TECHNOLOGY GUIDE

Intel Corporation

Kubernetes* Operators – Automated Lifecycle Management Technology Guide

Authors

Conor Nolan

1 Introduction

This document describes the Kubernetes Operator Pattern, which has rapidly become a common and trusted practice in the Kubernetes ecosystem for automation of application lifecycle management.

A Kubernetes Operator is an application-specific controller that extends the functionality of the Kubernetes API to create, configure, and manage instances of complex applications on behalf of a Kubernetes user.

Kubernetes is designed for automation. Out of the box, Kubernetes provides built-in automation for deploying, running and scaling workloads. However, some workloads and services require a deep knowledge of how the system should behave beyond the capabilities of core Kubernetes functionality. Operators are purpose-built with operational intelligence to address the individuality of such applications.

Easy-to-use tools for operator creation help developers build a great automation experience for cluster administrators and end users. By extending a common set of Kubernetes APIs and tools, Kubernetes operators can help provide ease of deployment and streamlined Day 1 and Day 2 operations.

The Kubernetes Operator Pattern has emerged as a leading solution to ensure the automation of these function-specific applications is possible in a Kubernetes cluster.

This document is targeted at those looking for an introduction to the Kubernetes Operator Pattern. Basic knowledge of Kubernetes is recommended.

This document is part of the Network Transformation Experience Kit, which is available at https://networkbuilders.intel.com/intel-technologies/network-transformation-exp-kits.

Table of Contents

Figures

Tables

1.1 Terminology

Table 1. Terminology

ABBREVIATION	DESCRIPTION
API	Application Programming Interface
CLI	Command-line interface
CPU	Central Processing Unit
CR	Custom Resource
CRD	Custom Resource Definition
GPU	Graphic Processing Unit
MBA	Memory Bandwidth Allocation
NIC	Network Interface Controller
QAT	QuickAssist Technology
RMD	Resource Management Daemon
SDK	Software Development Kit
SR-IOV	Single Root Input Output Virtualization
SST-BF	Speed Select Technology – Base Frequency
SST-CP	Speed Select Technology – Core Power

1.2 Reference Documentation

Table 2. Reference Documents

REFERENCE	SOURCE
Red Hat: What is a Kubernetes Operator?	https://www.redhat.com/en/topics/containers/what-is-a-kubernetes-operator
Extending Kubernetes	https://kubernetes.io/docs/concepts/extend-kubernetes/
Operator Pattern	https://kubernetes.io/docs/concepts/extend-kubernetes/operator
Kubebuilder	https://book.kubebuilder.io/
Operator Framework	https://operatorframework.io/
Operator SDK	https://sdk.operatorframework.io/
Operator Hub	https://operatorhub.io/
Resource Management Daemon (RMD)	https://github.com/intel/rmd
RMD Operator	https://github.com/intel/rmd-operator
Intel Device Plugins for Kubernetes	https://github.com/intel/intel-device-plugins-for-kubernetes

2 Overview

2.1 Challenges Addressed

Many of the current applications possess complex requirements beyond those catered to by existing core Kubernetes constructs and APIs. These requirements might include support for more granular resource allocations such as memory bandwidth or advanced CPU capabilities. In order to cater to such applications in a Kubernetes environment, it is often necessary to extend existing Kubernetes functionality and APIs using the operator pattern.

2.2 Use Cases

A human operator who manages a specific application is responsible for full software lifecycle management. This includes deploying, monitoring, and manually applying desired changes to that application. This entire process can be fully automated by the Kubernetes operator.

More specifically, Day 1 of the software lifecycle includes development and deployment of the software application as part of a continuous integration and continuous deployment (CI/CD) pipeline. Day 2 is when the product is made available to the customer. Then, the focus is on maintaining, monitoring, and optimizing the system. A feedback loop on current behavior is very important for the system to react correctly to constantly changing circumstances until the end of the application's life. Both Day 1 and Day 2 operations can be automated considerably with the Kubernetes operator pattern as it caters to automatic deployment and custom control loops to monitor application behavior.

The Kubernetes operator pattern also lends itself to complex system configuration. Hardware-specific features such as Intel® Speed Select Technology (Intel® SST), Cache Allocation Technology (CAT) and Memory Bandwidth Allocation (MBA) are outside the knowledge and scope of native Kubernetes. These features can be configured and provisioned by Kubernetes operators and in turn utilized by performance-critical workloads.

