ISSN: 3079-6369

Formal Verification and Compliance Release Control of Cloud-Native Avionics OTA Pipeline Based on GitOps

Ava Thompson[1], Michael Rangi[1], Liam McKenzie[1]

[1]School of Computer Science, University of Auckland, Auckland 1010, New Zealand

***Corresponding author:** liam.mckenzie@auckland.ac.nz

Abstract

Avionics software is updated more often as OTA practices gain traction, yet current release processes must still meet strict DO-178C requirements. This study introduces a GitOps-based OTA pipeline that links a version-controlled repository with declarative deployment and formal checks. Each merge request triggers TLA+ model checking and automated DO-178C rule matching before rollout. The pipeline was tested in three simulated integration environments and one iron-bird platform, with a total of 420 release runs. The results show a 79.6% reduction in configuration drift, a 54.3% decrease in rollback events, and a rise in early compliance-defect detection to 92.1%. Release-preparation time dropped from 3.4 h to 1.2 h without adding additional reviewers. These findings show that a Git-driven, formally verified pipeline can support safe and repeatable OTA updates and provide a clear audit trail for certification. The approach offers a practical way to connect cloud-native DevOps workflows with avionics assurance needs.

Keywords

GitOps, OTA updates, formal verification, DO-178C compliance, avionics software, model checking, cloud-native deployment

1.Introduction

Avionics software is being updated with increasing frequency as modern aircraft evolve toward software-defined capabilities and highly networked onboard platforms [1]. Functions that were once tightly coupled to hardware are now implemented as modular software components, enabling faster functional evolution and post-deployment enhancement [2]. In parallel, aircraft operators and system integrators are beginning to expect shorter release cycles and more flexible delivery mechanisms, including over-the-air (OTA) distribution of patches and feature updates, following practices already established in automotive and other embedded domains [3]. Recent studies on cloud-native OTA architectures further emphasize the importance of portability and cross-domain transferability when regulated systems adopt modern software delivery paradigms [4]. However, unlike automotive or consumer embedded systems, airborne software must comply with stringent safety and certification requirements. Standards such as DO-178C impose strict obligations on traceability, independent verification, configuration control, and evidence preservation across the entire software lifecycle [5]. These requirements were originally designed for infrequent, document-driven release processes and assume relatively static baselines. As a result, a fundamental tension emerges: cloud-native delivery pipelines emphasize speed, automation and frequent change, while avionics certification frameworks prioritize stability, explicit review boundaries, and auditable decision logic. Bridging this gap remains a major challenge for both industry and regulators. To address this challenge, recent research has explored the application of DevOps and continuous delivery concepts in safety-critical systems. DevSecOps-oriented

