ДОДАТОК А

Код Kubernetes оператора

Файл securitytesting_controller.go

```
package securitytesting

import (
	"context"

	appv1alpha1 "security_diploma_operator/pkg/apis/app/v1alpha1"

	corev1 "k8s.io/api/core/v1"
	"k8s.io/apimachinery/pkg/api/errors"
	metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
	"k8s.io/apimachinery/pkg/runtime"
	"k8s.io/apimachinery/pkg/types"
	"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/handler"
	logf "sigs.k8s.io/controller-runtime/pkg/log"
	"sigs.k8s.io/controller-runtime/pkg/manager"
	"sigs.k8s.io/controller-runtime/pkg/reconcile"
	"sigs.k8s.io/controller-runtime/pkg/source"
)

var log = logf.Log.WithName("controller_securitytesting")
func Add(mgr manager.Manager) error {
	return add(mgr, newReconciler(mgr))
}

// newReconciler returns a new reconcile.Reconciler
func newReconciler(mgr manager.Manager) reconcile.Reconciler {
	return &ReconcileSecurityTesting{client: mgr.GetClient(), scheme: mgr.GetScheme()}
}

// add adds a new Controller to mgr with r as the reconcile.Reconciler
func add(mgr manager.Manager, r reconcile.Reconciler) error {
	// Create a new controller
	c,       err       :=       controller.New("securitytesting-controller",       mgr,
controller.Options{Reconciler: r})
	if err != nil {
		return err
	}
	// Watch for changes to primary resource SecurityTesting
	err	=	c.Watch(&source.Kind{Type:	&appv1alpha1.SecurityTesting{}},
&handler.EnqueueRequestForObject{})
	if err != nil {
		return err
	}

	// Watch for changes to secondary resource Pods and requeue the owner SecurityTesting
	err = c.Watch(&source.Kind{Type: &corev1.Pod{}}, &handler.EnqueueRequestForOwner{
		IsController: true,
		OwnerType:    &appv1alpha1.SecurityTesting{},
	})
	if err != nil {
		return err
	}

	return nil
}

// blank assignment to verify that ReconcileSecurityTesting implements reconcile.Reconciler
var _ reconcile.Reconciler = &ReconcileSecurityTesting{}

func newReconciler(mgr manager.Manager) reconcile.Reconciler {
	scheme := mgr.GetScheme()
	addKnownTypes(scheme)
	client := mgr.GetClient()
	pt := helper.GetPlatformTypeEnv()
	ps, _ := platform.NewPlatformService(pt, scheme, &client)
	return &ReconcileSecurityTesting{
		client: client,
		scheme: scheme,
		ps:     ps,
	}
}

func addKnownTypes(scheme *runtime.Scheme) {
	scheme.AddKnownTypes(SchemeGroupVersion,
		&pipev1alpha1.Stage{},
		&pipev1alpha1.StageList{},
		&pipev1alpha1.CDPipeline{},
		&pipev1alpha1.CDPipelineList{},
	)
	metav1.AddToGroupVersion(scheme, SchemeGroupVersion)
}

// ReconcileSecurityTesting reconciles a SecurityTesting object
type ReconcileSecurityTesting struct {
	// This client, initialized using mgr.Client() above, is a split client
	// that reads objects from the cache and writes to the apiserver
	client client.Client
	scheme *runtime.Scheme
}

// Reconcile reads that state of the cluster for a SecurityTesting object and makes changes
based on the state read
// and what is in the SecurityTesting.Spec
// Note:
// The Controller will requeue the Request to be processed again if the returned error is
non-nil or
// Result.Requeue is true, otherwise upon completion it will remove the work from the
queue.
func (r *ReconcileSecurityTesting) Reconcile(request reconcile.Request) (reconcile.Result,
error) {
	reqLogger := log.WithValues("Request.Namespace", request.Namespace, "Request.Name",
request.Name)
	reqLogger.Info("Reconciling SecurityTesting")

	// Fetch the SecurityTesting instance
	instance := &appv1alpha1.SecurityTesting{}
	err := r.client.Get(context.TODO(), request.NamespacedName, instance)
	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 and don't requeue
			return reconcile.Result{}, nil
		}
		// Error reading the object - requeue the request.
		return reconcile.Result{}, err
	}

	// Define a new Pod object
	pod := newPodForCR(instance)

	// Set SecurityTesting instance as the owner and controller
	if err := controllerutil.SetControllerReference(instance, pod, r.scheme); err != nil
{
		return reconcile.Result{}, err
	}

	// Check if this Pod already exists
	found := &corev1.Pod{}
	err = r.client.Get(context.TODO(), types.NamespacedName{Name: pod.Name, Namespace: 
pod.Namespace}, found)
	if err != nil && errors.IsNotFound(err) {
		reqLogger.Info("Creating a new Pod", "Pod.Namespace", pod.Namespace, 
"Pod.Name", pod.Name)
		err = r.client.Create(context.TODO(), pod)
		if err != nil {
			return reconcile.Result{}, err
		}

		// Pod created successfully - don't requeue
		return reconcile.Result{}, nil
	} else if err != nil {
		return reconcile.Result{}, err
	}

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

// newPodForCR returns a busybox pod with the same name/namespace as the cr
func newPodForCR(cr *appv1alpha1.SecurityTesting) *corev1.Pod {
	labels := map[string]string{
		"app": cr.Name,
	}
	return &corev1.Pod{
		ObjectMeta: metav1.ObjectMeta{
			Name:      cr.Name + "-pod",
			Namespace: cr.Namespace,
			Labels:    labels,
		},
		Spec: corev1.PodSpec{
			Containers: []corev1.Container{
				{
					Name:    "busybox",
					Image:   "busybox",
					Command: []string{"sleep", "3600"},
				},
			},
		},
	}
}

func (r *ReconcileSecurityTesting) updateStatus(instance *v2v1alpha1.SecurityTesting, 
newStatus string) error {
	reqLogger := log.WithValues("Request.Namespace", instance.Namespace, "Request.Name", instance.Name).WithName("status_update")
	currentStatus := instance.Status.Status
	instance.Status.Status = newStatus
	instance.Status.LastTimeUpdated = time.Now()
	err := r.client.Status().Update(context.TODO(), instance)
	if err != nil {
		err := r.client.Update(context.TODO(), instance)
		if err != nil {
			return errorsf.Wrapf(err, "Couldn't update status from '%v' to '%v'", currentStatus, newStatus)
		}
	}
	reqLogger.Info(fmt.Sprintf("Status has been updated to '%v'", newStatus))
	return nil
}

func (r ReconcileSecurityTesting) updateAvailableStatus(instance *v2v1alpha1.SecurityTesting, value bool) error {
	reqLogger := log.WithValues("Request.Namespace", instance.Namespace, "Request.Name", instance.Name).WithName("status_update")
	if instance.Status.Available != value {
		instance.Status.Available = value
		instance.Status.LastTimeUpdated = time.Now()
		err := r.client.Status().Update(context.TODO(), instance)
		if err != nil {
			err := r.client.Update(context.TODO(), instance)
			if err != nil {
				return errorsf.Wrapf(err, "Couldn't update availability status to %v", value)
			}
		}
	}
	reqLogger.Info(fmt.Sprintf("Availability status has been updated to '%v'", value))
	return nil
}

func (r ReconcileSecurityTesting) updateInstanceStatus(instance *v2v1alpha1.SecurityTesting) error {
	reqLogger := log.WithValues("Request.Namespace", instance.Namespace, "Request.Name", instance.Name).WithName("status_update")
	instance.Status.LastTimeUpdated = time.Now()
	err := r.client.Status().Update(context.TODO(), instance)
	if err != nil {
		err := r.client.Update(context.TODO(), instance)
		if err != nil {
			return errorsf.Wrapf(err, "Couldn't update instance status")
		}
	}
	reqLogger.Info(fmt.Sprintf("Instance status has been updated"))
	return nil
```

