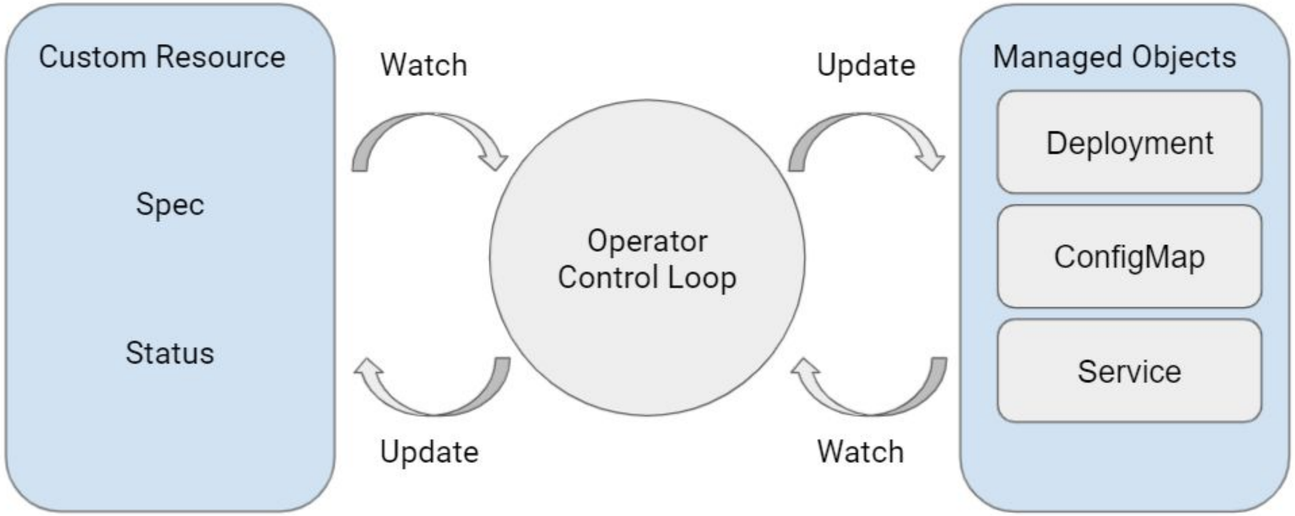
Kubernetes Operator Tutorial

Ruohang Yin (e0661412@u.nus.edu)

Table of Contents

Operator Recap

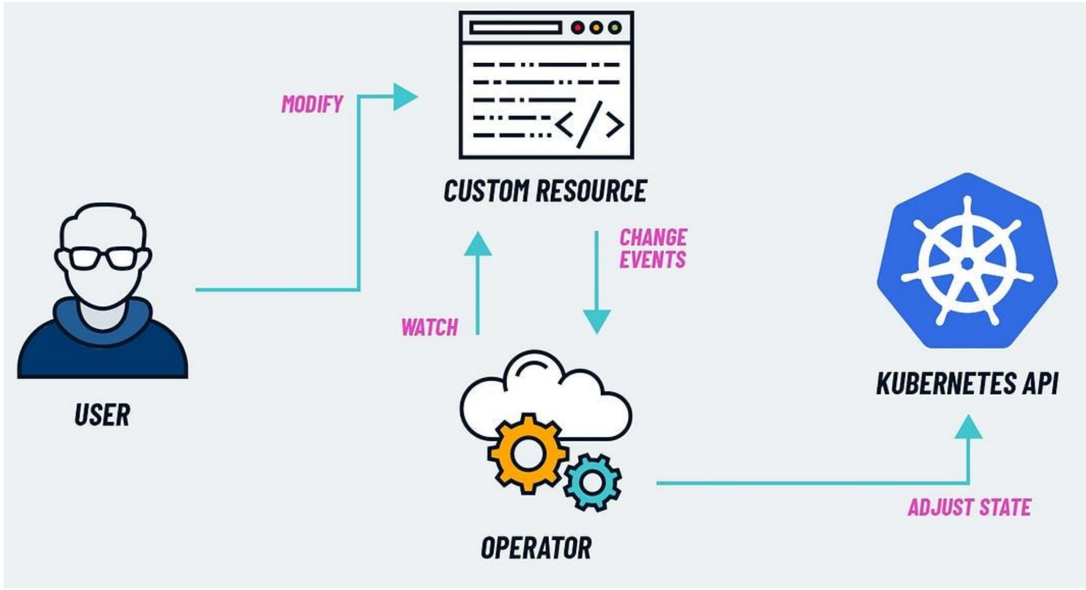
- Operator = Controller + CRD
- Manages application lifecycle on the cloud automatically using custom resources



Tools to Build an Operator

Operator Inner workings - Reconcile Loop

- Steps of a reconcile loop:
  - Check for changes in custom resource instance
  - Check kubernetes cluster state
  - Adjust kubernetes state based on custom resource spec
- Building an operator = implementing reconcile loop

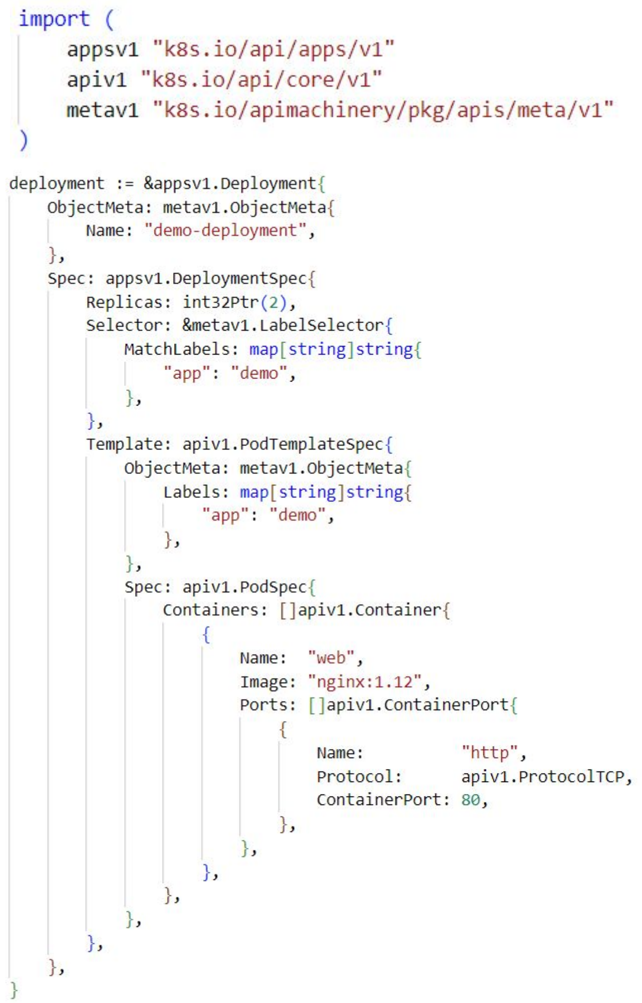


Interacting with Kubernetes Programmatically

- Operators need to manage resources on Kubernetes in a program
- Kubernetes is written in Golang
  - Extensive libraries and APIs are provided by Kubernetes Group to interact with Kubernetes
- Kubectl commands can be done in Golang
  - Acts as a client using Kubernetes