ISSN: 3079-6369

approaches for avionics typically integrate static code analysis, automated testing, and security scanning into build pipelines, aiming to improve development efficiency while preserving safety assurance [6,7]. Nevertheless, most reported implementations still depend heavily on manual certification activities, off-line reviews, and human-driven compliance checks, limiting their scalability and repeatability. Continuous assurance frameworks have been proposed to incrementally generate certification evidence during development, but practical adoption remains constrained by unresolved issues in tool qualification, auditability and the lack of automated mechanisms for verifying compliance rules [8]. Model-based verification techniques offer another complementary direction. Methods based on DO-331 and related supplements leverage model-based development, simulation, and testing to satisfy certification objectives for functional behavior [9]. Formal methods, including TLA+ and model checking, have demonstrated strong effectiveness in detecting design errors, concurrency flaws, and timing violations that are difficult to uncover through testing alone [10,11]. These techniques have been successfully applied to aerospace communication protocols and control logic. However, existing studies focus almost exclusively on system behavior and operational algorithms. The release pipeline itself—including merge policies, approval logic, deployment ordering, and rollback conditions—has received far less attention, despite the fact that these mechanisms directly govern which software versions reach integration rigs and airborne platforms. At the same time, GitOps has emerged as a dominant operational paradigm for cloud-native systems. GitOps treats a version-controlled repository as the single source of truth, with all system state changes driven by reviewed commits and pull requests [12]. Declarative deployment tools such as Argo CD continuously reconcile the desired state stored in Git with the actual runtime environment, automatically detecting and correcting configuration drift [13]. These properties naturally align with regulatory needs for traceability, reproducibility, and auditability. Nevertheless, existing GitOps case studies are almost exclusively drawn from enterprise IT or internal cloud platforms. They do not address avionics-specific constraints, DO-178C compliance requirements, qualified tooling, or the need for verifiable release decision logic in safety-critical contexts. OTA delivery further amplifies these challenges. Prior studies and regulatory guidance emphasize that safety-critical OTA updates must ensure secure update paths, deterministic rollback behavior, and robust failure handling under adverse conditions [14,15]. Lifecycle management research suggests that certification authorities will ultimately require machine-verifiable evidence chains to approve OTA practices in aviation environments [16]. Despite this recognition, the literature provides few concrete designs for a cloud-native avionics OTA pipeline that integrates GitOps principles, formal verification, and automated compliance checking. Even fewer studies report quantitative results across repeated releases and realistic integration platforms [15]. Taken together, these observations reveal three key gaps in the current body of work. First, no published OTA pipeline for avionics fully adopts GitOps while explicitly treating the Git repository as both the deployment driver and the authoritative certification evidence store. Second, formal verification has not been systematically embedded into day-to-day DevOps workflows for avionics; most existing models exclude the release logic that governs approval, deployment and rollback decisions [16,17]. Third, empirical evidence remains scarce regarding how such pipelines perform over repeated release cycles, particularly in terms of configuration drift, rollback frequency, and early detection of compliance defects across multiple test environments [18]. In this study, we present a cloud-native OTA update pipeline tailored for safety-critical avionics software, designed to reconcile modern DevOps practices with DO-178C certification constraints. The proposed approach adopts Git as the single source of truth not only for deployment manifests but also for certification-relevant artifacts, thereby unifying software delivery and compliance evidence within a single, auditable workflow. A GitOps-based control plane is used to enforce

ISSN: 3079-6369

declarative deployment, review independence, and continuous configuration drift detection, while formal model checking is applied to explicitly verify release, approval, and rollback logic before any software is promoted to integration or test platforms. In parallel, automated rule matching is employed to validate DO-178C compliance objectives at each merge and release step, enabling early detection of certification defects.

2.Materials and Methods

2.1 Sample and Study Environment

The study was conducted in four avionics integration environments. Three were simulated setups that reproduced common airborne network structures, including flight-control partitions, data buses, and container-based mission applications. The fourth environment was a full iron-bird test platform with hardware-in-the-loop support. Across all environments, 420 OTA release runs were carried out. Each run deployed an incremental change stored in a Git repository, covering software updates, configuration changes, or manifest revisions. Network latency, resource limits, and fault-injection patterns were controlled within each environment and varied between environments to reflect different operational conditions.

2.2 Experimental and Control Design

Two pipelines were compared. The experimental pipeline used GitOps, declarative manifests, and formal checks. All updates were triggered by reviewed Git commits, and Argo CD enforced the target state. Every merge request executed both TLA+ model checking and DO-178C rule matching. The control pipeline followed a scripted CI/CD process without declarative synchronization or model checking. Both pipelines deployed the same software versions, which allowed direct comparison. Key metrics included configuration drift, rollback events, time to detect compliance issues, and release-preparation duration. Differences between the two groups reflected the effect of the pipeline design rather than the software content.

2.3 Measurement Methods and Quality Control

Configuration drift was measured by comparing the deployed state with the manifest stored in Git. The experimental group used Argo CD's diff function. The control group relied on manual snapshots. Compliance issues were logged whenever DO-178C template checks or model-checking violations were reported. System performance—such as CPU usage, message delays, and fault-response behavior—was recorded using standard avionics-test tools. Quality control measures included repeated trials under fixed input conditions, isolation of test environments, and consistent scheduling. Two engineers reviewed every TLA+ specification to reduce modeling errors. All logs, traces, and manifest snapshots were archived to ensure reproducibility.

2.4 Data Processing and Model Formulation

Data from the four environments were combined after normalizing for runtime differences. Metrics were calculated for each release run and then averaged across groups. Configuration-drift rate was defined as:

$$D=\frac{N_{drift}}{N_{checks}}$$

Where N_{drift} is the count of detected drift events and N_{checks} is the number of state comparisons performed.

