# **Automatic Rollout Options for Red Hat Developer Hub**

## **Using ArgoCD and GitHub Repository**

**Environment:** <namespace> namespace  
 **Goal:** Automatically restart RHDH deployment when app-config.yaml changes in Git  
 **Date:** October 14, 2025

## **Table of Contents**

1. [Overview](https://claude.ai/chat/7efd7743-8c69-4293-ae73-ab32f29b6fbf#overview)
2. [Option 1: Timestamp Annotation (Easiest)](https://claude.ai/chat/7efd7743-8c69-4293-ae73-ab32f29b6fbf#option-1-timestamp-annotation)
3. [Option 2: Kustomize ConfigMapGenerator (Recommended)](https://claude.ai/chat/7efd7743-8c69-4293-ae73-ab32f29b6fbf#option-2-kustomize-configmapgenerator)
4. [Option 3: ArgoCD PostSync Hook](https://claude.ai/chat/7efd7743-8c69-4293-ae73-ab32f29b6fbf#option-3-argocd-postsync-hook)
5. [Option 4: Helper Script](https://claude.ai/chat/7efd7743-8c69-4293-ae73-ab32f29b6fbf#option-4-helper-script)
6. [Comparison Table](https://claude.ai/chat/7efd7743-8c69-4293-ae73-ab32f29b6fbf#comparison-table)
7. [Recommendations](https://claude.ai/chat/7efd7743-8c69-4293-ae73-ab32f29b6fbf#recommendations)

## **Overview**

This document provides four options for automatically triggering Red Hat Developer Hub (RHDH) deployment rollouts when configuration files are updated in a Git repository and synced via ArgoCD.

**Current Setup:**

* Namespace: <namespace>
* Deployment: redhat-developer-hub
* ArgoCD syncs from GitHub repository
* Configuration files stored in Git
* Cannot install Reloader (no cluster-admin access)

**Challenge:** ConfigMap changes don't automatically trigger pod restarts in Kubernetes. When you update app-config.yaml in Git and ArgoCD syncs it, the pods continue running with old configuration.

**Solution Options:** Four approaches to solve this, ranging from simple manual updates to fully automated solutions.

## **Option 1: Timestamp Annotation**

### **Difficulty: ⭐ Easiest**

### **Setup Time: 2 minutes**

### **Future Effort: Update 2 files per change**

### **Automation: Manual**

### **Description**

Add a timestamp annotation to your deployment that you manually update whenever you change configuration files. When ArgoCD syncs and sees the deployment has changed, Kubernetes triggers a rollout.

### **Implementation**

**Step 1:** Edit your deployment file in Git

**File:** k8s/deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: redhat-developer-hub

namespace: <namespace>

spec:

template:

metadata:

annotations:

restartedAt: "2025-10-14T10:30:00Z" # Update this timestamp

spec:

containers:

- name: backstage

# ... your container configuration ...

**Step 2:** Commit the initial setup

git add k8s/deployment.yaml

git commit -m "Add restart annotation to deployment"

git push

### **Daily Workflow**

When you need to update configuration:

# 1. Edit your app-config

vim k8s/app-config.yaml

# 2. Update the timestamp in deployment

vim k8s/deployment.yaml

# Change restartedAt to current time: "2025-10-14T11:45:00Z"

# 3. Commit both files

git add k8s/app-config.yaml k8s/deployment.yaml

git commit -m "Update app-config"

git push

# ArgoCD syncs → deployment changes → rollout happens automatically

### **Pros**

* ✅ Extremely simple to understand and implement
* ✅ No special tools or permissions required
* ✅ Works immediately
* ✅ No cluster-side setup needed

### **Cons**

* ❌ Manual process - must remember to update timestamp
* ❌ Easy to forget the timestamp update
* ❌ Must edit two files for every config change
* ❌ Prone to human error

### **When to Use**

* Quick temporary solution
* You make infrequent configuration changes
* You want something working in 2 minutes
* Testing ArgoCD setup before committing to automation

## **Option 2: Kustomize ConfigMapGenerator**

### **Difficulty: ⭐⭐ Easy**

### **Setup Time: 10 minutes**

### **Future Effort: Edit 1 file per change**

### **Automation: Fully Automatic**

### **Description**

Kustomize automatically generates ConfigMaps with hash suffixes. When the ConfigMap content changes, Kustomize creates a new ConfigMap with a new hash (e.g., rhdh-app-config-abc123, then rhdh-app-config-def456), and updates the deployment to reference the new ConfigMap. This change in the deployment triggers an automatic rollout.

This is the **recommended industry-standard GitOps approach**.

