This module covers the creation of Helm operators using Operator SDK. [Helm](https://helm.sh/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkCO0201ENSkillsNetwork23008840-2021-01-01) is a package manager for Kubernetes. [Helm charts](https://helm.sh/docs/topics/charts/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkCO0201ENSkillsNetwork23008840-2021-01-01) are used to define, install, and upgrade Kubernetes applications. Operator SDK supports the creation of operators based on existing Helm charts as a way to quickly scaffold operators. However, it is not recommended to create a Helm operator if you're starting from scratch because Helm operators limit the amount of control you have over the behaviour of your operator.

**Anatomy of a Helm operator**

Diagram

Description automatically generated

The Operator SDK open source community maintains a server image that acts as the controller for Helm operators, sometimes referred to as the 'helm-operator. This base image must be configured with a specific Helm chart. A custom resource definition (CRD) is then scaffolded based on that Helm chart and deployed to the Kubernetes cluster.

This diagram shows the flow of how Helm charts are used to deploy an operator.

1. A Kubernetes user uses kubectl to create an instance of a custom resource type.
2. The request is sent to the API server, which validates that the request is formatted correctly, and passes it through to the Kubernetes etcd store.
3. A record is created with the requested state for foo
4. The controller takes the input from the spec of the custom resource instance.
5. The controller deploys the Helm chart. This is done in the same way that you can use values.yaml to override the defaults when using the Helm CLI to deploy a Helm chart manually.

As the core Kubernetes API server serves requests to manage custom resource types, our resource can be interacted with using kubectl like any Kubernetes resource.

This lab covers using [Operator SDK](https://github.com/operator-framework/operator-sdk) to scaffold a Helm operator from an existing [Wordpress Helm chart](https://github.com/bitnami/charts/tree/master/bitnami/wordpress" \t "_blank). You will create a new project, deploy your operator to your Kubernetes cluster, and create an instance of the WordPress chart.

We will create a new project, construct some basic reconciliation logic to deploy the various resources such as Deployments and Services to create the backend for our new WordPress type, and then deploy our operator to our Kubernetes cluster.

## Prerequisites

To complete the steps in this lab, you need have:

* [Operator-sdk](https://sdk.operatorframework.io/docs/installation/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkCO0201ENSkillsNetwork23008840-2021-01-01) installed
* [Helm V3](https://helm.sh/docs/intro/install/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkCO0201ENSkillsNetwork23008840-2021-01-01)
* [Docker](https://docs.docker.com/get-docker/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkCO0201ENSkillsNetwork23008840-2021-01-01) installed
* access to a Docker image repository such as [Docker Hub](https://hub.docker.com/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkCO0201ENSkillsNetwork23008840-2021-01-01)

## Helm operator overview

The Helm operator is intended to quickly build out an operator from existing Helm charts. The operator creates a custom resource definition (CRD) based on an existing Helm chart. The resulting custom resource (CR) is watched by a controller that is based on a generic helm-operator controller image which is maintained by the Operator SDK open source community.

With Help operators, you configure the controller image with information about your specific Helm chart. When an instance of the (CR) is created, the controller deploys an instance of the Helm chart to the cluster, configured with the information specified in the CR object's spec.

## Step 0: Get the Helm chart

[Helm](https://helm.sh/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkCO0201ENSkillsNetwork23008840-2021-01-01) uses a packaging format called [charts](https://helm.sh/docs/topics/charts/) to deploy applications more easily to Kubernetes. You can search the [Artifact Hub](https://artifacthub.io/) for the different available charts and the repositories that host them. Helm can reference charts in a number of ways, including a Helm repository and a chart archive. See the [Helm Install](https://helm.sh/docs/helm/helm_install/) doc for more details.

This lab uses the [WordPress chart](https://github.com/bitnami/charts/tree/master/bitnami/wordpress) as an example chart for creating an operator. You can reference the chart from the local Helm repository list by adding it to the Helm repo list. Follow the steps below if you would like to set it up in this way.

1. **Find where the WordPress chart is hosted:**
2. **helm search hub wordpress**
3. **helm repo add bitnami https://charts.bitnami.com/bitnami**

**$ mkdir wordpress-operator**

**$ cd wordpress-operator**

**$ operator-sdk init --plugins helm --helm-chart bitnami/wordpress --domain example.com --group helm-chart --version v1alpha1 --kind Wordpress**

* --helm-chart references a chart from a local Helm chart repository. You can also access remote Helm chart repositories with

--helm-chart-repo and a chart version with --helm-chart-version.