Kubernetes Resource in Go

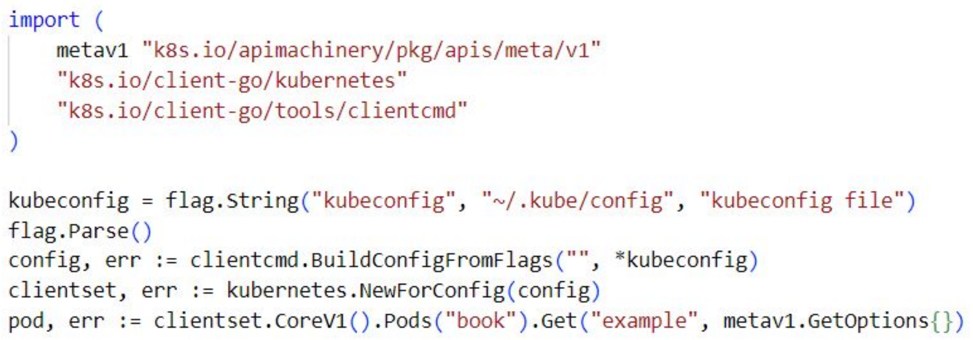
- Go: Define objects with structs
- Kubernetes Structs are imported through packages
- k8s.io/api/core/v1
    - Group: Core. Version: v1
    - Defines a Pod
- k8s.io/api/apps/v1
    - Group: Apps. Version: v1
    - More advanced resources such as Deployments
- k8s.io/apimachinery/pkg/apis/meta/v1
    - Common API tools that are not tied to any version



```go
import (
    appsv1 "k8s.io/api/apps/v1"
    apiv1 "k8s.io/api/core/v1"
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

deployment := &appsv1.Deployment{
    ObjectMeta: metav1.ObjectMeta{
        Name: "demo-deployment",
    },
    Spec: appsv1.DeploymentSpec{
        Replicas: int32Ptr(2),
        Selector: &metav1.LabelSelector{
            MatchLabels: map[string]string{
                "app": "demo",
            },
        },
        Template: apiv1.PodTemplateSpec{
            ObjectMeta: metav1.ObjectMeta{
                Labels: map[string]string{
                    "app": "demo",
                },
            },
            Spec: apiv1.PodSpec{
                Containers: []apiv1.Container{
                    {
                        Name:  "web",
                        Image: "nginx:1.12",
                        Ports: []apiv1.ContainerPort{
                            {
                                Name:          "http",
                                Protocol:      apiv1.ProtocolTCP,
                                ContainerPort: 80,
                            },
                        },
                    },
                },
            },
        },
    },
}
```

client-go

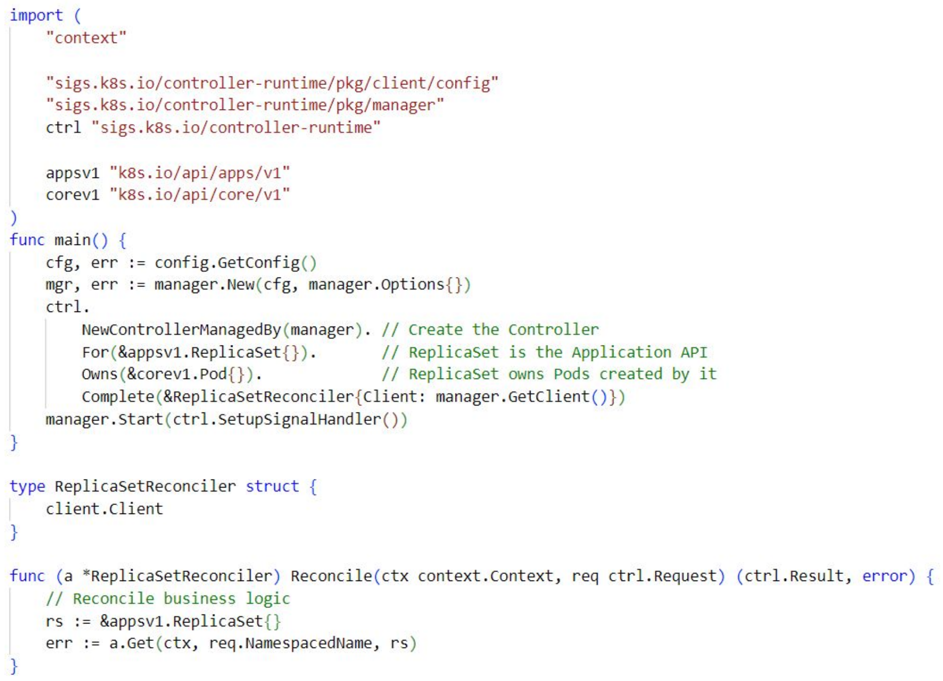
- To perform `kubectl` commands you need a Kubernetes client in go
- https://pkg.go.dev/k8s.io/client-go
- Kubernetes client is created from kubeconfig.
- Types of Clients:
    - Static client: Useful for existing statically defined resources. (Pods, Deployment)
    - Dynamic clients: Useful for custom resources that is defined on runtime



```go
import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
    "k8s.io/client-go/kubernetes"
    "k8s.io/client-go/tools/clientcmd"
)

kubeconfig = flag.String("kubeconfig", "~/.kube/config", "kubeconfig file")
flag.Parse()
config, err := clientcmd.BuildConfigFromFlags("", *kubeconfig)
clientset, err := kubernetes.NewForConfig(config)
pod, err := clientset.CoreV1().Pods("book").Get("example", metav1.GetOptions{})
```

controller-runtime

- Kubernetes Special Interest Group (Kubernetes SIGs) package for operator development
- Abstracts client-go logic to a **Manager** that manages **Controllers**
- https://pkg.go.dev/sigs.k8s.io/controller-runtime/
- Might seem complicated. But: There is a way to simplify building operators



```
import (
    "context"

    "sigs.k8s.io/controller-runtime/pkg/client/config"
    "sigs.k8s.io/controller-runtime/pkg/manager"
    ctrl "sigs.k8s.io/controller-runtime"

    appsv1 "k8s.io/api/apps/v1"
    corev1 "k8s.io/api/core/v1"
)
func main() {
    cfg, err := config.GetConfig()
    mgr, err := manager.New(cfg, manager.Options{})
    ctrl.
        NewControllerManagedBy(manager). // Create the Controller
        For(&appsv1.ReplicaSet{}).       // ReplicaSet is the Application API
        Owns(&corev1.Pod{}).             // ReplicaSet owns Pods created by it
        Complete(&ReplicaSetReconciler{Client: manager.GetClient()})
    manager.Start(ctrl.SetupSignalHandler())
}

type ReplicaSetReconciler struct {
    client.Client
}

func (a *ReplicaSetReconciler) Reconcile(ctx context.Context, req ctrl.Request) (ctrl.Result, error) {
    // Reconcile business logic
    rs := &appsv1.ReplicaSet{}
    err := a.Get(ctx, req.NamespacedName, rs)
}
```

Operator Framework

- A set of open-source tools to manage Kubernetes operators.
- Simplifies the process of automating application deployment, scaling, and life-cycle management.
- Consists of main components:
  - Operator SDK: A toolkit to build, test, and package Operators.
  - Operator Lifecycle Manager (OLM): Manages the installation, updates, and overall lifecycle of Operators.
  - Operator Hub: Collection of open-source Kubernetes Operators for different applications

OperatorSDK

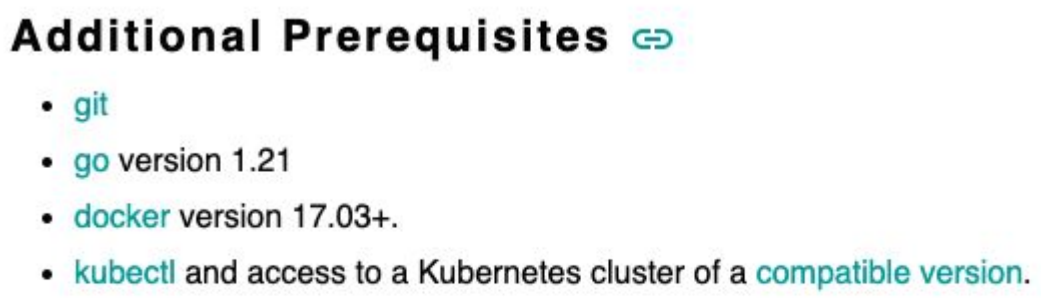
- A framework that simplifies the development of Kubernetes Operators.
- Versions: Golang, Java
- Key Features:
  - Scaffolding: Provides code scaffolding to jumpstart Operator development.
  - Testing Framework: Built-in testing tools to validate Operator behavior.
  - Packaging: Tools to package and distribute Operators.

