**Continuous Delivery Approach Using ArgoCD and GitOps for Development and QA Deployment**

### **Overview**

This section of the document outlines the continuous delivery approach for deploying a collection of bugs and features into the development environment and selectively promoting them to the QA environment using ArgoCD, GitOps principles, and semantic versioning. The artifacts include APIs (containerized) and UX zip files deployed directly to Azure Storage.

Our approach leverages GitOps principles to ensure that the desired state of our applications is always defined in Git repositories. This enables automated deployments, version control of configurations, and easy rollbacks when needed. The semantic versioning strategy helps maintain clear tracking of changes and facilitates controlled promotions between environments.

### **GitOps Repository Structure**

The GitOps repository structure follows key organizational principles to enable clear separation of concerns, environment-specific configurations, and maintainable infrastructure as code:

1. **Separation by Application Type (api-apps vs ux-apps)**
   * APIs and UX applications have different deployment patterns and requirements
   * APIs are deployed as containers to Kubernetes while UX files go to Azure Storage
   * This separation allows for specialized deployment configurations and workflows
2. **Application-Specific Directories**
   * Each application (e.g., catalog-api, home-page-ux) has its own directory
   * Contains all configurations needed to deploy that specific application
   * Enables independent versioning and deployment of each application
3. **Base/Overlay Pattern**
   * The base directory contains the common configuration shared across environments
   * overlays directory contains environment-specific customizations (dev/qa)
   * Uses Kustomize to merge base configs with environment-specific changes
   * Reduces duplication while maintaining environment differences
4. **ApplicationSets Directory**
   * Contains ArgoCD ApplicationSet definitions
   * Separate ApplicationSets for APIs and UX deployments
   * Enables automated application creation and management
   * Provides scalable way to handle multiple applications and environments

This structure supports GitOps principles by: - Maintaining all configurations in version control - Providing clear separation between environments - Enabling easy rollbacks and version tracking - Supporting declarative infrastructure management

### **Understanding Kustomize, Overlays, and ApplicationSets**

#### Kustomize

Kustomize is a powerful configuration management tool for Kubernetes that allows you to customize application configurations without modifying the original YAML files. Key benefits include:

* **Base/Overlay Pattern**: Maintain a base configuration and create variants (overlays) for different environments
* **No Templates**: Uses pure Kubernetes YAML files without the need for templating language
* **Composable**: Can combine and layer multiple configurations
* **Built into kubectl**: Native support in Kubernetes CLI with kubectl apply -k

For example, you can have a base deployment.yaml:

> \*\*Note\*\*: Decide if we structure the repo by environment first and api and ux second ?

└── gitops-repo  
 ├── api-apps  
 │ ├── catalog-api  
 │ │ ├── kustomization.yaml  
 │ │ ├── base  
 │ │ │ ├── deployment.yaml  
 │ │ │ ├── service.yaml  
 │ │ │ └── configmap.yaml  
 │ │ └── overlays  
 │ │ ├── dev  
 │ │ │ └── kustomization.yaml  
 │ │ └── qa  
 │ │ └── kustomization.yaml  
 ├── ux-apps  
 │ └── home-page-ux  
 │ ├── kustomization.yaml  
 │ ├── base  
 │ │ └── storage-config.yaml  
 │ └── overlays  
 │ ├── dev  
 │ │ └── kustomization.yaml  
 │ └── qa  
 │ └── kustomization.yaml  
 └── applicationsets  
 ├── api-applicationset.yaml  
 └── ux-applicationset.yaml

### **Kustomization Config for API (Development and QA Overlays)**

**Development Overlay** (api-apps/catalog-api/overlays/dev/kustomization.yaml):

resources:  
 - ../../base/deployment.yaml  
 - ../../base/service.yaml  
 - ../../base/configmap.yaml  
  
images:  
 - name: catalog-api  
 newTag: "1.2.3-dev"  
  
configMapGenerator:  
 - name: catalog-api-config  
 literals:  
 - environment=development  
 - logLevel=debug

