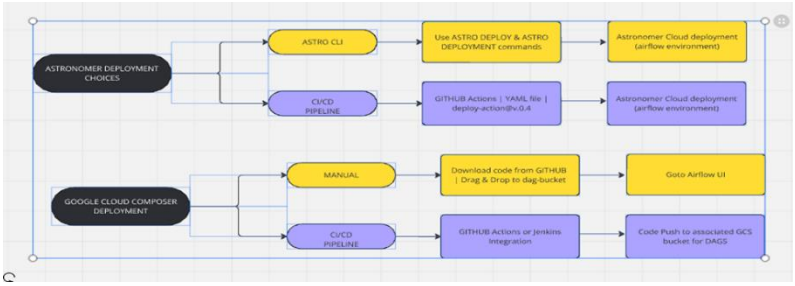
The key factors we are going to consider to compare are Hosting-Management, Infrastructure, Flexibility, Airflow Versioning, Customization, Scaling, Integration, Development, Pricing, Support to name a few.

Deployment: Astronomer offers both self-hosted and fully managed via astro cloud, whereas GCC is a fully managed service in GCP. In Astronomer you first start off with creating a dedicated cluster (specific for the Organization), which will be used in one or more 'deployments' (a.k.a. airflow environments) that follows next. In Astronomer terms, 'deployment' means an 'Airflow instance'. To elaborate more, an organization may have multiple deployment instances (e.g. dev-deployment, qa-deployment, prod-deployment) - first two are nonprod, and last one is prod as name suggests. All the above three usually are spun up from the same dedicated cluster created for Organization. The code deployment to these environments can be through one of the following options

(1) Astro CLI: Astro CLI is very convenient command line tool that helps manage workspaces, deployments, code pushes etc. A great documentation to help is available in astronomer website.

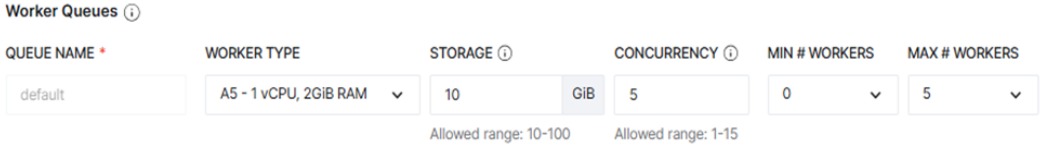
(2) CI/CD: If you opt for CI/CD enforced deployment, this is super convenient and the astronomer provided python package helps with seamless integration of GITHUB-ACTIONS -> Astronomer. The management of deployment is super convenient and code upgrades are way easier. All one has to do is update the dockerFile configuration to point to latest astronomer runtime and it will upgrade the 'deployment' in few minutes. The complete code will reside in github repository in this case, and astronomer doesn't give access to the artifacts in the cloud-console.

Coming to GCC management, its all managed by GCP platform, and one will go to console and just create a composer environment. In GCC, each composer environment comes with associated dag bucket, where the workflow artifacts resides. So you have access to the artifact via GCS bucket. Each composer environment gets its own GKE cluster ensuring isolation between environments. Complete GKE infrastructure is managed by google like security patches and node updates. Code deployment can be manual move to the dag folder or one can integrate via Jenkins or equivalent tool. One can also create a YAML file in github-actions, and setup with the help of Service Account (and JSON Key generated for this), environment, location (specific to the composer environment we would like to integrate) to accomplish this.

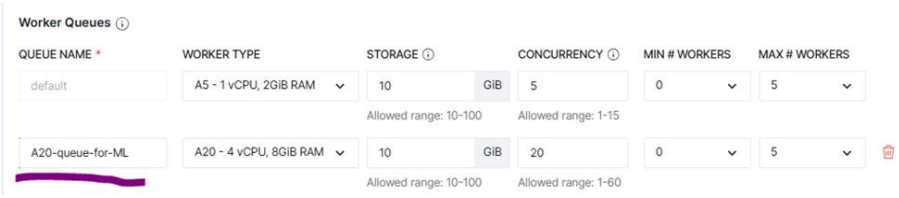


Executors: GCC runs on GKE which provides more isolation, and tightly integrated with GCP and managed by google. In this regard, Astronomer comes with more granular options of (1) Celery executor (2) Kubernetes executor.

"Celery Executor" - Celery is one of the highly preferred choice for production environments for its robustness and scalability. This basically comes with scheduler, workers and a metadata db. Scheduler component determines the ready-to-be-executed tasks and queue it to message brokers for the 'workers' to pick up. Multiple workers executes tasks simultaneously. Executors adds/removes workers dynamically to aid the scalability. This is distributed, scalable and resilient. The default worker queue configuration comes with A5 machine. Other parameters to choose are Storage, Concurrency, Min and max workers in any point in time.



