Building a Kubernetes Operator with Kubebuilder 🚀

1. Pre-requisites 📋

Before we start, make sure you have the following installed:

- Go (v1.13 or higher)
- Docker (for building and pushing the operator image)
- `kubectl` (to interact with your cluster)
- A Kubernetes cluster (like Minikube or KIND)

2. Install Kubebuilder 📥

Install the latest release of `kubebuilder` from the official release page.

3. Initialize Your Operator Project 🛠️

Let's start by creating a new directory for our operator and navigating into it:

```
mkdir  -p /mnt/c/project/nimble-opti-adapter
cd /mnt/c/project/nimble-opti-adapter
```

Now, initialize the project:

```
kubebuilder init --domain uri-tech.github.io --repo github.com/uri-
tech/nimble-opti-adapter
```

4. Define Your Custom Resource (CR) 📄

CRs are the heart of your operator. Let's define one:

```
kubebuilder create api --group=adapter --version=v1 --kind=NimbleOpti --
resource=true --controller=true
```

This will scaffold the `NimbleOpti` CRD and its associated controller.

5. Implement the Operator Logic 🧠

Navigate to the generated controller file, likely at:

`controllers/adapter_nimbleopti_controller.go`

Here, implement the reconciliation logic inside the `Reconcile` method. This is where the magic happens!

6. Watch for Ingress and NimbleOpti Resources 👮

To ensure our operator reacts to changes in Ingress and NimbleOpti resources, add watches for these types in the controller's setup.

Modify the SetupWithManager method in `controllers/adapter_nimbleopti_controller.go` to include:

```
// SetupWithManager sets up the controller with the Manager. The Manager
will set fields on the Controller
// and Start it when the Manager is Started.
func (r *NimbleOptiReconciler) SetupWithManager(mgr ctrl.Manager) error {
    // debug
    klog.InfoS("debug - SetupWithManager")

    // NewControllerManagedBy returns a builder for a controller
    // managed by mgr that will be started with mgr.Start.
    b := ctrl.NewControllerManagedBy(mgr)

    // For the primary resource type that this controller watches
    b = b.For(&adapterv1.NimbleOpti{})

    // Owns specifies objects that are owned by the primary resource
    // The argument here must be a runtime object that will have its
    // Group, Version, and Kind filled in.
    b = b.Owns(&networkingv1.Ingress{})

    // WithEventFilter specifies a Predicate that will be used to filter
    // events before they are sent to event handlers.
    // The GenerationChangedPredicate filters out objects that have not
changed their .metadata.generation field.
    // b = b.WithEventFilter(predicate.GenerationChangedPredicate{})

    // Watch Ingress objects - do not need it - do it from the
IngressWatcher
    // if err := b.Watches(&networkingv1.Ingress{},
&handler.EnqueueRequestForObject{}); err != nil {
    //  klog.Error(err, "unable to watch Ingress")
    // }

    // Watch for changes to Pods
    // if err := b.Watches(&source.Kind{Type: &corev1.Pod{}},
&handler.EnqueueRequestForObject{}); err != nil {
    //  klog.Error(err, "unable to watch Pods")
    // }

    // Call Complete to create the NimbleOptiReconciler. This step comes
at the end
    // as it finalizes the controller's configuration.
    return b.Complete(r)
}
```

