Pour créer un opérateur utilisant Operator SDK avec Go pour gérer un workflow CI/CD, suivez ces étapes :

1. **Installer Operator SDK**:
   Assurez-vous d'avoir installé l'Operator SDK, Kubectl, et Docker.

2. **Créer un projet Go Operator**:

```
operator-sdk init --domain example.com --repo
github.com/example/go-operator
```

3. **Créer l'API pour la Custom Resource**:

```
operator-sdk create api --group cicd --version v1alpha1 --kind
Workflow --resource --controller
```

4. **Définir la structure de la Custom Resource**:

   Modifiez le fichier `api/v1alpha1/workflow_types.go` pour définir votre CRD. Voici un exemple :

```
package v1alpha1

import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

// EDIT THIS FILE!  THIS IS SCAFFOLDING FOR YOU TO OWN!

// WorkflowSpec defines the desired state of Workflow
type WorkflowSpec struct {
    Pipeline   string `json:"pipeline,omitempty"`
    Repository string `json:"repository,omitempty"`
    Branch     string `json:"branch,omitempty"`
}

// WorkflowStatus defines the observed state of Workflow
type WorkflowStatus struct {
    Status  string `json:"status,omitempty"`
    Message string `json:"message,omitempty"`
}

// +kubebuilder:object:root=true
// +kubebuilder:subresource:status

// Workflow is the Schema for the workflows API
```

```go
type Workflow struct {
    metav1.TypeMeta   `json:",inline"`
    metav1.ObjectMeta `json:"metadata,omitempty"`

    Spec   WorkflowSpec   `json:"spec,omitempty"`
    Status WorkflowStatus `json:"status,omitempty"`
}

// +kubebuilder:object:root=true

// WorkflowList contains a list of Workflow
type WorkflowList struct {
    metav1.TypeMeta `json:",inline"`
    metav1.ListMeta `json:"metadata,omitempty"`
    Items           []Workflow `json:"items"`
}

func init() {
    SchemeBuilder.Register(&Workflow{}, &WorkflowList{})
}
```

5. **Définir le contrôleur**:

   Modifiez le fichier `controllers/workflow_controller.go` pour implémenter la logique du contrôleur. Voici un exemple simplifié :

```go
package controllers

import (
    "context"
    "fmt"

    "github.com/go-logr/logr"
    "k8s.io/apimachinery/pkg/api/errors"
    "k8s.io/apimachinery/pkg/runtime"
    ctrl "sigs.k8s.io/controller-runtime"
    "sigs.k8s.io/controller-runtime/pkg/client"
    "sigs.k8s.io/controller-runtime/pkg/controller"
    "sigs.k8s.io/controller-runtime/pkg/controller/controllerutil"
    "sigs.k8s.io/controller-runtime/pkg/log"

    cicdv1alpha1 "github.com/example/go-operator/api/v1alpha1"
    corev1 "k8s.io/api/core/v1"
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

// WorkflowReconciler reconciles a Workflow object
type WorkflowReconciler struct {
    client.Client
    Scheme *runtime.Scheme
```

```
}

//+kubebuilder:rbac:groups=cicd.example.com,resources=workflows,ver
bs=get;list;watch;create;update;patch;delete
//+kubebuilder:rbac:groups=cicd.example.com,resources=workflows/sta
tus,verbs=get;update;patch
//+kubebuilder:rbac:groups=cicd.example.com,resources=workflows/fin
alizers,verbs=update
//+kubebuilder:rbac:groups="",resources=pods,verbs=create;update;de
lete;get;list;watch

func (r *WorkflowReconciler) Reconcile(ctx context.Context, req
ctrl.Request) (ctrl.Result, error) {
    _ = log.FromContext(ctx)

    // Fetch the Workflow instance
    workflow := &cicdv1alpha1.Workflow{}
    err := r.Get(ctx, req.NamespacedName, workflow)
    if err != nil {
        if errors.IsNotFound(err) {
            // Request object not found, could have been deleted
after reconcile request.
            // Owned objects are automatically garbage collected.
            // For additional cleanup logic use finalizers.
            return ctrl.Result{}, nil
        }
        // Error reading the object - requeue the request.
        return ctrl.Result{}, err
    }

    // Define a new Pod object
    pod := r.podForWorkflow(workflow)

    // Set Workflow instance as the owner and controller
    if err := controllerutil.SetControllerReference(workflow, pod,
r.Scheme); err != nil {
        return ctrl.Result{}, err
    }

    // Check if this Pod already exists
    found := &corev1.Pod{}
    err = r.Get(ctx, client.ObjectKey{Name: pod.Name, Namespace:
pod.Namespace}, found)
    if err != nil && errors.IsNotFound(err) {
        log.Log.Info("Creating a new Pod", "Pod.Namespace",
pod.Namespace, "Pod.Name", pod.Name)
        err = r.Create(ctx, pod)
        if err != nil {
            return ctrl.Result{}, err
        }

        // Pod created successfully - don't requeue
        return ctrl.Result{}, nil
```

```go
    } else if err != nil {
        return ctrl.Result{}, err
    }

    // Pod already exists - don't requeue
    log.Log.Info("Skip reconcile: Pod already exists",
"Pod.Namespace", found.Namespace, "Pod.Name", found.Name)
    return ctrl.Result{}, nil
}

// podForWorkflow returns a Workflow Pod object
func (r *WorkflowReconciler) podForWorkflow(m
*cicdv1alpha1.Workflow) *corev1.Pod {
    labels := map[string]string{
        "app": m.Name,
    }
    return &corev1.Pod{
        ObjectMeta: metav1.ObjectMeta{
            Name:      fmt.Sprintf("workflow-pod-%s", m.Name),
            Namespace: m.Namespace,
            Labels:    labels,
        },
        Spec: corev1.PodSpec{
            Containers: []corev1.Container{{
                Name:  "workflow",
                Image: "alpine",
                Command: []string{
                    "/bin/sh",
                    "-c",
                    fmt.Sprintf("echo Running pipeline %s for repo
%s on branch %s", m.Spec.Pipeline, m.Spec.Repository,
m.Spec.Branch),
                },
            }},
        },
    }
}

func (r *WorkflowReconciler) SetupWithManager(mgr ctrl.Manager)
error {
    return ctrl.NewControllerManagedBy(mgr).
        For(&cicdv1alpha1.Workflow{}).
        Owns(&corev1.Pod{}).
        Complete(r)
}
```

6. **Mettre à jour la configuration et déployer**:
   - Créez les fichiers nécessaires dans le dossier `config` pour le déploiement de l'opérateur.
   - Construisez et poussez l'image Docker.
   - Déployez les CRD et l'opérateur sur votre cluster Kubernetes.

---

```
make docker-build docker-push IMG=your-operator-image:tag
make deploy IMG=your-operator-image:tag
```

7. **Créer une instance de la ressource personnalisée**:

   Créez un fichier `workflow.yaml` pour définir une instance de votre CRD :

   ```yaml
   apiVersion: cicd.example.com/v1alpha1
   kind: Workflow
   metadata:
     name: example-workflow
   spec:
     pipeline: "example-pipeline"
     repository: "https://github.com/example/repo.git"
     branch: "main"
   ```

   Appliquez cette configuration :

   ```
   kubectl apply -f workflow.yaml
   ```

Avec ces étapes, vous avez un opérateur Kubernetes utilisant Go et Operator SDK qui peut gérer des workflows CI/CD. Vous pouvez adapter et étendre cette configuration en fonction de vos besoins spécifiques.