Intro to Operators

This article is the first in a series of articles and tutorials that show you how to build and deploy a Kubernetes Operator. In this article, learn what Kubernetes Operators are, what they do, and key features including the Operator SDK, Operator capability levels and Operator Hub. You will learn all of the basic knowledge needed to develop an operator implemented using Golang.

If you are already familiar with operators, you can either skim this article or skip ahead to the next Operator tutorial: Develop and deploy a Memcached Operator on OpenShift which shows how to develop and deploy your first operator to an OpenShift cluster. Alternately, read one of the other articles in the learning path.

Prerequisites

This article assumes you want to learn the basic concepts about Kubernetes Operators and steps needed to develop a Golang-based operator to manage Kubernetes resources.

- You have a basic understanding of Kubernetes concepts and how to install a workload
- You have little to no knowledge about Kubernetes Operators concepts
- You have little to no experience developing operators

Estimated time

This article should take roughly 15-30 minutes to complete.

What are operators?

Operators are extensions to Kubernetes that use custom resources to manage Kubernetes applications and their components. Operators are used to automate software configuration and maintenance activities that are typically performed by human operators. That's why they are called operators!

While Kubernetes is great at managing stateless applications, when you need more complex configuration details for a stateful application, such as a database, operators are very useful. An operator can also automate other more complex lifecycle management tasks such as version upgrades, failure recovery, and scaling.

Operators extend the Kubernetes control plane with specialized functionality to manage a workload on behalf of a Kubernetes admin. An operator includes these components:

- A custom resource definition (CRD) that defines a schema of settings available for configuring the workload
- A custom resource (CR) is the Kubernetes API extension created by a CRD
    - A custom resource instance sets specific values for the settings defined by the CRD to describe the configuration of a workload
- A controller customized for the workload that configures the current state of the workload to match the desired state represented by the values in the CR

Operators have the following features:

- The user provides configuration settings within a CR, and then the operator translates the configuration into low-level actions, using the operator's custom controller logic to implement the translation
- An operator introduces new object types through its custom resource definition. These objects can be handled by the Kubernetes API just like native Kubernetes objects, including interaction via Kubernetes client tools and inclusion in role-based access control policies (RBAC)

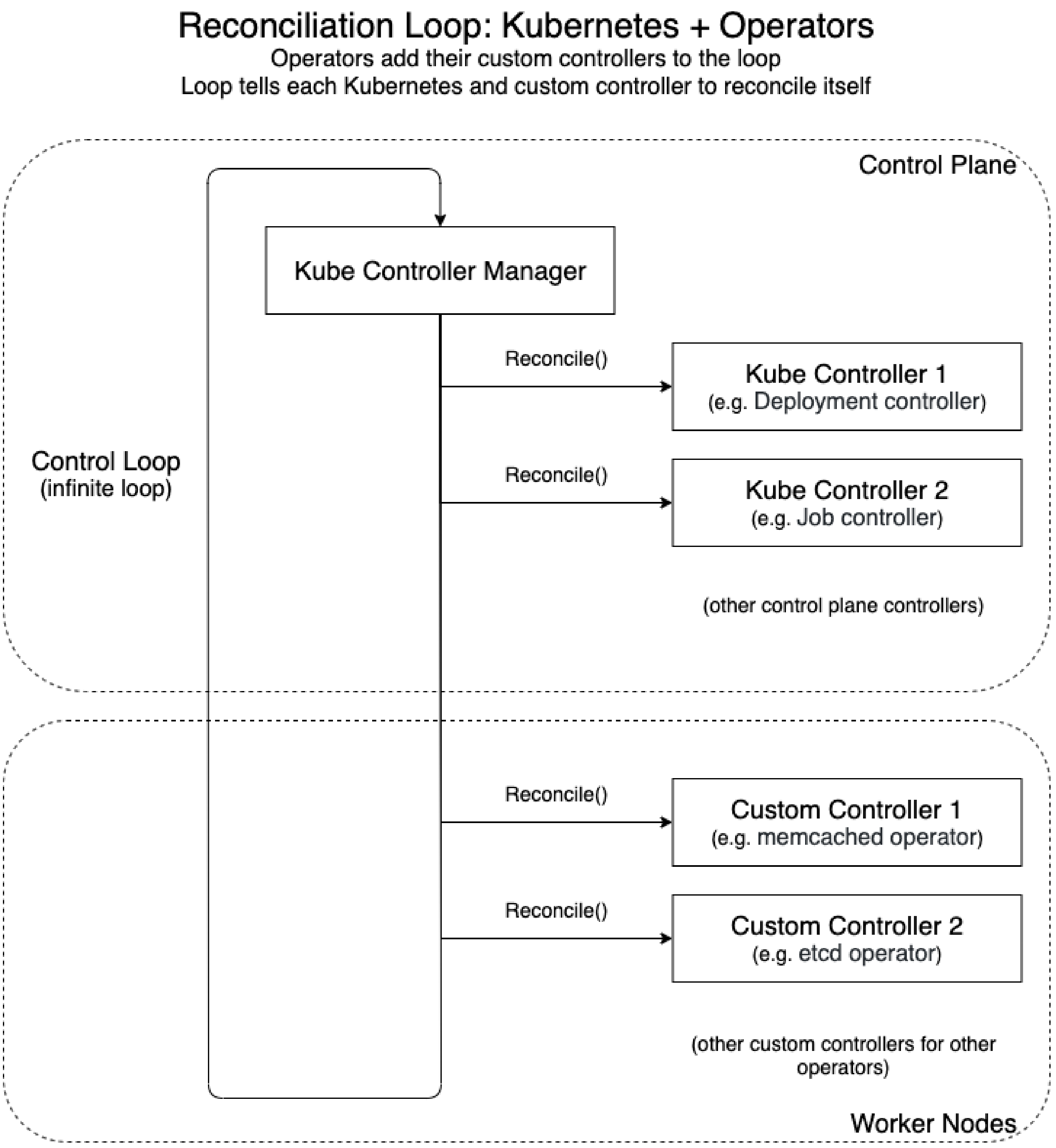
The article What is a Kubernetes operator? by Red Hat explains more details about operators.