7. Update the operator core files:

./cmd/main.go:

```
package main

import (
    "flag"
    "os"

    // Import all Kubernetes client auth plugins (e.g. Azure, GCP, OIDC,
etc.)
    // to ensure that exec-entrypoint and run can make use of them.
    "k8s.io/client-go/kubernetes"
    _ "k8s.io/client-go/plugin/pkg/client/auth"
    "k8s.io/klog/v2"

    "k8s.io/apimachinery/pkg/runtime"
    utilruntime "k8s.io/apimachinery/pkg/util/runtime"
    clientgoscheme "k8s.io/client-go/kubernetes/scheme"
    ctrl "sigs.k8s.io/controller-runtime"
    "sigs.k8s.io/controller-runtime/pkg/healthz"
    "sigs.k8s.io/controller-runtime/pkg/log/zap"

    adapterv1 "github.com/uri-tech/nimble-opti-adapter/api/v1"
    "github.com/uri-tech/nimble-opti-adapter/internal/controller"
    //+kubebuilder:scaffold:imports
)

var (
    scheme   = runtime.NewScheme()
    setupLog = ctrl.Log.WithName("setup")
)

func init() {
    // debug
    klog.InfoS("debug - init")

    utilruntime.Must(clientgoscheme.AddToScheme(scheme))

    utilruntime.Must(adapterv1.AddToScheme(scheme))
    //+kubebuilder:scaffold:scheme
}

func main() {
    // debug
    klog.InfoS("debug - main")

    // Define command-line flags.
    var metricsAddr string
    var enableLeaderElection bool
    var probeAddr string
```

```
    flag.StringVar(&metricsAddr, "metrics-bind-address", ":8080", "The
address the metric endpoint binds to.")
    flag.StringVar(&probeAddr, "health-probe-bind-address", ":8081", "The
address the probe endpoint binds to.")
    flag.BoolVar(&enableLeaderElection, "leader-elect", false,
        "Enable leader election for controller manager. "+
            "Enabling this will ensure there is only one active controller
manager.")
    opts := zap.Options{
        Development: true,
    }
    opts.BindFlags(flag.CommandLine)
    flag.Parse()

    // Set the logger for the controller-runtime package.
    ctrl.SetLogger(zap.New(zap.UseFlagOptions(&opts)))

    // Create a new manager to provide shared dependencies and start
components.
    mgr, err := ctrl.NewManager(ctrl.GetConfigOrDie(), ctrl.Options{
        Scheme:                 scheme,
        MetricsBindAddress:     metricsAddr,
        Port:                   9443,
        HealthProbeBindAddress: probeAddr,
        LeaderElection:         enableLeaderElection,
        LeaderElectionID:       "8f24f142.uri-tech.github.io",
        // LeaderElectionReleaseOnCancel defines if the leader should step
down voluntarily
        // when the Manager ends. This requires the binary to immediately
end when the
        // Manager is stopped, otherwise, this setting is unsafe. Setting
this significantly
        // speeds up voluntary leader transitions as the new leader don't
have to wait
        // LeaseDuration time first.
        //
        // In the default scaffold provided, the program ends immediately
after
        // the manager stops, so would be fine to enable this option.
However,
        // if you are doing or is intended to do any operation such as
perform cleanups
        // after the manager stops then its usage might be unsafe.
        // LeaderElectionReleaseOnCancel: true,
    })
    if err != nil {
        setupLog.Error(err, "unable to start manager")
        os.Exit(1)
    }

    // Initialize the Kubernetes client.
    kubernetesClient := kubernetes.NewForConfigOrDie(mgr.GetConfig())

    // Initialize the IngressWatcher.
```

```
    ingressWatcher := &controller.IngressWatcher{
        Client: kubernetesClient,
    }

    // Pass the KubernetesClient and IngressWatcher to the
NimbleOptiReconciler.
    if err = (&controller.NimbleOptiReconciler{
        Client:           mgr.GetClient(),
        Scheme:           mgr.GetScheme(),
        KubernetesClient: kubernetesClient,
        IngressWatcher:   ingressWatcher,
    }).SetupWithManager(mgr); err != nil {
        setupLog.Error(err, "unable to create controller", "controller",
"NimbleOpti")
        os.Exit(1)
    }
    //+kubebuilder:scaffold:builder

    // if err = (&controller.NimbleOptiReconciler{
    //  Client: mgr.GetClient(),
    //  Scheme: mgr.GetScheme(),
    // }).SetupWithManager(mgr); err != nil {
    //  setupLog.Error(err, "unable to create controller", "controller",
"NimbleOpti")
    //  os.Exit(1)
    // }
    // //+kubebuilder:scaffold:builder

    // Add a health check to the manager.
    if err := mgr.AddHealthzCheck("healthz", healthz.Ping); err != nil {
        setupLog.Error(err, "unable to set up health check")
        os.Exit(1)
    }

    // Add a readiness check to the manager.
    if err := mgr.AddReadyzCheck("readyz", healthz.Ping); err != nil {
        setupLog.Error(err, "unable to set up ready check")
        os.Exit(1)
    }

    // Start the manager and listen for the termination signal.
    setupLog.Info("starting manager")
    if err := mgr.Start(ctrl.SetupSignalHandler()); err != nil {
        setupLog.Error(err, "problem running manager")
        os.Exit(1)
    }
}
```