OperatorSDK - Bootstrap Project

Install OperatorSDK

- https://sdk.operatorframework.io/docs/installation/
- Homebrew: `brew install operator-sdk`
- `operator-sdk version`

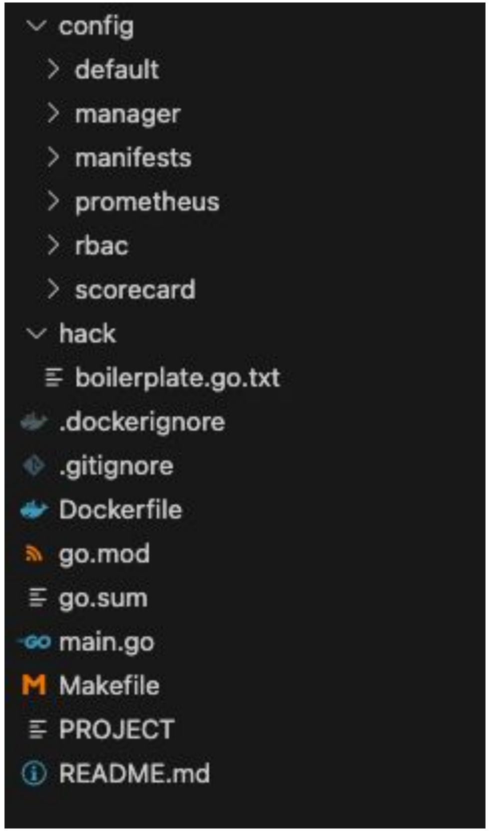
Creating An Operator



Additional Prerequisites

- git
- go version 1.21
- docker version 17.03+.
- kubectl and access to a Kubernetes cluster of a compatible version.

```
mkdir webpage-operator

cd webpage-operator

operator-sdk init --repo github.com/example/webpage-operator
```

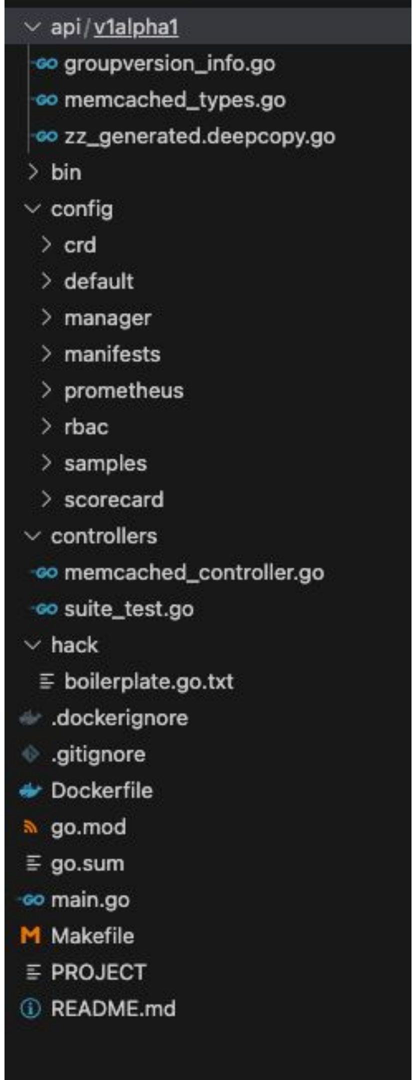


```
config
  default
  manager
  manifests
  prometheus
  rbac
  scorecard
hack
  boilerplate.go.txt
.dockerignore
.gitignore
Dockerfile
go.mod
go.sum
main.go
Makefile
PROJECT
README.md
```

Creating an API

```
operator-sdk create api --group app --version
v1alpha1 --kind WebPage --resource --controller
```

- api/version: CRD definition
  - File to modify: webpage_types.go
- controllers: Control logic
  - File to modify: webpage_controller.go



```
api/v1alpha1
  groupversion_info.go
  memcached_types.go
  zz_generated.deepcopy.go
bin
config
  crd
  default
  manager
  manifests
  prometheus
  rbac
  samples
  scorecard
controllers
  memcached_controller.go
  suite_test.go
hack
  boilerplate.go.txt
.dockerignore
.gitignore
Dockerfile
go.mod
go.sum
main.go
Makefile
PROJECT
README.md
```

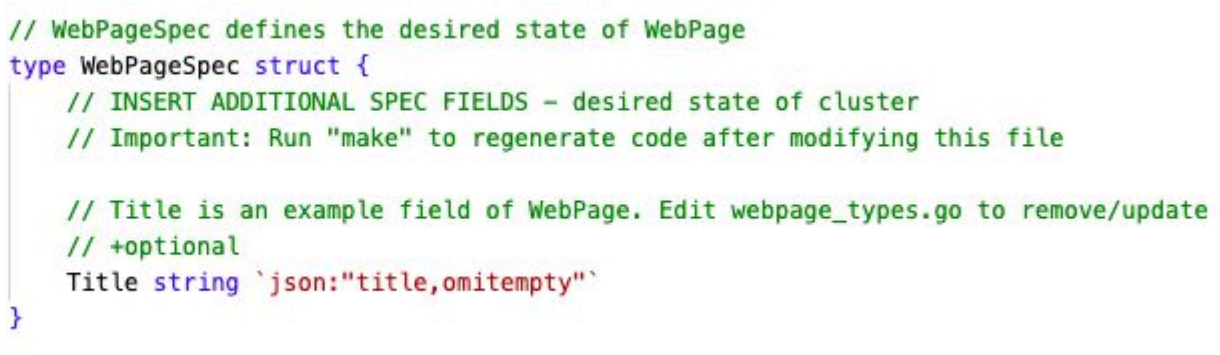
WebPage Types

- 1 Spec: Title (string)

`make generate`

`make manifests`

- CRD created automatically in config/crd/bases

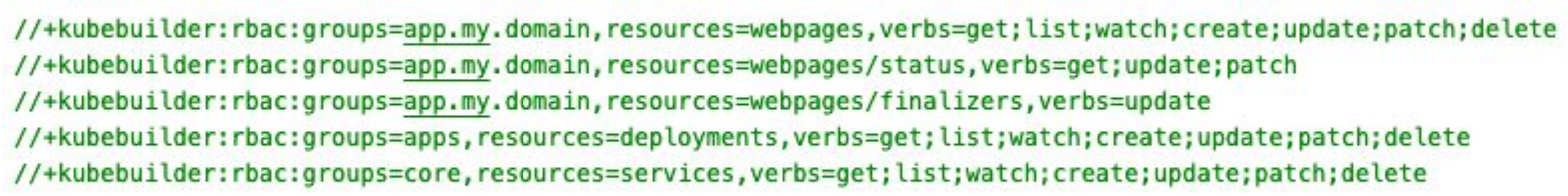


```go
// WebPageSpec defines the desired state of WebPage
type WebPageSpec struct {
	// INSERT ADDITIONAL SPEC FIELDS - desired state of cluster
	// Important: Run "make" to regenerate code after modifying this file

	// Title is an example field of WebPage. Edit webpage_types.go to remove/update
	// +optional
	Title string `json:"title,omitempty"`
}
```

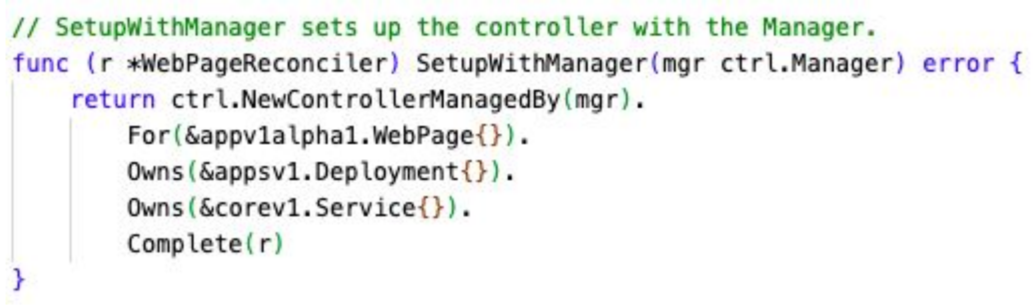
Custom Controller

controllers/webpage_controllers.go (Download from canvas)

- 2 Main functions for the reconciler
  - SetupWithManager: Tells the manager which resources to watch
  - Reconcile: Implement reconcile loop



