ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

SCIENTIFIC Research and Community

Review Article

Open Access

Adapting Airflow for Backend Infrastructure Automation

Nikhita Kataria

USA

ABSTRACT

This paper aims to present effectiveness of Airflow for management of common infrastructure management operations such as certificate renewal, operating system upgrades, configuration distribution and multi-region DR drill. Orchestration of these workflows are complex operations due to varied dependencies and a mesh of conditional workflows involving dry run validation and operation execution both in a reliable manner. Key features offered by Airflow such as DAG (Direct Acyclic Graph) based workflow execution, in-built retries, support for functional programming languages such as python allow engineers to automate day to day tasks especially for ops teams. In this paper we present example workflows tailored to infrastructure management operations.

***Corresponding author**
NikhitaKataria, USA.

Received: July 15, 2025; **Accepted:** July 18, 2025; **Published:** July 28, 2025

Keywords: Airflow, Workflows, DAG, Certificate Renewal, Upgrade, Failover, Configuration Distribution.

Introduction

Let's assume we have different personas managing infrastructure components in an organization including engineers and non-engineers such as executives. This demands a need for orchestration that is easy to understand, self-descriptive, offer scalability and are easy to audit, can support a broad set of use cases. auditing and can handle multiple scenarios.

Apache Airflow was originally developed for orchestration of data pipelines however it is gaining importance in infrastructure operations as well. It offers self-descriptive, visual representation of workflows with support for retries, branching, scheduling perfectly suited for such operations. In this paper, we present our approach to adapting Airflow for infrastructure-centric workflows. We demonstrate how it can be leveraged as a unified orchestration platform across a range of operational scenarios, including:

- **Configuration distribution** – Distribution of a configuration to a large number of hosts.
- **Certificate Renewal** – Managing SSL/TLS lifecycle to ensure continuous secure connectivity.
- **Operating System Upgrades** – Executing controlled, minimal-downtime upgrade workflows.
- **Multi-region Failover or drill** - orchestrating seamless failovers to test disaster recovery strategy.

Through these use cases, we show that Airflow can effectively serve as a central orchestrator for infrastructure workflows—bridging the gap between operational reliability and cross-functional accessibility.

Template Workflow

Infrastructure workflows were Modeled as Airflow DAGs, with the Following Typical Phases

- **Validation:** Making sure everything's ready to go before we start.
- **Pre-checks:** Running quick health checks—like confirming the app is healthy, the OS version is approved, and that the host actually needs work.
- **Execution:** Doing the main work of the task, broken down into manageable phases.
- **Audit Logging:** Keeping detailed logs so we can track what happened.
- **Monitoring and Notification:** Watching the process and sending updates or alerts when needed.
- **Cleanup:** Wrapping things up and tidying after everything finishes, using Airflow's TriggerRule.ALL_DONE to know when to start.

The DAG form for these steps will look like as depicted in Figure. 1. and in the subsequent sections we present implementation of these steps for various infrastructure use cases.

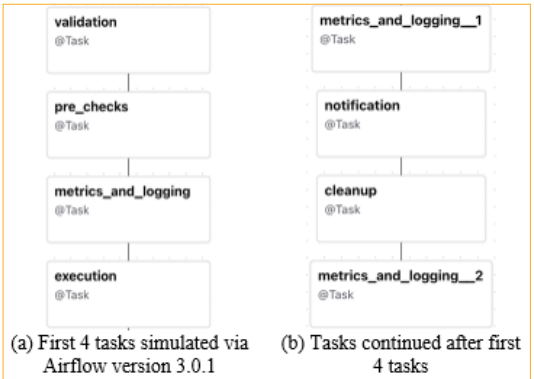

Figure 1: DAG Example for Template Workflow with Tasks for Each Required Step.

Citation: Nikhita Kataria (2024) Adapting Airflow for Backend Infrastructure Automation. Journal of Artificial Intelligence & Cloud Computing. SRC/JAICC-492. DOI: doi.org/10.47363/JAICC/2025(4)451

Configuration Distribution Workflow

Validation

Validation would consist of tasks ensuring configuration format is correct, schema is valid for underlying configuration and is semantically valid. This could depend also on application side validation. For instance on the application side a configuration of rate limit set to 0 might be invalid although a plain integer value of rate limit is syntactically valid.

Pre-Checks

This step is all about making sure the environment is actually ready to support the configuration we're about to roll out. For example, if we're changing a kernel parameter, we need to make sure the hosts are running an OS version that supports it. These kinds of checks happen during the pre-checks phase, so we can catch any issues early.

Execution

This is where the main logic for rolling out the configuration lives—and it runs in a phased manner. Inspired by patterns like those used in Zookeeper, the rollout starts with a small canary set, then gradually expands to a larger percentage of nodes. In Airflow, the best way to handle this is using a TaskFlow-based operator, which supports retries and lets you batch the rollout with controlled delays between batches. This helps reduce risk and gives time to catch issues early.