./api/v1/nimbleopti_types.go:

```
package v1
```

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

// EDIT THIS FILE!  THIS IS SCAFFOLDING FOR YOU TO OWN!
// NOTE: json tags are required.  Any new fields you add must have json
tags for the fields to be serialized.

// NimbleOptiSpec defines the desired state of NimbleOpti
type NimbleOptiSpec struct {
    // TargetNamespace is the namespace where the operator should manage
certificates
    // +kubebuilder:validation:MinLength=1
    TargetNamespace string `json:"targetNamespace"`

    // CertificateRenewalThreshold is the waiting time (in days) before
the certificate expires to trigger renewal
    // +kubebuilder:validation:Minimum=1
    CertificateRenewalThreshold int `json:"certificateRenewalThreshold"`

    // AnnotationRemovalDelay is the delay (in seconds) after removing the
"nginx.ingress.kubernetes.io/backend-protocol: HTTPS" annotation before
re-adding it
    // +kubebuilder:validation:Minimum=1
    AnnotationRemovalDelay int `json:"annotationRemovalDelay"`

    // RenewalCheckInterval is the interval (in minutes) for checking
certificate renewals
    // +kubebuilder:validation:Minimum=1
    RenewalCheckInterval int `json:"renewalCheckInterval"`
}

// NimbleOptiStatus defines the observed state of NimbleOpti
type NimbleOptiStatus struct {
    // Conditions are the conditions for this resource.
    Conditions []metav1.Condition `json:"conditions,omitempty"`

    // IngressPathsForRenewal is a list of ingress paths for which
certificates need to be renewed.
    IngressPathsForRenewal []string
`json:"ingressPathsForRenewal,omitempty"`
}

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

// NimbleOpti is the Schema for the nimbleoptiadapters API
type NimbleOpti struct {
    metav1.TypeMeta   `json:",inline"`
    metav1.ObjectMeta `json:"metadata,omitempty"`

    Spec   NimbleOptiSpec   `json:"spec,omitempty"`
    Status NimbleOptiStatus `json:"status,omitempty"`
}
```

```
//+kubebuilder:object:root=true

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

func init() {
    SchemeBuilder.Register(&NimbleOpti{}, &NimbleOptiList{})
}
```

For adding control of ingress in the ServiceAcount we modify `./config/rbac/auth_proxy_role.yaml`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/name: clusterrole
    app.kubernetes.io/instance: proxy-role
    app.kubernetes.io/component: kube-rbac-proxy
    app.kubernetes.io/created-by: nimble-opti-adapter
    app.kubernetes.io/part-of: nimble-opti-adapter
    app.kubernetes.io/managed-by: kustomize
  name: proxy-role
rules:
  - apiGroups:
      - authentication.k8s.io
    resources:
      - tokenreviews
    verbs:
      - create
  - apiGroups:
      - authorization.k8s.io
    resources:
      - subjectaccessreviews
    verbs:
      - create
  - apiGroups:
      - networking.k8s.io
    resources:
      - ingresses
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
```

8. Generate CRD manifests:

When you modify the API definitions, you should regenerate CRD manifests using kubebuilder or operator-sdk tools (whichever you are using)

```
make manifests
make build
```

9. Build and Test the Operator 🚧

If we are using the default Kubebuilder setup:

1. **make install**: This will install the CRDs into the cluster.
2. **make run**: This will run the operator outside of the cluster (for testing purposes).
3. **make docker-build**: This will build the docker image.
4. **make docker-push**: This will push the docker image to the specified registry.
5. **make deploy**: This will deploy the operator into the cluster.