### **Implementation**

**Step 1:** Create directory structure

mkdir -p k8s/base/configs

**Step 2:** Move your configuration file

# Move app-config to the configs directory

mv k8s/app-config.yaml k8s/base/configs/app-config.yaml

**Step 3:** Create Kustomization file

**File:** k8s/base/kustomization.yaml

apiVersion: kustomize.config.k8s.io/v1beta1

kind: Kustomization

namespace: <namespace>

# ConfigMap generator - automatically creates hash suffix

configMapGenerator:

- name: rhdh-app-config

files:

- configs/app-config.yaml

# Include deployment and other resources

resources:

- deployment.yaml

# Enable hash suffix (this is default behavior)

generatorOptions:

disableNameSuffixHash: false

**Step 4:** Update deployment to reference ConfigMap

**File:** k8s/base/deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: redhat-developer-hub

namespace: <namespace>

spec:

template:

spec:

containers:

- name: backstage

volumeMounts:

- name: app-config

mountPath: /opt/app-root/src/app-config.yaml

subPath: app-config.yaml

volumes:

- name: app-config

configMap:

# Kustomize automatically updates this with hashed name

name: rhdh-app-config

**Step 5:** Update ArgoCD Application

**File:** argocd/rhdh-application.yaml

apiVersion: argoproj.io/v1alpha1

kind: Application

metadata:

name: rhdh-app

namespace: argocd

spec:

project: default

source:

repoURL: https://github.com/your-org/your-repo

targetRevision: main

path: k8s/base # Point to kustomize directory

kustomize:

version: v5.0.0

destination:

server: https://kubernetes.default.svc

namespace: <namespace>

syncPolicy:

automated:

prune: true

selfHeal: true

**Step 6:** Commit everything

git add k8s/base/

git add argocd/rhdh-application.yaml

git commit -m "Setup Kustomize with ConfigMapGenerator"

git push

### **Daily Workflow**

From now on, configuration updates are simple:

# 1. Edit your app-config - THAT'S IT!

vim k8s/base/configs/app-config.yaml

# 2. Commit and push

git add k8s/base/configs/app-config.yaml

git commit -m "Update app-config: enable new feature"

git push

# ArgoCD automatically:

# - Detects the change

# - Generates new ConfigMap: rhdh-app-config-a1b2c3d4

# - Updates deployment to reference new ConfigMap

# - Kubernetes sees deployment change

# - Triggers automatic rollout with zero downtime

### **How It Works**

1. You edit k8s/base/configs/app-config.yaml
2. Kustomize generates: rhdh-app-config-a1b2c3d4 (with content hash)
3. Deployment spec changes to reference new ConfigMap name
4. Kubernetes sees deployment changed → triggers rollout
5. Old ConfigMap remains until deployment completes (zero downtime)
6. ArgoCD prunes old ConfigMap after rollout succeeds

### **Pros**

* ✅ Fully automatic after initial setup
* ✅ Edit only one file for config changes
* ✅ Industry-standard GitOps approach
* ✅ Built-in to Kubernetes ecosystem
* ✅ Zero downtime rollouts
* ✅ No manual checksum calculations
* ✅ True immutable infrastructure

### **Cons**

* ❌ Requires 10-minute initial setup
* ❌ Need to learn basic Kustomize concepts
* ❌ Directory structure changes required

### **When to Use**

* **RECOMMENDED for production environments**
* You want true GitOps automation
* You're comfortable with Kustomize
* You make frequent configuration changes
* You want industry best practices

## **Option 3: ArgoCD PostSync Hook**

### **Difficulty: ⭐⭐⭐ Medium**

### **Setup Time: 15 minutes**

### **Future Effort: Edit 1 file per change**

### **Automation: Fully Automatic**

### **Description**

ArgoCD hooks allow you to run jobs at specific points in the sync lifecycle. A PostSync hook runs after ArgoCD successfully syncs resources. This hook runs an OpenShift CLI command to restart the deployment automatically.

