Operator Developer Guide for Red Hat Partners

Community and Certified Operators

Prepared for: Red Hat Partners and Technical Staff
Version: 1 (Feb 2019)

Operator Developer Guide

This document covers high level objectives, benefits and distribution opportunities for Operators created by Red Hat partners and open source communities.

Red Hat strongly believes that Operators provide the best Day 1 and Day 2 management experience for modern applications. Red Hat's goal is twofold:

1. Provide a set of tools to build high quality Operators that fulfill the promise of hybrid cloud
2. Provide a discovery and distribution method to connect Red Hat customers with partners that solve the customer's business problem(s) with Operators. Red Hat needs you – as the expert in your arena – to make our customers successful.

Table of Contents

Operator Overview

What is an Operator?

Conceptually, an Operator takes human operational knowledge and encodes it into software that is more easily packaged and shared with consumers. Think of an Operator as an extension of the software vendor's engineering team that watches over a Kubernetes environment and uses its current state to make decisions in milliseconds. Advanced Operators are designed to handle upgrades seamlessly, react to failures automatically, and not take shortcuts, like skipping a software backup process to save time.

More technically, an Operator is a method of packaging, deploying and managing a Kubernetes application. A Kubernetes application is an application that is both deployed on Kubernetes and managed using the Kubernetes APIs and kubectl/oc tooling. You can think of Operators as the runtime that manages this type of application on Kubernetes.

Q: Why deploy on Kubernetes?
A: It contains all of the primitives needed to build complex distributed systems – secret handling, load balancing, service discovery, autoscaling – that work across providers.

Q: Why manage your app with Kubernetes APIs and kubectl tooling?
A: These APIs are feature rich, have clients for all platforms and plug into the cluster's access control/auditing. An Operator uses the Kubernetes' extension mechanism so your custom object, eg "MongoDB", looks and acts just like the built-in Kubernetes objects.

Q: Why not use a Service Broker?
A: A Service Broker is a step towards programmatic discovery and deployment of an application but it has one flaw: it's not a long running process, so it can't execute any Day 2 operations like upgrade, failover, scale out, etc. This is the team problem with technologies like Helm. Customizations and parameterization of tunables can only be provided at install time, versus an Operator that is constantly watching your cluster's current state. Off-cluster services continue to be a good match for a Service Broker, although Operators exist for these as well.

Desired Customer Experience

Before covering the mechanics of building Operators, it's important to cover the experience that we want to enable for our customers.

First, is a cloud-like experience – the power of running something like Google BigQuery – but on any infrastructure, including bare metal and offline clusters. This means utilizing all of the Kubernetes primitives for highly available stateless and stateful services, secret management, load balancing and autoscaling. The beauty of doing this via an Operator is you gain these industry-leading capabilities without having to build them yourself or re-invent any established best practices. These capabilities are also included in every Kubernetes cluster.

Second, is embedding best practices from the experts – you – into the Operator. The goal is to allow folks to run a production quality installation without needing to be an expert in how your app is upgraded, scaled out, failed over, etc. All of this is built into your Operator. Think of your Operator as an extension of an SRE team, but embedded in the cluster that can react in milliseconds to any re-configuration or unhealthy component.

Third, software upgrades are the only way to keep software secure in the age of Heartbleed, Shellshock, et al. An Operator must understand how to transition a running app from version A to version B. Powered by the Operator Hub, these updates can be provided to all OpenShift users and their clusters. The general practice is for the Operator Lifecycle Manager to update your Operator running on the cluster, which then contains the logic to get from A to B. If you strive to make this upgrade happen in a highly available way, your users don't need to be impacted at all. This is a critical part of the cloud-like experience, getting more features, bug fixes and better scalability without having to run scripts or worse, re-install the cluster.

Fourth, is to lower the "time to first value" to be as quick as possible. Providing a free tier of your app/service and eliminating any dependencies outside of the cluster allow for cluster users to quickly try out your database, monitoring stack, etc when building out a new application. Since your Operator is super smart, it can upgrade these instances to a production or licensed configuration afterwards.

