GitOps z Argo CD: klaster sam pobiera zmiany z Gitea

Poziom: zaawansowane

Dotyczy: Argo CD 3.5.3 (wykres argo-cd 10.9.6), Gitea 1.27 na docker01, Helm 4.3.0, k3s 1.36, wykres kolorowy-nginx 0.3.0 (wersje sprawdzone 3.10.2026)

Czas: ~90 min

Przenosimy wykres kolorowego nginksa do repozytorium w Gitea i instalujemy w klastrze Argo CD, który co kilka minut porównuje repozytorium z klastrem i sam dociąga różnice. Zmianę palety kolorów robimy commitem, a cofamy ją przez `git revert`, bez żadnego `helm upgrade` z konsoli.

Skróty użyte na tej stronie (10)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
CD	Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu
CI	Continuous Integration: automatyczne budowanie i testowanie kodu po każdej zmianie; w ITIL także configuration item, czyli element konfiguracji: każdy składnik usługi zapisany w CMDB
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
CNCF	Cloud Native Computing Foundation: fundacja pod Linux Foundation, formalny właściciel projektu Kubernetes od lipca 2015 roku.
DNS	Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP
GitOps	Git Operations: sposób prowadzenia klastra, w którym pożądany stan jest zapisany w repozytorium Git, a agent w klastrze sam doprowadza rzeczywistość do tego stanu
HPA	HorizontalPodAutoscaler: obiekt Kubernetesa, który sam zwiększa albo zmniejsza liczbę podów Deploymentu, gdy zużycie procesora lub pamięci odbiega od celu
HTTPS	HTTP zabezpieczony szyfrowaniem TLS
TLS	Transport Layer Security: szyfrowanie połączeń, np. w HTTPS

Fiszka

Komendy wykonujemy z CLI bash na stacji administratora, z kubeconfigiem do `k3s01` i dostępem do repozytorium przez HTTPS. Zakładamy, że stos monitoringu ze strony Obserwowalność klastra: metryki, logi i alerty bez bólu już stoi, bo wykres w wersji 0.3.0 tworzy ServiceMonitor i PrometheusRule. Gitea działa na `docker01` pod adresem `git.zielona.example`, tak jak opisują strony książki o Gitea.

□□ KLIKAMY W INTERFEJSIE

Utworzenie repozytorium w Gitea

```
Gitea → + (u góry po prawej) → New Repository → Owner: zielona, Repository Name: kolory-k8s, Visibility: Private → Create Repository
```

Zakładamy w organizacji `zielona` prywatne, puste repozytorium. Pierwszy commit i push wykonujemy poniżej z komputera.

□□ KLIKAMY W INTERFEJSIE

Utworzenie tokenu tylko do odczytu dla Argo CD

```
Gitea → Settings → Applications → Generate New Token → Token Name: argocd, Select scopes: read:repository → Generate Token
```

Token tworzymy na koncie serwisowym `svc.argocd`, któremu w organizacji dajemy tylko odczyt repozytorium `kolory-k8s`. Zapisujemy go od razu, bo Gitea pokazuje go jeden raz.

□□ **CD** = Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu

□□ TYLKO ODCZYT

Sklonowanie pustego repozytorium: git clone

```
git clone https://git.zielona.example/zielona/kolory-k8s.git
cd kolory-k8s
```

Przykład:

```
Cloning into 'kolory-k8s'...
warning: You appear to have cloned an empty repository.
```

git = system kontroli wersji (nazwa to brytyjski slang: nieznośny typ) · **cd** = change directory: przejdź do katalogu

ZMIENIA SYSTEM

Skopiowanie wykresu do repozytorium: cp

```
cp -r ../kolorowy-nginx ./kolorowy-nginx
mkdir -p argocd
```

Wykres w wersji 0.3.0 trafia do katalogu `kolorowy-nginx`, a manifest aplikacji Argo CD z sekcji niżej do katalogu `argocd`. Do repozytorium dopisujemy też plik `kolorowy-nginx/values-lab.yaml`, który pokazujemy w sekcji o układzie repozytorium.

