Ansible Operators

Ansible is a tool for the provisioning, configuration, and management of application infrastructure on compute resources. While it supports many backends, the [Kubernetes collection](https://github.com/ansible-collections/community.kubernetes) specifically allows for the direct provisioning of Kubernetes resources. When combined with the operator design pattern, this allows you to extend the API of a Kubernetes cluster, similar to other operators.

Ansible operators allow a greater degree of control than Helm operators, as you can define custom Ansible tasks that you directly control. Ansible operators may be a good fit if you are already familiar with Ansible or have existing Ansible configuration or infrastructure.

## **Anatomy of an Ansible operator**

As mentioned, Ansible allows for the programmatic provisioning of infrastructure. An abtracted Ansible workflow looks something like the following:

A picture containing text, clock, clipart

Description automatically generated

Note that an Ansible server can exist in several forms. The server can only exist dynamically at runtime on a developer's machine, or it could be a long-running server running somewhere in the cloud. To create an Ansible operator, you will fill in the basic operator architecture with Ansible components, like so:

A screenshot of a cell phone

Description automatically generated with low confidence

In an Ansible operator, the controller is a running Ansible server. This server is based on an Operator SDK community-maintained image, sometimes referred to as the 'ansible-operator'. This image is configured with Ansible files such as roles and playbooks that describe the form and function of the Kubernetes operator. Kubernetes Custom Resource Definitions are created based on these Ansible files and added to the Kubernetes cluster. When a user interacts with the operator, for example, by provisioning an instance of the custom resource, the ansible-operator controller reconciles this by provisioning Kubernetes resources via the Ansible Kubernetes collection. Additonal configuration of the ansible-operator controller allows for information flow in the other direction, to allow for things such as updating the status fields of the original Kubernetes object.

# Module 3 - Building your first Ansible Operator

Ansible is a tool for the provisioning, configuration, and management of application infrastructure on compute resources. Operator SDK allows someone familiar with Ansible to quickly scaffold an operator based on Ansible playbooks and roles. You can use these Ansible resources to configure a controller (referenced as ansible-operator) based on a image maintained by Operator SDK. If you have existing Ansible-based infrastructure or are already familiar with Ansible abstractions, the Ansible operator is a good place to start.

This module covers using [Operator SDK](https://github.com/operator-framework/operator-sdk) to scaffold a basic Ansible operator similar to the Memcached operator you created in Module 1. In this lab, you create a new project, construct some basic reconciliation logic to deploy the various resources such as deployments and services to create the backend for your new Memcached type, and then deploy your operator to your Kubernetes cluster.

## **Step 1: Create a project**

First, create a new directory and init the project in it:

$ mkdir ansible-operator

$ cd ansible-operator

$ operator-sdk init --plugins=ansible --domain example.com

--domain is used for any API groups the operator creates, so yours will be all \*.example.com. An *API group* is simply a group of related functionality, such as your operator. An API group you might already 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 you can append to any API groups that you define to help avoid name collisions.

Let's take a look at what was generated:

* a watches.yaml file that defines the mapping between your API and your Ansible playbooks and roles
* a Dockerfile for the controller image, based on Operator SDK's generic ansible-operator image
* a PROJECT file with the domain and project layout configuration
* a Makefile to build, deploy, and undeploy the project
* an Ansible requirements.yaml file that contains the Ansible dependencies of your project
* Playbooks and Roles directories to contain the corresponding Ansible files
* a molecule directory that contains scaffolding for use with Molecule, an Ansible test framework
* 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

## **Step 2: Create an API**

Run the following command to create an Memcached API:

$ operator-sdk create api --group cache --version v1alpha1 --kind Memcached --generate-role

This command creates a new API by creating a new CRD in the 'cache' API group. 'v1alpha1' is the Kubernetes version of the CRD. --generate-role causes Operator SDK to also generate an Ansible role that serves as the backing for your custom resource type.