* --domain is used for any API groups the operator creates, so the one in this lab will be \*.example.com. An *API group* is simply a group of related functionality, such as your operator. An API group you might be familiar with is rbac.authorization.k8s.io, which is where the functionality for creating RBAC resources such as ClusterRoles and ClusterBindings is usually set up on a Kubernetes cluster. operator-sdk allows you to specify a custom domain

to be appended to any API groups we might define to help avoid name collisions.

* --group serves a similar function to --domain, and will be appended to the domain for our resources. Most of the resources you create in this lab have the API version 'helm-chart.example.com'.

Let's take a look at what was generated:

* a helm-charts directory with copies of the chart(s) to build releases from
* a watches.yaml file that defines the mapping between your API and a Helm chart
* a Dockerfile for the controller image, based on Operator SDK's generic helm-operator image
* a PROJECT file with the domain and project layout configuration
* a Makefile to build, deploy, and undeploy the project
* a config directory that contains:
  + base manifests for deploying the CRD and controller of the operator
  + Kustomization YAML for customizing manifests
  + RBAC YAML to authorize the various components to interact with each other
  + a Patch file for enabling Prometheus metrics
  + some sample YAML for creating a simple instance of our new type

Step 2: Build and deploy your operator

Now that you generated the scaffolding for your operator, let's deploy it to a cluster! In general, there are three different ways to deploy an operator:

1. Run as a deployment inside the Kubernetes cluster. This lab walks you through steps to deploy this way.
2. As a program executing outside the Kubernetes cluster. This might be done for development purposes or for security concerns of the data contained in the cluster. The Makefile contains the target make install run to run the operator locally, but this method is not the focus of this lab.
3. Deployed and managed by Operator Lifecycle Manager (OLM). This is recommended for production use because OLM has additional features for the management and upgrade of a running operator.

For now, let's start by building and pushing the Docker image for your controller. This lab uses the base Dockerfile, which is based on an image maintained by the Operator SDK working group that provides the reconciliaton logic out-of-the-box. In this lab, I use Docker Hub as my image repository but any repo you have push/pull access to should work.

1. Build and push the image:

$ export USERNAME=<docker-registry-username>

$ make docker-build docker-push IMG=docker.io/$USERNAME/wordpress-operator:v1.0.0

[...]

Note that you can set the default name and tag of the image in your Makefile.

1. Deploy the operator:

$ make deploy IMG=docker.io/$USERNAME/wordpress-operator:v1.0.0

cd config/manager && projects/wordpress-operator/bin/kustomize edit set image controller=docker.io/user/wordpress-operator:v1.0.0

projects/wordpress-operator/bin/kustomize build config/default | kubectl apply -f -

namespace/wordpress-operator-system unchanged

customresourcedefinition.apiextensions.k8s.io/wordpresses.helm-chart.example.com created

role.rbac.authorization.k8s.io/wordpress-operator-leader-election-role unchanged

clusterrole.rbac.authorization.k8s.io/wordpress-operator-manager-role unchanged

clusterrole.rbac.authorization.k8s.io/wordpress-operator-metrics-reader unchanged

clusterrole.rbac.authorization.k8s.io/wordpress-operator-proxy-role unchanged

rolebinding.rbac.authorization.k8s.io/wordpress-operator-leader-election-rolebinding unchanged

clusterrolebinding.rbac.authorization.k8s.io/wordpress-operator-manager-rolebinding unchanged

clusterrolebinding.rbac.authorization.k8s.io/wordpress-operator-proxy-rolebinding unchanged

service/wordpress-operator-controller-manager-metrics-service unchanged

deployment.apps/wordpress-operator-controller-manager configured

It will take a bit for the cluster to pull the images and start everything. You can check if the the various components of the operator are up.

1. Check the custom resource definitions:

$ kubectl get crds | grep wordpress

wordpresses.helm-chart.example.com 2021-03-05T01:05:02Z

1. Check the controller deployment and pods:

$ kubectl get deployment -n wordpress-operator-system

NAME READY UP-TO-DATE AVAILABLE AGE

wordpress-operator-controller-manager 1/1 1 1 16m

$ kubectl get pods -n wordpress-operator-system

NAME READY STATUS RESTARTS AGE

wordpress-operator-controller-manager-9cf5dc88d-chjbn 2/2 Running 0 16m

1. Check the RBAC:

$ kubectl get clusterroles | grep wordpress

wordpress-operator-manager-role 2021-03-05T01:05:03Z

wordpress-operator-metrics-reader 2021-03-05T01:05:03Z

wordpress-operator-proxy-role 2021-03-05T01:05:04Z

$ kubectl get clusterrolebindings | grep wordpress

wordpress-operator-manager-rolebinding ClusterRole/wordpress-operator-manager-role 16m

wordpress-operator-proxy-rolebinding ClusterRole/wordpress-operator-proxy-role 16m

$ kubectl get roles -n wordpress-operator-system

NAME CREATED AT

wordpress-operator-leader-election-role 2021-03-05T01:05:03Z

$ kubectl get rolebindings -n wordpress-operator-system

NAME ROLE AGE

wordpress-operator-leader-election-rolebinding Role/wordpress-operator-leader-election-role 16m

Once everything is up and running, you can use your operator to deploy an instance of Wordpress.