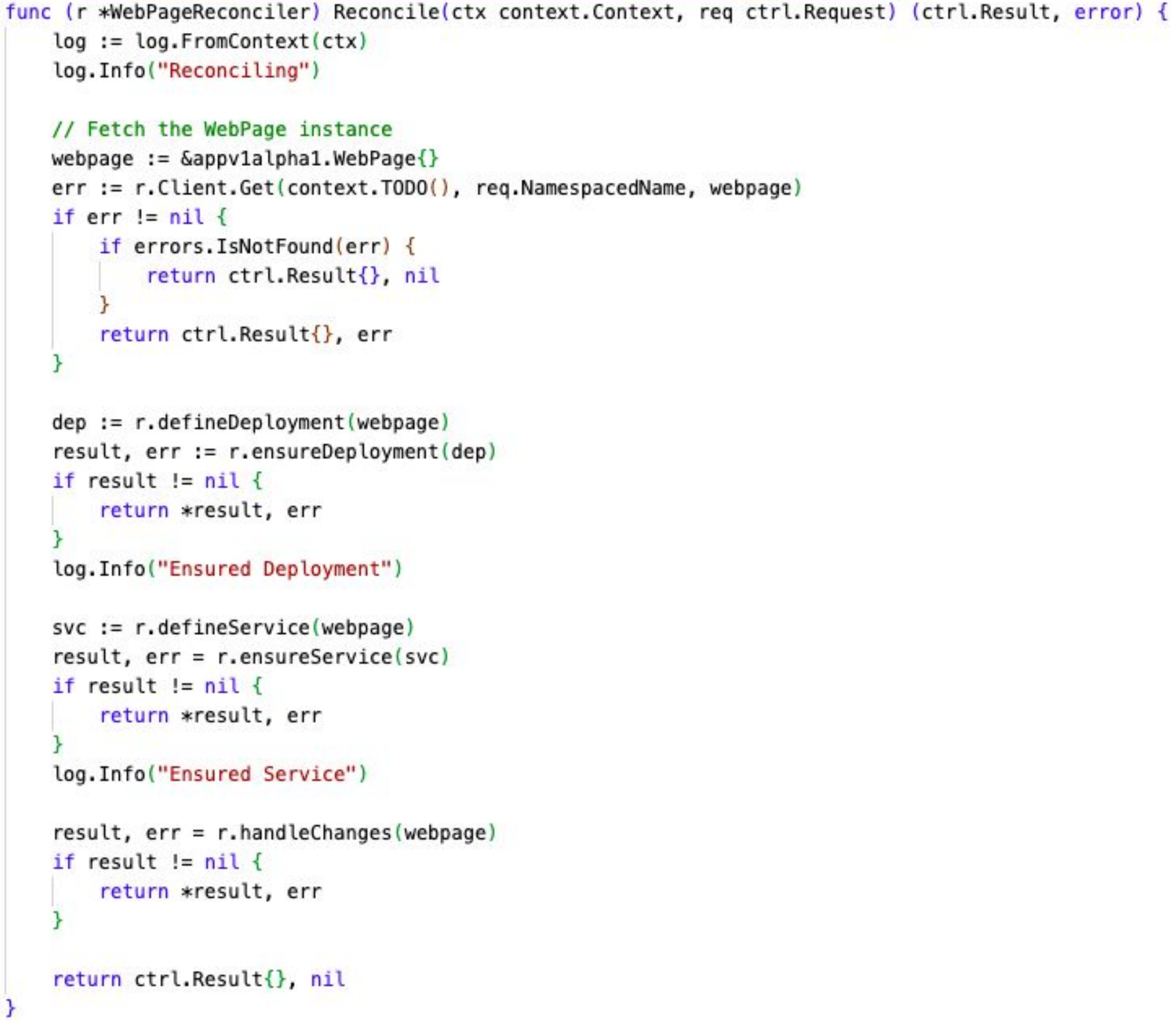
```
//+kubebuilder:rbac:groups=app.my.domain,resources=webpages,verbs=get;list;watch;create;update;patch;delete
//+kubebuilder:rbac:groups=app.my.domain,resources=webpages/status,verbs=get;update;patch
//+kubebuilder:rbac:groups=app.my.domain,resources=webpages/finalizers,verbs=update
//+kubebuilder:rbac:groups=apps,resources=deployments,verbs=get;list;watch;create;update;patch;delete
//+kubebuilder:rbac:groups=core,resources=services,verbs=get;list;watch;create;update;patch;delete
```



```
// SetupWithManager sets up the controller with the Manager.
func (r *WebPageReconciler) SetupWithManager(mgr ctrl.Manager) error {
    return ctrl.NewControllerManagedBy(mgr).
        For(&appv1alpha1.WebPage{}).
        Owns(&appsv1.Deployment{}).
        Owns(&corev1.Service{}).
        Complete(r)
}
```

Reconcile Loop Overview

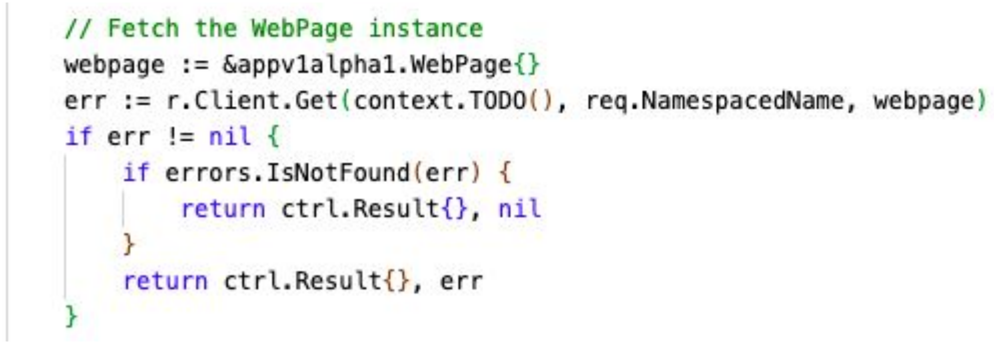
1. Fetch custom resource
2. Ensure deployment is deployed on cluster
3. Ensure service is deployed on cluster
4. Handle changes to custom resource by updating frontend deployment



```go
func (r *WebPageReconciler) Reconcile(ctx context.Context, req ctrl.Request) (ctrl.Result, error) {
    log := log.FromContext(ctx)
    log.Info("Reconciling")

    // Fetch the WebPage instance
    webpage := &appv1alpha1.WebPage{}
    err := r.Client.Get(context.TODO(), req.NamespacedName, webpage)
    if err != nil {
        if errors.IsNotFound(err) {
            return ctrl.Result{}, nil
        }
        return ctrl.Result{}, err
    }

    dep := r.defineDeployment(webpage)
    result, err := r.ensureDeployment(dep)
    if result != nil {
        return *result, err
    }
    log.Info("Ensured Deployment")

    svc := r.defineService(webpage)
    result, err = r.ensureService(svc)
    if result != nil {
        return *result, err
    }
    log.Info("Ensured Service")

    result, err = r.handleChanges(webpage)
    if result != nil {
        return *result, err
    }

    return ctrl.Result{}, nil
}
```