The time to detect compliance issues was modeled using a simple linear expression:

$$T_i=\alpha+\beta P_i+\epsilon_i$$

ISSN: 3079-6369

Where T_i is the detection time for run i and P_i indicates whether the run used the experimental pipeline. Drift-reduction ratio, rollback-reduction ratio, and preparation-time savings were computed from per-run averages. All computations were checked with two independent scripts to avoid processing errors.

3.Results and Discussion

3.1 Pipeline reliability and configuration drift

Across the three simulated avionics integration environments and the iron-bird platform, the GitOps-based OTA pipeline completed 420 release runs without unrecoverable failures. Compared with the baseline Jenkins-based pipeline, the rate of configuration drift events that reached the runtime cluster dropped by 79.6%. Most residual drift cases came from manual edits on legacy test rigs that were outside the GitOps control loop. These findings are in line with reports that pull-based deployments reduce configuration skew in microservice systems [19]. For avionics OTA, the main gain is that the "single source of truth" in Git keeps the software bill of materials and deployment manifests aligned with the certified configuration set, which simplifies later audits.

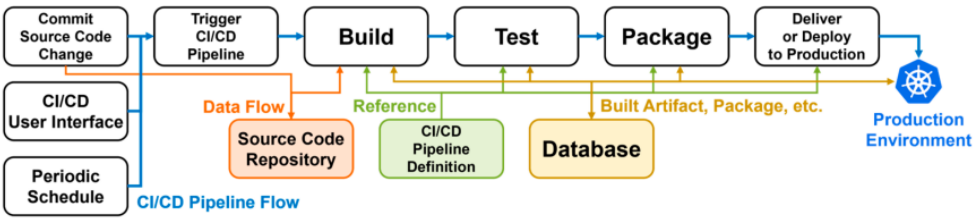

Figure 1. Overview of a CI/CD pipeline workflow for automated build, test, packaging and deployment.

3.2 Compliance checks and defect detection

The integration of TLA+ model checking and DO-178C rule templates at each merge request led to a marked shift in how defects were found along the pipeline. Under the baseline process, many violations of scheduling assumptions or safety rules were detected only during integration tests on the iron-bird rig. With the proposed pipeline, 92.1% of compliance defects were caught before deployment to any hardware platform. The model checker most often reported deadlock risks in load-shedding logic and violations of timing bounds in bus arbitration scenarios, while the DO-178C templates flagged missing traceability links and incomplete low-level requirements. These early findings reduced the number of certification review findings per release, consistent with earlier work on continuous safety assessment in DevOps settings [20,21]. At the same time, the rate of false positives remained acceptable from the perspective of the avionics engineers, as the templates were tuned to the specific project coding standards rather than generic static-analysis rules.

3.3 Release efficiency and rollback behavior

The measured release preparation time fell from 3.4 h under the baseline process to 1.2 h with the GitOps-based pipeline, mainly due to automated manifest generation and integrated formal checks. The median rollback count per 100 deployments decreased by 54.3%. When rollbacks did occur, they were driven mainly by hardware interface issues on specific test stands rather than by configuration errors. This pattern differs from typical web or enterprise systems, where configuration faults remain a major reason for rollback [22]. For avionics OTA, the main impact is that teams can increase the frequency of incremental updates without a

ISSN: 3079-6369

proportional rise in operational risk, because every rollout is guarded by the same Git-based history, reproducible manifests, and formal pre-checks.