Deploy a WordPress instance

1. Create a demo WordPress custom resource YAML manifest:

Note: This is equivalent to a [Helm values override file](https://helm.sh/docs/chart_template_guide/values_files/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkCO0201ENSkillsNetwork23008840-2021-01-01).

* Copy the generated CR manifest:

$ cp config/samples/helm-chart\_v1alpha1\_wordpress.yaml config/samples/wordpress-demo.yaml

* Change the WordPress chart release name from the default generated name:

Update name: wordpress-sample to name: wordpress-demo

* Change one of the values in the manifest to demonstrate that CR manifest exposes the chart values:

Change spec.wordpressBlogName to The Demo Blog!. We check this later when the server is running.

1. Deploy WordPress:

$ kubectl create ns wordpress-demo

namespace/wordpress-demo created

$ kubectl apply -f config/samples/wordpress-demo.yaml -n wordpress-demo --validate=false

wordpress.helm-chart.example.com/wordpress-demo created

Note: You need to turn off validation (--validate=false) because the WordPress Helm chart uses null values for some defaults, and this can cause the Kubernetes schema validation to fail. This should not be done in production systems.

1. Verify that WordPress is running:

$ kubectl get pods -n wordpress-demo

NAME READY STATUS RESTARTS AGE

pod/wordpress-demo-646d4d897d-qzjbz 1/1 Running 0 88m

pod/wordpress-demo-mariadb-0 1/1 Running 0 88m

Note: You can also check that Helm deployed the WordPress chart from the operator:

$ helm ls -A

NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION

wordpress-demo wordpress-demo 1 2021-02-12 18:30:19.542550642 +0000 UTC deployed wordpress-10.6.4 5.6.1

1. Check that you can access WordPress:

* The details on how to access the WordPress site are in the chart notes when you deploy the chart. You can get the details in one of two ways:
  + Check the operator manager logs for the output with the following command:
  + $ kubectl logs deployment.apps/wordpress-operator-controller-manager -n wordpress-operator-system -c manager
  + Get the notes using the Helm client as follows:
  + $ helm get notes wordpress-demo -n wordpress-demo
* Open a browser and access WordPress using the obtained URL. You should see the updated value of THE DEMO BLOG! on the page.

Clean up

**WARNING:** This command will remove the WordPress deployment, the operator, and all other resources created.

$ kubectl delete -f config/samples/wordpress-demo.yaml -n wordpress-demo

wordpress.helm-chart.example.com "wordpress-demo" deleted

$ make undeploy

projects/operators/helm-charts/wordpress-operator/bin/kustomize build config/default | kubectl delete -f -

namespace "wordpress-operator-system" deleted

customresourcedefinition.apiextensions.k8s.io "wordpresses.helm-chart.example.com" deleted

role.rbac.authorization.k8s.io "wordpress-operator-leader-election-role" deleted

clusterrole.rbac.authorization.k8s.io "wordpress-operator-manager-role" deleted

clusterrole.rbac.authorization.k8s.io "wordpress-operator-metrics-reader" deleted

clusterrole.rbac.authorization.k8s.io "wordpress-operator-proxy-role" deleted

rolebinding.rbac.authorization.k8s.io "wordpress-operator-leader-election-rolebinding" deleted

clusterrolebinding.rbac.authorization.k8s.io "wordpress-operator-manager-rolebinding" deleted

clusterrolebinding.rbac.authorization.k8s.io "wordpress-operator-proxy-rolebinding" deleted

service "wordpress-operator-controller-manager-metrics-service" deleted

deployment.apps "wordpress-operator-controller-manager" deleted

$ kubectl delete -n wordpress-demo all --all

persistentvolumeclaim "data-wordpress-demo-mariadb-0" deleted

Note: You first need to uninstall the release before removing the operator because the operator could be stopped before cleanup and can therefore hang the undeploy waiting. See this [Operator SDK issue](https://github.com/operator-framework/operator-sdk/issues/4383) for more details. As a final step, you need to clean up the PersistentVolumeClaim (PVC) that was created when the chart was deployed. This is because Helm does not remove PVCs by design. See [Helm issue](https://github.com/helm/helm/issues/5156) for more details.