Reconcile Loop

Step 1: Fetch custom resource object

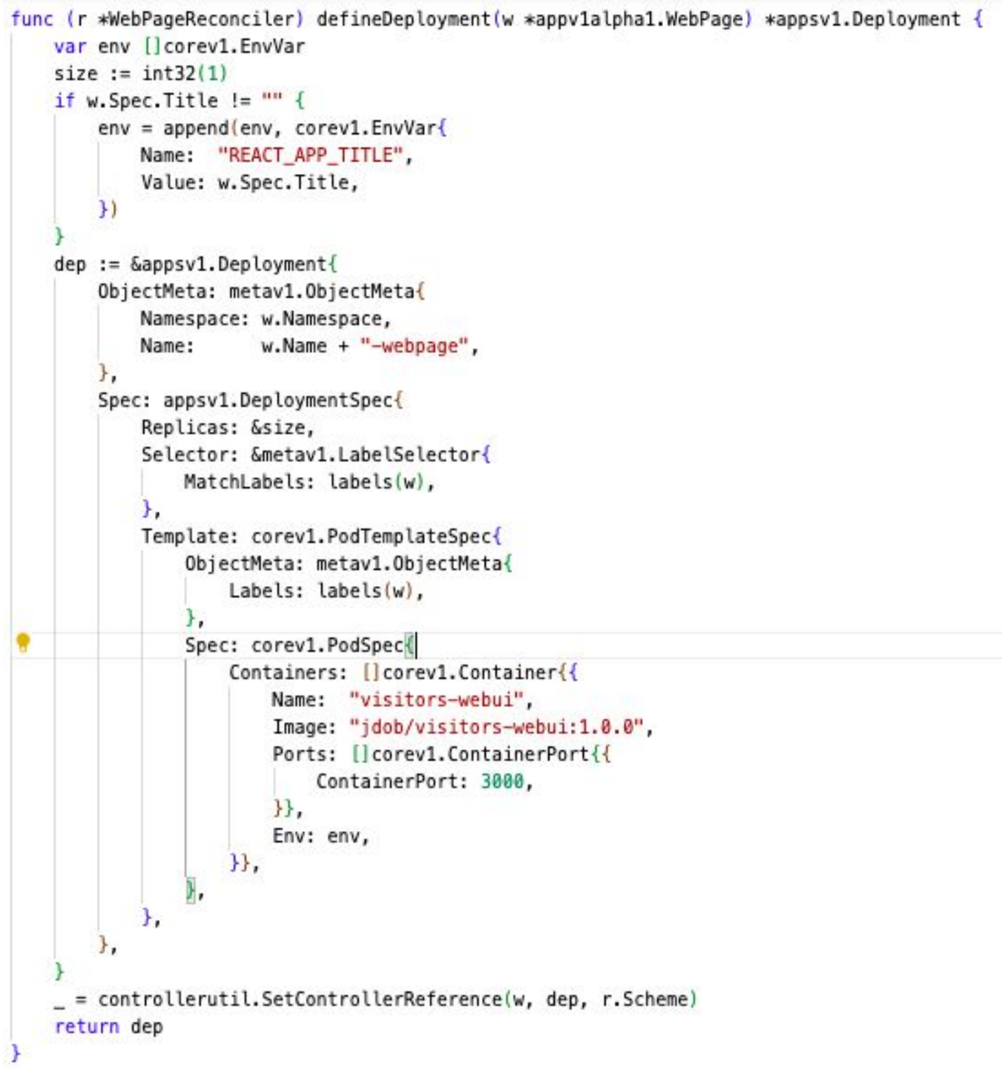


```go
// Fetch the WebPage instance
webpage := &appv1alpha1.WebPage{}
err := r.Client.Get(context.TODO(), req.NamespacedName, webpage)
if err != nil {
    if errors.IsNotFound(err) {
        return ctrl.Result{}, nil
    }
    return ctrl.Result{}, err
}
```

Reconcile Loop

Step 2: Define deployment

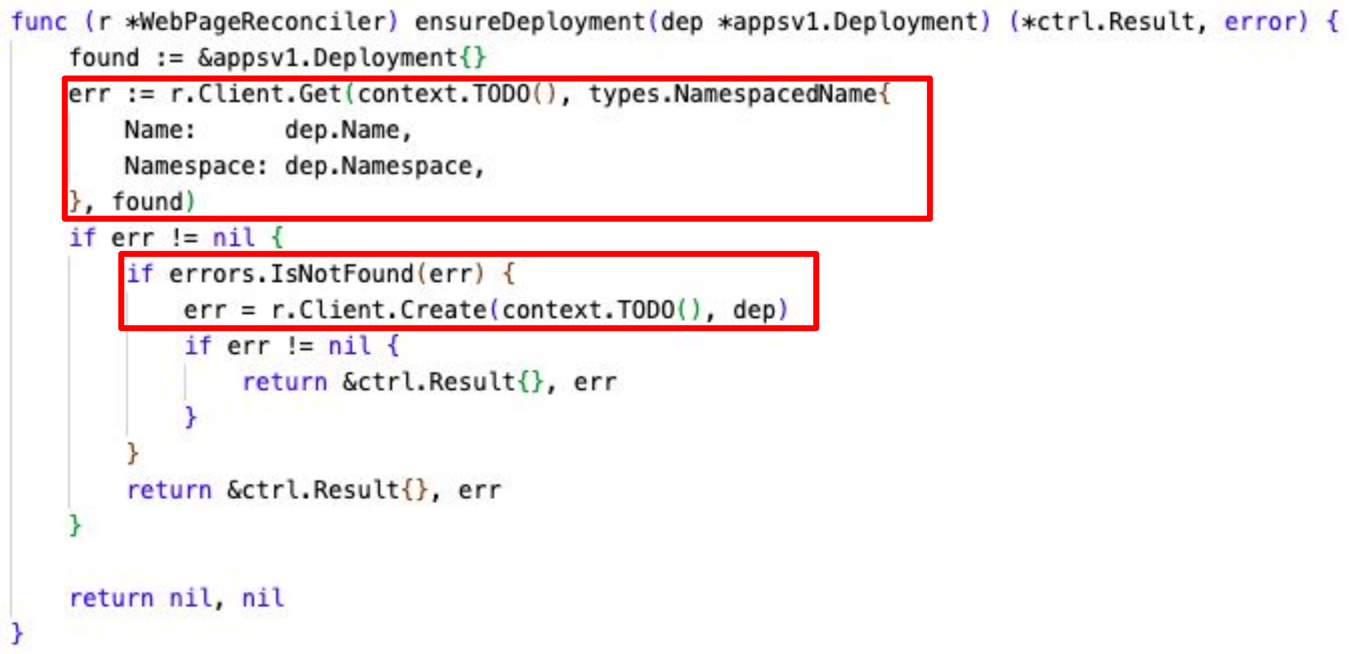
- Spec.Title is used as a ENV variable in the container on the pods
- Deployment's reference is set to the custom resource



```go
func (r *WebPageReconciler) defineDeployment(w *appv1alpha1.WebPage) *appsv1.Deployment {
    var env []corev1.EnvVar
    size := int32(1)
    if w.Spec.Title != "" {
        env = append(env, corev1.EnvVar{
            Name:  "REACT_APP_TITLE",
            Value: w.Spec.Title,
        })
    }
    dep := &appsv1.Deployment{
        ObjectMeta: metav1.ObjectMeta{
            Namespace: w.Namespace,
            Name:      w.Name + "-webpage",
        },
        Spec: appsv1.DeploymentSpec{
            Replicas: &size,
            Selector: &metav1.LabelSelector{
                MatchLabels: labels(w),
            },
            Template: corev1.PodTemplateSpec{
                ObjectMeta: metav1.ObjectMeta{
                    Labels: labels(w),
                },
                Spec: corev1.PodSpec{
                    Containers: []corev1.Container{{
                        Name:  "visitors-webui",
                        Image: "jdob/visitors-webui:1.0.0",
                        Ports: []corev1.ContainerPort{{
                            ContainerPort: 3000,
                        }},
                        Env: env,
                    }},
                },
            },
        },
    }
    _ = controllerutil.SetControllerReference(w, dep, r.Scheme)
    return dep
}
```

Reconcile Loop

Step 3: Ensure deployment is on the cluster



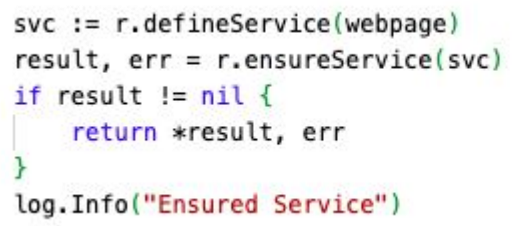
```go
func (r *WebPageReconciler) ensureDeployment(dep *appsv1.Deployment) (*ctrl.Result, error) {
	found := &appsv1.Deployment{}
	err := r.Client.Get(context.TODO(), types.NamespacedName{
		Name:      dep.Name,
		Namespace: dep.Namespace,
	}, found)
	if err != nil {
		if errors.IsNotFound(err) {
			err = r.Client.Create(context.TODO(), dep)
			if err != nil {
				return &ctrl.Result{}, err
			}
		}
		return &ctrl.Result{}, err
	}

	return nil, nil
}
```