First, let's build our operator's Docker image:

```
docker build -t nimbleopti/nimble-opti-adapter:latest .
docker push nimbleopti/nimble-opti-adapter:latest
```

Crafting a Helm Chart for the Nimble-Opti-Adapter Operator 🚀

1. Pre-requisites 📋

Ensure you've got these tools in your developer's toolbox:

- Helm 3.x installed
- `kubectl` (to interact with your cluster)
- Access to a Kubernetes cluster

2. Initializing the Helm Chart 🏗️

To create a new chart, we'll be using the `helm create` command:

```
cd /mnt/c/project/nimble-opti-adapter
helm create nimble-opti-adapter
```

This will generate a new directory called `nimble-opti-adapter` with the structure of a standard Helm chart.

3. Understanding the Chart Structure 🗺️

Navigate to your `nimble-opti-adapter` directory, and you'll see a structure similar to:

```
nimble-opti-adapter/
│
├── charts/ # Dependencies (other charts this chart depends on)
├── templates/ # Kubernetes YAML templates with variables
│ ├── deployment.yaml
│ ├── service.yaml
│ ├── _helpers.tpl # Template helpers and definitions
│ └── ...
├── values.yaml # Default configuration values for the templates
└── Chart.yaml # Information about the chart
```

4. Tailoring the Chart for Our Operator 📐 ✂

1. **Chart.yaml** 📝 : Begin by updating the `Chart.yaml` with relevant details like the version, description, and maintainers.
2. **values.yaml** 💼 : This file holds the default values for your Helm chart. Update it with parameters specific to the `nimble-opti-adapter` operator, like image repository, tag, resources, etc.
3. **Templates** 🎨 : In the `templates/` directory, customize the existing templates or add new ones. Given this is an operator, you'd likely have:
    - A Deployment for the operator
    - CustomResourceDefinitions (CRDs) the operator will manage
    - RBAC roles and role bindings
    - Any additional resources the operator might need

Remember to make use of Helm's templating functions to ensure flexibility and configurability.

4. **complete chart structure** :

```
nimble-opti-adapter/             # Root directory of your Helm chart.
│
├── Chart.yaml                   # Contains metadata about the Helm chart
like its version, description, etc.
├── README.md                    # Documentation for how to use the chart.
├── charts                       # Directory for Helm chart dependencies
(other charts that this chart relies upon).
│
├── crds                         # Custom Resource Definitions (CRDs) that
should be installed before any other resources.
│   └── nimbleopti.yaml          # The CRD definition for NimbleOpti.
│
├── templates                    # Kubernetes resource templates.
│   ├── NOTES.txt                # Notes that will be displayed to the
```

```
user after helm install or helm upgrade.
│   ├── _helpers.tpl             # Define helper methods and variables for
templating.
│   ├── deployment.yaml          # Template for the Deployment resource.
Can be used to deploy applications, controllers, etc.
│   └── nimbleopti-instance.yaml # Template for an instance of the
NimbleOpti CRD.
│
└── values.yaml                  # Default configuration values that the
templates will use. Can be overridden by the user during install/upgrade.
```

5. Linting & Testing 🧪

Before deploying, it's essential to ensure your chart is free from errors:

```
helm lint nimble-opti-adapter
```

It should return:

```
==> Linting nimble-opti-adapter
[INFO] Chart.yaml: icon is recommended
1 chart(s) linted, 0 chart(s) failed
```

To test your Helm chart's rendering:

```
helm install nimble-opti-adapter ./nimble-opti-adapter --dry-run --debug
```