Файл securitytesting_types.go

```
package v1alpha1

import (
	"time"

	coreV1Api "k8s.io/api/core/v1"
	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.

// SecurityTestingSpec defines the desired state of SecurityTesting
// +k8s:openapi-gen=true

type SecurityTestingSpec struct {
	// INSERT ADDITIONAL SPEC FIELDS - desired state of cluster
	// Important: Run "operator-sdk generate k8s" to regenerate code after modifying
this file
	// Add custom validation using kubebuilder tags: https://book-
v1.book.kubebuilder.io/beyond_basics/generating_crd.html
	Image            string                             `json:"image"`
	Version          string                             `json:"version"`
	InitImage        string                             `json:"initImage"`
	BasePath         string                             `json:"basePath,omitempty"`
	ImagePullSecrets                                     []coreV1Api.LocalObjectReference
`json:"imagePullSecrets,omitempty"`
	Volumes          []SecurityTestingVolumes           `json:"volumes,omitempty"`
	Plugin  []SecurityTestingSharedLibraries          `json:"plugin,omitempty"`
	ReportType     KeycloakSpec                       `json:"plugin"`
}

type SecurityTestingVolumes struct {
	Name         string `json:"name"`
	StorageClass string `json:"storageClass"`
	Capacity     string `json:"capacity"`
}

type SecurityTestingPlugins struct {
	Name       string `json:"name"`
	Id         string `json:"id"`
	Repository string `json:"repository"`
	Version    string `json:"version"`
}

// SecurityTestingStatus defines the observed state of SecurityTesting
// +k8s:openapi-gen=true
type SecurityTestingStatus struct {
	// INSERT ADDITIONAL STATUS FIELD - define observed state of cluster
	// Important: Run "operator-sdk generate k8s" to regenerate code after modifying
this file
	// Add custom validation using kubebuilder tags: https://book-
v1.book.kubebuilder.io/beyond_basics/generating_crd.html
	Available       bool           `json:"available,omitempty"`
	LastTimeUpdated time.Time      `json:"lastTimeUpdated,omitempty"`
	Status          string         `json:"status,omitempty"`
	AdminSecretName string         `json:"adminSecretName,omitempty"`
	JobProvisions   []JobProvision `json:"jobProvisions,omitempty"`
}

type JobProvision struct {
	Name  string `json:"name"`
	Scope string `json:"scope"`
}

type ReportTypeSpec struct {
	Enabled bool   `json:"enabled"`
	Report  string `json:"file,omitempty"`
}

// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

// SecurityTesting is the Schema for the SecurityTesting API
// +k8s:openapi-gen=true
// +kubebuilder:subresource:status
type SecurityTesting struct {
	metav1.TypeMeta   `json:",inline"`
	metav1.ObjectMeta `json:"metadata,omitempty"`

	Spec   SecurityTestingSpec   `json:"spec,omitempty"`
	Status SecurityTestingStatus `json:"status,omitempty"`
}

// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

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

func init() {
	SchemeBuilder.Register(&SecurityTesting{}, &SecurityTestingList{})
}
}
```