Operator Framework

The Operator Framework is a family of tools and capabilities to deliver on the customer experience describe above. It's not just about writing code. Testing, delivery and updating Operators are just as important. The Framework components consists of open source tools to tackle these problems:

Operator SDK - build, test and package Operators
Operator Lifecycle Manager - controls the install/upgrade/RBAC of Operators on a cluster
Operator Metering - collecting operational metrics about Operators on the cluster for Day 2 management and aggregating usage metrics
OperatorHub - GUI for discovering and installing Operators on one or more clusters

These tools are designed to be composable, so you can use any that are useful to you.

Categories of Operators

Operators fall into a few different categories, based on the type of applications they run:

Service Operators:
Imagine the software that powers Amazon RDS behind the scenes: installing, upgrading, monitoring and healing thousands of instances of Postgres automatically. This category helps solve the "How do I scale services without scaling humans?" and "As a vendor, how do I easily deploy my app the same way across X customers in Y environments?" problems.
Examples: MongoDB Operator, Spark Operator, Nginx Operator

Platform Operators:

Software to configure, audit and remediate changes to the platform you run. For example, a security scanning vendor could watch for any vulnerable software running and either report it, prevent it from running, or upgrade it, if desired. Most extensions to OpenShift cluster itself fall into this category.
Examples: Aggregated Logging, Security Scanning, Namespace Management

Resource Operators:
Software to manage precious resources, such as network adapters, GPUs, or out of tree kernel modules, plus the required userspace and monitoring code that comprises a full enterprise-quality deployment. An advanced Resource Operator also handles upgrades and security patching, Day 2 management and operations of resources that it is responsible for.
Examples: CNI Operators, Hardware Management Operators (see appendix), Telco/Cellular Radios

Full Solutions:
A complete software solution can combine all of these types together, eg an AI/ML product with an Operator to manage GPU resources. Everything from the UI dashboard for managing training sets to scaling out workloads that populate the machine learning models can be configured via the Operator.

Advantages of an Operator

Operators provide a model of installation and Day 2 maintenance that is not possible with other methods:

- **Pod startup ordering:** Kubernetes does not provide guarantees of container- or pod-startup ordering without sophisticated use of concepts like init containers.
- **Familiar experience on any infrastructure:** Your customers have the same deployment and day 2 experience regardless of infrastructure provider - anywhere OpenShift runs, your Operator should run.
- **Provider maintained:** De-couple your OSR from the release cadence of Kubernetes/OpenShift itself, and deliver features, bug- and security-fixes to your customers as they become available.
- **Ease of installation:** We have heard from vendors providing out-of-tree kernel modules that installation (DKMS or manual rpmbuild / similar) end up in a poor experience (even for sophisticated customers). Operators provide a one-click method for installation and operations of your hardware and software stack.
- **Standardized operational model and reduce support burden:** Operators provide a way for you to standardize deployments, upgrades and configuration to only what you support. This avoids “snowflake” deployments and ensures smooth upgrades.
- **Upgrades:** Operators allow you to encode best-practices (e.g. straight from your documentation), reducing configuration drift in customer deployments, increasing deployment success ratio and thereby reducing support costs.
- **Network adapter integrations**

    - “White list” of supported SR-IOV card/driver combinations for the device and CNI Multus plugins

Building Operators

Operators are long running programs that watch for changes in desired state, either reconfiguration caused by an administrator or a deviation based on a failure or errant change, and executes any action required to bring the current state back to the desired state.

Any programming language can be used to create an Operator. Red Hat has produced an SDK that covers most of the scaffolding required to start building an Operator, to save you time and get to market quicker.

Operators are built and run as containers. They can be plugged into any CI/CD tools or workflows you may already have for containers.