CD = Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu · **cp** = copy: skopiuj · **mkdir** = make directory: utwórz katalog

ZMIENIA SYSTEM

Pierwszy commit i wysłanie do Gitea: git commit i git push

```
git add .
git commit -m "Wykres kolorowy-nginx 0.3.0 i wartości laboratorium"
git push origin main
```

Przykład:

```
[main (root-commit) 4be1a07] Wykres kolorowy-nginx 0.3.0 i wartości laboratorium
 12 files changed, 438 insertions(+)
To https://git.zielona.example/zielona/kolory-k8s.git
 * [new branch]      main -> main
```

git = system kontroli wersji (nazwa to brytyjski slang: nieznośny typ)

TYLKO ODCZYT

Usunięcie dotychczasowego wydania Helma: helm uninstall

```
helm uninstall kolorowy-nginx --namespace kolory
```

Od teraz obiektami w przestrzeni nazw `kolory` zarządza Argo CD, więc ręczne wydanie z poprzednich stron usuwamy, żeby dwa narzędzia nie walczyły o te same obiekty. Pierwszy raz Argo CD utworzy je od nowa.

CD = Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu · **helm** = menedżer paczek aplikacji dla Kubernetesa

ZMIENIA SYSTEM

Dodanie repozytorium wykresów Argo: helm repo add

```
helm repo add argo https://argoproj.github.io/argo-helm
helm repo update
```

helm = menedżer paczek aplikacji dla Kubernetesa

TYLKO ODCZYT

Przejrzenie ustawień ingressa w wykresie Argo CD: helm show values

```
helm show values argo/argo-cd --version 10.9.6 | grep -n -A 12 "^  ingress:"
```

Klucze plików wartości zmieniają się między wersjami wykresu, więc przed instalacją sprawdzamy, że `server.ingress` i `global.domain` mają w przypiętej wersji wykresu te same nazwy, co w pliku `argocd-values.yaml`.

CD = Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu · **helm** = menedżer paczek aplikacji dla Kubernetesa

ZMIENIA SYSTEM

Instalacja Argo CD: helm install argo-cd

```
helm install argocd argo/argo-cd --version 10.9.6 \
  --namespace argocd --create-namespace -f argocd-values.yaml --wait --timeout 5m
```

Przykład:

```
NAME: argocd
LAST DEPLOYED: Wed Oct  7 09:15:22 2026
NAMESPACE: argocd
```

```
STATUS: deployed
REVISION: 1
DESCRIPTION: Install complete
```

Wykres 10.9.6 instaluje Argo CD w wersji 3.5.3. Plik `argocd-values.yaml` pokazujemy w sekcji o instalacji.

CD = Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu · **helm** = menedżer paczek aplikacji dla Kubernetesa

TYLKO ODCZYT

Sprawdzenie podów Argo CD: kubectl get pods

```
kubectl get pods -n argocd
```

Przykład:

```
NAME                                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0                     1/1     Running   0          2m
argocd-applicationset-controller-6f8c7d9b4d-lq5xv   1/1     Running   0          2m
argocd-notifications-controller-5b9d6c7f84-r2kzt    1/1     Running   0          2m
argocd-redis-7c6d8f9b5-hn4wp                        1/1     Running   0          2m
argocd-repo-server-68d7c5b9f6-xw9tj                 1/1     Running   0          2m
argocd-server-5c9b7d6f8d-vj6mq                      1/1     Running   0          2m
```

Zestaw podów zależy od wersji wykresu i od włączonych komponentów. Przy wyłączonym dex nie ma `argocd-dex-server`.

CD = Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Odczyt hasła początkowego administratora: kubectl get secret

```
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}"
| base64 -d; echo
```

Przykład:

```
Q7mL2xV9tK4pRw8Z
```

Login to `admin`. Hasło zmieniamy po pierwszym zalogowaniu, a Secret początkowy usuwamy.

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Podejrzenie interfejsu Argo CD bez Ingressa: kubectl port-forward

```
kubectl port-forward svc/argocd-server -n argocd 8080:80
```

Przykład:

```
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

Interfejs jest wtedy pod `http://localhost:8080`, niezależnie od DNS i Traefika.

CD = Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu · **DNS** = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Zarejestrowanie repozytorium w Argo CD: kubectl create secret

```
kubectl create secret generic repo-kolory-k8s -n argocd \
  --from-literal=type=git \
  --from-literal=url=https://git.zielona.example/zielona/kolory-k8s.git \
  --from-literal=username=svc.argocd \
  --from-literal=password='TokenZGitea'
kubectl label secret repo-kolory-k8s -n argocd argocd.argoproj.io/secret-type=repository
```

Przykład:

```
secret/repo-kolory-k8s created
secret/repo-kolory-k8s labeled
```

Argo CD szuka danych dostępowych do repozytoriów w Secretach z etykietą `argocd.argoproj.io/secret-type=repository`. Token z Gitea trafia tylko do tego Secretu, nigdy

do repozytorium.

CD = Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Utworzenie aplikacji Argo CD: kubectl apply -f