**Note:** This approach restarts the deployment on EVERY ArgoCD sync, not just config changes.

### **Implementation**

**Step 1:** Create ServiceAccount with permissions

oc apply -f - <<EOF

apiVersion: v1

kind: ServiceAccount

metadata:

name: argocd-rollout-sa

namespace: <namespace>

---

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

name: argocd-rollout-role

namespace: <namespace>

rules:

- apiGroups: ["apps"]

resources: ["deployments"]

verbs: ["get", "patch"]

- apiGroups: ["apps"]

resources: ["deployments/status"]

verbs: ["get"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: argocd-rollout-binding

namespace: <namespace>

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: argocd-rollout-role

subjects:

- kind: ServiceAccount

name: argocd-rollout-sa

namespace: <namespace>

EOF

**Step 2:** Create PostSync Hook Job

**File:** k8s/hooks/restart-rhdh.yaml

apiVersion: batch/v1

kind: Job

metadata:

generateName: restart-rhdh-

namespace: <namespace>

annotations:

# Run after ArgoCD syncs resources

argocd.argoproj.io/hook: PostSync

# Delete previous hook jobs before creating new one

argocd.argoproj.io/hook-delete-policy: BeforeHookCreation

spec:

# Clean up job after 5 minutes

ttlSecondsAfterFinished: 300

backoffLimit: 2

template:

spec:

serviceAccountName: argocd-rollout-sa

restartPolicy: Never

containers:

- name: restart

image: registry.redhat.io/openshift4/ose-cli:latest

command:

- /bin/bash

- -c

- |

set -e

echo "🔄 Restarting RHDH deployment after config sync..."

oc rollout restart deployment/redhat-developer-hub -n <namespace>

echo "⏳ Waiting for rollout to complete..."

oc rollout status deployment/redhat-developer-hub -n <namespace> --timeout=10m

echo "✅ Rollout completed successfully!"

**Step 3:** Optionally add sync waves for ordering

**File:** k8s/configmap-app-config.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: rhdh-app-config

namespace: <namespace>

annotations:

# Sync ConfigMap first

argocd.argoproj.io/sync-wave: "1"

data:

app-config.yaml: |

# Your configuration content here

**Step 4:** Commit hook to Git

git add k8s/hooks/restart-rhdh.yaml

git add k8s/configmap-app-config.yaml

git commit -m "Add PostSync hook for automatic restarts"

git push

### **Daily Workflow**

# 1. Edit app-config in Git

vim k8s/configmap-app-config.yaml

# 2. Commit and push

git add k8s/configmap-app-config.yaml

git commit -m "Update app-config"

git push

# ArgoCD automatically:

# - Syncs ConfigMap changes

# - Runs PostSync hook job

# - Job restarts deployment

# - Waits for rollout to complete

# - Reports success/failure

### **Verification**

Check if the hook executed:

# View recent jobs

oc -n <namespace> get jobs

# Check job logs

oc -n <namespace> logs job/restart-rhdh-xxxxx

# View ArgoCD sync status

argocd app get rhdh-app

### **Pros**

* ✅ Fully automatic after setup
* ✅ Works with any file changes in Git
* ✅ Don't need to touch deployment file
* ✅ Built into ArgoCD functionality
* ✅ Visible in ArgoCD UI

### **Cons**

* ❌ Requires ServiceAccount creation
* ❌ Restarts on ANY sync (not just config changes)
* ❌ More complex to troubleshoot
* ❌ Requires understanding of ArgoCD hooks
* ❌ Job logs needed for debugging

### **When to Use**

* You want automation but can't use Kustomize
* You have permissions to create ServiceAccounts
* You're comfortable with ArgoCD features
* You want visibility in ArgoCD UI

## **Option 4: Helper Script**

### **Difficulty: ⭐⭐ Easy**

### **Setup Time: 5 minutes**

### **Future Effort: Run script for each change**

### **Automation: Semi-Automatic**

### **Description**

Create a bash script that automates the manual steps: editing the config file, calculating a checksum, and updating the deployment annotation. This provides automation without requiring cluster changes or new tools.

### **Implementation**

**Step 1:** Add checksum annotation to deployment

**File:** k8s/deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: redhat-developer-hub

namespace: <namespace>

spec:

template:

metadata:

annotations:

config-checksum: "initial" # Script will update this

spec:

containers:

- name: backstage

# ... your container configuration ...

**Step 2:** Create helper script

**File:** update-config.sh

#!/bin/bash

set -e

CONFIG\_FILE="k8s/app-config.yaml"

DEPLOYMENT\_FILE="k8s/deployment.yaml"

echo "📝 Opening config file for editing..."

${EDITOR:-vim} "$CONFIG\_FILE"

echo "📊 Calculating checksum..."