What do operators do?

In Kubernetes, controllers in the control plane run in a control loop that repeatedly compares the desired state of the cluster to its current state. If the states don't match,
then the controller takes action to adjust the current state to more closely match the desired state.

Similarly, the controller in an operator watches a specific CR type and takes application-specific actions to make the workload's current state match the desired state expressed in the CR.

This diagram illustrates how the control plane runs the controllers in a loop, where some controllers are built into Kubernetes and some are part of operators:


Reconciliation Loop: Kubernetes + Operators
Operators add their custom controllers to the loop
Loop tells each Kubernetes and custom controller to reconcile itself
Control Plane
Kube Controller Manager
Control Loop
(infinite loop)
Reconcile()
Kube Controller 1
(e.g. Deployment controller)
Reconcile()
Kube Controller 2
(e.g. Job controller)
(other control plane controllers)
Reconcile()
Custom Controller 1
(e.g. memcached operator)
Reconcile()
Custom Controller 2
(e.g. etcd operator)
(other custom controllers for other operators)
Worker Nodes

The controllers in the control plane are optimized for stateless workloads and one set of controllers works for all stateless workloads because they're all very similar. The controller in an operator is customized for one particular stateful workload. Each stateful workload has its own operator with its own controller that knows how to manage this workload.

Why does Kubernetes need operators?

Kubernetes needs operators to automate tasks that are normally performed manually by IT operations personnel. A stateful workload is more difficult to manage than a stateless workload. The state in a workload changes how a workload: needs to be installed, upgrades to a new version, recovers from failures, needs to be monitored, and scales out and back in again. The operator will take care of everything needed to make sure the service keeps running successfully. An operator makes its service more self-managing, enabling an application team to spend less effort managing the service so that it can spend more effort using the service.

How do I use an operator?

To use an operator, you must first deploy the operator as a workload in your cluster. You can deploy it like any other workload, via the Kubernetes CLI or using a Helm chart. Additionally, you can make use of Operator Hub or deploy your operator through the Operator Lifecycle Manager.

Operator SDK

The Operator SDK is a set of open source tools to build, test, and package operators. The SDK CLI enables you to scaffold a project and also provides commands to generate code. The SDK uses make, a build automation tool that runs commands configured in a Makefile to generate executable code and libraries from source code.