Reconcile Loop

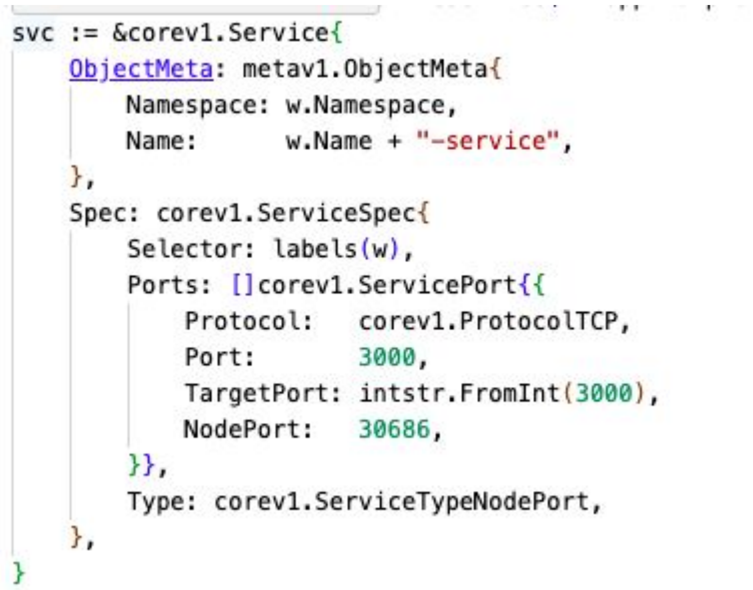
Step 4: define service

Step 5: ensure service

Similar to Step 2 and 3



```go
svc := r.defineService(webpage)
result, err = r.ensureService(svc)
if result != nil {
    return *result, err
}
log.Info("Ensured Service")
```

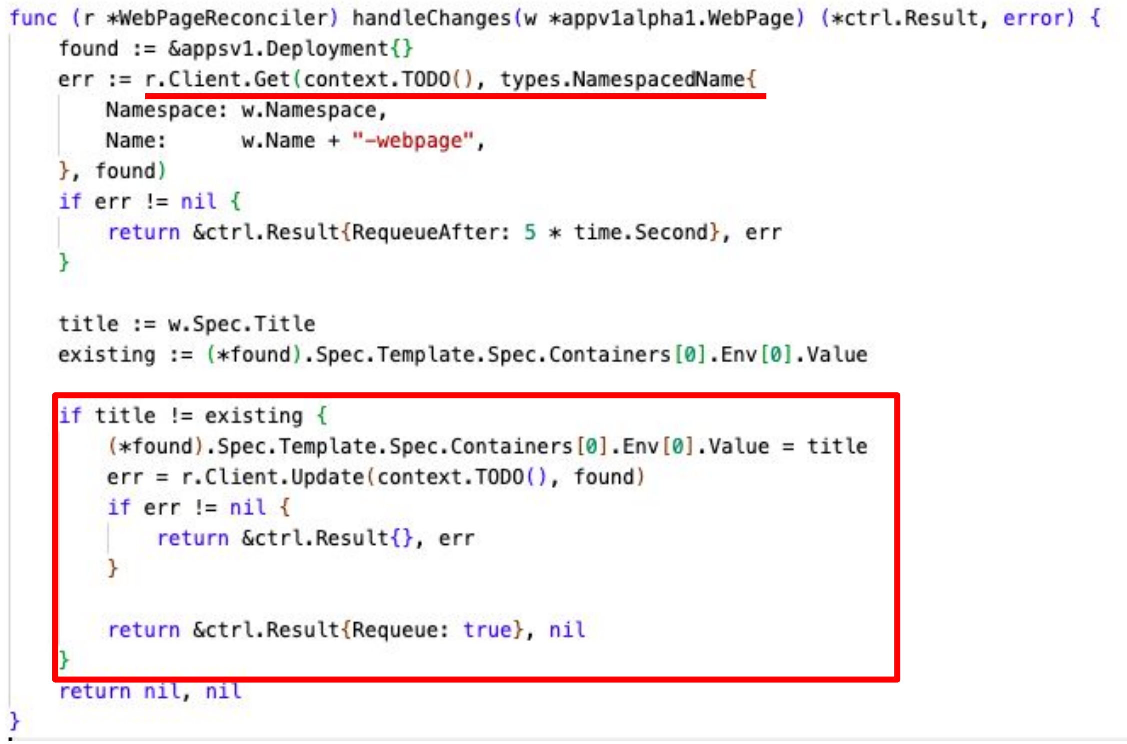


```go
svc := &corev1.Service{
    ObjectMeta: metav1.ObjectMeta{
        Namespace: w.Namespace,
        Name:      w.Name + "-service",
    },
    Spec: corev1.ServiceSpec{
        Selector: labels(w),
        Ports: []corev1.ServicePort{{
            Protocol:   corev1.ProtocolTCP,
            Port:       3000,
            TargetPort: intstr.FromInt(3000),
            NodePort:   30686,
        }},
        Type: corev1.ServiceTypeNodePort,
    },
}
```

Reconcile Loop

Step 6: Handle changes to custom resource

1. Re-fetch deployment
2. If not found, requeue reconcile loop after 5 seconds
3. Check if the title field is the same as the environment variable of the deployment
    a. If it isn't the same, change the field to the desired title and update the deployment using the K8s client and immediately requeue the reconcile loop



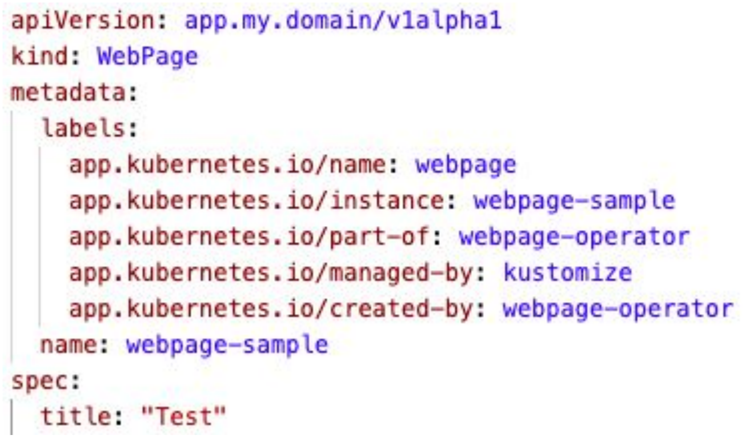
```go
func (r *WebPageReconciler) handleChanges(w *appv1alpha1.WebPage) (*ctrl.Result, error) {
    found := &appsv1.Deployment{}
    err := r.Client.Get(context.TODO(), types.NamespacedName{
        Namespace: w.Namespace,
        Name:      w.Name + "-webpage",
    }, found)
    if err != nil {
        return &ctrl.Result{RequeueAfter: 5 * time.Second}, err
    }

    title := w.Spec.Title
    existing := (*found).Spec.Template.Spec.Containers[0].Env[0].Value

    if title != existing {
        (*found).Spec.Template.Spec.Containers[0].Env[0].Value = title
        err = r.Client.Update(context.TODO(), found)
        if err != nil {
            return &ctrl.Result{}, err
        }

        return &ctrl.Result{Requeue: true}, nil
    }
    return nil, nil
}
```

Reconcile Loop return values

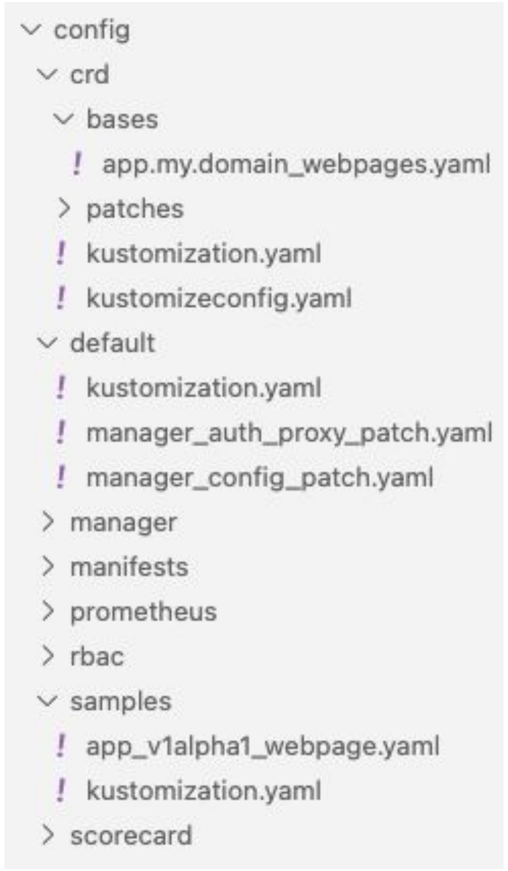
- By default, reconcile loop triggers when there is a change in the resources it is watching
- Returning different ctrl.Result objects will have different behaviors
  - `return ctrl.Result{}, err`
    - No requeue with error
  - `return ctrl.Result{}, nil`
    - No requeue, no error
  - `return ctrl.Result{Requeue: true}, nil`
    - Requeue with no error
  - `return ctrl.Result{RequeueAfter: 5 * time.Minute}, nil`
    - Requeue after X time duration

