УДК 004.4:004.75

ВИЯВЛЕННЯ ТА КОНТРОЛЬ КОНФІГУРАЦІЙНОГО ДРЕЙФУ В GITOPS-СЕРЕДОВИЩАХ KUBERNETES

Білоконь Т.І.[1]; Трофименко О.Г. к.т.н., доц.[2]
[1,2] Національний університет «Одеська юридична академія»
[1,2] Україна, Одеса
[1] belokontimofej4@gmail.com; [2]ORCID: 0000-0001-7626-0886

Анотація. *Розглянуто причини конфігураційного дрейфу в GitOps-середовищах Kubernetes та механізми його виявлення й усунення в Argo CD. Узагальнено ризики автоматичної синхронізації, зокрема її взаємодії з автомасштабуванням, і запропоновано практичні принципи контрольованого узгодження станів.*
Ключові слова*: GitOps; конфігураційний дрейф; Kubernetes; Argo CD; реконсиляція; автоматизація розгортання.*

Вступна частина. GitOps формалізує керування розгортанням через декларативний опис бажаного стану, його зберігання з версіонуванням (як правило, у Git), автоматичне отримання конфігурації програмними агентами та безперервне узгодження з фактичним станом системи [1]. Такий підхід підвищує відтворюваність змін і простежуваність історії розгортань, однак сам по собі не гарантує постійної відповідності кластера репозиторію. Конфігураційний дрейф, тобто відхилення між задекларованою та реальною конфігурацією, може виникати через ручні зміни, автоматичні контролери або затримки синхронізації. Для мікросервісних систем із великою кількістю ресурсів Kubernetes це створює ризики нестабільності та ускладнює діагностику інцидентів.

Мета дослідження – систематизувати основні причини конфігураційного дрейфу в GitOps-середовищах, зіставити засоби його виявлення й корекції та сформулювати рекомендації щодо контрольованого впровадження автоматичної синхронізації.

Основна частина. Причини дрейфу мають різну природу. По-перше, ручні втручання через Kubernetes API або командний рядок змінюють ресурси поза контрольованим Git-процесом. По-друге, автоматичні контролери, зокрема механізми автомасштабування, можуть змінювати окремі параметри під час роботи. Такі зміни не завжди є помилковими: вони можуть бути очікуваною реакцією системи на навантаження. Проблема виникає тоді, коли GitOps-процес не враховує ці зміни або постійно намагається повернути параметр до значення, яке задає конфігурація. По-третє, дрейф спричиняють процесні чинники: паралельне редагування середовищ, конфлікти злиття, неповний опис ресурсів та інтервал між оновленням репозиторію й черговою перевіркою. Показово, що в описаному дослідженні впровадження GitOps в одній із компаній конфлікти злиття стали частішими саме через одночасні зміни конфігурацій різних середовищ, а керування секретами й робота із застарілою інфраструктурою потребували додаткових інструментів і процедур [2].

Базовим механізмом виявлення є безперервне порівняння бажаного стану з фактичним. Argo CD позначає застосунок як OutOfSync, коли маніфести в Git і живий стан кластера розходяться, та може автоматично синхронізувати ресурси [3]. Водночас за замовчуванням зміни, зроблені безпосередньо в кластері, автоматичну синхронізацію не запускають: для цього потрібно явно ввімкнути параметр selfHeal. Опція prune, яку також вмикають окремо, видаляє ресурси, що більше не описані в Git [3]. Отже, ці механізми мають різні наслідки: selfHeal коригує розбіжності наявних ресурсів, тоді як prune може спричинити їх видалення.

Частота перевірок визначає затримку виявлення дрейфу: інтервал автоматичної синхронізації задається параметром timeout.reconciliation, що за замовчуванням дорівнює 120 с із випадковим зсувом до 60 с, тобто не перевищує трьох хвилин [3]. Для критичних ресурсів, наприклад просторів імен, документація передбачає опцію Prune=confirm, яка вимагає ручного підтвердження перед видаленням [4].

Порівняно з pull-моделлю GitOps, у традиційних IaC-процесах зміни частіше застосовують за принципом push: такі інструменти, як Terraform чи Ansible, виконують їх під час конкретного запуску, тоді як GitOps-агент (Argo CD або Flux) безперервно порівнює стан у Git зі станом системи й узгоджує їх [2]. Це не означає, що push-інструменти взагалі не виявляють дрейф: Terraform, наприклад, має окремі механізми планування та зіставлення керованих ресурсів. Тому коректно порівнювати не «наявність або відсутність» виявлення як таку, а безперервність контролю й модель застосування змін. Водночас у згаданому дослідженні кількість випадків дрейфу після впровадження GitOps суттєво зменшилася, але вибірка була малою (п’ять інтерв’ю), а спостереження тривали 4–5 місяців [2].

Окремої уваги потребує взаємодія GitOps з автомасштабуванням. Типовий приклад: у Git задано spec.replicas = 3, а Horizontal Pod Autoscaler змінює це значення відповідно до навантаження. Без додаткового налаштування Argo CD фіксує розбіжність, а за ввімкненого selfHeal повертає значення з Git, тобто вступає в конфлікт із контролером Kubernetes. Документація пропонує кілька способів вирішення: ігнорувати окремі поля під час порівняння через ignoreDifferences (за JSON-шляхом, JQ-виразом або за менеджером поля, наприклад kube-controller-manager) [5] і додатково ввімкнути опцію RespectIgnoreDifferences=true, щоб ці поля не перезаписувалися й під час самої синхронізації [4]. Альтернатива – вилучити поле replicas із маніфестів і залишити його під керуванням автомасштабувача.