The SDK includes pre-built make commands that we will use to develop our operator, such as:

- `make manifests` -- generates YAML manifests based on `kubebuilder` markers
- `make install` -- compiles source code into executables
- `make generate` -- updates the generated code based on an operator's API schema
- `make docker-build` -- builds the operator's Docker container image
- `make docker-push` -- pushes the Docker image
- `make deploy` -- deploys all of the operator's resources to the cluster
- `make undeploy` -- deletes all of the operator's deployed resources from the cluster

These commands in the SDK greatly simplify implementing an operator.

Operator Lifecycle Manager

The SDK also enables you to install the Operator Lifecycle Manager (OLM) using the `operator-sdk olm install` command. The OLM is a set of cluster resources that manage the lifecycle of an operator. Once installed, you can get the status of the OLM using `operator-sdk olm status`, which verifies whether the SDK can successfully communicate with the OLM components in the cluster.

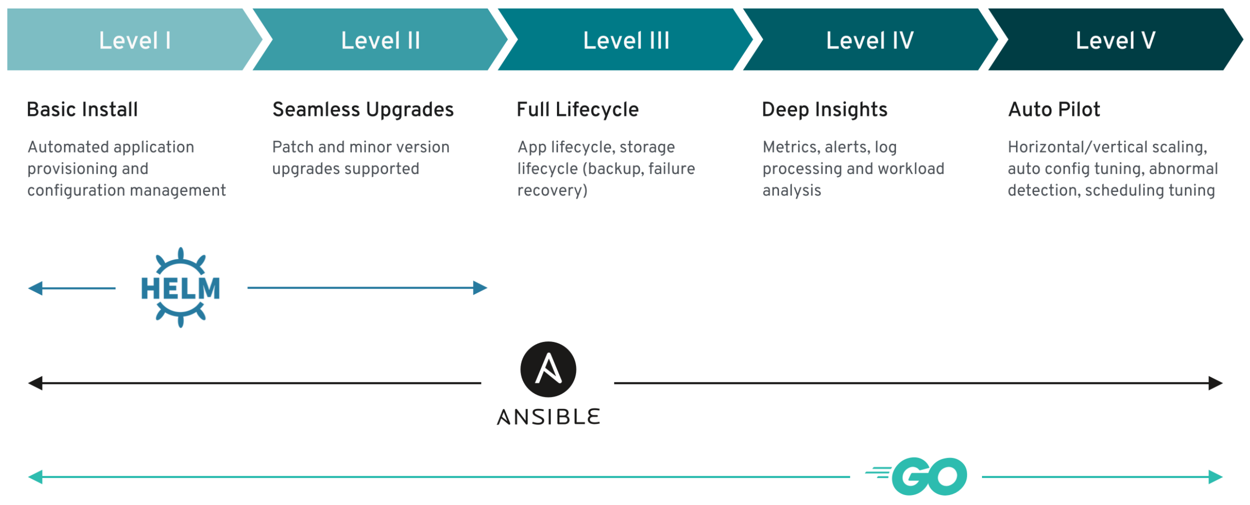
Additional Operator SDK concepts

In addition to operator, custom resource, and custom resource definition, you need to understand what we mean by the Operator SDK *operand* and *managed resource*:

- Operand - the managed workload provided by the Operator as a service
- Managed resources - the Kubernetes objects or off-cluster services the Operator uses to constitute an Operand (also known as secondary resources)

Operator capability Levels

Some operators are more sophisticated at managing their operand's lifecycle than others. The Operator Capability Levels model defines five levels of sophistication, as illustrated here:


Level I
Level II
Level III
Level IV
Level V
Basic Install
Automated application provisioning and configuration management
Seamless Upgrades
Patch and minor version upgrades supported
Full Lifecycle
App lifecycle, storage lifecycle (backup, failure recovery)
Deep Insights
Metrics, alerts, log processing and workload analysis
Auto Pilot
Horizontal/vertical scaling, auto config tuning, abnormal detection, scheduling tuning
HELM
ANSIBLE
GO

This model aims to provide guidance for the features that users can expect from a particular operator. As the picture shows, you can only use Ansible and Go to achieve all five capability levels. Helm can only be used to achieve the first two levels.

Capability levels build on top of one another. That means that if an operator has level 3 capabilities, then it should also have all of the capabilities required from level 1 and level 2.