2.3 Technology Description

The Custom Resource Definitions (CRD) feature has been a part of Kubernetes since v1.7. This feature enables the developer to extend the Kubernetes API with their own object types (i.e. Custom Resources) that cater to their application's specific requirements. Since the introduction of this feature, it has been possible to manually create both CRDs to extend the default Kubernetes distribution and custom controllers to manage them. However, this process, which we now commonly refer to as the Kubernetes Operator Pattern, has been simplified and popularized by utilities such as The Operator Framework and Kubebuilder.

These tools make the process of building an operator much more lightweight for developers. They provide a command-line interface to facilitate user friendly CRD creation and controller scaffolding from which the developer can build and run custom, application-specific functionality. These tools also provide the utilities to containerize the operator software itself. This means that the operator can be deployed into a Kubernetes cluster like any other application and can be managed as such with existing Kubernetes constructs, APIs, and CLIs. These tools also include capabilities such as deployment manifest generation and end to end testing frameworks to help verify the integrity of the operator software. A growing number of operators for a wide variety of applications can be viewed at operatorhub.io.

3 Deployment

Operators are software components that extend Kubernetes functionality to manage applications and cater to their specific use cases.

The Kubernetes operator pattern is achieved in two steps. Firstly, the Kubernetes API is extended with a CRD that defines the specification for the application object. The standard Kubernetes distribution ships many native objects such as Pods, Deployments, StatefulSets etc. The CRD feature enables users to manage their own objects known as "custom resources". Once a CRD is created, it becomes an extension to the Kubernetes API and can be utilized like any native Kubernetes object, leveraging all Kubernetes utilities such as its API services, CLI, and storage of child custom resources in the *Etcd* control plane component.

Secondly, the operator is developed to manage all instances of this custom resource with a custom controller. The control loop principle is a cornerstone of the Kubernetes architecture. This is the practice of observing the current state of an object, comparing that current state to the object's desired state, and finally acting to correct the current state if it does not align with the desired state. This simple process of observation and reconciliation is also fundamental to the operator pattern. In its simplest form, a Kubernetes operator is a custom controller watching a custom resource and taking action to modify the custom resource status based on the custom resource specification. This custom controller is created by the developer with functionality specific to the custom resource it reconciles. It is also worth noting that a Kubernetes operator can be designed to consist of multiple CRDs and controllers.

The operator itself is packaged into a container image. It can then be deployed to a Kubernetes cluster using an existing construct such as a pod or deployment. Once deployed, the operator will continuously reconcile its corresponding custom resources with custom functionality as created by the developer.