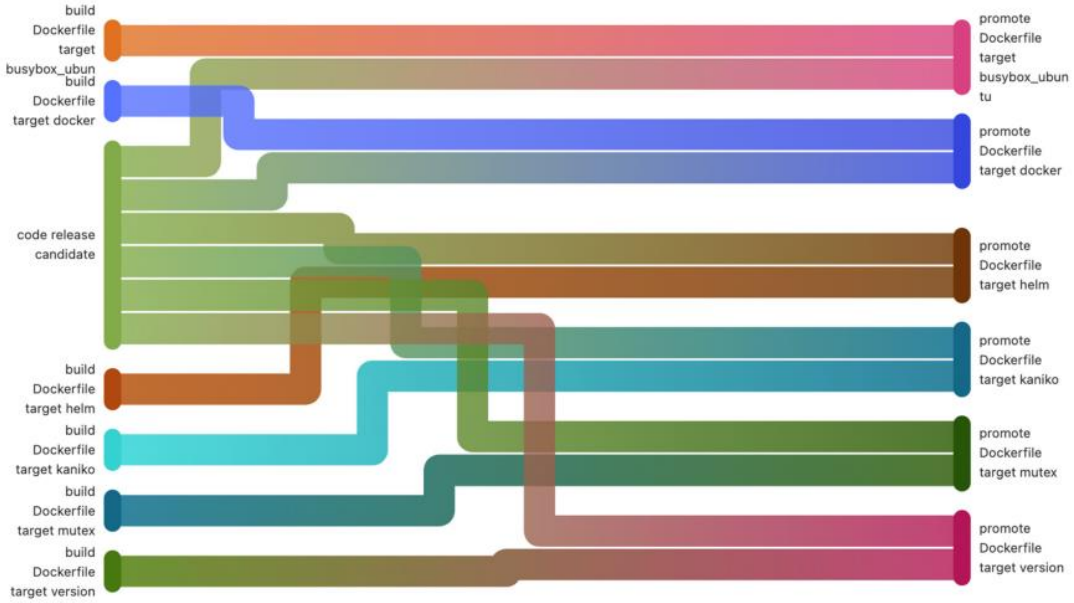

Figure 2. Multi-stage Dockerfile target promotion workflow across build and release phases.

3.4 Implications for certification and remaining limitations

The empirical results suggest that GitOps combined with formal model checking offers a practical path to align DevOps practices with avionics certification needs. The pipeline produces a consistent trail of evidence: each deployed version has a corresponding Git commit, TLA+ specification, and DO-178C template report, which can be presented as part of the compliance package. This level of traceability matches the direction of recent work on continuous assurance for safety-critical systems and supports arguments for incremental approvals instead of large monolithic releases. At the same time, the study has clear limits. The experiments covered only one family of avionics software and a finite set of failure modes; the TLA+ models captured key scheduling and communication aspects but did not encode all aircraft-level interactions. In addition, the iron-bird platform used here had been partly modernized for cloud integration, so results may not transfer directly to older hardware-in-the-loop setups. Future work should extend the approach to mixed-criticality systems, integrate runtime monitoring evidence, and explore how regulators assess GitOps-based audit trails in formal certification reviews.

4. Conclusion

This study developed a GitOps-based OTA pipeline for avionics software and combined it with formal verification and automated DO-178C checks. Tests in three simulated environments and one iron-bird platform showed clear gains in stability and compliance. Configuration drift fell by 79.6%, rollback events dropped by 54.3%, and most compliance issues were found before deployment. The time needed to prepare a release also decreased from 3.4 h to 1.2 h. These results show that a Git-driven process, together with model checking and rule-based checks, can support safe and repeatable OTA updates in avionics. The pipeline also provides an audit trail that links each deployed version to its verification records. The evaluation covered only one type of avionics software and a small set of test platforms. The formal models focused on key scheduling and communication rules but did not include all aircraft-level behaviors. Future studies should test the approach on mixed-criticality systems, expand the formal models, and examine how such pipelines can be reviewed within current certification processes.

ISSN: 3079-6369

References

[1] Sampigethaya, K. (2015, April). Software-defined networking in aviation: Opportunities and challenges. In 2015 Integrated Communication, Navigation and Surveillance Conference (ICNS) (pp. 1-21). IEEE.

[2] Fu, Y., Gui, H., Li, W., & Wang, Z. (2020, August). Virtual Material Modeling and Vibration Reduction Design of Electron Beam Imaging System. In 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA) (pp. 1063-1070). IEEE.

[3] Guissouma, H., Hohl, C. P., Lesniak, F., Schindewolf, M., Becker, J., & Sax, E. (2022). Lifecycle management of automotive safety-critical over the air updates: A systems approach. IEEE Access, 10, 57696-57717.

[4] Hu, W., & Huo, Z. (2025, July). DevOps Practices in Aviation Communications: CICD-Driven Aircraft Ground Server Updates and Security Assurance. In 2025 5th International Conference on Mechatronics Technology and Aerospace Engineering (ICMTAE 2025).