CHECKSUM=$(sha256sum "$CONFIG\_FILE" | cut -d' ' -f1 | cut -c1-16)

echo "🔄 Updating deployment annotation..."

sed -i "s/config-checksum: \".\*\"/config-checksum: \"$CHECKSUM\"/" "$DEPLOYMENT\_FILE"

echo ""

echo "✅ Configuration updated successfully!"

echo " New checksum: $CHECKSUM"

echo ""

echo "📤 Ready to commit. Run:"

echo " git add $CONFIG\_FILE $DEPLOYMENT\_FILE"

echo " git commit -m 'Update app-config'"

echo " git push"

**Step 3:** Make script executable

chmod +x update-config.sh

git add update-config.sh k8s/deployment.yaml

git commit -m "Add config update helper script"

git push

### **Daily Workflow**

# 1. Run the helper script

./update-config.sh

# This will:

# - Open your editor for app-config.yaml

# - Calculate new checksum automatically

# - Update deployment.yaml with new checksum

# 2. Review changes

git diff

# 3. Commit both files

git add k8s/app-config.yaml k8s/deployment.yaml

git commit -m "Update app-config: add new integration"

git push

# ArgoCD syncs → deployment changes → rollout happens

### **Enhanced Version with Auto-Commit**

For even more automation:

#!/bin/bash

set -e

CONFIG\_FILE="k8s/app-config.yaml"

DEPLOYMENT\_FILE="k8s/deployment.yaml"

# Edit config

${EDITOR:-vim} "$CONFIG\_FILE"

# Calculate checksum

CHECKSUM=$(sha256sum "$CONFIG\_FILE" | cut -d' ' -f1 | cut -c1-16)

# Update deployment

sed -i "s/config-checksum: \".\*\"/config-checksum: \"$CHECKSUM\"/" "$DEPLOYMENT\_FILE"

# Auto-commit

git add "$CONFIG\_FILE" "$DEPLOYMENT\_FILE"

git commit -m "Update app-config (checksum: $CHECKSUM)"

echo "✅ Changes committed! Ready to push."

echo "Run: git push"

### **Pros**

* ✅ Semi-automated process
* ✅ Quick 5-minute setup
* ✅ No cluster changes needed
* ✅ No new tools to learn
* ✅ Easy to customize script

### **Cons**

* ❌ Must remember to run script
* ❌ Not fully automated
* ❌ Script could fail to update correctly
* ❌ Requires bash environment

### **When to Use**

* You want more automation than manual timestamps
* You can't or won't use Kustomize
* You don't have permissions for PostSync hooks
* You make moderate configuration changes

## **Comparison Table**

| **Feature** | **Timestamp** | **Kustomize** | **PostSync Hook** | **Helper Script** |
| --- | --- | --- | --- | --- |
| **Setup Time** | 2 minutes | 10 minutes | 15 minutes | 5 minutes |
| **Complexity** | ⭐ Very Simple | ⭐⭐ Simple | ⭐⭐⭐ Medium | ⭐⭐ Simple |
| **Automation Level** | ❌ Manual | ✅ Full Auto | ✅ Full Auto | 🟡 Semi-Auto |
| **Future Effort** | Edit 2 files | Edit 1 file | Edit 1 file | Run script |
| **Files Modified** | 2 per change | 1 per change | 1 per change | 2 per change |
| **Cluster Setup** | None | None | ServiceAccount | None |
| **New Tools** | None | Kustomize | ArgoCD hooks | Bash script |
| **Error Prone** | High | Low | Low | Medium |
| **GitOps Native** | ✅ Yes | ✅ Yes | ✅ Yes | ✅ Yes |
| **Industry Standard** | ❌ No | ✅ Yes | 🟡 Common | ❌ No |
| **Visibility** | Direct | Direct | ArgoCD UI | Direct |
| **Rollback** | Easy | Automatic | Automatic | Easy |

### **Automation Comparison**

* **Fully Automatic:** Kustomize, PostSync Hook
* **Semi-Automatic:** Helper Script
* **Manual:** Timestamp Annotation

### **Recommended By Use Case**

| **Your Situation** | **Best Option** |
| --- | --- |
| **Want it working NOW** | Timestamp Annotation |
| **Production environment** | **Kustomize (RECOMMENDED)** |
| **Frequent config changes** | **Kustomize (RECOMMENDED)** |
| **Can't use Kustomize** | PostSync Hook |
| **No ServiceAccount permissions** | Helper Script or Timestamp |
| **Learning new tools is OK** | Kustomize |
| **Want industry best practice** | **Kustomize (RECOMMENDED)** |
| **Quick temporary solution** | Timestamp Annotation |
| **Want ArgoCD integration** | PostSync Hook |

## **Recommendations**

### **Primary Recommendation: Kustomize ConfigMapGenerator**

**For most production use cases, Kustomize is the best choice:**

**Why Kustomize?**

* Industry-standard GitOps approach
* Fully automatic after 10-minute setup
* Edit only configuration files, never deployment
* Built into Kubernetes ecosystem
* Zero downtime rollouts
* Easy rollback via Git
* ArgoCD has native support
* Scales to multiple environments

**Implementation Path:**

1. Spend 10 minutes on initial setup
2. Move configs to k8s/base/configs/
3. Create kustomization.yaml
4. Update ArgoCD Application
5. From that point forward: edit one file, commit, push - done!