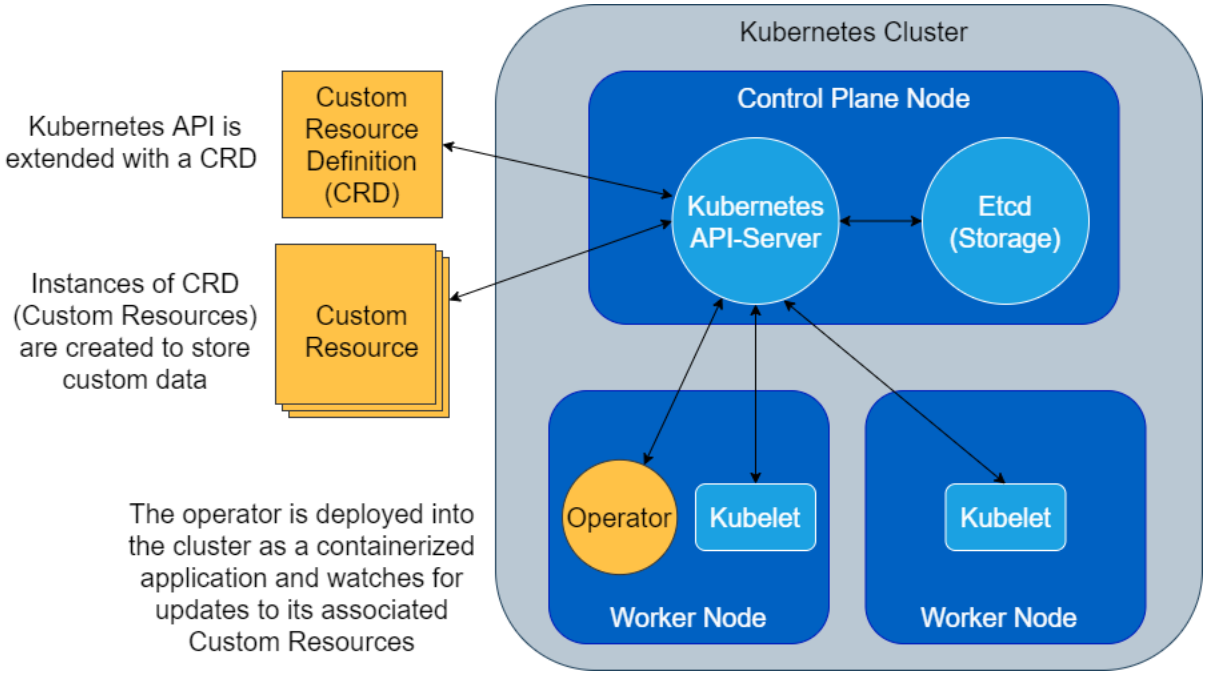

Figure 1. Deployment Diagram

4 Implementation Example

4.1 Resource Management Daemon Operator (RMD-Operator)

The Intel RMD Operator is a Kubernetes operator designed to provision and manage Resource Management Daemon (RMD) instances in a Kubernetes cluster. The operator is responsible for lifecycle management of RMD on suitable nodes in the cluster, advertisement of extended resources (layer 3 cache ways) per node and orchestration of RMD workloads to all RMD components throughout the cluster.

4.2 Device Plugins Operator

The goal of the Intel Device Plugins Operator is to serve the installation and lifecycle management of Intel Device Plugins for Kubernetes and provide one-click installation. Currently the operator has limited support for the QAT and GPU device plugins, validating container image references and extending reported statuses. This support will be extended to more Kubernetes device plugins such as FPGA and VPU in future releases.

4.3 Power Operator

Intel is currently developing a Kubernetes operator to provide power management capabilities from the latest Intel® Xeon® processors to CPUs allocated to containers in a Kubernetes cluster. These capabilities include *Intel® Speed Select Technology – Base Frequency (Intel® SST-BF)* and *Speed Select Technology – Core Power (SST—CP).* The goal of the operator is to automate configuration of power capabilities for workloads, thus removing the need for complex and tedious manual setups.

Intel® SST is currently supported in Kubernetes through Intel's CPU Manager for Kubernetes (CMK). While this approach is functional, it is also tied to static provisioning of node resources and lacks flexibility as a result. By utilizing the operator pattern, these capabilities can be implemented in a more dynamic fashion. Such complex hardware capabilities can now be represented in a custom resource as *Power Profiles.* From there, preferred settings and power related configurations can be applied automatically to suitable workloads on-the-fly by the operator's custom controller(s). This automated workflow both optimizes resource utilization on the host and unshackles the user from much of the pre-provisioning overhead associated with previous workflows.

5 Benefits

The extensibility and customizability of an operator lends itself to all manner of applications that require life-cycle-management of feature specific software in a Kubernetes cluster.

Cluster administrators can develop operators to automate tasks associated with cluster management and reduce the management overhead. Developers can build operators to control the applications they are delivering to customers. Thus, enabling the customer to simply deploy an application-specific operator to handle the deployment and management of the application, without the need for in depth knowledge and hands-on management. The ultimate benefit lies in the increased consumability of the application and platform feature as a result of the automation provided by the operator.

6 Summary

The Kubernetes Operator Pattern has established itself as a trusted methodology for application lifecycle management. Due to this popularization, Kubernetes operators are used extensively within the open-source community. This widescale community adoption and investment has resulted in the advancement of operator building tools such as Kubebuilder and Operator-SDK. These utilities greatly simplify the process of designing, creating, and deploying complex operators and have helped make operators a favored practice among developers within the cloud native ecosystem. The true value of Kubernetes operators exists in their ability to make complex applications more consumable to end users through automation of application deployment and management.

Kubernetes operators are currently and will continue to be leveraged by Intel to enable applications that promote consumption of Intel Architecture such as Power Management features, CAT, MBA, SR-IOV NICs, GPUs, Accelerators, and much more.

0121/DN/WIPRO/PDF 634303-001US