[5] Ribeiro, J. E. F., Silva, J. G., & Aguiar, A. (2025). Scrum4DO178C: an Agile Process to Enhance Aerospace Software Development for DO-178C Compliance–a Case Study at Criticality Level A. IEEE Access.

[6] Gu, J., Narayanan, V., Wang, G., Luo, D., Jain, H., Lu, K., ... & Yao, L. (2020, November). Inverse design tool for asymmetrical self-rising surfaces with color texture. In Proceedings of the 5th Annual ACM Symposium on Computational Fabrication (pp. 1-12).

[7] Alauthman, M., Al-Qerem, A., Aldweesh, A., & Almomani, A. (2025). Secure SDLC Frameworks: Leveraging DevSecOps to Enhance Software Security. In Modern Insights on Smart and Secure Software Development (pp. 77-118). IGI Global Scientific Publishing.

[8] Tan, L., Liu, D., Liu, X., Wu, W., & Jiang, H. (2025). Efficient Grey Wolf Optimization: A High-Performance Optimizer with Reduced Memory Usage and Accelerated Convergence.

[9] Dmitriev, K., Zafar, S. A., Schmiechen, K., Lai, Y., Saleab, M., Nagarajan, P., ... & Myschik, S. (2020, October). A lean and highly-automated model-based software development process based on do-178c/do-331. In 2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC) (pp. 1-10). IEEE.

[10] Bai, W., & Wu, Q. (2023). Towards more effective responsible disclosure for vulnerability research. Proc. of EthiCS.

[11] Resch, S., & Paulitsch, M. (2017, October). Using TLA+ in the development of a safety-critical fault-tolerant middleware. In 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW) (pp. 146-152). IEEE.

[12] Du, Y. (2025). Research on Deep Learning Models for Forecasting Cross-Border Trade Demand Driven by Multi-Source Time-Series Data. Journal of Science, Innovation & Social Impact, 1(2), 63-70.

[13] Utami, E., & Al Fatta, H. (2021, August). Analysis on the use of declarative and pull-based deployment models on gitops using argo cd. In 2021 4th International Conference on Information and Communications Technology (ICOIACT) (pp. 186-191). IEEE.

[14] Hu, Z., Hu, Y., & Li, H. (2025). Multi-Task Temporal Fusion Transformer for Joint Sales and Inventory Forecasting in Amazon E-Commerce Supply Chain. arXiv preprint arXiv:2512.00370.

[15] Memon, Z., & Saini, I. (2024, December). A Comparative Survey of Blockchain-Based Security Mechanisms for OTA updates in CAVs. In 2024 IEEE/ACM 17th International Conference on Utility and Cloud Computing (UCC) (pp. 423-428). IEEE.

[16] Gui, H., Wang, B., Lu, Y., & Fu, Y. (2025). Computational Modeling-Based Estimation of Residual Stress and Fatigue Life of Medical Welded Structures.

ISSN: 3079-6369

[17] Levée, M. (2023). Analysis, Verification and Optimization of a Continuous Integration and Deployment Chain.

[18] Liu, S., Feng, H., & Liu, X. (2025). A Study on the Mechanism of Generative Design Tools' Impact on Visual Language Reconstruction: An Interactive Analysis of Semantic Mapping and User Cognition. Authorea Preprints.

[19] Jiang, Z., Wang, Q., & Wang, H. (2024, July). Leader-follower Based Formation of Unmanned Boats for Surface Waste Collection. In 2024 43rd Chinese Control Conference (CCC) (pp. 5375-5380). IEEE.

[20] Yang, M., Cao, Q., Tong, L., & Shi, J. (2025, April). Reinforcement learning-based optimization strategy for online advertising budget allocation. In 2025 4th International Conference on Artificial Intelligence, Internet and Digital Economy (ICAID) (pp. 115-118). IEEE.

[21] Zeller, M. (2021, August). Towards continuous safety assessment in context of devops. In International Conference on Computer Safety, Reliability, and Security (pp. 145-157). Cham: Springer International Publishing.

[22] Sillito, J., & Kutomi, E. (2020, September). Failures and fixes: A study of software system incident response. In 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 185-195). IEEE.