Audit Logging

It is essential to log changes to a distributed logging service to ensure logs can be manipulated or filtered quickly in case a task throws an error.

Monitoring & Notification

Airflow automatically logs each task and tracks a basic set of metrics out of the box. In this step, we build on top of that by integrating with either proprietary or open-source monitoring systems. The goal is to make sure any errors trigger alerts and the right service owners get notified—so issues don't go unnoticed.

Cleanup

This step handles the final cleanup—removing any temporary files, marking the DAG as complete, and recording the final set of metrics.

Figure 2: Summarizes The Workflow into a Skeleton Airflow DAG for Configuration Distribution as Described in This Section.

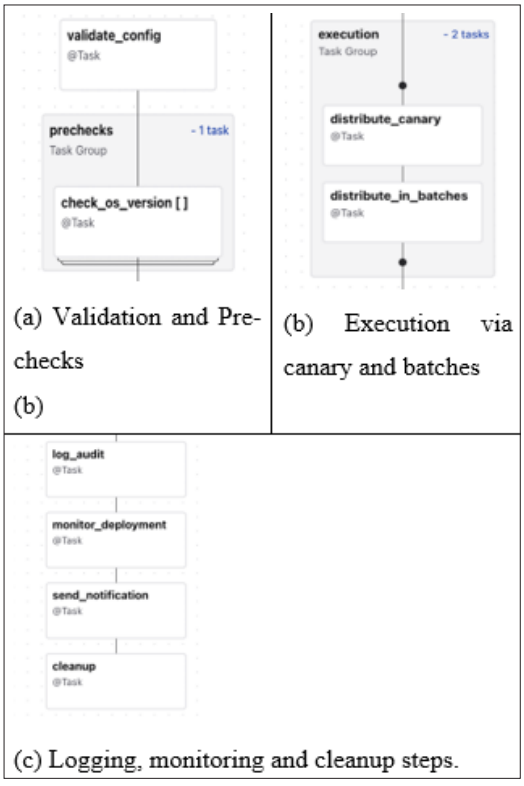

Figure 2: Sample Airflow Workflow for Configuration Distributions Simulated on Airflow version 3.0.1.

Operating System Upgrades

Validation

Validation an operating system is not a simple task as it involves plain vanilla system level validation and validation post running sample application and then probably by replaying customer traffic. This step is to validate if necessary, checks have happened, and a release is certified.

Pre-Checks

OS upgrades are generally more complex than simple configuration rollouts and require multiple layers of pre-checks. For instance, we need to verify that the target host is reachable, hasn't previously failed an upgrade, and isn't currently handling a critical workload—such as a long-running AI process—to avoid any risk of disruption or data loss.

Execution
Once all the checks are out of the way, the upgrade itself is routine. It starts with a canary host to test the waters, then rolls out in batches to the rest. Each host handles the usual steps—grab the OS image, run the upgrade, reboot, and then we make sure it comes back up and is reachable. It's a repeatable flow once things are in motion.

Logging
Logging at this stage is important—it helps us track exactly when each upgrade starts and when it finishes. These timestamps aren't just for show; they give us insight into how long the process is taking. If there's a noticeable delay on a single host, it could point to a local issue. But if a bunch of hosts are slow or stuck, it might signal a broader problem with the OS itself. Keeping a close eye on timing helps catch these patterns early before they snowball into real downtime.

Monitoring
A key part of the monitoring will be to ensure that the host is coming back healthy after each step of the upgrade. Recording timestamp, hostname, percentage of the data center rolled out over time are key metadata metrics.

Audit and Logging
Most certainly we need a final notification to record a host is done with the upgrade and this notification or alert can be aggregated across entire data center to gather success rate of the upgrade as well. In terms of the DAG visualization, the skeleton would look similar to the one described for configuration validation with certan tweaks especially in an additional step of rollback if any of the tasks fails via TriggerRule.ONE_FAILED. Figure 3. illustrates the template DAG for this scenario.

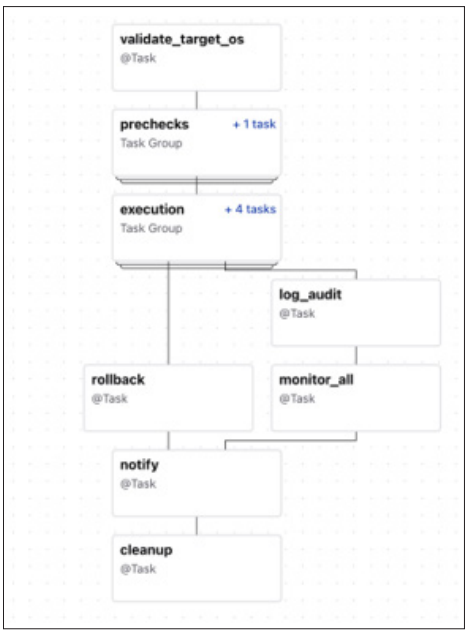

Figure 3: Sample Airflow Workflow for OS Upgrade Simulated on Airflow Version 3.0.1.