Файл app.security.kokhanevych.com_v1alpha1_securitytesting_cr.yaml

```
apiVersion: app.security.kokhanevych.com/v1alpha1
kind: SecurityTesting
metadata:
  name: example-securitytesting
spec:
  version: "3.1.1"
  image: "devsecopsat/spotbugs"
  initImage: "busybox"
  volumes:
    - name: "data"
      storageClass: "gp2"
      capacity: "1Gi"
  plugin: maven
  reportType: file
```

ДОДАТОК Б

Скрипт Jenkins пайплайна

```
Файл app.security.kokhanevych.com_v1alpha1_securitytesting_cr.yaml
podTemplate(yaml: '''
apiVersion: v1
kind: Pod
metadata:
 labels:
   job: security-build-deploy
spec:
 containers:
 - name: maven
   image: maven:3.6.0-jdk-11-slim
   command: ["cat"]
   tty: true
 - name: docker
   image: docker:18.09.2
   command: ["cat"]
   tty: true
   volumeMounts:
   - name: docker-sock
     mountPath: /var/run/docker.sock
 - name: helm-cli
   image: alpine/helm:3.1.3
   command: ["cat"]
   tty: true
 volumes:
 - name: docker-sock
   hostPath:
     path: /var/run/docker.sock
''') {
       node(POD_LABEL) {
          stage ('checkout') {
          checkout([$class: 'GitSCM', branches: [[name: "master"]],
              doGenerateSubmoduleConfigurations: false, extensions: [],
              submoduleCfg: [],
              userRemoteConfigs: [[credentialsId: "github",
              url: "git@github.com:YevheniiKokhanevych/cdp_k8s_test.git"]]])
          }
          stage ('compile') {
            container('maven') {
              sh 'mvn clean compile test-compile'
            }
          }
          stage ('test') {
            container('maven') {
              try {
                    sh """
                        mvn compile -B --settings ${vars.mavenSettings}
                        $SPOTBUGS_HOME/bin/spotbugs          -textui          -pluginList
$SPOTBUGS_HOME/plugin/findsecbugs-plugin-1.8.0.jar \
                        -xml:withMessages   -output   $DOJO_API_FILE   -low   -progress   -
effort:max -bugCategories SECURITY .
                    """
                    archiveArtifacts artifacts: 'spotbugs-result.html'
              } catch (Exception ex) {
                  println("[JENKINS][ERROR] SPOTBUGS stage has failed. Reason - ${ex}")
              }
            }
          }
          stage ('build artifact') {
            container('maven') {
              sh "mvn package -Dmaven.test.skip"
            }
            container('docker') {
              registryIp = sh(script: "kubectl get services -n default | grep docker-
registry | awk '{print \$3}'", returnStdout: true).trim()
              sh "docker build . -t ${registryIp}/cdp-k8s-app:${BUILD_NUMBER}"
            }
          }
          stage ('deploy') {
            container('helm-cli') {
              secretValue = credentials("${ENV}-secret")
              def replicaNumber = 1
              if (ENV == 'prod') {
                replicaNumber = 3
              }
              sh "/usr/bin/helm upgrade cdp-k8s-app helm/ --install --namespace ${ENV} --
set version=${BUILD_NUMBER} --set registry=${registryIp} --set replicas=${replicaNumber}
--set password=${secretValue}"
            }
          }
       }
}
```