Types of Operators

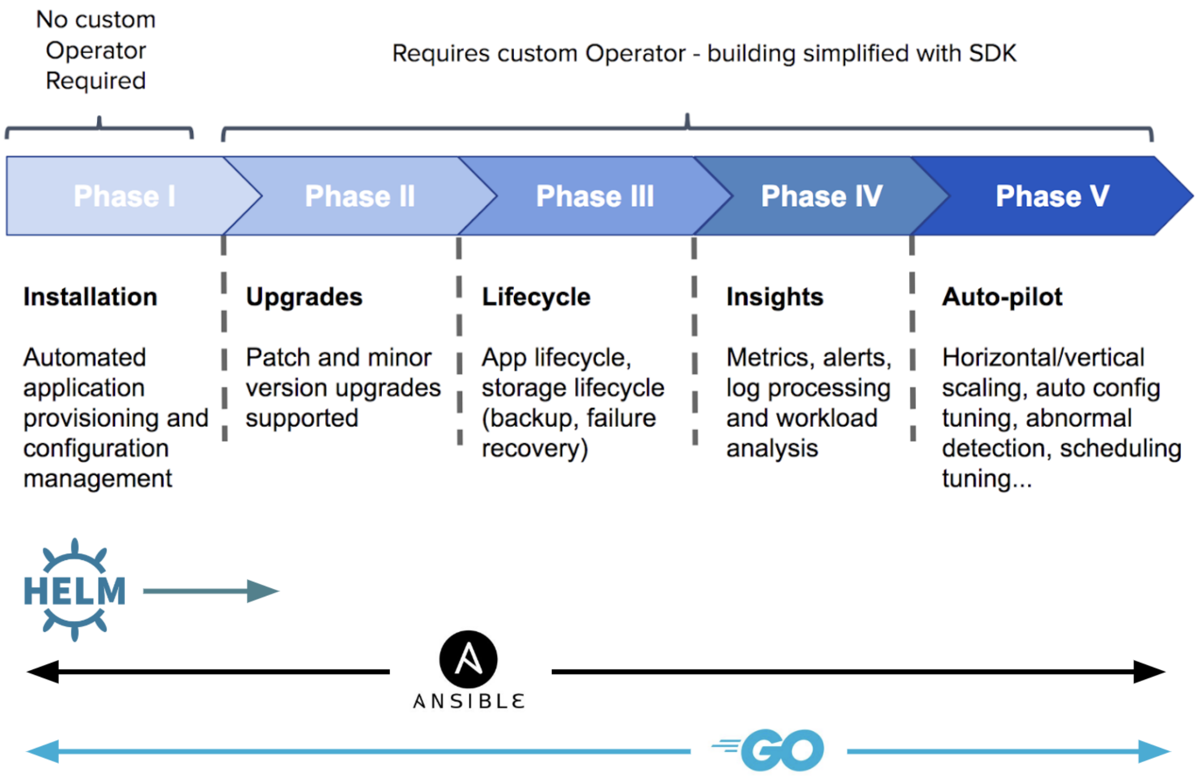
The business logic for an Operator – all of your expert knowledge – can be implemented in several pieces of technology. Each have their own SDK

Go:
Ideal for traditional software development teams that want to get to a fully auto-pilot Operator. Ability to leverage the same Kubernetes libraries the upstream projects uses under the hood. Go Getting Started guide.

Ansible
Useful for infrastructure-focused teams that have investment in Ansible modules, but want to use them in a Kubernetes-native way. Also useful for using Ansible to configure off-cluster objects like hardware load balancers. Ansible Getting Started guide.

Helm
Easiest way to get started. Useful for running Helm charts in a more secure way (no Tiller) that doesn’t rely on manual invocation of Helm to reconfigure your apps. Helm Getting Started guide.

Other Languages
Operators have been built in many languages, including Python and Java, but no SDKs exist for these.


No custom Operator Required
Requires custom Operator - building simplified with SDK
Phase I
Phase II
Phase III
Phase IV
Phase V
Installation
Automated application provisioning and configuration management
Upgrades
Patch and minor version upgrades supported
Lifecycle
App lifecycle, storage lifecycle (backup, failure recovery)
Insights
Metrics, alerts, log processing and workload analysis
Auto-pilot
Horizontal/vertical scaling, auto config tuning, abnormal detection, scheduling tuning...
HELM
ANSIBLE
GO

https://github.com/operator-framework/operator-sdk/blob/master/doc/images/Operator-Maturity-Model.png

Operator SDK

The Operator SDK is a CLI tool and set of pre-generated code to help you quickly build an Operator that handles install, upgrade, full lifecycle control and more. The SDK's goal is to require an Operator author to provide the business logic of your application and everything else is handled for you.