```
kubectl apply -f argocd/aplikacja-kolory.yaml
```

Przykład:

```
application.argoproj.io/kolorowy-nginx created
```

To jedyny raz, kiedy tworzymy obiekt ręcznie. Dalej zmiany idą przez Git.

CD = Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Sprawdzenie stanu aplikacji: kubectl get applications

```
kubectl get applications -n argocd
```

Przykład:

```
NAME             SYNC STATUS   HEALTH STATUS
kolorowy-nginx   Synced        Healthy
```

Stan `Synced` znaczy, że klaster zgadza się z Gitem, a `Healthy`, że obiekty działają. Stan `OutOfSync` oznacza różnicę, a `Missing` obiekt, którego w klastrze brakuje.

kubectl = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Zmiana palety przez commit: git commit i git push

```
git add kolorowy-nginx/values-lab.yaml
git commit -m "Paleta: dokładamy turkusowy, zdejmujemy czerwony"
git push origin main
```

Przykład:

```
[main 9a3c1d2] Paleta: dokładamy turkusowy, zdejmujemy czerwony
 1 file changed, 2 insertions(+), 2 deletions(-)
To https://git.zielona.example/zielona/kolory-k8s.git
   4be1a07..9a3c1d2  main -> main
```

Edycję pliku `values-lab.yaml` opisujemy w sekcji o zmianie palety. Od tej chwili czekamy na Argo CD.

CD = Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu · **git** = system kontroli wersji (nazwa to brytyjski slang: nieznośny typ)

TYLKO ODCZYT

Wymuszenie natychmiastowego sprawdzenia repozytorium: kubectl annotate

```
kubectl annotate application kolorowy-nginx -n argocd argocd.argoproj.io/refresh=normal
--overwrite
```

Przykład:

```
application.argoproj.io/kolorowy-nginx annotated
```

Zamiast czekać do trzech minut na kolejny cykl, prosimy Argo CD o odświeżenie od razu. Adnotacja znika sama po wykonaniu odświeżenia.

CD = Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Obserwacja aktualizacji podów: kubectl get pods -w

```
kubectl get pods -n kolory -w
```

Przykład:

```
NAME                              READY   STATUS            RESTARTS   AGE
kolorowy-nginx-7d9c6b5f8d-4tkq2   2/2     Running           0          14m
kolorowy-nginx-7d9c6b5f8d-m8zr5   2/2     Running           0          14m
kolorowy-nginx-5f6b8c7d9c-q7tzn   0/2     Init:0/1          0          3s
kolorowy-nginx-5f6b8c7d9c-q7tzn   2/2     Running           0          9s
kolorowy-nginx-7d9c6b5f8d-4tkq2   2/2     Terminating       0          14m
```

Nowy ReplicaSet powstaje, bo zmiana palety zmienia zmienną `KOLORY` w szablonie poda. Dalej działa ta sama aktualizacja krocząca z `maxUnavailable: 0` co zawsze.

□□ **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

□□ ZMIENIA SYSTEM

Wycofanie zmiany przez Git: git revert

```
git revert --no-edit HEAD
git push origin main
```

Przykład:

```
[main c5e8f01] Revert "Paleta: dokładamy turkusowy, zdejmujemy czerwony"
 1 file changed, 2 insertions(+), 2 deletions(-)
```

Cofnięcie jest nowym commitem, który odwraca poprzedni. Argo CD synchronizuje go jak każdy inny, więc historia w Gitea pokazuje, kto, co i kiedy zmienił, także przy cofnięciu.

□□ **CD** = Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu ·**git** = system kontroli wersji (nazwa to brytyjski slang: nieznośny typ)

□□ OSTROŻNIE · NIEODWRACALNE

Sprawdzenie samoleczenia: usunięcie Service ręcznie

```
kubectl delete service kolorowy-nginx -n kolory
kubectl get service -n kolory -w
```

Przykład:

```
service "kolorowy-nginx" deleted
NAME             TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)           AGE
kolorowy-nginx   ClusterIP   10.43.118.52   <none>        80/TCP,9113/TCP   2s
```

Z włączonym `selfHeal` Argo CD odtwarza usunięty ręcznie obiekt w ciągu kilku sekund, bo repozytorium mówi, że Service ma istnieć. Nowy Service dostaje nowy adres ClusterIP.

CD = Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

Z życia wzięte

Z ŻYCIA WZIĘTE

Środa, 16:40. Tomek w pośpiechu zmienia ręcznie `kubectl set image` na laboratoryjnym kolorowym nginksie, żeby sprawdzić nowszą wersję, i zapomina o tym. W czwartek rano Ola widzi na stronie ten sam kolor co wczoraj, a Argo CD pokazuje `OutOfSync` z czerwoną różnicą w polu `image`. Po włączeniu `selfHeal` taka ręczna zmiana wraca do stanu z Gita sama, a Tomek uczy się, że nowszą wersję wpisuje się w plik wartości i wysyła commitem.

Jak to działa

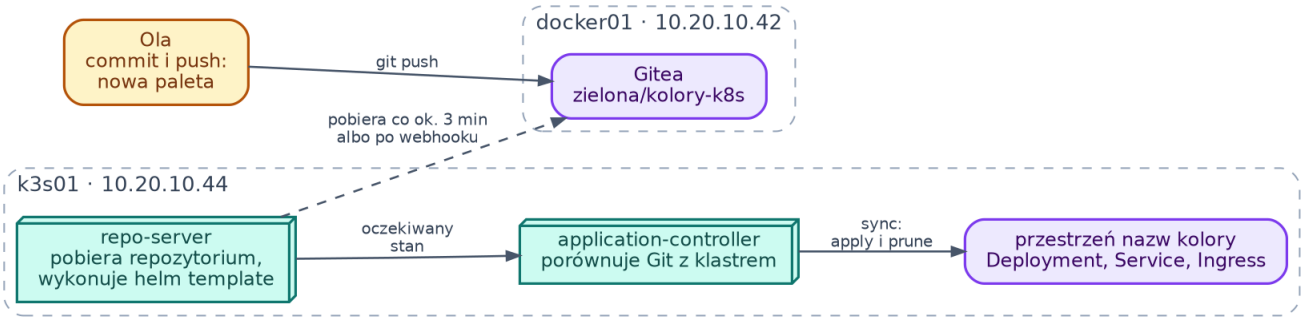
Rozwiń: jak to działa pod spodem

> **Pojęcie: GitOps.** Sposób prowadzenia infrastruktury, w którym pożądany stan klastra jest zapisany w repozytorium Git, a osobny program w klastrze dba o to, żeby rzeczywistość się z nim zgadzała. Zmiana oznacza commit, zatwierdzenie oznacza przegląd w Gitea, a cofnięcie oznacza `git revert`. Klaster nie ma ręcznych poprawek, o których nikt nie wie.

Cztery zasady, które wynikają z tej definicji, odróżniają GitOps od zwykłego wdrażania skryptem:

- stan jest deklaratywny, jak w manifestach z pierwszych stron książki
- repozytorium jest jedynym źródłem prawdy, więc zmiany w klastrze bez commitu uznajemy za błąd

- agent w klastrze sam pobiera zmiany, zamiast czekać na wypchnięcie
- agent stale porównuje stan rzeczywisty z pożądanym i usuwa różnice



Ola wysyła commit do Gitea na docker01, repo-server Argo CD pobiera repozytorium i wykonuje helm template, application-controller porównuje wynik z klastrem i synchronizuje przestrzeń nazw kolory (kliknięcie otwiera pełny rozmiar)

Wdrażanie z wypchnięciem kontra z pobraniem

W podejściu push system CI, na przykład Jenkins na `ci01`, ma dostęp do klastra i sam wykonuje `kubectl apply` albo `helm upgrade`. To wygodne, ale wymaga trzymania w Jenkinsie `kubeconfig` z szerokimi uprawnieniami i otwartego API klastra z `ci01`. Przejęcie Jenkinsa daje wtedy dostęp do klastra.

W podejściu pull agent działa wewnątrz klastra i sam wychodzi do Gitea. Klaster nie musi przyjmować połączeń od CI, a do repozytorium wystarcza token tylko do odczytu. Uprawnienia do zmian klastra ma wyłącznie Argo CD, a to, co ma się zmienić, ustala tylko to, co przejdzie przez commit.



W podejściu push Jenkins na ci01 trzyma kubeconfig i wdraża do klastra, w podejściu pull Argo CD w klastrze sam pobiera dane z Gitea tylko do odczytu (kliknięcie otwiera pełny rozmiar)

Z czego składa się Argo CD

Argo CD to kilka komponentów w przestrzeni nazw `argocd`. `argocd-repo-server` pobiera repozytorium i generuje manifesty, w naszym przypadku wykonując `helm template`. `argocd-application-controller` porównuje wygenerowane manifesty z obiektami w klastrze i synchronizuje różnice. `argocd-server` udostępnia interfejs i API, `argocd-redis` buforuje wyniki, a opcjonalny kontroler ApplicationSet rozmnaża aplikacje po wielu klastrach lub katalogach.

> **Pojęcie: Application.** Obiekt Kubernetesa, który mówi Argo CD, skąd brać manifesty (repozytorium, gałąź i katalog), dokąd je wdrażać (klaster i przestrzeń nazw) i jak synchronizować (ręcznie albo automatycznie). Jedna aplikacja to w naszym laboratorium jedno wydanie kolorowego nginksa.

> ⚠ Argo CD nie wykonuje `helm install`. Użyje Helma tylko do wygenerowania manifestów przez `helm template` i zastosuje je samodzielnie. Polecenie `helm list -n kolory` pokazuje więc pustą listę, a `helm rollback` nie ma czego cofać. Historię i cofanie wydania zapewnia Git.

Układ repozytorium

Repozytorium `zielona/kolory-k8s` ma prosty układ. Wykres leży w katalogu o nazwie wykresu, obok niego plik wartości laboratorium, a manifest Argo CD w osobnym katalogu.