Принципово інший підхід запропоновано в препринті DriGitScale [6]. Автори розглядають автомасштабування як задачу керування узгодженістю й оцінюють зважену суму трьох відхилень: прогнозного, ресурсного (між потрібною та фактичною кількістю реплік) і конфігураційного (між кількістю реплік у Git та в кластері). Рішення про масштабування ухвалюється лише тоді, коли сукупний показник перевищує поріг (в експериментах $\theta = 0{,}05$), інакше зберігається значення з Git. Саме масштабування виконується не прямою зміною ресурсів через API, а комітом оновленого маніфеста в репозиторій, який далі синхронізує GitOps-агент [6]. Таким чином Git залишається єдиним джерелом істини, що в принципі усуває причину конфлікту між контролером і GitOps-агентом. Автори повідомляють про зниження показника фактичної нестабільності (частоти змін кількості реплік) з 0,1315 до 0,0097 на трасах NASA [6]. Цей результат слід інтерпретувати обережно: експерименти проводили на трасах навантаження та синтетичних даних, масштабування оцінювали лише за завантаженням процесора, а затримки синхронізації репозиторію й реконсиляції в моделі не враховано [6]. Тому він не доводить переваги методу для будь-якого виробничого кластера.

На основі проведеного аналізу можна сформулювати практичні принципи контрольованого узгодження станів. По-перше, усі планові зміни слід проводити через pull request із перевіркою та історією схвалення, а прямий доступ для редагування продуктивних ресурсів обмежувати й аудіювати. По-друге, автосинхронізацію варто впроваджувати поетапно: спочатку в тестовому середовищі, потім для безпечних типів ресурсів і лише після оцінювання наслідків – у продуктивному; надмірна автоматизація продуктивного середовища без етапів погодження чи поетапної перевірки може спричиняти непередбачені

збої [2]. По-третє, для критичних ресурсів доцільно вимагати підтвердження видалення (Prune=confirm) [4], а для секретів застосовувати спеціалізоване шифрування та контроль доступу без збереження їх у відкритому вигляді. По-четверте, для кожного параметра, яким керують і автомасштабувач, і Git, потрібно явно визначити єдине джерело істини: закріпити його через ignoreDifferences [5] або вносити зміни через коміт, як у DriGitScale [6]. Нарешті, слід відстежувати тривалість розбіжності, частоту невдалих синхронізацій і час реконсиляції; ці показники доповнюють метрики DORA (частота розгортань, час від коміту до розгортання, частка невдалих змін, час відновлення), які застосовують для оцінювання ефективності GitOps-процесів [2]. Усі ці заходи узгоджуються з принципами декларативності, версіонування, автоматичного отримання та безперервного узгодження, сформульованими спільнотою OpenGitOps [1].

Висновки. Конфігураційний дрейф у GitOps-середовищах є наслідком не лише ручних втручань, а й взаємодії автоматичних контролерів, процесів розроблення та параметрів синхронізації. Argo CD надає інструменти для виявлення розбіжностей і їх автоматичного усунення, однак усталено не реагує на зміни в кластері, а використання selfHeal та prune має враховувати критичність ресурсів і можливі побічні ефекти. Для параметрів, якими керують автомасштабувачі, потрібні явні правила: ignoreDifferences разом із RespectIgnoreDifferences або внесення змін через Git, як у підході DriGitScale. Подальші дослідження доцільно спрямувати на оцінювання адаптивних порогів дрейфу, впливу затримок реконсиляції та узгодження GitOps із автомасштабуванням на відтворюваних тестових сценаріях у реальних кластерах.

Використання інструментів штучного інтелекту. ChatGPT (GPT-6) використано для перевірки правопису тексту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] OpenGitOps. (n.d.). *GitOps principles v1.0.0*. https://opengitops.dev/

[2] Tseradze, G., Kuchava, G., Sturua, T., Khinchikashvili, A., Somkhishvili, E., Kartvelishvili, I., Mantskava, M., Chokhonelidze, S., & Momtselidze, N. (2026). Performance and efficiency evaluation of GitOps-based deployment workflows. *Computational and Applied Science,* 1(1), 66–83. https://doi.org/10.68218/mcsk3n57

[3] Argo Project. (n.d.). *Automated sync policy*. In *Argo CD documentation*. https://argo-cd.readthedocs.io/en/latest/user-guide/auto_sync/

[4] Argo Project. (n.d.). *Sync options*. In *Argo CD documentation*. https://argo-cd.readthedocs.io/en/latest/user-guide/sync-options/

[5] Argo Project. (n.d.). *Diff customization*. In *Argo CD documentation*. https://argo-cd.readthedocs.io/en/latest/user-guide/diffing/

[6] AboElazm, A. F., Yassin, A. S., & Fahmy, H. (2026). DriGitScale: Drift-aware GitOps for consistency-aware predictive autoscaling in Kubernetes [Preprint]. *Research Square*. https://doi.org/10.21203/rs.3.rs-11019483/v1

Bilokon T.I., Trofymenko O.G.

Detection and control of configuration drift in Kubernetes GitOps environments

Abstract*. The causes of configuration drift in Kubernetes GitOps environments and the mechanisms for its detection and* remediation *in Argo CD are considered. The risks of automated synchronization, including its interaction with autoscaling, are summarized, and practical principles for controlled state reconciliation are proposed.*

Keywords*: GitOps; configuration drift; Kubernetes; Argo CD; reconciliation; deployment automation.*