Certificate Renewal
Validation
Validation for certificate renewal is generally easier than the first two use cases we examined. It involves steps such as ensuring the certificate has not expired and it is signed by a trusted certificate authority. A few metadata hecks such as the service principals are the intended ones and match the correct target domains. We can use tools like openssl or some handy Python libraries to do this automatically.

Pre-Checks
Once the certificate itself looks good, we need to make sure the machines we're updating are ready. This means making sure we can actually reach each host over the network, there's enough disk space to put the new certificate, and the right folders and permissions are in place. It's also smart to back up the existing certificates before we start, just in case something goes wrong. Plus, we want to be sure that restarting the services that use these certificates (like NGINX or Envoy) won't cause any downtime or trouble.

Execution
The actual execution has a very high impact for certificate renweal because service principals might be in-use by multiple applications serving critical traffic. We follow the same rollout pattern however canarying here means we start small with a couple of less critical machines. We copy over the new certificates and perform restarts if needed. Next, we ensure that the services are able to take and there is no regression in certificate requests. This could mean running a quick HTTPS check to confirm the new certificate is in use. If all goes well, we move on to the rest of the machines in batches, carefully updating and verifying each group to keep things safe and steady.

Audit Logging
During this process there could be multiple levels of failures due to wide impact and logging key critical metadata such as certificate fingerprint, hosts that got the new certificate successfully vs the ones that errored are essential. These logs are super helpful—not just for keeping track of what happened, but also if we need to troubleshoot or prove compliance later.

Monitoring and Notification
After the certificates are deployed, we need to monitor the services to ensure they're still running properly and serving the new certificates. Automated checks can help catch any problems early. If something goes wrong during this task, we'll send notifications to the appropriate teams—most likely a central team responsible for certificate renewal—via Slack or email, so everyone stays in the loop.

Cleanup
In the final step, once the rollout is complete, we clean up any temporary files created during the process and remove old certificates—after making sure it's safe to do so. We also update our records to reflect which certificates are currently in use and clear any locks or flags we set during deployment to signal that the job is finished.

Citation: Nikhita Kataria (2024) Adapting Airflow for Backend Infrastructure Automation. Journal of Artificial Intelligence & Cloud Computing. SRC/JAICC-492. DOI: doi.org/10.47363/JAICC/2025(4)451

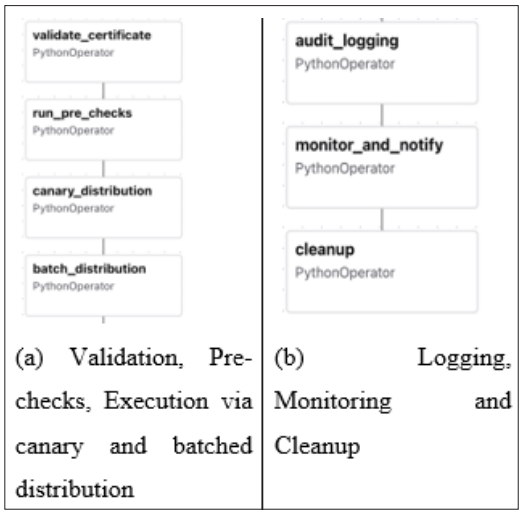

Figure 4: Sample Airflow Workflow for Certificate Renewal Simulated on Airflow Version 3.0.1.

Multi-Region Failover or Drill

Validation

Disaster Recovery plan might be different depending on the scope of failover. In this pase, we focus on verifying if the strategy is comprehensive and well suited for the scenario. We verify is all scope specific configurations and network setups are in place. For instance if it's a regional ourage, we validate that all region configurations are in place, verify network reachability and ensure infrastructure definitions are correctly in place. We validate if DNS settings, routing rules and replication mechanisms are run in a dry run mode to validate semantic functionality, and we are ready for failover.

Pre-Checks

Post validation, we need to run readiness checks to ensure the current state and target state post the DR is intended. This means confirming that both the primary and secondary regions are healthy and meeting their service commitments. We validate the network to ensure its stable and reachable and has needed backups. We also need to ensure there are no other maintenances or incidents on going that might impact the DR flow or cause degradations of any kind.