```
kolory-k8s/
  README.md
  argocd/
    aplikacja-kolory.yaml
  kolorowy-nginx/
    Chart.yaml
    values.yaml
    values-lab.yaml
    templates/
      _helpers.tpl
      deployment.yaml
      service.yaml
      ingress.yaml
      hpa.yaml
      configmap.yaml
      servicemonitor.yaml
      prometheusrule.yaml
      NOTES.txt
```

Plik `values.yaml` to wartości domyślne wykresu, a `values-lab.yaml` to wszystko, co różni laboratorium od wartości domyślnych. Zmiany w klastrze robimy w pliku laboratorium, a do pliku domyślnego zaglądamy tylko wtedy, gdy zmieniamy sam wykres.

```
# Wartości laboratorium Zielonej: ten plik zmieniamy commitem
autoscaling:
  enabled: true
  minReplicas: 2
  maxReplicas: 6

metrics:
  enabled: true
  serviceMonitor:
    enabled: true
  alerts:
    enabled: true
```

```
ingress:
  host: kolory.zielona.example

paleta:
  - { nazwa: czerwony, hex: "#ef4444" }
  - { nazwa: zielony, hex: "#22c55e" }
  - { nazwa: niebieski, hex: "#3b82f6" }
  - { nazwa: zolty, hex: "#eab308" }
  - { nazwa: fioletowy, hex: "#a855f7" }
  - { nazwa: pomaranczowy, hex: "#f97316" }
```

Instalacja Argo CD przez Helm

Argo CD instalujemy wykresem z przypiętą wersją 10.9.6, który wdraża Argo CD w wersji 3.5.3. Plik wartości robi trzy rzeczy: wyłącza TLS w samym Argo CD, bo kończy je Traefik, dodaje certyfikat urzędu CA-Zielona, żeby `repo-server` ufał certyfikatowi `git.zielona.example`, i wyłącza komponent Dex, którego bez logowania przez zewnętrzną tożsamość nie potrzebujemy.

> □□ Po przenosinach urzędu według strony Przenosiny urzędu z dc01 bez przerwy: dwa urzędy równolegle i wygaszenie CA-Zielona dokładamy tu w ten sam sposób zaufanie do `Zielona-Root-CA`, a `CA-Zielona.crt` zostaje obok do wygaszenia starego urzędu.

```
# Argo CD w laboratorium na k3s01
global:
  domain: argocd.zielona.example

configs:
  params:
    # TLS kończy Traefik, sam Argo CD słucha po HTTP
    server.insecure: true
  tls:
    certificates:
      git.zielona.example: |
```