9. Deploying the Operator ⛵

Option 1 - using helm:

install the CRDs and deploy the operator:

```
helm install nimble-opti-adapter ./nimble-opti-adapter
```

And monitor the release:

```
helm list
```

or using this gide: install using helm

Option 2 - directly:

Deploy the CRDs:

```
make install
```

Now, deploy the operator in your cluster:

```
make deploy IMG=nimbleopti/nimble-opti-adapter:latest
```

10. Using github action to auto build and push the image:

create the dir:

```
mkdir  -p /mnt/c/project/nimble-opti-adapter/.github/workflows
cd /mnt/c/project/nimble-opti-adapter/.github/workflows
```

create the file `NimbleOpti_CI.yml`:

```
name: NimbleOpti_CI

on:
  workflow_dispatch:
  push:
    branches:
      - "main"
    # pull_request:
    #   branches: [ "main" ]
    # paths:
    #   # - "controllers/**"
    #   # - "Dockerfile"
    #   # - "main.go"
    #   - "*"
    #   # Exclude README.md from triggering the workflow
    paths-ignore:
      - "README.md"
jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Check the repo.
        uses: actions/checkout@v3

      - name: Login to docker hub registry.
        run: |
```

```
          docker login docker.io -u ${{ secrets.DOCKER_USERNAME }} -p ${{
secrets.DOCKER_PASSWORD }}

      # - name: Build and push "NimbleOpti".
      #   run: |
      #     IMAGE_TAG=$(date +%s)
      #     docker build ./NimbleOpti --file ./NimbleOpti/Dockerfile --tag
nimbleopti/nimble-opti-adapter:$IMAGE_TAG --tag nimbleopti/nimble-opti-
adapter:latest
      #     docker push nimbleopti/nimble-opti-adapter:$IMAGE_TAG
      #     docker push nimbleopti/nimble-opti-adapter:latest

      - name: Build and push "NimbleOpti".
        run: |
          IMAGE_TAG=$(date +%s)
          docker run --rm --privileged tonistiigi/binfmt --install all
          docker buildx create --use
          DOCKER_TARGET_PLATFORM="linux/arm64,linux/amd64"
          DOCKER_USERNAME="nimbleopti"
          DOCKER_IMAGE_NAME="nimble-opti-adapter"
          docker buildx build . --platform $DOCKER_TARGET_PLATFORM \
          --build-arg BUILD_DATE=$(date -u +'%Y-%m-%dT%H:%M:%SZ') \
          --build-arg VCS_REF=$(git rev-parse --short HEAD) \
          --build-arg VERSION=0.1 \
          --tag $DOCKER_USERNAME/$DOCKER_IMAGE_NAME:$IMAGE_TAG --tag
$DOCKER_USERNAME/$DOCKER_IMAGE_NAME:latest  \
          --file ./Dockerfile \
          --output type=image,push=true
```

11. Watch Your Operator in Action 🍿

Create instances of your `NimbleOpti` custom resource and watch your operator ensure the desired state!

12. Celebrate! 🎉

Congratulations! We just finish building a Kubernetes operator using `kubebuilder`! 🥳

docker info:

```
nimbleopti
&mQ6S2L2Yg57$xY
```

problem fix:

```
go clean -modcache
go mod tidy
go mod download
go get sigs.k8s.io/controller-runtime@latest
```

