# Automated Version Updates in GitOps with GitHub Actions

This guide explains how to use GitHub Actions to automatically update application versions in a GitOps repository when new versions are released.

## Overview

When you release a new version of your application (API or UI), you typically want to update the corresponding version references in your GitOps repository. This process can be automated using GitHub Actions.

The workflow includes: 1. Triggering on release publication 2. Checking out your GitOps repository 3. Updating the version references in Kustomize files 4. Committing and pushing the changes back 5. Creating a PR to maintain proper approval process

## Implementation

### GitHub Action Workflow

Create a file named .github/workflows/update-gitops-version.yml in your application repository:

name: Update GitOps Version  
  
on:  
 release:  
 types: [published]  
  
jobs:  
 update-gitops:  
 runs-on: ubuntu-latest  
 steps:  
 - name: Extract version  
 id: extract\_version  
 run: echo "VERSION=${GITHUB\_REF#refs/tags/}" >> $GITHUB\_OUTPUT  
  
 - name: Checkout GitOps repository  
 uses: actions/checkout@v4  
 with:  
 repository: your-org/gitops-repo  
 token: ${{ secrets.GITOPS\_PAT }}  
 path: gitops-repo  
  
 - name: Update application version  
 run: |  
 cd gitops-repo  
   
 # Determine app name from repository (or use a fixed name)  
 APP\_NAME=$(echo $GITHUB\_REPOSITORY | cut -d '/' -f 2)  
   
 # Update version in all environments or specific ones  
 # Example for dev environment:  
 sed -i "s|image: $APP\_NAME:.\*|image: $APP\_NAME:${{ steps.extract\_version.outputs.VERSION }}|g" \  
 environments/dev/kustomization.yaml  
   
 # For production environments, you might only update a specific reference  
 # to maintain more controlled promotion process  
   
 echo "Updated $APP\_NAME to version ${{ steps.extract\_version.outputs.VERSION }}"  
  
 - name: Create Pull Request  
 uses: peter-evans/create-pull-request@v5  
 with:  
 path: gitops-repo  
 token: ${{ secrets.GITOPS\_PAT }}  
 commit-message: "chore: update ${{ github.repository }} to ${{ steps.extract\_version.outputs.VERSION }}"  
 title: "Update ${{ github.repository }} to ${{ steps.extract\_version.outputs.VERSION }}"  
 body: |  
 This PR updates the version of ${{ github.repository }} to ${{ steps.extract\_version.outputs.VERSION }}.  
   
 Release link: ${{ github.event.release.html\_url }}  
   
 This is an automated PR created by GitHub Actions.  
 branch: update-${{ github.repository\_owner }}-${{ github.event.repository.name }}-${{ steps.extract\_version.outputs.VERSION }}  
 base: main

### For Kustomize-Specific Updates

If you’re working with Kustomize files, you’ll typically be updating:

1. **The image version in a kustomization.yaml file:**

# Example original file: environments/dev/kustomization.yaml  
apiVersion: kustomize.config.k8s.io/v1beta1  
kind: Kustomization  
resources:  
 - ../../base  
images:  
 - name: myapp  
 newName: myregistry.com/myapp  
 newTag: 1.0.0 # This will be updated to the new version

The script to update this could look like:

# Update Kustomize file  
yq e ".images[] |= select(.name == \"$APP\_NAME\").newTag = \"$VERSION\"" -i environments/dev/kustomization.yaml

1. **Updating a ConfigMap generator:**

# Example original file with ConfigMap generator  
apiVersion: kustomize.config.k8s.io/v1beta1  
kind: Kustomization  
configMapGenerator:  
 - name: app-config  
 literals:  
 - version=1.0.0 # This will be updated

The script to update this could look like:

# Update version in ConfigMap generator  
sed -i "s/version=.\*/version=$VERSION/g" environments/dev/kustomization.yaml

## Setup Requirements

1. **Personal Access Token (PAT)**: Create a PAT with repo permissions and store it as a repository secret named GITOPS\_PAT.
2. **Repository Configuration**: Ensure your GitOps repository is structured with environment-specific Kustomize configurations.
3. **Branch Protection**: Consider implementing branch protection rules on your GitOps repository to ensure changes go through proper reviews.

## Advanced Options

### Selective Environment Updates

For a more controlled promotion process, you might want to:

1. Only update dev environments automatically
2. Create a PR for QA/staging environments that needs manual approval
3. Leave production environments unchanged (requiring manual promotion)

- name: Update selective environments  
 run: |  
 cd gitops-repo  
   
 # Always update dev  
 yq e ".images[] |= select(.name == \"$APP\_NAME\").newTag = \"$VERSION\"" -i environments/dev/kustomization.yaml  
   
 # Update QA only if this is a stable release (not a pre-release)  
 if [[ ! "${{ github.event.release.prerelease }}" == "true" ]]; then  
 yq e ".images[] |= select(.name == \"$APP\_NAME\").newTag = \"$VERSION\"" -i environments/qa/kustomization.yaml  
 fi  
   
 # Never automatically update production  
 echo "Production environment requires manual promotion"

### Multiple Application Components

If your GitOps repo manages multiple components of the same application:

- name: Update specific component  
 run: |  
 cd gitops-repo  
   
 # For frontend component  
 if [[ "${{ github.repository }}" == \*-frontend ]]; then  
 yq e ".images[] |= select(.name == \"frontend\").newTag = \"$VERSION\"" -i environments/dev/kustomization.yaml  
 fi  
   
 # For backend component  
 if [[ "${{ github.repository }}" == \*-backend ]]; then  
 yq e ".images[] |= select(.name == \"backend\").newTag = \"$VERSION\"" -i environments/dev/kustomization.yaml  
 fi

## Best Practices

1. **Atomic Updates**: Update all relevant files in a single commit to ensure consistency.
2. **Validation**: Add a validation step to verify the updates were made correctly.
3. **Notifications**: Configure the workflow to notify relevant teams on success or failure.
4. **Audit Trail**: The PR created provides a clear audit trail of when and why versions were updated.
5. **Review Process**: Even automated updates should go through code review to ensure correctness.