```
        -----BEGIN CERTIFICATE-----
        TUTAJ_WKLEJAMY_CERTYFIKAT_CA_ZIELONA
        -----END CERTIFICATE-----

server:
  ingress:
    enabled: true
    ingressClassName: traefik

# Bez logowania przez zewnętrznego dostawcę tożsamości oszczędzamy jeden pod
dex:
  enabled: false
```

Certyfikat urzędu wklejamy w całości w miejsce linii `TUTAJ_WKLEJAMY_CERTYFIKAT_CA_ZIELONA`. Bez niego `repo-server` zgłasza błąd `x509: certificate signed by unknown authority` przy pierwszej próbie pobrania repozytorium.

Application wskazująca wykres Helm z repozytorium

Manifest aplikacji wskazuje repozytorium, gałąź `main`, katalog z wykresem i plik wartości laboratorium. Argo CD odnajduje wykres po pliku `Chart.yaml` w katalogu `path`, a nazwa aplikacji staje się nazwą wydania.

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: kolorowy-nginx
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default
  source:
    repoURL: https://git.zielona.example/zielona/kolory-k8s.git
    targetRevision: main
    path: kolorowy-nginx
```

```
    helm:
      valueFiles:
        - values-lab.yaml
  destination:
    server: https://kubernetes.default.svc
    namespace: kolory
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
```

Trzy pola w `syncPolicy` wymagają osobnego opisu. Wpis `automated` włącza automatyczną synchronizację po wykryciu różnicy, `prune: true` usuwa z klastra obiekty, które zniknęły z repozytorium, a `selfHeal: true` cofa ręczne zmiany w klastrze. Opcja `CreateNamespace=true` zakłada przestrzeń nazw `kolory`, a finalizator `resources-finalizer.argocd.argoproj.io` sprawia, że usunięcie Application usuwa także obiekty, które utworzyła.

Argo CD sprawdza repozytorium domyślnie co 120 sekund z dodatkowym losowym opóźnieniem do 60 sekund, czyli najwyżej co trzy minuty. Czekanie można skrócić webhookiem. Argo CD obsługuje webhooki z Gogs, a Gitea potrafi wysłać taki typ webhooka na adres `/api/webhook` serwera Argo CD, ale szczegóły ustawiamy według dokumentacji Argo CD, w rozdziale o konfiguracji webhooków.

Zmiana palety przez commit

W pliku `kolorowy-nginx/values-lab.yaml` zdejmujemy czerwony i dodajemy turkusowy. Lista palety w pliku wartości zastępuje całą listę domyślną, więc zapisujemy ją w całości.

```
paleta:
  - { nazwa: zielony, hex: "#22c55e" }
  - { nazwa: niebieski, hex: "#3b82f6" }
  - { nazwa: zolty, hex: "#eab308" }
  - { nazwa: fioletowy, hex: "#a855f7" }
  - { nazwa: pomaranczowy, hex: "#f97316" }
  - { nazwa: turkusowy, hex: "#14b8a6" }
```

Po commicie i pushu przebieg jest zawsze taki sam. Argo CD pobiera nową wersję repozytorium, `repo-server` wykonuje `helm template` z nowymi wartościami, a `application-`

`controller` widzi różnicę w zmiennej `KOLORY` Deploymentu. Synchronizacja zmienia Deployment, a ten uruchamia aktualizację kroczącą. Po kilku minutach w odpowiedziach nginksa pojawia się turkusowy, a czerwonego już nie.

Sprawdzamy to zliczeniem kolorów, jak na stronie Autoskalowanie: HPA i requests, limits na kolorowym nginx:

```
for i in $(seq 1 40); do
  curl -s --resolve kolory.zielona.example:80:10.20.10.44 http://kolory.zielona.example/ |
grep -o "Kolor: [a-z]*"
done | sort | uniq -c
```

Rollback przez git revert

Interfejs Argo CD ma przycisk wycofania do wcześniejszej rewizji, ale przy włączonej automatycznej synchronizacji jest on zablokowany. Argo CD ustawia tak celowo: automat natychmiast zsynchronizowałby klaster z powrotem do stanu z Gita. Dlatego cofamy zmianę w Gicie, poleceniem `git revert` z fiszki wyżej, a Argo CD wdraża cofnięcie jak każdy inny commit.

Dzięki temu cofnięcie zostawia ślad: kto, kiedy i dlaczego wrócił do poprzedniej palety. Gdy zmian było więcej, `git revert` wskazuje konkretny commit zamiast `HEAD`. Gdy nie ma czasu na przegląd, zawieszamy automatyczną synchronizację w Application i dopiero wtedy cofamy ręcznie, ale na stałe zmianę zapisujemy w Gicie.

Argo CD czy Flux

Flux to drugie powszechne narzędzie GitOps, także w projekcie CNCF. Oba robią to samo, różnią się architekturą i wygodą. W Zielonej wybieramy Argo CD, bo interfejs graficzny ułatwia naukę i pokazuje różnicę między Gitem a klastrem.