[Interactive Operator SDK tutorial on Katacoda]()

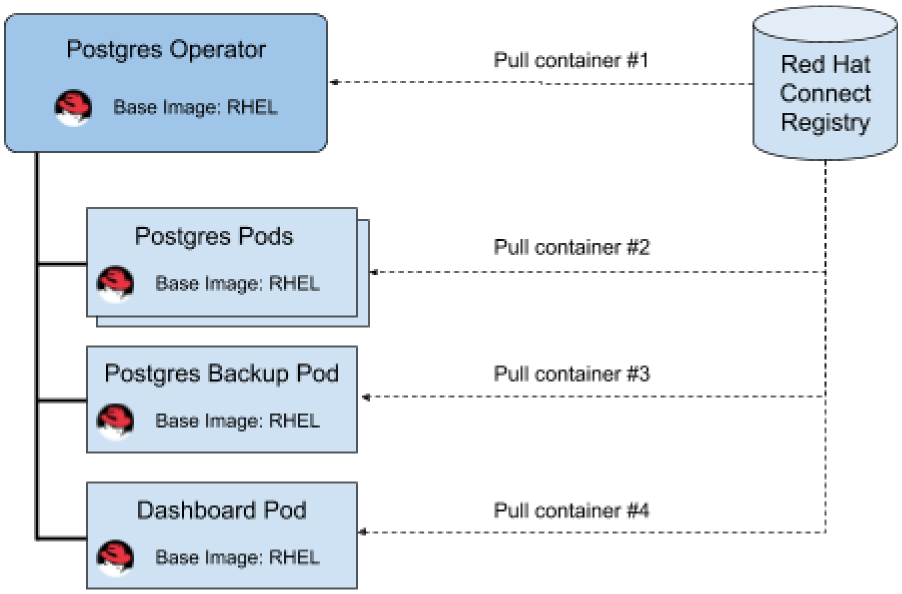
Base Images

Red Hat is serious about security for Operators and using a Red Hat supported base image is required for inclusion in the OpenShift OperatorHub. RHEL and OpenShift have many security certifications, which require RHEL usage throughout the entire stack for compatibility. As a partner you have access to licenses for OpenShift and RHEL for testing.

Before 2019, redistributing a RHEL base image was not allowed. A new "universal base image" is available with relaxed restrictions, to allow you to used any CI tool to build your Operator and store your Operator images within any registry.

Community Operators may use any base image, but it is highly suggested to use the RHEL universal base image for the best security.

All of the containers used by your certified Operator (described below), including the Operator and all images that it deploys, must use a RHEL base image and be distributed from the Red Hat partner registry. This ensures that Red Hat (and you by extension) can comply with legal export compliance. Below is an example of an Operator that uses 4 total images:



Each of the 4 containers shown above must also remain published and accessible forever, so that OpenShift clusters can continue to start these workloads as Kubernetes moves them around the cluster.

Operator Scorecard

The main test for if your Operator is compatible with OpenShift is to use our Operator Scorecard. This is an automated test you can run locally and a test that we will use as part of our tests of your Operator. Here's a summary of the tests:

- Basic Tests
  - Spec Block Exists
  - Status Block Exists
  - Operator Action Are Reflected In Status
  - Writing Into CRs Has An Effect
- OLM Compatibility Tests
  - Owned CRDs Have Resources Listed
  - CRs Have At Least 1 Example
  - Spec Fields With Descriptors

- Status Fields With Descriptors

[Read more about the Scorecard]

Distributing an Operator

There are two streams to distribute an Operator, based on its maturity. Red Hat Certified Operators are listed in OpenShift by default, while Community Operators are optionally shown by the cluster admin.

Community Operators

Community Operators are maintained by the relevant representatives on the community GitHub repo. These repo holds all of the manifests required to start the Operator under OLM. There is not official support promised for these Community Operators, but you can add links to documentation, issue trackers and other relevant materials.

The following steps are required to ship a new version of an Operator.