Let's examine the capabilities of a level 1 operator in more detail.

Operator Capability Level 1 - Basic install

In level 1, your Operator can provision an application as described by a custom resource. The CR specifies all of the configuration details. Avoid making the user create and manage configuration files outside of Kubernetes — that's what the CR is for.

Level 1 Example - Install the Operand

When a custom resource is created, that triggers the Operator, which responds by creating and installing the Operand. If the custom resource is deleted, then the Operator removes the Operand.

To install an Operand, the controller creates the managed resources for that Operand and installs them, which causes the cluster to install the Operand. These managed resources are typical Kubernetes workload kinds such as a `Deployment` and a `Service`, as well as other kinds like a `ConfigMap`, a `Secret`, and a `PersistentVolumeClaim`.

You can specify the configuration of these managed resources in the custom resource's specification (that is, the `spec` section of its YAML); I cover this in greater detail in the next article.

A simple controller may simply copy values out of the CR's specification and into the appropriate fields of the managed resources, maybe transforming the values as needed. Once the cluster installs the Operand, the controller gathers the status of those managed resources and updates the custom resource's status (that is, the `status` section of its YAML); I cover this in greater detail in my next article. Status is how the CR remembers the managed resources that were created for its Operand.

Level 1 Example - Manage the Operand

Now, let's say that you want to increase the capacity of the Operand. How would you do this using the Operator?

You can do this by updating the custom resource's specification — perhaps it has a `size` setting, and you increase it. When you update the custom resource's specification, you are specifying a different configuration for the Operand. The controller notices the changes in the custom resource and responds by changing the configuration of the managed resources — perhaps to increase the number of pods or create new persistent volume claims.

Once the controller applies these changes to the managed resources, the cluster responds by applying them the Operand, thereby scaling the Operand.

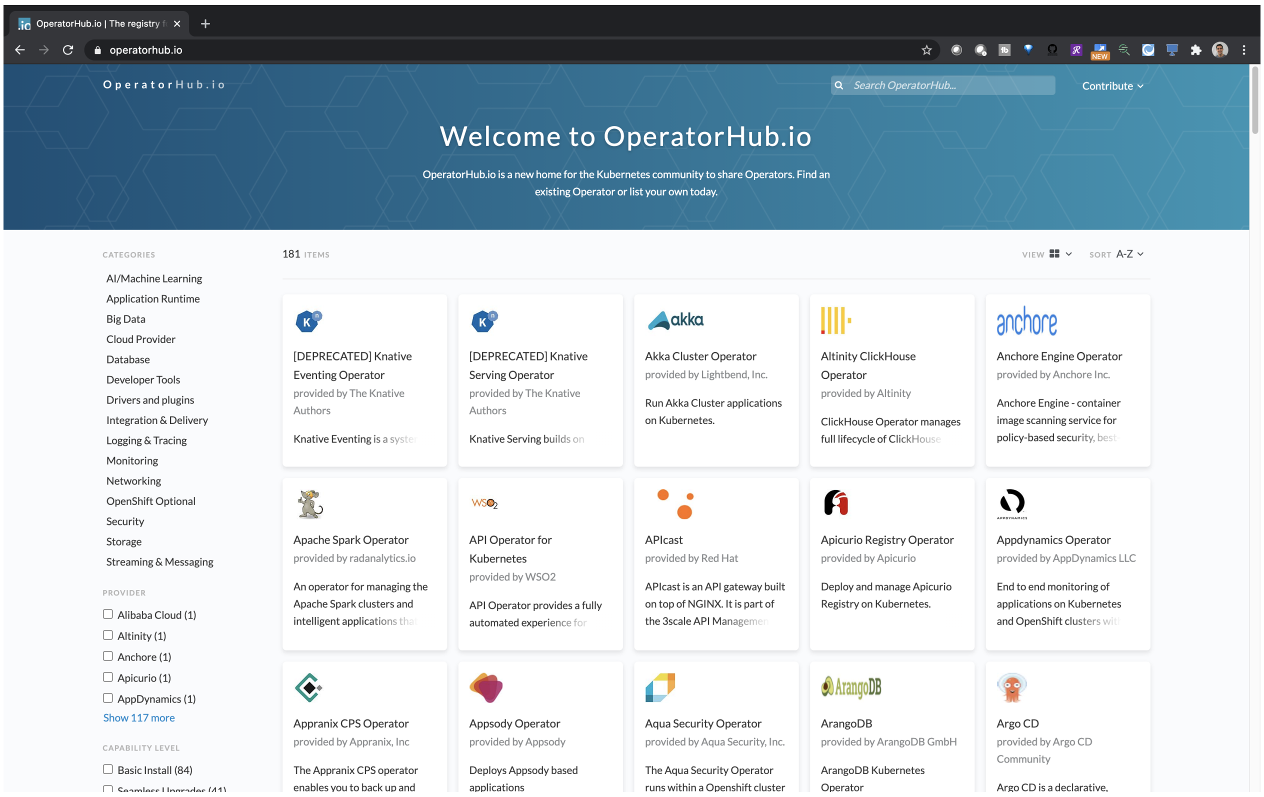
Operator Capability Levels in the Operator SDK documentation also describes the behavior of the other capability levels.