Note: The --version flag is for the [Kubernetes API version](https://kubernetes.io/docs/concepts/overview/kubernetes-api/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkCO0201ENSkillsNetwork23008840-2021-01-01#api-groups-and-versioning) of the operator, not a [semantic version](https://ssemver.org/). As such, do NOT use . for --version.

Let's see what the command generates:

* a Memcached custom resource definition
* an Ansible role to back that CRD
* scaffolding to configure the ansible-operator server image to connect them
* RBAC to allow the ansible-operator server to edit the needed Kubernetes resources
* sample YAML in config/samples/ to create an example Memcached custom resource object
* changes in watches.yaml based on the new Memcached custom resource
* changes in PROJECT to show the new API

## **Step 3: Configure the Ansible role**

An Ansible role is a collection of tasks and related scaffolding for how to execute those tasks such as scripts, environment variables, and the like. For this example operator, you will create a single Ansible role to provide the backing for your CRD. Note that Ansible is a feature-rich system with plenty of methods for customization, for further information you should check [the Ansible docs](https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkCO0201ENSkillsNetwork23008840-2021-01-01).

Update roles/memcached/tasks/main.yml to contain the following:

---

- name: start memcached

community.kubernetes.k8s:

definition:

kind: Deployment

apiVersion: apps/v1

metadata:

name: '{{ ansible\_operator\_meta.name }}-memcached'

namespace: '{{ ansible\_operator\_meta.namespace }}'

spec:

replicas: "{{size}}"

selector:

matchLabels:

app: memcached

template:

metadata:

labels:

app: memcached

spec:

containers:

- name: memcached

command:

- memcached

- -m=64

- -o

- modern

- -v

image: "docker.io/memcached:1.4.36-alpine"

ports:

- containerPort: 11211

This role contains a single task named 'start memcached'. It uses the interface defined by the Ansible Kubernetes collection to create a deployment configured with the information available from an instance of our Memcached custom resource. It is configured to set the replicas of the deployment to the size variable from our custom resource's spec.

Note that for an Ansible operator, this file configures the anisble-operator controller and actually creates the mapping that causes the operator to function.

Due to the simple nature of this lab, we won't edit the CRD itself to add the **Size** field to the spec, as Kubernetes objects allow for arbitrary field entry. In an actual use case, it is recommended that you fill in the CRD with the fields corresponding to the fields in your Ansible roles, so they are visible to Kubernetes users interacting with the custom resource.

In addition, it is good practice to define defaults here in the Ansible files, so edit roles/memcached/defaults/main.yml to include the following:

---

Now you're ready to deploy your operator! You may why we didn't add any fields to the custom resource definition (CRD) or change any Golang files. By default, the CRD allows arbitrary field entry, and the actual mapping is based on the Ansible role. Thus, you don't actually need to define any fields in the CRD itself. For this lab, you won't modify the CRD beyond the minimum needed to make things work. For an actual Ansible operator, it is recommended that you fill in the CRD with the fields users are expected to fill in.

## **Step 4: Build and deploy the operator**

Build and push the image to a Docker repository.

$ export USERNAME=<docker-registry-username>

$ make docker-build docker-push IMG=docker.io/$USERNAME/ansible-operator:1.0.0

$ make deploy IMG=docker.io/$USERNAME/ansible-operator:1.0.0

After a bit, you should see the pods containing the controller come up:

$ kubectl get pods --n ansible-operator-system

NAMESPACE NAME READY STATUS RESTARTS AGE

ansible-operator-system ansible-operator-controller-manager-7b8555fb4b-wv8mh 0/2 ContainerCreating 0 14s

Check to see that the CRD has been created:

$ kubectl get crds

NAME CREATED AT

memcacheds.cache.example.com 2021-03-13T00:20:18Z

And the RBAC to tie it all together:

$ kubectl get clusterroles | grep ansible

ansible-operator-manager-role 2021-03-13T00:20:18Z

ansible-operator-metrics-reader 2021-03-13T00:20:18Z

ansible-operator-proxy-role 2021-03-13T00:20:18Z

$ kubectl get clusterrolebindings | grep ansible

ansible-operator-manager-rolebinding ClusterRole/ansible-operator-manager-role 2m27s

ansible-operator-proxy-rolebinding ClusterRole/ansible-operator-proxy-role 2m27s

$ kubectl get roles -n ansible-operator-system

NAME CREATED AT

ansible-operator-leader-election-role 2021-03-13T00:20:18Z

$ kubectl get rolebindings -n ansible-operator-system

NAME ROLE AGE

ansible-operator-leader-election-rolebinding Role/ansible-operator-leader-election-role 3m23s

Once the controller pods are up and running, the operator is ready to use.

### **Create a Memcached object**

Edit config/samples/cache\_v1alpha1\_memcached.yaml to include a size of 3:

apiVersion: cache.example.com/v1alpha1

kind: Memcached

metadata:

name: memcached-sample

spec:

size: 3

Then, create the memcached object:

$ kubectl apply -f config/samples/cache\_v1alpha1\_memcached.yaml

You should see a deployment with three pods created:

$ kubectl get deployments

NAME READY UP-TO-DATE AVAILABLE AGE

memcached-sample-memcached 3/3 3 3 37s

$ kubectl get pods

NAMESPACE NAME READY STATUS RESTARTS AGE

default memcached-sample-memcached-54946946b-6chnn 0/1 ContainerCreating 0 38s

default memcached-sample-memcached-54946946b-8qwbz 0/1 ContainerCreating 0 38s

default memcached-sample-memcached-54946946b-k9z26 0/1 ContainerCreating 0 38s

The Ansible information for the task is available in the status of the CR object:

$ kubectl get memcached memcached-sample -o yaml

apiVersion: cache.example.com/v1alpha1

kind: Memcached

metadata:

creationTimestamp: "2021-03-13T00:34:25Z"

generation: 1

managedFields:

- apiVersion: cache.example.com/v1alpha1

fieldsType: FieldsV1

fieldsV1:

f:status:

.: {}

f:conditions: {}

manager: ansible-operator

operation: Update

time: "2021-03-13T00:34:25Z"

- apiVersion: cache.example.com/v1alpha1

fieldsType: FieldsV1

fieldsV1:

f:spec:

.: {}

f:size: {}

manager: kubectl

operation: Update

time: "2021-03-13T00:34:25Z"

name: memcached-sample

namespace: default

resourceVersion: "1608"

uid: 4ebe4e9f-b41a-429b-89c3-e4eaebbd3554

spec:

size: 3

status:

conditions:

- ansibleResult:

changed: 0

completion: 2021-03-13T00:34:39.490213

failures: 0

ok: 1

skipped: 0

lastTransitionTime: "2021-03-13T00:34:25Z"

message: Awaiting next reconciliation

reason: Successful

status: "True"

type: Running

Note the Ansible status in the Kubernetes status. The ansible-operator controller populated this data automatically. If you want to include additional data, you define those additional tasks in your Ansible role to collect the data and populate the Kubernetes status.

Go ahead and change the size of the object to see the controller reconcile your object. Edit config/samples/cache\_v1alpha1\_memcached.yaml to have a size of 2:

apiVersion: cache.example.com/v1alpha1

kind: Memcached

metadata:

name: memcached-sample

spec:

size: 2

And then apply it:

$ kubectl apply -f config/samples/cache\_v1alpha1\_memcached.yaml

And see the pods drop down to 2:

$ kubectl get pods

NAME READY STATUS RESTARTS AGE

memcached-sample-memcached-54946946b-6chnn 1/1 Running 0 11m

memcached-sample-memcached-54946946b-8qwbz 0/1 Terminating 0 11m

memcached-sample-memcached-54946946b-k9z26 1/1 Running 0 11m

Feel free to play around with it to get a feel for the behavior of the Custom Resource.

## **Step 5: Cleanup**

When you're done, you can clean up the deployed operator by running the following commands:

$ kubectl delete memcached memcached-sample

$ make undeploy