Cecha	Argo CD 3.5.3	Flux 2.9.5
Model	obiekt Application wskazuje repozytorium i katalog	osobne obiekty: GitRepository (źródło) i Kustomization albo HelmRelease (co wdrożyć)
Interfejs graficzny	wbudowany, z widokiem różnic	brak w samym Flux, są dodatki innych projektów
Instalacja na k3s	wykres Helm albo manifesty z repozytorium Argo CD	polecenie `flux bootstrap`, także `flux bootstrap gitea`, które samo zapisuje konfigurację w repozytorium

Cecha	Argo CD 3.5.3	Flux 2.9.5
Wykresy Helm	generuje manifesty przez `helm template` i stosuje je sam, więc `helm list` jest pusty	kontroler helm-controller instaluje prawdziwe wydania, widoczne w `helm list`
Cofanie	przy automatycznej synchronizacji tylko przez `git revert`	`git revert`, a obiekt HelmRelease może sam cofać nieudane aktualizacje
Wiele zespołów	projekty (AppProject) ograniczają repozytoria, klastry i przestrzenie nazw	wielodostęp przez konta usługi, pod które Flux się podszywa
Zużycie zasobów	kilka komponentów, w tym Redis	zwykle lżejszy, osobny kontroler na każdą rolę
Nasz wybór w Zielonej	tak, do nauki i do pokazywania zmian zespołowi	rozważamy przy większej liczbie klastrów

> Sekretów nie wkładamy do repozytorium GitOps w jawnej postaci. Hasła i tokeny zapisujemy przez Sealed Secrets albo External Secrets Operator, jak opisuje strona ConfigMap i Secret. W tym laboratorium jedynym sekretem poza Gitem jest token odczytu w Secret `repo-kolory-k8s`.

Częste błędy

Objaw	Przyczyna	Co robimy
Application ma `ComparisonError`, w komunikacie `x509: certificate signed by unknown authority`	`repo-server` nie ufa urzędowi, który wystawił certyfikat `git.zielona.example`	dopisujemy certyfikat CA-Zielona w `configs.tls.certificates` i robimy `helm upgrade` wykresu Argo CD
`authentication required` albo `repository not found`	zły token, zły adres albo konto bez prawa odczytu	sprawdzamy Secret `repo-kolory-k8s`, token wygenerowany z zakresem `read:repository` i uprawnienia konta serwisowego
`Unable to resolve 'main' to a commit SHA`	gałąź nazywa się inaczej albo repozytorium jest puste	sprawdzamy gałąź w Gitea i pole `targetRevision`
`values-lab.yaml: no such file or directory`	ścieżka w `valueFiles` jest względna względem katalogu wykresu, a plik leży gdzie indziej	plik trzymamy obok `Chart.yaml` albo poprawiamy ścieżkę

Objaw	Przyczyna	Co robimy
Synchronizacja kończy się błędem `no matches for kind "ServiceMonitor"`	w klastrze nie ma jeszcze typów obiektów Prometheus Operatora	instalujemy stos monitoringu przed aplikacją albo wyłączamy `metrics.serviceMonitor` i `metrics.alerts` w `values-lab.yaml`
Application ciągle jest `OutOfSync` z powodu pola `replicas`	Deployment ma stałe `replicas`, a HPA je zmienia	szablon pomija `replicas` przy włączonym HPA, a w innym przypadku dodajemy `ignoreDifferences` dla `/spec/replicas`
Commit jest w Gitea, a w klastrze nic się nie dzieje	Argo CD sprawdza repozytorium co około 3 minuty	adnotacja `argocd.argoproj.io/refresh=normal` albo webhook z Gitea
`helm list -n kolory` pokazuje pustą listę	Argo CD nie tworzy wydań Helma, tylko stosuje manifesty z `helm template`	to normalne, stan sprawdzamy w `kubectl get applications` i w interfejsie Argo CD
Ręczna zmiana w klastrze znika po kilku sekundach	`selfHeal: true` przywraca stan z Gita	zmianę robimy commitem, a do testów wyłączamy `selfHeal` w Application
Plik zniknął z repozytorium, a razem z nim obiekt w klastrze	`prune: true` usuwa obiekty, których nie ma w Gicie	to zamierzone działanie, więc przed usunięciem pliku sprawdzamy, czy obiekt ma zniknąć
Po usunięciu Application w klastrze zostają obiekty	brakuje finalizatora `resources-finalizer.argocd.argoproj.io`	dodajemy finalizator do Application albo usuwamy obiekty ręcznie
Argo CD nie widzi `git.zielona.example`	brak rozwiązywania nazwy w klastrze albo zapora blokuje ruch z `k3s01`	`kubectl run test --rm -it --image=busybox:1.37.0 -- nslookup git.zielona.example`, sprawdzamy DNS i reguły zapory `fw01`