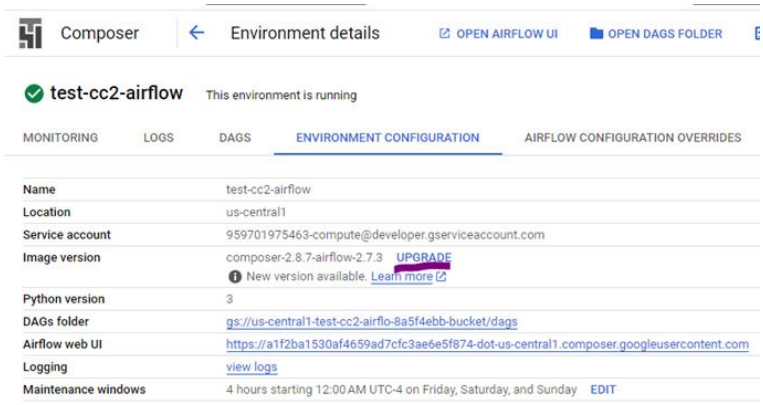
Ideally, for most workflows A5 will be more than enough. For resource intensive tasks that asks for more powerful workers, there is a choice to add more queues and the specific task in dag can be made to use the queue created. For e.g. below queue uses A20 type.



More powerful the worker type, the cost escalates respectively.
"Kubernetes Executor" – Here pods are created dynamically for each task instance which runs in an isolated environment. Pods are terminated once done thereby free up the resources. This is also used when there is a need for a custom docker image and resource intensive tasks.

Scaling: In GCC, worker scaling is automatically happening with HPA (Horizontal Pod Autoscaling), which auto adjusts the number of pods based on utilization and metrics. There is an option to set minimum and maximum worker counts. Composer 2 comes with more efficient autoscaling algorithm over Composer 1.
Astronomer scaling happens by adding or removing worker pods dynamically as needed by celery executor, and in case of kub executor, it spins up pods for each task. With Celery executor, you can control the worker types to use by creating custom-queues as showed in previous section.
Consider the example scenarios –
Sample-queue-1: A20-4vCPU-8GiB RAM – Concurrency 4
Sample-queue-2: A40-8vCPU-16GiB RAM – Concurrency 10
Sample-queue-3: A20-4vCPU-8GiB RAM – Concurrency 2
Sample-queue-4: A40-8vCPU-16GiB RAM – Concurrency 4
In above options, both the sample-queue-3 & sample-queue-4 allocates same level of resources for each task. However, keep in mind that if a single task needs more compute than the worker, the task might need to use queue-4 than the queue-3.
"Usage" metrics in billing section tells us the resource utilization graphically. Based on that we can adjust the concurrency factor as well.

Version Upgrade: Whichever environment is upgraded; it is very important to apply the upgrade in test environment to make sure all dags and associated python packages are compatible with the new version. There are always changes in the python packages and one cannot afford to assume and move to production without checking that compatibility. This applies to both google and astronomer versions.
Astronomer: Upgrade here is very easy. All you will do is point to the newest runtime version that you would like in the dockerfile (a config file provided by astronomer), and once deployed, it rebuilds the docker image and in minutes you will have a airflow instance with the mentioned upgrade. The packages needed for dags are controlled by the requirements.txt (another config file in the root folder of astronomer). The OS level packages that needs to be installed similarly are controlled in packages.txt. For e.g. the installs of vim, curl, gnupg etc goes in packages config file, where as the python packages like pysftp, pyodbc, pymongo, thefuzz etc goes in requirements config file. Again one has to do all these in test environment with upgraded runtime version to ensure compatibility.
(picture of packages and requirements side by side here)
GCC: Here upgrades can be performed via UI or gcloud CLI. There are commands provided by google to run compatibility test. Set the downtime, turn off all dags, and upgrades can be performed. The packages here is controlled in the UI of composer environment.



Pricing: In case of GCC, the cost of various parameters are different between various regions. For e.g. cpu compute cost for IOWA (us-central1) is 0.045$ per 1000 mCPU hours, whereas for Johannesberg (Africa-south1) it is 0.059$ per 1000 mCPU hours.

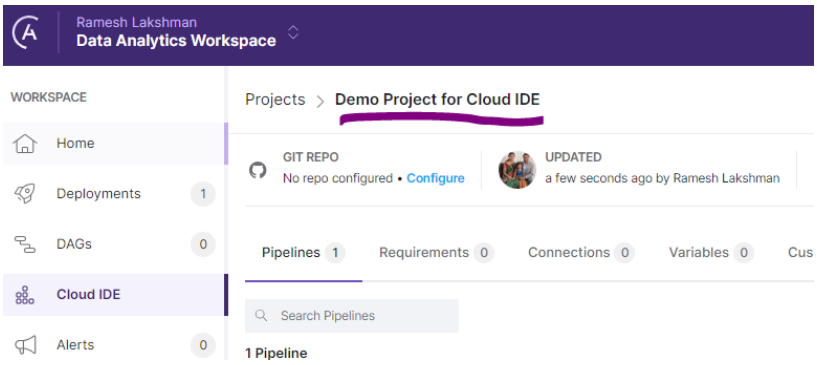
When it comes to Astronomer, the pricing is based on Cluster, Deployments active, Workers utilized. A dedicated cluster comes with a cost of $4 USD per unit, Deployments (airflow instances) comes with 0.35$ to 1.54$ (per hour) depending on the size of scheduler, and then worker costs range from 0.13$ per hour(A2) to $4.16 per hour (A160).