**QA Overlay** (api-apps/catalog-api/overlays/qa/kustomization.yaml):

resources:  
 - ../../base/deployment.yaml  
 - ../../base/service.yaml  
 - ../../base/configmap.yaml  
  
images:  
 - name: catalog-api  
 newTag: "1.2.3"  
  
configMapGenerator:  
 - name: catalog-api-config  
 literals:  
 - environment=qa  
 - logLevel=info

### **Kustomization Config for UX (Development and QA Overlays)**

**Development Overlay** (ux-apps/home-page-ux/overlays/dev/kustomization.yaml):

resources:  
 - ../../base/storage-config.yaml  
  
configMapGenerator:  
 - name: ux-config  
 literals:  
 - storageAccount=dev-storage-account  
 - zipVersion=1.2.3-dev

**QA Overlay** (ux-apps/home-page-ux/overlays/qa/kustomization.yaml):

resources:  
 - ../../base/storage-config.yaml  
  
configMapGenerator:  
 - name: ux-config  
 literals:  
 - storageAccount=qa-storage-account  
 - zipVersion=1.2.3

### **ApplicationSet for API**

apiVersion: argoproj.io/v1alpha1  
kind: ApplicationSet  
metadata:  
 name: api-deployments  
spec:  
 generators:  
 git:  
 repoURL: https://github.com/your-org/gitops-repo.git  
 revision: main  
 directories:  
 - path: api-apps/\*/overlays/\*  
 template:  
 metadata:  
 name: '{{path.basename}}-{{path.dirname.basename}}'  
 spec:  
 project: default  
 source:  
 repoURL: https://github.com/your-org/gitops-repo.git  
 targetRevision: main  
 path: '{{path}}'  
 destination:  
 server: https://kubernetes.default.svc  
 namespace: '{{path.dirname.basename}}'  
 syncPolicy:  
 automated:  
 prune: true  
 selfHeal: true

### **ApplicationSet for UX**

apiVersion: argoproj.io/v1alpha1  
kind: ApplicationSet  
metadata:  
 name: ux-deployments  
spec:  
 generators:  
 git:  
 repoURL: https://github.com/your-org/gitops-repo.git  
 revision: main  
 directories:  
 - path: ux-apps/\*/overlays/\*  
 template:  
 metadata:  
 name: '{{path.basename}}-{{path.dirname.basename}}'  
 spec:  
 project: default  
 source:  
 repoURL: https://github.com/your-org/gitops-repo.git  
 targetRevision: main  
 path: '{{path}}'  
 destination:  
 server: https://kubernetes.default.svc  
 namespace: '{{path.dirname.basename}}'  
 syncPolicy:  
 automated:  
 prune: true  
 selfHeal: true

## Moving zip to Azure Storage

### **Azure Storage Deployment Configuration**

To move zip files to Azure Storage using ArgoCD and ApplicationSets, follow these configuration steps:

1. First, create a Kubernetes Secret for Azure Storage credentials:

apiVersion: v1  
kind: Secret  
metadata:  
 name: azure-storage-secret  
type: Opaque  
data:  
 AZURE\_STORAGE\_ACCOUNT: <base64-encoded-storage-account-name>  
 AZURE\_STORAGE\_KEY: <base64-encoded-storage-key>  
 # Or use SAS token  
 AZURE\_STORAGE\_SAS\_TOKEN: <base64-encoded-sas-token>

1. Create an Argo Workflow Template

apiVersion: argoproj.io/v1alpha1  
kind: Workflow  
metadata:  
 name: azure-blob-upload  
spec:  
 entrypoint: upload-to-blob  
 templates:  
 - name: upload-to-blob  
 container:  
 image: mcr.microsoft.com/azure-cli  
 command: ["/bin/sh", "-c"]  
 args:  
 - |  
 # Install required tools  
 apk add --no-cache curl  
  