1. Test the Operator with OLM installed on a Kubernetes cluster
2. Submit a PR to the community Github repo with all changes required for the new version
    a. Create a new CSV file with the version
    b. If it replaces an existing version, ensure it uses the `replaces: foo.v1.0.1` parameter
    c. Add any new CRDs that are required for new functionality
    d. Update any existing CRDs if their API version has changed
3. PR will be reviewed and tested automatically for technical compatibility
4. PR is merged, at which point the new version will show up in OpenShift.
5. Existing installations will be notified of the new version via Operator Hub. These clusters:
    a. Must be connected to the internet
    b. Must have installed a previous version of the Operator (`replaces` from above) that matches a channel
    c. May upgrade your Operator automatically if configured to do so

View the readme on the community repo for more info

Red Hat Partners

Red Hat Partners have special status due to their backing of the Operator(s) with formal support, testing and joint go-to-market activities. Red Hat customers are some of the largest enterprises in the world, and Red Hat wants to provide high quality experiences with your product at the center.

Partners should register for Red Hat Connect, which includes test product licenses and an auto-build feature which you can connect to your code repositories. You may also connect an existing CI pipeline you may use.

A formal support escalation path must exist for OpenShift customers with established SLAs between Red Hat and the partner. Red Hat uses a service called TSANet to exchange joint support cases. Using TSANet is free for the Partner.

For each new version of your Operator, you will submit both the container images and the OLM manifests required to install your Operator on a cluster. These assets will be used to automatically scan, test and publish your product. The following will occur for each new version:

6. Test the Operator yourself with OpenShift
7. Submit a PR to your code repo with all changes required for the new version. This can be done along any release-related PR on your Operator code.
    a. Create a new CSV file with the version
    b. If it replaces an existing version, ensure it uses the `replaces: foo.v.1.0.1` parameter
    c. Add any new CRDs that are required for new functionality
    d. Update any existing CRDs if their API version has changed
8. New version will be reviewed and tested automatically for technical compatibility
9. Automated testing passes and the Operator is published
10. Existing installations will be notified of the new version via Operator Hub. These clusters:
    a. Must be connected to the internet
    b. Must have installed a previous version of the Operator (`replaces` from above) that matches a channel
    c. May upgrade your Operator automatically if configured to do so

Register for Red Hat Connect

Certification Requirements

Red Hat Partners will need to meet both Certified Container (the Operator image itself) and Certified Operator requirements. These are described below, and Red Hat is happy to walk you through this process.

Certified Container Image

	Certified
Onboarding	
Provider is a member of TSANet	Required [1]
Provider is a commercial entity	Required
Provider agrees to program legal terms	Required
Provider passes approval of RH export controls	Required [2]
Initial Certification	
Container image is built from a healthy RHEL/RH Base Image	Required [3]
Container image is scanned and verified by Red Hat	Required
Container image includes labels according to program requirements	Required
Container image does not run as root	Required
Container image includes appropriate license file	Required
Provider provides metadata required for Container Catalog listing	Required
All dependent images are also certified or from Red Hat	Required
Image health grade must be B or higher to pass initial certification	Required
Provider uses Red Hat's certified image build service	Optional
Provider uses Red Hat's certified image auto-rebuild service	Optional
Container image is commercially supported by partner	Required [4]
Ongoing Certification	
Image grades must sustain a B or higher	Required

[1] Not initially required but will be added as a requirement

[2] if distributed through a Red Hat managed registry
[3] Will be changed to UBI requirement once UBI is available
[4] We currently allow both community and commercial content from partners. In the future, images labeled not community should be commercially supported.