### **When NOT to Use Kustomize**

**Use Timestamp Annotation if:**

* You need something working in 2 minutes for testing
* This is a temporary/development environment
* You make very infrequent changes (once a month or less)

**Use PostSync Hook if:**

* Your organization prohibits Kustomize
* You already heavily use ArgoCD hooks
* You want restarts on ANY repository change

**Use Helper Script if:**

* You can't use Kustomize or hooks
* You want semi-automation
* You're comfortable with bash scripting

## **Getting Started**

### **Quick Start with Kustomize (Recommended)**

# 1. Create directory structure

mkdir -p k8s/base/configs

# 2. Move your config

mv k8s/app-config.yaml k8s/base/configs/app-config.yaml

# 3. Create kustomization.yaml

cat > k8s/base/kustomization.yaml <<EOF

apiVersion: kustomize.config.k8s.io/v1beta1

kind: Kustomization

namespace: <namespace>

configMapGenerator:

- name: rhdh-app-config

files:

- configs/app-config.yaml

resources:

- deployment.yaml

EOF

# 4. Update your deployment to mount the ConfigMap

# (see Option 2 for deployment YAML example)

# 5. Update ArgoCD to use Kustomize path

# (see Option 2 for ArgoCD Application example)

# 6. Commit and push

git add k8s/base/

git commit -m "Setup Kustomize for automatic rollouts"

git push

# 7. Verify in ArgoCD

argocd app sync rhdh-app

### **Quick Start with Timestamp (Fastest)**

# 1. Add annotation to deployment

# In k8s/deployment.yaml add:

# annotations:

# restartedAt: "2025-10-14T10:00:00Z"

# 2. Commit

git add k8s/deployment.yaml

git commit -m "Add restart annotation"

git push

# 3. When you update config, update timestamp too

## **Troubleshooting**

### **ArgoCD Not Syncing**

# Check ArgoCD application status

argocd app get rhdh-app

# Force sync

argocd app sync rhdh-app --force

# Check sync errors

argocd app logs rhdh-app

### **Deployment Not Restarting**

# Check if deployment actually changed

oc -n <namespace> get deployment redhat-developer-hub -o yaml

# Check pod age

oc -n <namespace> get pods

# Manually trigger rollout to test

oc -n <namespace> rollout restart deployment/redhat-developer-hub

### **Kustomize Errors**

# Test kustomize build locally

cd k8s/base

kustomize build .

# Check ArgoCD kustomize version

argocd app get rhdh-app -o yaml | grep kustomize

### **PostSync Hook Not Running**

# Check jobs

oc -n <namespace> get jobs

# Check job logs

oc -n <namespace> logs job/restart-rhdh-xxxxx

# Check ServiceAccount permissions

oc -n <namespace> auth can-i patch deployment --as=system:serviceaccount:<namespace>:argocd-rollout-sa

## **Additional Resources**

### **Documentation Links**

* **Kustomize Documentation:** https://kustomize.io/
* **ArgoCD Documentation:** https://argo-cd.readthedocs.io/
* **ArgoCD Resource Hooks:** https://argo-cd.readthedocs.io/en/stable/user-guide/resource\_hooks/
* **OpenShift CLI Reference:** https://docs.openshift.com/container-platform/latest/cli\_reference/openshift\_cli/getting-started-cli.html
* **RHDH Documentation:** https://access.redhat.com/documentation/en-us/red\_hat\_developer\_hub/

### **Example Repositories**

Look for example implementations:

* Search GitHub for "kustomize configmap generator"
* Search for "argocd postsync hook"
* Red Hat Developer Hub examples

## **Conclusion**

For production Red Hat Developer Hub deployments in the <namespace> namespace, **Kustomize ConfigMapGenerator** provides the best balance of automation, simplicity, and industry best practices.

**Investment:** 10 minutes setup time  
 **Return:** Automatic rollouts forever with single-file edits

If you need something working immediately for testing, start with **Timestamp Annotation** and migrate to Kustomize later.

For any questions or implementation assistance, consult your DevOps team or Red Hat support.

**Document Version:** 1.0  
 **Last Updated:** October 14, 2025  
 **Environment:** OpenShift / Red Hat Developer Hub  
 **Namespace:** <namespace>