 # Upload file to Azure Blob Storage  
 az storage blob upload \  
 --account-name $AZURE\_STORAGE\_ACCOUNT \  
 --container-name your-container-name \  
 --name destination-filename.zip \  
 --file /path/to/your/source.zip \  
 --auth-mode key \  
 --account-key $AZURE\_STORAGE\_KEY  
 env:  
 - name: AZURE\_STORAGE\_ACCOUNT  
 valueFrom:  
 secretKeyRef:  
 name: azure-storage-secret  
 key: AZURE\_STORAGE\_ACCOUNT  
 - name: AZURE\_STORAGE\_KEY  
 valueFrom:  
 secretKeyRef:  
 name: azure-storage-secret  
 key: AZURE\_STORAGE\_KEY

1. Create a Kustomization File

apiVersion: kustomize.config.k8s.io/v1beta1  
kind: Kustomization  
  
resources:  
 - azure-secret.yaml  
 - workflow.yaml  
  
# Optional: Add configMapGenerator if you need to configure paths or other variables  
configMapGenerator:  
 - name: blob-config  
 literals:  
 - CONTAINER\_NAME=your-container-name  
 - BLOB\_PATH=your/blob/path

1. Apply the Configuration

# Create base64 encoded secrets first  
echo -n "your-storage-account-name" | base64  
echo -n "your-storage-key" | base64  
  
# Update the secrets in azure-secret.yaml with the base64 values  
  
# Apply using kustomize  
kubectl apply -k .  
  
# Submit the Argo workflow  
argo submit --watch workflow.yaml

**Important Notes:**

1. **Security Considerations:**
   * Store sensitive Azure credentials in Kubernetes secrets
   * Consider using Azure Managed Identity if running in AKS
   * Use SAS tokens with limited permissions when possible
2. **Alternative Using Azure Managed Identity:**

apiVersion: argoproj.io/v1alpha1  
kind: Workflow  
metadata:  
 name: azure-blob-upload  
spec:  
 entrypoint: upload-to-blob  
 templates:  
 - name: upload-to-blob  
 container:  
 image: mcr.microsoft.com/azure-cli  
 command: ["/bin/sh", "-c"]  
 args:  
 - |  
 # Using managed identity  
 az storage blob upload \  
 --account-name $AZURE\_STORAGE\_ACCOUNT \  
 --container-name your-container-name \  
 --name destination-filename.zip \  
 --file /path/to/your/source.zip \  
 --auth-mode login

1. **For Large Files:**
   * Consider using azcopy instead of az cli for better performance:

* apiVersion: argoproj.io/v1alpha1  
  kind: Workflow  
  metadata:  
   name: azure-blob-upload  
  spec:  
   entrypoint: upload-to-blob  
   templates:  
   - name: upload-to-blob  
   container:  
   image: mcr.microsoft.com/azure-cli  
   command: ["/bin/sh", "-c"]  
   args:  
   - |  
   # Install azcopy  
   wget https://aka.ms/downloadazcopy-v10-linux  
   tar -xvf downloadazcopy-v10-linux  
   cp ./azcopy\_linux\_amd64\_\*/azcopy /usr/bin/  
    
   # Upload using azcopy  
   azcopy copy "/path/to/your/source.zip" \  
   "https://$AZURE\_STORAGE\_ACCOUNT.blob.core.windows.net/your-container-name/destination-filename.zip$AZURE\_STORAGE\_SAS\_TOKEN"

1. **Monitoring and Logging:**
   * Add appropriate logging to track upload progress
   * Consider adding retry logic for resilience
   * Use Argo’s built-in error handling:

* spec:  
   templates:  
   - name: upload-to-blob  
   retryStrategy:  
   limit: 3  
   retryPolicy: "Always"  
   backoff:  
   duration: "10s"  
   factor: 2

To monitor the workflow:

argo list  
argo get azure-blob-upload  
argo logs azure-blob-upload

Remember to replace placeholders like your-container-name, /path/to/your/source.zip, and the Azure credentials with your actual values.

### **Considerations**

* Separate ApplicationSets allow independent control over API and UX deployments.
* Kustomize overlays ensure environment-specific configuration.
* Direct UX deployments to Azure Storage ensure efficient asset handling.
* Validation processes are built into the deployment lifecycle.

This detailed approach ensures a structured, scalable, and efficient CI/CD pipeline for both API and UX components using ArgoCD, Kustomize, and GitOps principles.