Certified Operator

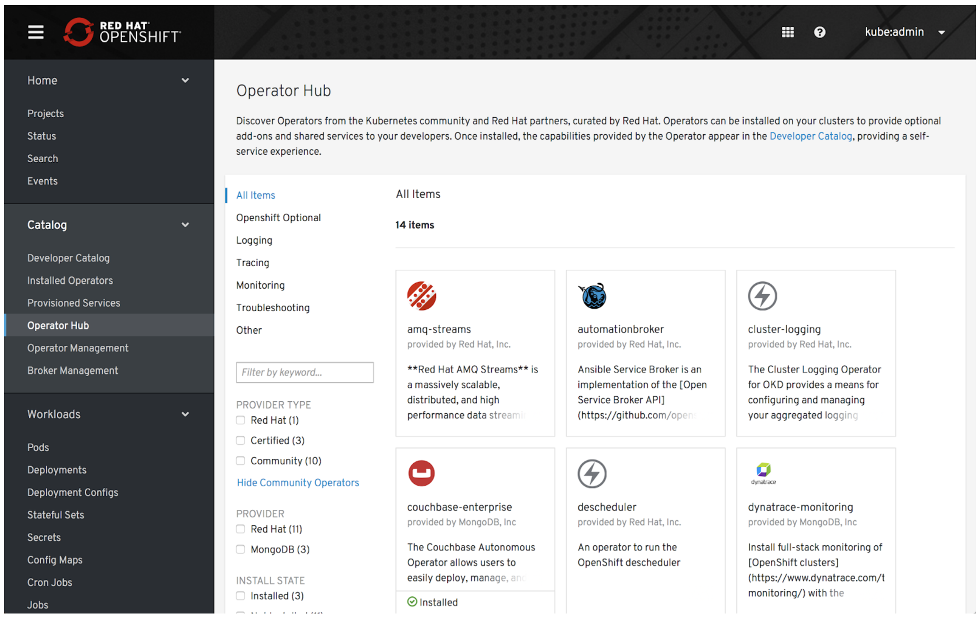
	Community	Certified
Onboarding		
Provider is a member of TSANet	Not Required	Required
Provider is a commercial entity	Not Required	Required
Provider agrees to program legal terms	Not Required	Required
Provider passes approval of Red Hat export controls	Not Required	Required [1]
Initial Certification		
Operator is built from a healthy RHEL/Red Hat Base Image	Not Required	Required [2]
Operator is scanned and verified by Red Hat	Not Required	Required
Operator is free of know vulnerabilities (Health grade A or B)	Not Required	Required
Operator includes labels according to program requirements	Not Required	Required
Operator includes metadata required for OperatorHub listing	Required	Required
Provider provides instructions for obtaining support	Not Required	Required
Operator includes appropriate license file	Not Required	Required
All dependent images are also certified or from Red Hat	Not Required	Required
Operator includes functional tests that can be verified on specified OpenShift version(s)	Not Required	Required
Operator can be installed via Operator Lifecycle Manager	Required	Required

Operator can be updated via Operator Lifecycle Manager	Required	Required
Operator is commercially supported by partner	Not Required	Required

[1] if distributed through a Red Hat managed registry
[2] Will be changed to Universal Bse Image (UBI) requirement once available

OperatorHub

OperatorHub is the graphical interface that OpenShift admins use to discover, install and upgrade Operators. With one click, these Operators can be installed on the cluster, ready for engineering teams to self-service manage your product in dev, test and prod environments.



Operator Hub

Discover Operators from the Kubernetes community and Red Hat partners, curated by Red Hat. Operators can be installed on your clusters to provide optional add-ons and shared services to your developers. Once installed, the capabilities provided by the Operator appear in the Developer Catalog, providing a self-service experience.

All Items

14 items

amq-streams
provided by Red Hat, Inc.
Red Hat AMQ Streams is a massively scalable, distributed, and high performance data streami

automationbroker
provided by Red Hat, Inc.
Ansible Service Broker is an implementation of the [Open Service Broker API] (https://github.com/open

cluster-logging
provided by Red Hat, Inc.
The Cluster Logging Operator for OKD provides a means for configuring and managing your aggregated logging

couchbase-enterprise
provided by MongoDB, Inc
The Couchbase Autonomous Operator allows users to easily deploy, manage, and
Installed

descheduler
provided by Red Hat, Inc.
An operator to run the OpenShift descheduler