Config Folder

- Uses Kustomization tool to generate yaml
- Notable files:
  - config/default/kustomization.yaml
    - Changing namespace can change default namespace
  - config/samples/app_v1alpha1_webpage.yaml
    - Auto-generated sample file to apply



```yaml
apiVersion: app.my.domain/v1alpha1
kind: WebPage
metadata:
  labels:
    app.kubernetes.io/name: webpage
    app.kubernetes.io/instance: webpage-sample
    app.kubernetes.io/part-of: webpage-operator
    app.kubernetes.io/managed-by: kustomize
    app.kubernetes.io/created-by: webpage-operator
  name: webpage-sample
spec:
  title: "Test"
```



```
v config
  v crd
    v bases
      ! app.my.domain_webpages.yaml
    > patches
    ! kustomization.yaml
    ! kustomizeconfig.yaml
  v default
    ! kustomization.yaml
    ! manager_auth_proxy_patch.yaml
    ! manager_config_patch.yaml
  > manager
  > manifests
  > prometheus
  > rbac
  v samples
    ! app_v1alpha1_webpage.yaml
    ! kustomization.yaml
  > scorecard
```

Testing your operator

```
make install run
```

- Compiles and runs it as a local program

```
kubectl apply -f config/samples/app_v1alpha1_webpage.yaml
```

- Access webpage through the NodePort: 30686

Useful make commands

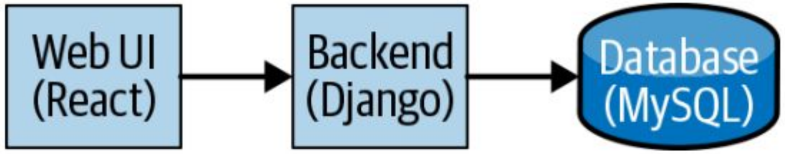
- `make generate`
  - Update zz_generated.deepcopy.go
- `make manifests`
  - Generates config/crd/bases (your CRD)
- `make docker-build`
  - Builds controller into a docker image
- `make docker-push`
  - Pushes docker image to dockerhub
- `make deploy`
  - Builds and runs your controller as a deployment on kubernetes
- `make undeploy`
  - Uninstalls Operator from Kubernete
- `make install run`
  - Locally run your controller as a program

Note: You need a DockerHub account to push docker images so that AWS EKS can access the image when using make deploy

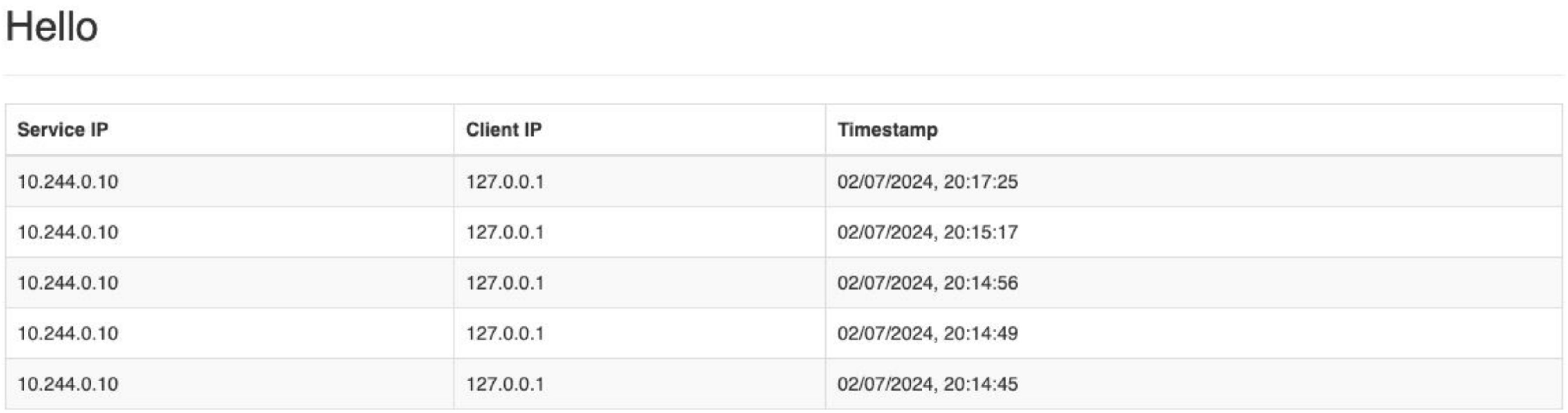
Example Operator with OperatorSDK

Visitors Website

Website to record visitors



$$\begin{tikzcd}\text{Web UI (React)} \arrow[r] & \text{Backend (Django)} \arrow[r] & \text{Database (MySQL)}\end{tikzcd}$$

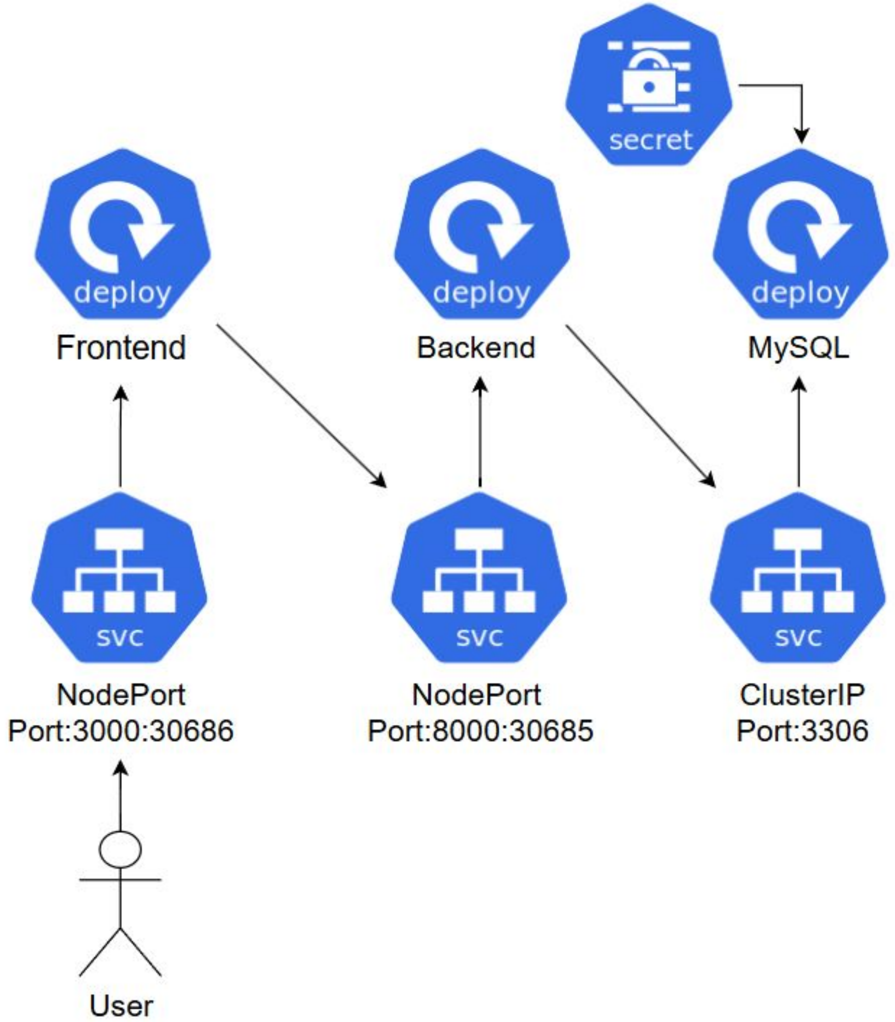


Hello

Service IP	Client IP	Timestamp
10.244.0.10	127.0.0.1	02/07/2024, 20:17:25
10.244.0.10	127.0.0.1	02/07/2024, 20:15:17
10.244.0.10	127.0.0.1	02/07/2024, 20:14:56
10.244.0.10	127.0.0.1	02/07/2024, 20:14:49
10.244.0.10	127.0.0.1	02/07/2024, 20:14:45