Execution

This phase is basically executing the actual failover. We consider adding the steps in execution into a TaskGroup. Essentially execution consists of 3 tasks. First is canarying to redirect a small portion of traffic to the new DR region. Second step is to run exhaustive test suite to ensure there is no degradation from user perspective and services in all tiers are responding correctly. Last step is to ensure the rollout is done in a batched manner.

Audit Logging

After the failover is complete, a thorough audit logging needs to be ensured to capture everything that happened. This step is tricky because as part of the failover we might loose the original region from where the redirection is happening. We need to gather logs in phased approach as well to ensure and note any incidents or alerts for quick discovery and audit trail.

Monitoring and Notification

We keep everyone in the loop by sending out notifications that recap how the drill went and share any important takeaways. After the failover, monitoring stays on to catch any late or new issues in the backup region. Alerts are set up to go off if performance or uptime drops below certain levels, so we can jump on any problems quickly.

Cleanup

During cleanup, we figure out if it's time to switch back to the primary region or keep things running in the backup. We get rid of any temporary stuff like test deployments or traffic routes so nothing gets messy or causes problems. If we're switching back, we reset routing and DNS to how they usually are.

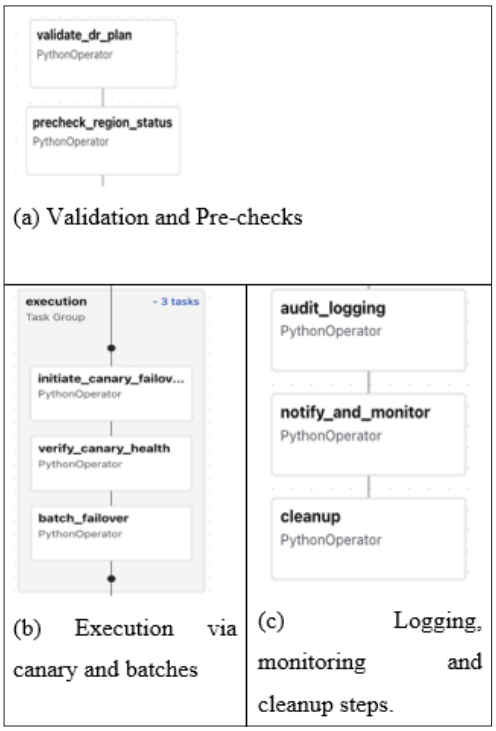

Figure 5: Sample Airflow Workflow for Multi-Region DR Drill Simulated on Airflow Version 3.0.1.

Other Use Cases

In this paper we present a couple of use cases however there exist multiple other use cases onto which same pattern as outlined in this paper can be applied which include and is not limited to:

- Application relocation from one host to another—ensuring smooth transfer and minimal downtime.
- Remediation of a data center host along with seamless application movement—maintaining service continuity.
- Mass restarts and reboots—coordinating large-scale reboot events with safety and observability.
- Host remediation—resolving hardware or software issues without impacting service availability.

Conclusion

Throughout this paper, we've looked at a few routine infrastructure tasks that software teams handle regularly. Airflow really stands out as a solid tool for day-to-day automation. It's easy to get up and running, and it gives you just enough structure to manage things like certificate renewals, OS upgrades, and config rollouts without getting in your way. What makes it especially useful is how it brings visibility, built-in retries, and a clear sense of flow to otherwise complex processes. For many teams, it hits that sweet spot between simplicity and capability—making it a go-to choice for keeping infrastructure running smoothly.

References

1. Apache Airflow, "Getting Started," Apache Software Foundation (2024) https://airflow.apache.org/docs/apache-airflow/stable/start.html
2. Apache Airflow, "Logging and Monitoring Metrics," Apache Software Foundation (2024) https://airflow.apache.org/docs/apache-airflow/stable/administration-and-deployment/logging-monitoring/metrics.html
3. Apache ZooKeeper, "Documentation: Release 3.5.9," Apache Software Foundation (2020) https://zookeeper.apache.org/doc/r3.5.9/index.html
4. Mozilla Foundation, "Web Security Guidelines: TLS," (2023) https://infosec.mozilla.org/guidelines/web_security#tls
5. Let's Encrypt, "Documentation," Internet Security Research Group (2024) https://letsencrypt.org/docs/
6. Google SRE, "Site Reliability Engineering Books," Google (2022) https://sre.google/books/
7. OpenSSL Project, "openssl-x509 - Certificate Display and Signing Utility," (2021) https://www.openssl.org/docs/man1.1.1/man1/openssl-x509.html
8. National Institute of Standards and Technology (NIST), "SP 1800-16: Securing Web Transactions," Final (2022) https://csrc.nist.gov/publications/detail/sp/1800-16/final
9. G. Kim, P. Debois, J. Willis, and J. Humble (2016) *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*, IT Revolution Press https://dl.acm.org/doi/10.5555/3044729.