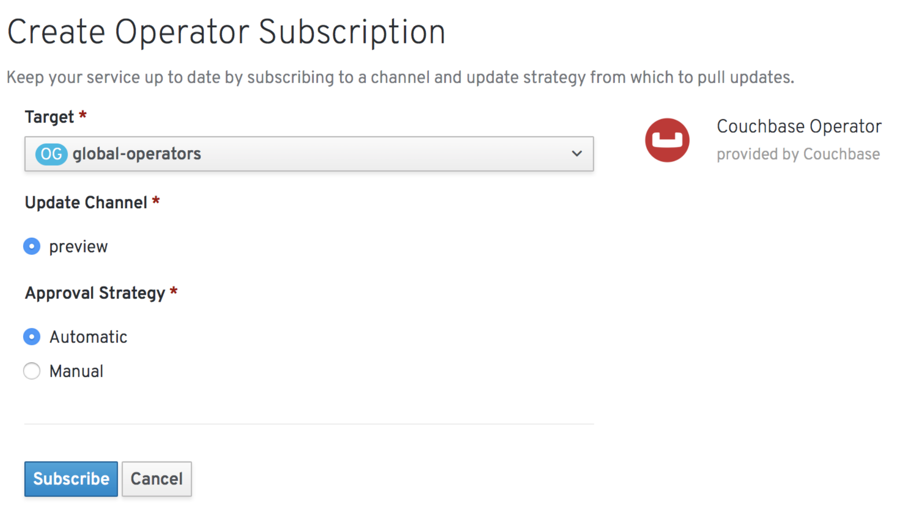
dynatrace-monitoring
provided by MongoDB, Inc
Install full-stack monitoring of [OpenShift clusters] (https://www.dynatrace.com/t monitoring/) with the

OperatorHub in OpenShift 4.0

After your Operator is approved by Red Hat, it shows up immediately in the OperatorHub for all OpenShift admins to discover and install.

Shipping Operator Updates

The OperatorHub serves as a conduit into the OpenShift clusters that have installed your Operator. When you indicate that a new version of your Operator replaces an older one, these clusters can update the Operator automatically (or per approval). This allows you to quickly release new features and bug fixes directly.


Create Operator Subscription

Keep your service up to date by subscribing to a channel and update strategy from which to pull updates.

Target *: global-operators

Update Channel *: [✓ preview]

Approval Strategy *: [✓ Automatic] Manual

Options for manual or automatic upgrades in OpenShift 4.0

Of course, this is powerful and must be done correctly. Here's a few tips to think about:

- Your Operator must understand past and future versions of your application in order to seamlessly transition from one to another, eg. a new feature (backed by a new component) was introduced in v2, so any v1 users need to have that component installed and configured by v2 Operator.
- Migrations always need to be done for long lived application installs, eg. if you change the format of a ConfigMap, the Operator needs to migrate that and keep it up to date.

Next Steps

Red Hat is here to help you on this journey, please contact us to assistance and questions with your Operator engineering efforts. Here is a quick set of next steps:

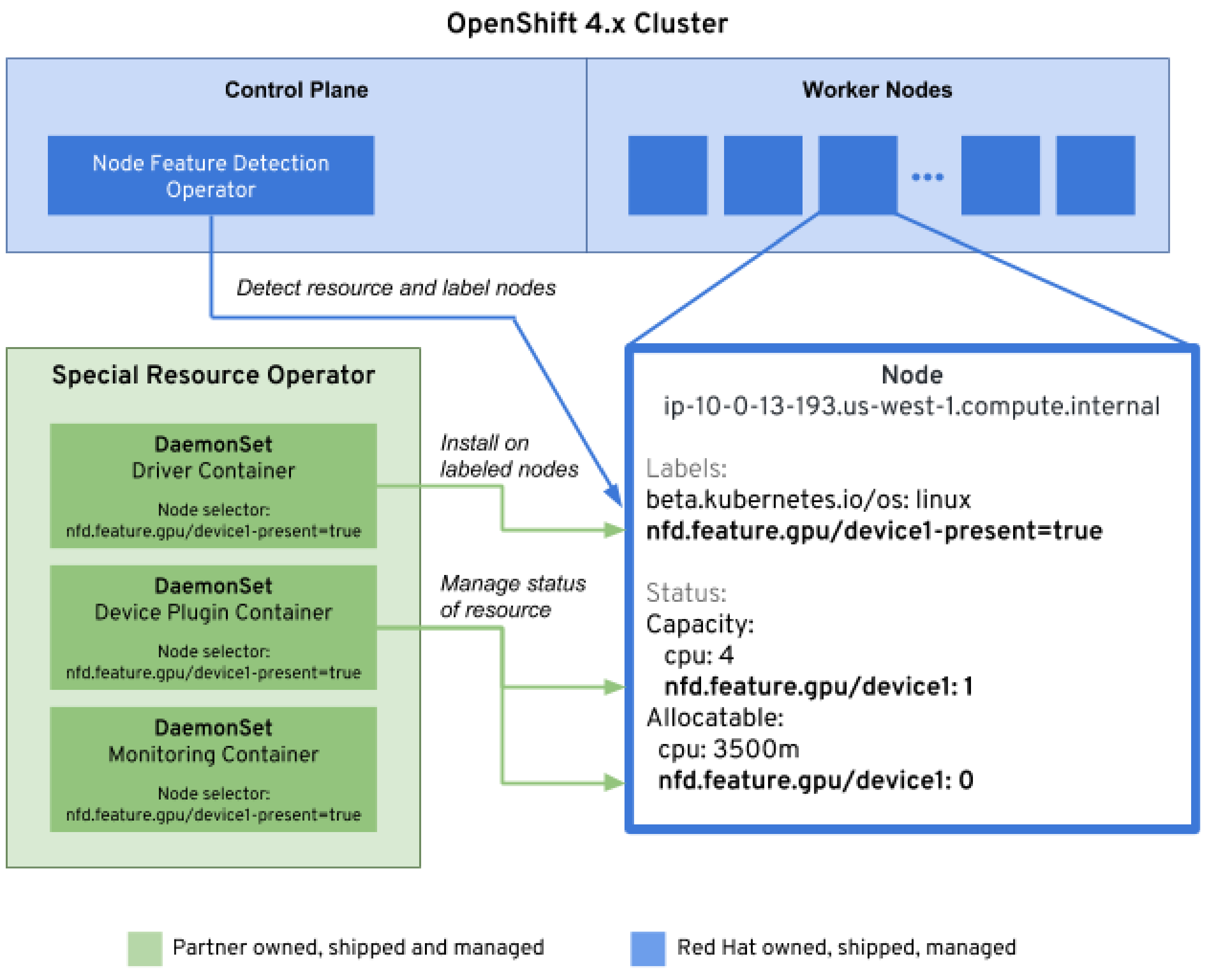
1. Download the Operator SDK tool
2. Build an initial proof-of-concept Operator
3. Test against a live cluster
4. Run the scorecard against your Operator
5. Ship your first Operator release
6. Submit your Operator
    a. Community Repo
    b. Red Hat Certification

Appendix

Example Special Resource Operators Workflow

Below shows the components of an Operator that manages GPU resources. Feature detection is done centrally across the cluster, and a Partner's Operator only needs to key off of these labels to deploy their software. In this example, the Operator is responsible for:

- Deploy a container via DaemonSet that installs any required drivers
- Deploy a container via DaemonSet that manages the device
- Deploy a container via DaemonSet that monitors the device and reports Prometheus metrics



OpenShift 4.x Cluster
Control Plane
Node Feature Detection Operator
Worker Nodes
...
Detect resource and label nodes
Special Resource Operator
DaemonSet
Driver Container
Node selector:
nfd.feature.gpu/device1-present=true
DaemonSet
Device Plugin Container
Node selector:
nfd.feature.gpu/device1-present=true
DaemonSet
Monitoring Container
Node selector:
nfd.feature.gpu/device1-present=true
Install on labeled nodes
Manage status of resource
Node
ip-10-0-13-193.us-west-1.compute.internal
Labels:
beta.kubernetes.io/os: linux
nfd.feature.gpu/device1-present=true
Status:
Capacity:
cpu: 4
nfd.feature.gpu/device1: 1
Allocatable:
cpu: 3500m
nfd.feature.gpu/device1: 0
Partner owned, shipped and managed
Red Hat owned, shipped, managed

Any cluster workloads that require a GPU can be steered towards these Nodes by the same label selector on their Deployments and other Kubernetes resources.