scrpit - create local env and install operator:

```
#!/bin/bash
set -e

DOCKER_USERNAME=${DOCKER_USERNAME:-nimbleopti}
DOCKER_IMAGE_NAME=${DOCKER_IMAGE_NAME:-${DOCKER_USERNAME}/nimble-opti-
adapter:latest}
CERT_MANAGER_VERSION=${CERT_MANAGER_VERSION:-v1.11.0}
SLEEP_TIME=${SLEEP_TIME:-1}

# echo "Login to docker..."
# echo $DOCKER_PASSWORD | docker login -u $DOCKER_USERNAME --password-
stdin

echo "Deleting Minikube..."
minikube delete

echo "Starting Minikube..."
minikube start

echo "Making manifests..."
make manifests

echo "Installing..."
make install

echo "Building Docker image..."
docker build -t $DOCKER_IMAGE_NAME .

echo "Pushing Docker image..."
docker push $DOCKER_IMAGE_NAME

echo "Deploying..."
make deploy IMG=$DOCKER_IMAGE_NAME

echo "Patching deployment..."
kubectl patch deployment nimble-opti-adapter-controller-manager -n nimble-
opti-adapter-system -p '{"spec":{"template":{"spec":{"containers":
[{"name":"kube-rbac-proxy","imagePullPolicy":"Always"},
{"name":"manager","imagePullPolicy":"Always"}]}}}}'

echo "Setting Minikube context..."
kubectl config use-context minikube

echo "Adding helm repo..."
helm repo add jetstack https://charts.jetstack.io

echo "Updating helm repo..."
helm repo update
```

```
echo "Installing cert-manager..."
helm install \
    cert-manager jetstack/cert-manager \
    --namespace cert-manager \
    --create-namespace \
    --version $CERT_MANAGER_VERSION \
    --set installCRDs=true \
    --wait

echo "Enabling Minikube ingress..."
minikube addons enable ingress

echo "Starting Minikube dashboard..."
minikube dashboard --url &

# Sleep for 1 second to allow the dashboard to start
sleep $SLEEP_TIME

echo "Setup complete."
```

scrpit - update operator:

```
#!/bin/bash

# causes the shell to exit if any invoked command exits with a non-zero
status
set -e

DOCKER_USERNAME=${DOCKER_USERNAME:-nimbleopti}
DOCKER_IMAGE_NAME=${DOCKER_IMAGE_NAME:-${DOCKER_USERNAME}/nimble-opti-
adapter:latest}

echo "Patching deployment..."
kubectl patch deployment nimble-opti-adapter-controller-manager -n nimble-
opti-adapter-system -p '{"spec":{"template":{"spec":{"containers":
[{"name":"kube-rbac-proxy","imagePullPolicy":"Always"},
{"name":"manager","imagePullPolicy":"Always"}]}}}}'

echo "Making manifests..."
make manifests

echo "Installing..."
make install

echo "Building Docker image..."
docker build -t $DOCKER_IMAGE_NAME .

echo "Pushing Docker image..."
docker push $DOCKER_IMAGE_NAME

echo "Deploying..."
make deploy IMG=$DOCKER_IMAGE_NAME
```

```
echo "Rolling out updates..."
kubectl rollout restart deployment/nimble-opti-adapter-controller-manager
-n nimble-opti-adapter-system
kubectl rollout status deployment/nimble-opti-adapter-controller-manager -
n nimble-opti-adapter-system

echo "Patching deployment..."
kubectl patch deployment nimble-opti-adapter-controller-manager -n nimble-
opti-adapter-system -p '{"spec":{"template":{"spec":{"containers":
[{"name":"kube-rbac-proxy","imagePullPolicy":"Always"},
{"name":"manager","imagePullPolicy":"Always"}]}}}}'

echo "Update complete."
```

scrpit - testing the operator:

```
#!/bin/bash

# causes the shell to exit if any invoked command exits with a non-zero
status
set -e

echo "Applying Ingress resource..."

cat <<EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
  labels:
    nimble.opti.adapter/enabled: "true"
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
EOF

echo "Ingress resource applied successfully."

echo "Applying NimbleOpti resource..."
```

```
cat <<EOF | kubectl apply -f -
apiVersion: adapter.uri-tech.github.io/v1
kind: NimbleOpti
metadata:
  name: example-nimbleopti
spec:
  # Add any fields defined in the NimbleOpti spec here
  # For instance:
  # someField: someValue
EOF

echo "NimbleOpti resource applied successfully."
```