□□ Ćwiczenia

1. Zmieniamy w `values-lab.yaml` wartość `maxReplicas` z 6 na 4, commitujemy i pushujemy. Co dzieje się z HPA i kiedy?

Odpowiedź

Po najbliższym odświeżeniu repozytorium, najpóźniej po około trzech minutach, Argo CD zsynchronizuje obiekt HPA ze zmienioną wartością `maxReplicas`. Liczba podów nie zmienia się natychmiast, bo HPA ocenia ją co 15 sekund i zmniejszy ją do czterech dopiero wtedy, gdy będzie ich więcej. Całość widzimy w `kubectl get hpa -n kolory`.

1. Zmieniamy poleceniem `kubectl edit hpa kolorowy-nginx -n kolory` pole `maxReplicas` z 6 na 10. Co robi Argo CD i dlaczego?

Odpowiedź

Argo CD wykrywa różnicę i, przy włączonym `selfHeal`, w ciągu kilku sekund przywraca wartość z Gita. Dzieje się tak, bo stanem pożądanym jest repozytorium, a nie klaster. Zmianę, którą chcemy zachować, robimy commitem.

1. Podajemy trzy powody, dla których po `git revert` Argo CD może nie dojść do poprzedniego stanu.

Odpowiedź

Pierwszy: ktoś zmienił między commitami inne pola i `git revert` zgłasza konflikt, który trzeba rozwiązać ręcznie przed pushem. Drugi: Argo CD nie widzi nowego commitu, bo repozytorium sprawdza co około trzy minuty, więc czekamy albo odświeżamy adnotacją. Trzeci możliwy powód to wyłączona synchronizacja automatyczna, wtedy zmianę trzeba zsynchronizować ręcznie.

1. Czym różni się ryzyko przejęcia Jenkinsa w podejściu push od przejęcia tokenu odczytu Gitea w podejściu pull?

Odpowiedź

Z kubeconfigiem Jenkinsa można zmieniać klaster, więc przejęcie `ci01` daje przejęcie wdrożeń. Token odczytu pozwala tylko czytać repozytorium: napastnik zobaczy manifesty, ale nie zmieni klastra bez zmiany w Gicie. Dlatego zapis do gałęzi `main` chronimy przeglądem zmian, a zakres tokenu ograniczamy do odczytu.

1. Dlaczego Argo CD wdrażające nasz wykres nie pokazuje wydania w `helm list`, a Flux z HelmRelease pokazuje?

Odpowiedź

Argo CD używa Helma tylko jako narzędzia do szablonów (`helm template`) i sam stosuje manifesty, więc nie zapisuje w klastrze Secretów wydania. Kontroler Flux helm-controller wykonuje prawdziwe operacje Helma, które zapisują wydanie jak zwykły `helm install`.

Powiązane: Helm: kolorowy nginx jako pakiet · Autoskalowanie: HPA i requests, limits na kolorowym nginx · Obserwowalność klastra: metryki, logi i alerty bez bólu · Pierwsze repozytorium: od pustego projektu do pierwszego pusha · ConfigMap i Secret · RBAC i bezpieczeństwo klastra

Źródła: argo-cd.readthedocs.io (Getting Started, Automated Sync Policy: odświeżanie co 120 sekund plus 60 sekund losowego opóźnienia, ograniczenie rollbacku przy automatycznej synchronizacji, Webhook Configuration, Declarative Setup) · github.com/argoproj/argo-helm, wykres argo-cd 10.9.6 (aplikacja v3.5.3) · github.com/argoproj/argo-cd/releases (v3.5.3 z 14.09.2026) · fluxcd.io/flux/installation/bootstrap/gitea · github.com/fluxcd/flux2/releases (v2.9.5 z 31.08.2026) · docs.gitea.com (tokeny dostępu, webhooki) · cncf.io/projects (Argo i Flux w CNCF)

Revision #17
Created 2026-10-03 09:39:11 UTC by Tomasz Konieczny
Updated 2026-10-26 14:17:24 UTC by Tomasz Konieczny