VisitorsApp

- Deployment
- Service (NodePort)
- Service (ClusterIP)
- Secret



Deploying the application on Kubernetes

- Need to create and deploy
  - 3 Deployment YAML files
  - 3 Service YAML files
  - 1 Secret YAML file

YAML Files

- To deploy the app on Kubernetes
- Need to create all the YAML files and do `kubectl -apply`
- Consistent configuration is needed for the app to work
  - Parameters for different YAML files must match
- What if there is a mistake?
  - Very hard to spot mismatch in values
- How to simplify deployment and upgrade process?
  - Use operators

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: visitors-backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: visitors
      tier: backend
  template:
    metadata:
      labels:
        app: visitors
        tier: backend
    spec:
      containers:
        - name: visitors-backend
          image: "jdob/visitors-service:1.0.0"
          imagePullPolicy: Always
          ports:
            - name: visitors
              containerPort: 8000
          env:
            - name: MYSQL_DATABASE
              value: visitors_db
            - name: MYSQL_SERVICE_HOST
              value: mysql-service
            - name: MYSQL_USERNAME
              valueFrom:
                secretKeyRef:
                  name: mysql-auth
                  key: username
            - name: MYSQL_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-auth
                  key: password
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
  labels:
    app: visitors
    tier: mysql
spec:
  clusterIP: None
  ports:
    - port: 3306
  selector:
    app: visitors
    tier: mysql
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysql-auth
type: Opaque
stringData:
  username: visitors-user
  password: visitors-pass
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: visitors
      tier: mysql
  template:
    metadata:
      labels:
        app: visitors
        tier: mysql
    spec:
      containers:
        - name: visitors-mysql
          image: "mysql:5.7"
          imagePullPolicy: Always
          ports:
            - name: mysql
              containerPort: 3306
              protocol: TCP
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: password
            - name: MYSQL_DATABASE
              value: visitors_db
            - name: MYSQL_USER
              valueFrom:
                secretKeyRef:
                  name: mysql-auth
                  key: username
            - name: MYSQL_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-auth
                  key: password
```

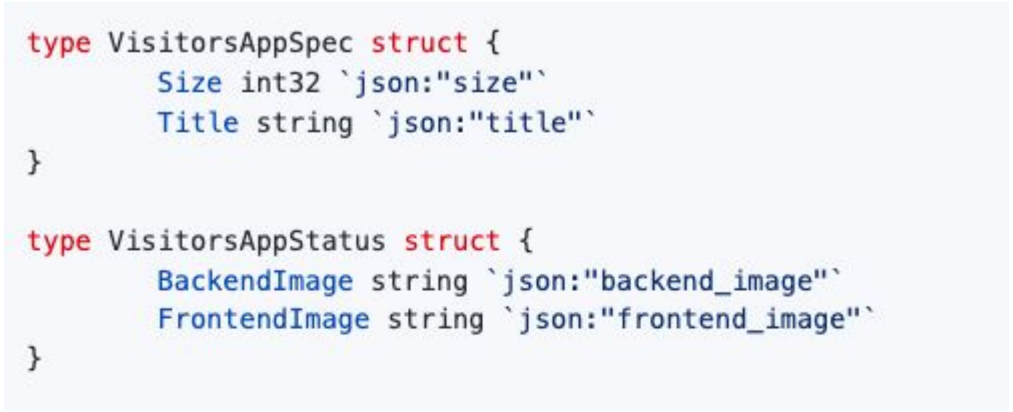
Installing the Operator

```
git clone https://github.com/yin72257/visitors-operator.git

make deploy

kubectl get deployments
```

- Wait for deployment to be ready (May take some time to pull image)
- Controller is deployed as a deployment on Kubernetes

VisitorsApp CRD

api/v1alpha1/visitorsapp_types.go

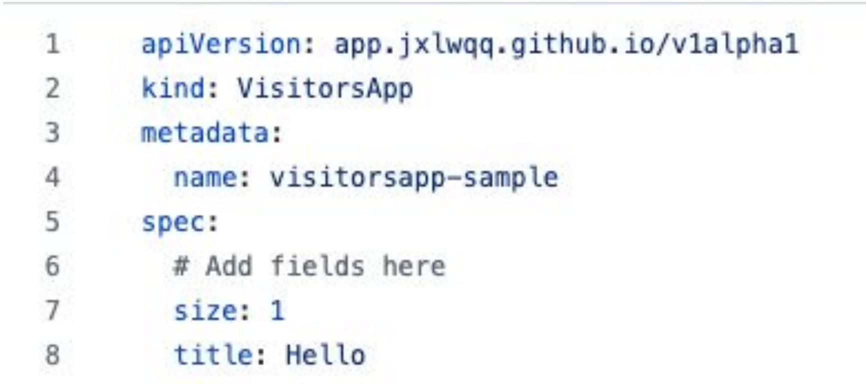


```go
type VisitorsAppSpec struct {
        Size int32 `json:"size"`
        Title string `json:"title"`
}

type VisitorsAppStatus struct {
        BackendImage string `json:"backend_image"`
        FrontendImage string `json:"frontend_image"`
}
```

Using the Operator

Apply a Custom Resource:

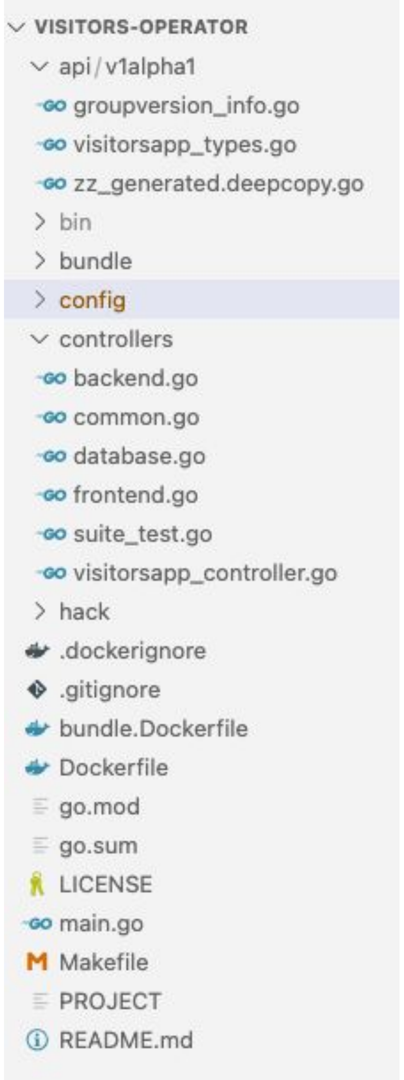
- Deploying the app
  - `kubectl apply -f config/samples/visitor_sample.yaml`
- Wait for pods
  - `kubectl get pods`
- Accessing the Frontend:
  - ${EC2 Public DNS}:30686



```yaml
apiVersion: app.jxlwqq.github.io/v1alpha1
kind: VisitorsApp
metadata:
  name: visitorsapp-sample
spec:
  # Add fields here
  size: 1
  title: Hello
```

Code structure

- common.go: Shared common utility methods
- backend.go: Backend management
- database.go: SQL database management
- frontend.go: Frontend management
- visitorsapp_controller.go: Central reconcile loop



```
VISITORS-OPERATOR
  api / v1alpha1
    groupversion_info.go
    visitorsapp_types.go
    zz_generated.deepcopy.go
  bin
  bundle
  config
  controllers
    backend.go
    common.go
    database.go
    frontend.go
    suite_test.go
    visitorsapp_controller.go
  hack
  .dockerignore
  .gitignore
  bundle.Dockerfile
  Dockerfile
  go.mod
  go.sum
  LICENSE
  main.go
  Makefile
  PROJECT
  README.md
```

Operator Use Cases

- Prometheus Operator
  - Widely used monitoring and metrics system
- Kafka Operator (Strimzi)
  - Distributed Message Queue
- MySQL Operator
- Many more…