WHAT IS THE KUBERNETES OPERATOR PATTERN?

A software extension to Kubernetes, operators function by capturing the knowledge of a human operator. Engineers who use Kubernetes have a unique perspective on how apps and services should behave, and how to react when problems arise. Those same engineers like to automate repeatable tasks—and the operator pattern is essential to this automation.

In this article, I'll shed light on the operator pattern, including how it applies to Kubernetes, when to use it, and some tools to help get started with creating one.

Understanding the K8S operator

Using a Kubernetes **operator** means you can operate a stateful application by writing a custom controller with domain specific knowledge built into it. If you are new to Kubernetes, the idea of an operator can be confusing, so let's look at a simple example.

Say you have a java app that connects to a database. You want to deploy that to your k8s cluster. Ideally, you'd want to run something like a "deployment" for the java app exposed with a service, and for the backend, run a "statefulset" for the database application. There are two parts to this setup:

- The **stateless** part, the Java app
- The **stateful** part, the database

To understand an operator, think of the stateful part of the setup—the database or any application that stores data, like etcd. So, in our example, we can apply what we know about how the application relates to the database and create a controller that will do certain things when the application behaves in a certain way.

Site reliability engineers and operational engineers are often tasked with, or interested in, automating things like backup, updates, data restore, etc. How these tasks are achieved varies depending on the application itself and the business use case (domain knowledge). This is exactly what k8s operators do: act on behalf of the user when you need to perform certain tasks that an SRE/Ops engineer normally has to perform.

Operators follow K8S patterns

Operator is built on two key principles of Kubernetes: a custom resource and a custom controller.

Custom resource

In Kubernetes, a **resource** is an endpoint in the k8s API that stores a bunch of API objects of a specific kind. It allows us to extend k8s by adding more objects of a kind to the cluster. After that, we can use kubectl to access our object just like any other built-in object.

Take, for example, a pod or deployment. When you write a manifest, you have to specify a kind (pod or deployment) in the yaml file. A custom resource is simply a resource that does not come bundled with k8s out of the box.

Custom controller

A **controller** is a control loop that watches the cluster for changes to a specific resource (custom resource) and makes sure that the current state matches the desired state. As a matter of fact, we are already using some form of controller already built into k8s.

A good example is a deployment wherein you kill a pod, and another one spins up. The controller sees that the number of pods you desire does not match the current state, so it spins another one up to match the desired state.

So, why aren't those built-in controllers called operators? Because those controllers are not specific to a particular application; they are upstream controllers used with built-in resources, like deployment, jobs, etc.

When to use an operator

It is important to know that all operators are controllers but not all controllers are operators. For a controller to be considered an operator, it must have application domain knowledge in it to perform automated tasks on behalf of the user (SRE/Ops engineer).

- Use an operator whenever you need to encapsulate your stateful application business logic controlling everything with Kubernetes API. This allows automation around your application built into the k8s ecosystem.
- Use an operator whenever you need to build a tool that watches your applications for changes and perform certain SRE/Ops tasks when certain things happen.

How to build an operator

There are several ways to build an operator:

- ClientGo connects to the Kubernetes API.
    - The benefit: It uses the same resources as your built-in resources so you can rest easy knowing that the code is tested and versioned according to k8s standards.
    - The downside: It has a steep learning curve if you don't understand the Go programming language.
- Kubebuilder, part of the k8s sigs organization, is written in go and uses the controller-runtime.
- Operator SDK, originally written by core OS and now run by RedHat, is a framework that comes with helper functions to create operators in Go, HEML, or Ansible.

How to deploy an operator

You can deploy an operator in two ways:

- Using yaml just like any other Kubernetes manifest.
- Using Helm chart to deploy both CRD and controller as a package.

Best practices for creating an operator

K8S controllers are for the cluster itself, and operators are controllers for your deployed stateful applications.

When creating an operator, follow these best pattern practices:

- Take advantage of built-in kinds to create your custom kinds. That way, you are leveraging already tested and proven kinds.
- Ensure no other outside code is needed for your controller to function. So, running kubectl install should be all you need to deploy the controller.
- If the operator is stopped, make sure your application can still function as expected.
- Employ sufficient tests for your controller code.

When you're ready to create your own application-specific custom resources that can be reconciled with your custom controller which allows you to extend the normal behavior of Kubernetes—you are ready to use operators.

Additional resources

For more on Kubernetes, explore these resources:

- Kubernetes Guide, with 20+ articles and tutorials
- BMC DevOps Blog
- Bring Kubernetes to the Serverless Party
- How eBay is Reinventing Their IT with Kubernetes & Replatforming Program