Access management: GCC access management is controlled through Google's IAM that allows you to manage access at project, environment and resource level. One can have fine-grained access control through custom roles. Otherwise you can use the predefined roles like Composer Admin, Composer Worker, Composer User, Composer Viewer etc. Other googles services can be accessed using the associated service account that comes with the composer. GCC has more robust and customizable access controls.

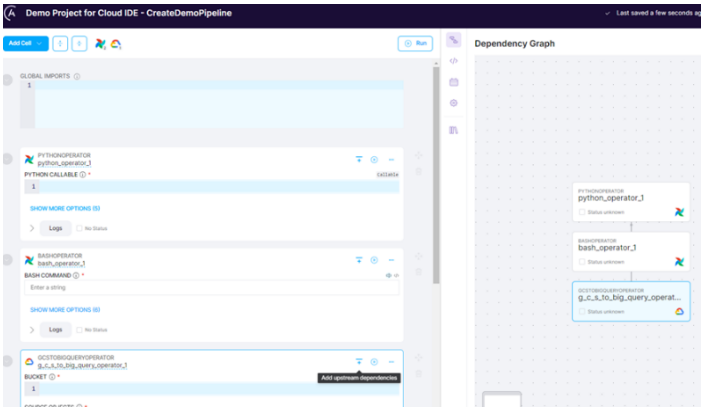
In the case of astronomer, the access is controlled only at the Workspace level (Group of Deployments), meaning multiple airflow instances role up to a Workspace. So one cannot have different access level specific to a deployment over the another one under the same workspace. However one can have any number of workspaces created and tie the deployments accordingly. For e.g. a prod and dev deployment cannot co-exist under same workspace, as we cannot have distinct access types. We had to separate them into two different workspaces so the users can have only 'read/view' on prod, and 'full access' on dev in this example. However, the ENTERPRISE package of Astronomer comes with the ability to grant RBAC roles (meaning – we can have deployment level custom access setup)

Cloud IDE (astronomer): This is a cloud based, notebook-inspired IDE from astronomer (currently it is available for free), to develop quickly with lots of operators available in drop down to choose from, and help members that doesn't know airflow as well to write code and test quickly. This is a super cool feature from Astronomer which is not available in GCC.

You will first create a 'Project', and then a 'Pipeline' to start using this feature in the 'Cloud IDE' section on the left in cloud console.



Below sample screen shows how convenient this one is. ‘Add Cell’ will give you the options to add tasks with operators listed. ‘Global imports’ is to add all the import statements needed pre-requisite. Right side dependency graph shows the tasks and dependency will be set/configured in the left side at the tasks itself.



4. Conclusion

From above analysis, it is evident that both has it’s own pros and cons, and we may choose based on what better suits to our IT infrastructure and goals. In our case, the switch helped save cut down costs by 48%, and there were no compromises. If in future for any resource intensive DAGs, we would opt for task that uses ‘kubernetes operator’ instead of default ‘celery operator’ and that would help address the need for powerful CPU and memory need. For regular mundane activities, we were able to save costs. On top of this, it comes with ‘Cloud IDE’ feature for quick development, easier to even for people that doesn’t know airflow coding.

References

[1]. Astro CLI Get Started & Overview - https://www.astronomer.io/docs/astro/cli/overview
[2]. Astronomer CI/CD automation - https://www.astronomer.io/docs/astro/automation-overview
[3]. Best practices for runtime version upgrade: https://www.astronomer.io/docs/astro/best-practices/upgrading-astro-runtime
[4]. Celery executors in astronomer: https://www.astronomer.io/docs/astro/celery-executor
[5]. Cloud composer 2 pricing model explained: https://cloud.google.com/composer/pricing#composer-2
[6]. Astronomer pricing explained: https://www.astronomer.io/pricing/?utm_term=&utm_campaign=KB+-+PMAX+-+Free+Trial&utm_source=adwords&utm_medium=ppc&hsa_acc=4274135664&hsa_cam=21095825972&hsa_grp=&hsa_ad=&hsa_src=x&hsa_tgt=&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gad_source=1&gclid=CjwKCAjw2dG1BhB4EiwA998cqOdRwmyEfEuizw1VLOCNXieyDstdsvfB7K-uKcyqCR-JepSTnVNV9RoCfo8QAvD_BwE
[7]. Astro Cloud IDE overview: https://www.astronomer.io/docs/astro/cloud-ide