5. Operator Hub

OperatorHub.io is a repository for operators, a public website where you can find over 180 different operators and share the operators you create. OperatorHub.io is very important since this is where you can use other operators to automate the configuration of your Kubernetes applications, and submit your own operator to be published in the hub.

Check out How to contribute an Operator for details of how to package, test, preview, and submit your operator for addition to the Hub.

Its homepage looks like this:



Welcome to OperatorHub.io

OperatorHub.io is a new home for the Kubernetes community to share Operators. Find an existing Operator or list your own today.

CATEGORIES

AI/Machine Learning
Application Runtime
Big Data
Cloud Provider
Database
Developer Tools
Drivers and plugins
Integration & Delivery
Logging & Tracing
Monitoring
Networking
OpenShift Optional
Security
Storage
Streaming & Messaging

PROVIDER

☐ Alibaba Cloud (1)
☐ Altinity (1)
☐ Anchore (1)
☐ Apicurio (1)
☐ AppDynamics (1)
Show 117 more

CAPABILITY LEVEL

☐ Basic Install (84)

181 ITEMS

[DEPRECATED] Knative Eventing Operator
provided by The Knative Authors
Knative Eventing is a syste

[DEPRECATED] Knative Serving Operator
provided by The Knative Authors
Knative Serving builds on

Akka Cluster Operator
provided by Lightbend, Inc.
Run Akka Cluster applications on Kubernetes.

Altinity ClickHouse Operator
provided by Altinity
ClickHouse Operator manages full lifecycle of ClickHouse

Anchore Engine Operator
provided by Anchore Inc.
Anchore Engine - container image scanning service for policy-based security, best

Apache Spark Operator
provided by radanalytics.io
An operator for managing the Apache Spark clusters and intelligent applications tha

API Operator for Kubernetes
provided by WSO2
API Operator provides a fully automated experience for

APIcast
provided by Red Hat
APIcast is an API gateway built on top of NGINX. It is part of the 3scale API Managemen

Apicurio Registry Operator
provided by Apicurio
Deploy and manage Apicurio Registry on Kubernetes.

Appdynamics Operator
provided by AppDynamics LLC
End to end monitoring of applications on Kubernetes and OpenShift clusters wi

Apprantix CPS Operator
provided by Appranix, Inc
The Appranix CPS operator enables you to back up and

Appsody Operator
provided by Appsody
Deploys Appsody based applications

Aqua Security Operator
provided by Aqua Security, Inc.
The Aqua Security Operator runs within a Openshift cluster

ArangoDB
provided by ArangoDB GmbH
ArangoDB Kubernetes Operator

Argo CD
provided by Argo CD Community
Argo CD is a declarative,

Conclusion

In this article, you learned how operators can extend the base Kubernetes functionality using custom controllers and custom resources. You also learned that the Operator SDK offers code scaffolding tools to enable you to write your Operator more easily, and offers guidelines for the capability levels of an operator. Lastly, I introduced you to OperatorHub.io where you can browse existing operators and submit your own.

The next article, Anatomy of an Operator, demystified, takes a closer look at the Kubernetes architecture that enables Operators to work.

If you would rather go straight to developing an Operator, go to the intermediate level tutorial Develop and deploy and operator to OpenShift.